

StarRocks 2.5

Copyright (c) 2025 The Linux Foundation

StarRocks

StarRocks is a next-gen, high-performance analytical data warehouse that enables real-time, multi-dimensional, and highly concurrent data analysis. StarRocks has an MPP architecture and is equipped with a fully vectorized execution engine, a columnar storage engine that supports real-time updates, and is powered by a rich set of features including a fully-customized cost-based optimizer (CBO), intelligent materialized view and more. StarRocks supports real-time and batch data ingestion from a variety of data sources. It also allows you to directly analyze data stored in data lakes with zero data migration.

StarRocks is also compatible with MySQL protocols and can be easily connected using MySQL clients and popular BI tools. StarRocks is highly scalable, available, and easy to maintain. It is widely adopted in the industry, powering a variety of OLAP scenarios, such as real-time analytics, ad-hoc queries, data lake analytics and more.

StarRocks is licensed under Apache 2.0, available at the StarRocks GitHub repository (see the StarRocks license). StarRocks (i) links to or calls functions from third party software libraries, the licenses of which are available in the folder licenses-binary; and (ii) incorporates third party software code, the licenses of which are available in the folder licenses.

Join our web forum for asking general questions. Join our Slack channel for chat. For community news, read the StarRocks.io Blog. You can also follow us on LinkedIn to get first-hand updates on new features, events, and sharing.

Popular topics

Introduction

OLAP, features, architecture

Quick Start

Get up and running quickly.

Data Loading

Clean, transform, and load

Table Design

Tables, indexing, acceleration

Data Lakes

Iceberg, Hive, Delta Lake, ...

Work with semi-structured data

JSON, map, struct, array

Integrations

BI tools, IDEs, Cloud authentication, ...

Administration

Scale, backups, roles and privileges, ...

Reference

SQL, functions, error codes, ...

FAQs

Frequently asked questions.

Benchmarks

DB performance comparison benchmarks.

Introduction

What is StarRocks?

StarRocks is a next-generation, blazing-fast massively parallel processing (MPP) database designe...

Architecture

StarRocks has a simple architecture. The entire system consists of only two types of components, ...

Features

StarRocks offers a rich set of features to deliver a blazing-fast, real-time analytics experience on d...

What is StarRocks?

StarRocks is a next-generation, blazing-fast massively parallel processing (MPP) database designed to make real-time analytics easy for enterprises. It is built to power sub-second queries at scale.

StarRocks has an elegant design. It encompasses a rich set of features including fully vectorized engine, newly designed cost-based optimizer (CBO), and intelligent materialized view. As such, StarRocks can deliver a query speed far exceeding database products of its kind, especially for multi-table joins.

StarRocks is ideal for real-time analytics on fresh data. Data can be ingested at a high speed and updated and deleted in real time. StarRocks empowers users to create tables that use various schemas, such as flat, star, and snowflake schemas.

Compatible with MySQL protocols and standard SQL, StarRocks has out-of-the-box support for all major BI tools, such as Tableau and Power BI. StarRocks does not rely on any external components. It is an integrated data analytics platform that allows for high scalability, high availability, and simplified management and maintenance.

Scenarios

StarRocks meets varied enterprise analytics requirements, including OLAP multi-dimensional analytics, real-time analytics, high-concurrency analytics, customized reporting, ad-hoc queries, and unified analytics.

OLAP multi-dimensional analytics

The MPP framework and vectorized execution engine enable users to choose between various schemas to develop multi-dimensional analytical reports. Scenarios:

- User behavior analysis
- User profiling, label analysis, user tagging
- High-dimensional metrics report

- Self-service dashboard
- Service anomaly probing and analysis
- Cross-theme analysis
- Financial data analysis
- System monitoring analysis

Real-time analytics

StarRocks uses the Primary Key table to implement real-time updates. Data changes in a TP database can be synchronized to StarRocks in a matter of seconds to build a real-time warehouse.

Scenarios:

- Online promotion analysis
- Logistics tracking and analysis
- Performance analysis and metrics computation for the financial industry
- Quality analysis for livestreaming
- Ad placement analysis
- Cockpit management
- Application Performance Management (APM)

High-concurrency analytics

StarRocks leverages performant data distribution, flexible indexing, and intelligent materialized views to facilitate user-facing analytics at high concurrency:

- Advertiser report analysis
- Channel analysis for the retail industry
- User-facing analysis for SaaS

- Multi-tabbed dashboard analysis

Unified analytics

StarRocks provides a unified data analytics experience.

- One system can power various analytical scenarios, reducing system complexity and lowering TCO.

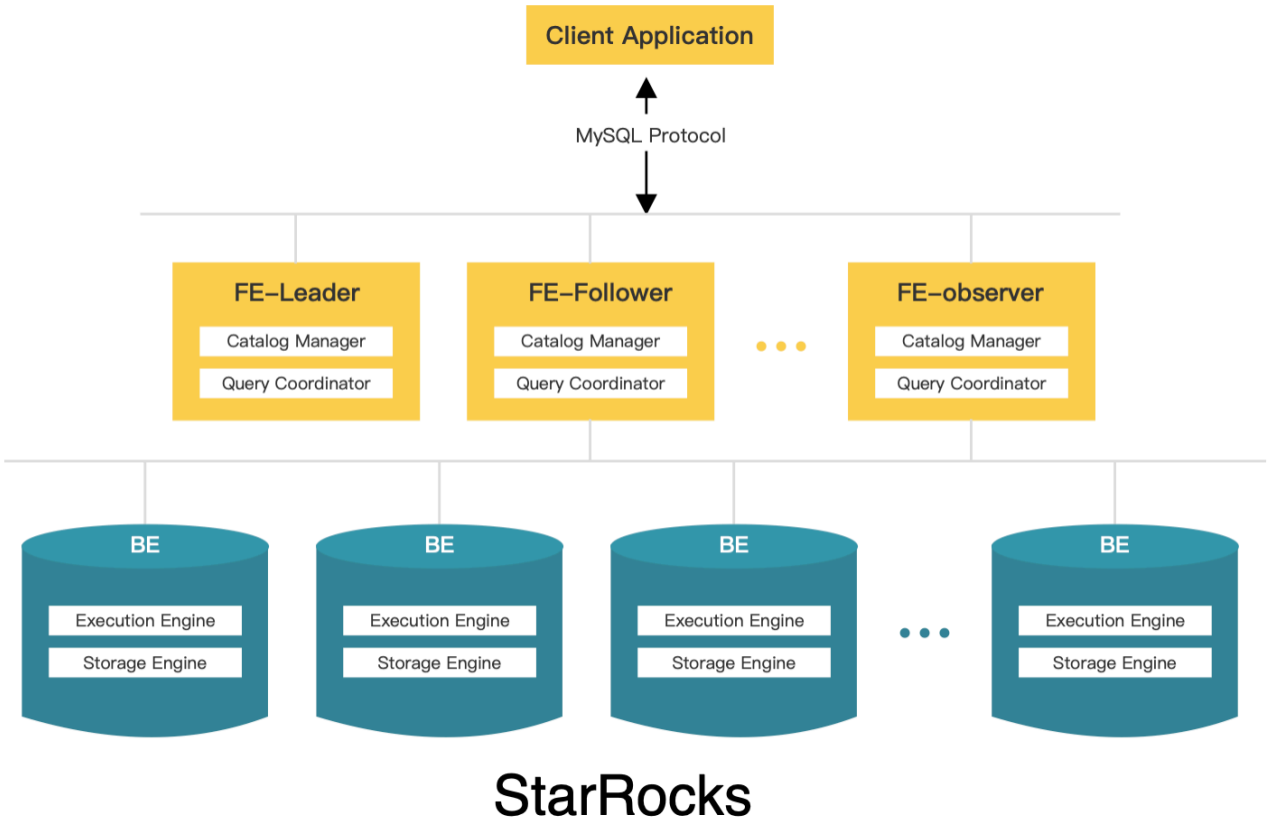
- StarRocks unifies data lakes and data warehouses. Data in a lakehouse can be managed all in StarRocks. Latency-sensitive queries that require high concurrency can run on StarRocks. Data in data lakes can be accessed by using external catalogs or external tables provided by StarRocks.

Architecture

StarRocks has a simple architecture. The entire system consists of only two types of components, frontends (FEs) and backends (BEs). StarRocks does not rely on any external components, simplifying deployment and maintenance. FEs and BEs can be horizontally scaled without service downtime. In addition, StarRocks has a replica mechanism for metadata and service data, which increases data reliability and efficiently prevents single points of failure (SPOFs).

StarRocks is compatible with MySQL protocols and supports standard SQL. Users can easily connect to StarRocks from MySQL clients to gain instant and valuable insights.

The following figure shows the architecture of StarRocks.



FE and BE

FEs are responsible for metadata management, client connection management, query planning, and query scheduling. Each FE stores and maintains a complete copy of metadata in its memory, which guarantees indiscriminate services among the FEs. FEs can work as the leader, followers, and observers. Followers can elect a leader according to the Paxos-like BDB JE protocol. BDB JE is short for Berkeley DB Java Edition.

- Leader
    - The leader FE is elected from follower FEs. To perform leader election, more than half of the follower FEs in the cluster must be active. When the leader FE fails, follower FEs will start another round of leader election.
    - The leader FE reads and writes metadata. Follower and observer FEs can only read metadata. They route metadata write requests to the leader FE. The leader FE updates the metadata and then uses BDE JE to synchronize the metadata changes to the follower and observer FEs. Data writes are considered successful only after the metadata changes are synchronized to more than half of the follower FEs.
- Follower
    - Followers can only read metadata. They synchronize and replay logs from the leader FE to update metadata.
    - Followers participate in leader election, which requires more than half of the followers in the cluster be active.
- Observer
    - Observers are mainly used to increase the query concurrency of the cluster.
    - Observers do not participate in leader election and therefore, will not add leader selection pressure to the cluster.
    - Observers synchronize and replay logs from the leader FE to update metadata.

BEs are responsible for data storage and SQL execution.

- Data storage: BEs have equivalent data storage capabilities. FEs distribute data to BEs based on predefined rules. BEs transform the ingested data, write the data into the required format, and generate indexes for the data.
- SQL execution: When an SQL query arrives, FEs parse it into a logical execution plan according to the semantics of the query, and then transform the logical plan into physical execution plans that can be executed on BEs. BEs that store the destination

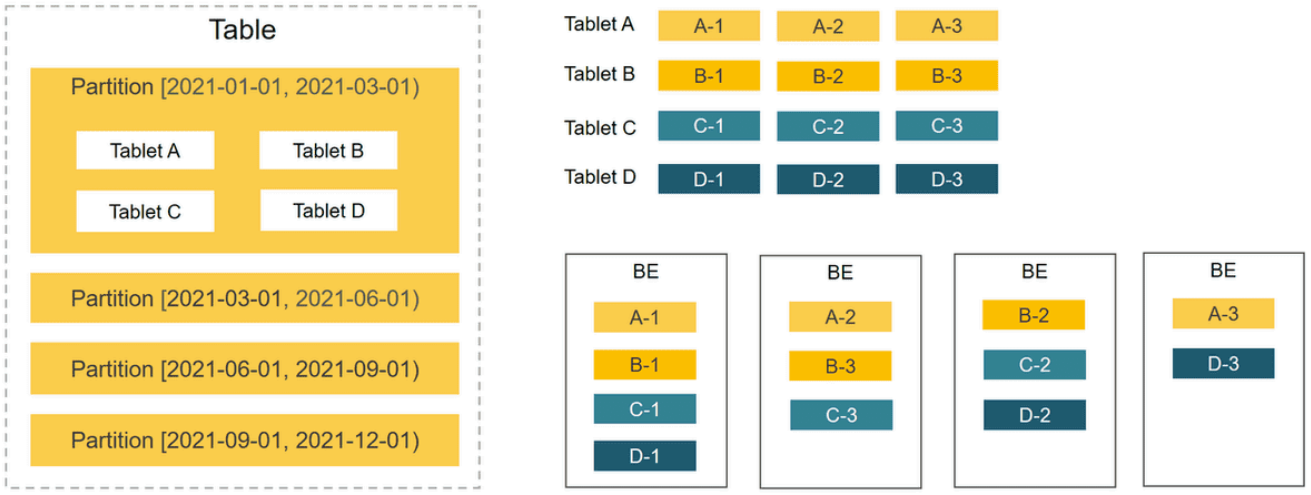
data execute the query. This eliminates the need for data transmission and copy, achieving high query performance.

Data management

StarRocks is a column-oriented database system. It uses the partitioning and bucketing mechanism to manage data. Data in a table is first divided into multiple partitions and then into multiple tablets. Tablets are the basic logical units of data management in StarRocks. Each tablet can have multiple replicas that can be stored across different BEs. You can specify the number of tablets and leave StarRocks to take care of the tablets.

Partitions and tablets reduce table scans and increase query concurrency. Replicas facilitate data backup and restoration, preventing data loss.

In the following figure, the table is divided into four partitions based on time. Data in the first partition is further split into four tablets. Each tablet has three replicas, which are stored on three different BEs.



Since one table is split into multiple tablets, StarRocks can schedule one SQL statement to all the tablets for parallel processing, fully utilizing the computing power of multiple physical machines and cores. This also helps offload the query pressure to multiple nodes, increasing service availability. You can add physical machines on demand to achieve high concurrency.

The distribution of tablets is not affected or confined by physical nodes. If the number of BEs changes (for example, when you add or remove a BE), ongoing services can proceed without any interruption. Node changes will trigger automatic migration of tablets. If BEs are

added, some tablets will be automatically migrated to the new BEs for more even data distribution. If BEs are removed, tablets on these BEs will be automatically migrated to other BEs, ensuring unchanged number of replicas. Auto tablet migration helps easily achieve auto scaling of StarRocks clusters, eliminating the need for manual data re-distribution.

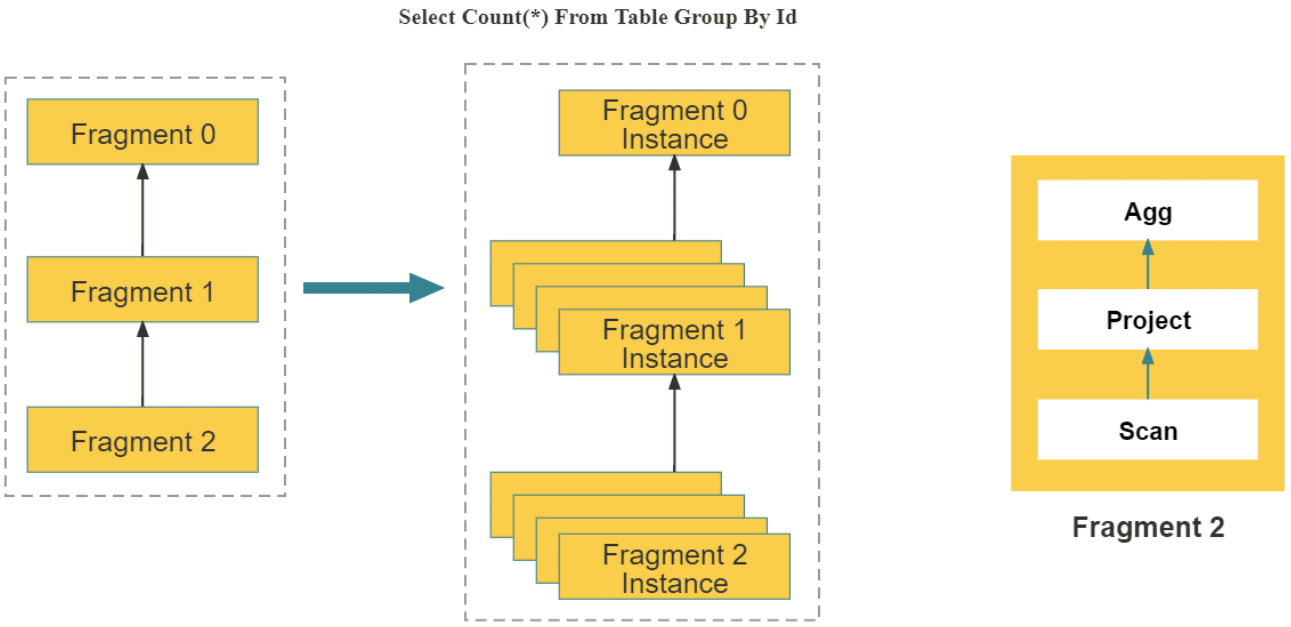
StarRocks uses a multi-replica mechanism (3 by default) for tablets. Replicas ensure high data reliability and service availability. The failure of one node will not affect the availability of the overall services. You can also increase the number of replicas to achieve high query concurrency.

Features

StarRocks offers a rich set of features to deliver a blazing-fast, real-time analytics experience on data at scale.

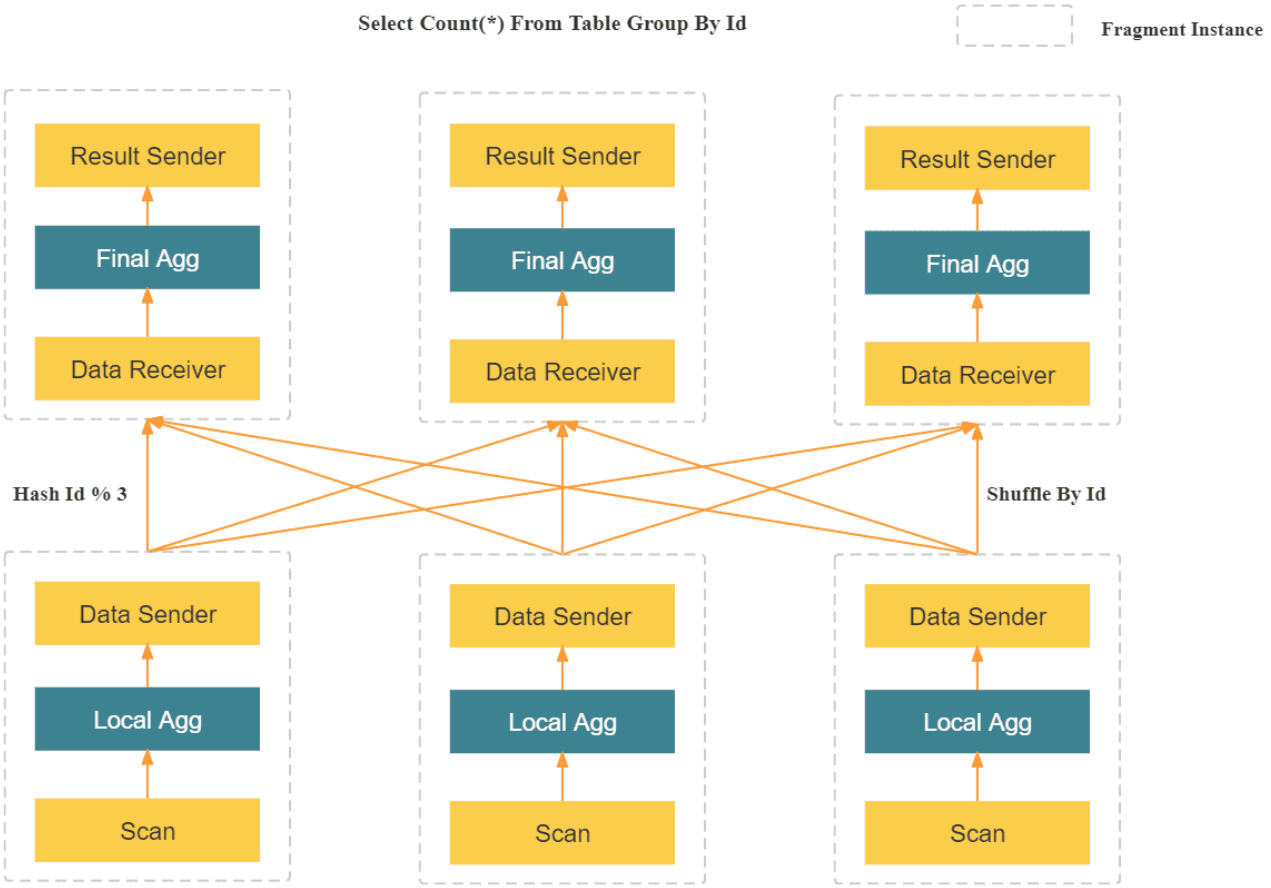
MPP framework

StarRocks adopts the massively parallel processing (MPP) framework. One query request is split into multiple physical computing units that can be executed in parallel on multiple machines. Each machine has dedicated CPU and memory resources. The MPP framework fully uses the resources of all CPU cores and machines. The performance of a single query can continuously increase as the cluster is scaled out.



In the preceding figure, StarRocks parses an SQL statement into multiple logical execution units (query fragments) according to the semantics of the statement. Each fragment is then implemented by one or multiple physical execution units (fragment instances) based on the computing complexity. A physical execution unit is the smallest scheduling unit in StarRocks. They will be scheduled to backends (BEs) for execution. One logical execution unit can contain one or more operators, such as the Scan, Project, and Agg operators, as shown on the right side of the figure. Each physical execution unit processes only part of

the data and the result will be merged to generate the final data. **Parallel execution of logical execution units fully utilizes the resources of all CPU cores and physical machines and accelerates the query speed.**



Unlike the Scatter-Gather framework used by many other data analytics systems, the MPP framework can utilize more resources to process query requests. In the Scatter-Gather framework, only the Gather node can perform the final merge operation. In the MPP framework, data is shuffled to multiple nodes for merge operations. For complex queries, such as Group By on high-cardinality fields and large table joins, StarRocks' MPP framework has noticeable performance advantages over the Scatter-Gather framework.

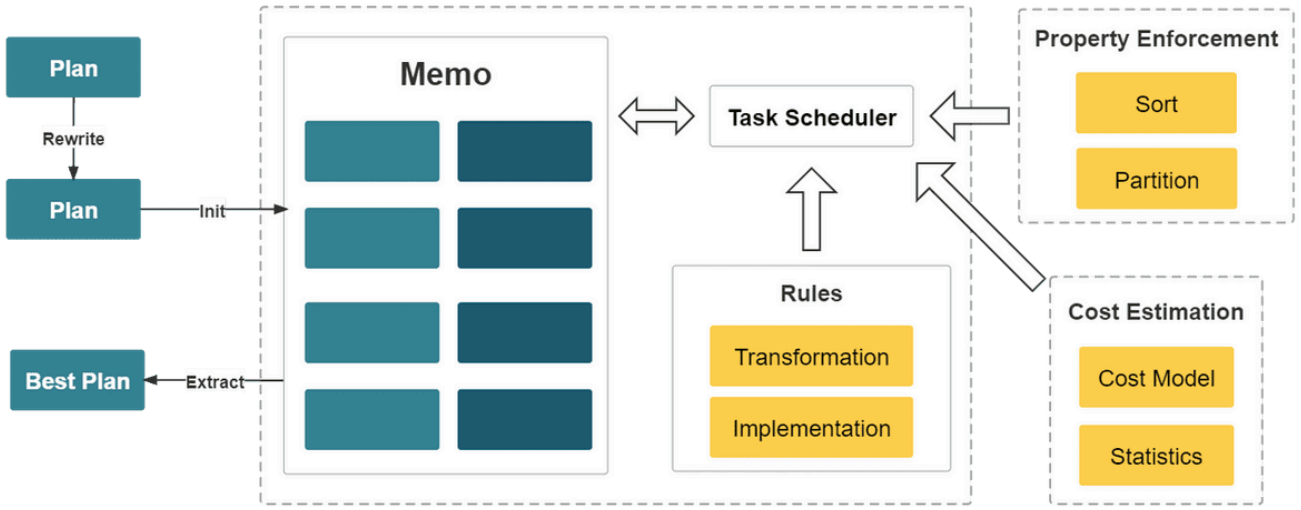
Fully vectorized execution engine

The fully vectorized execution engine makes more efficient use of CPU processing power because this engine organizes and processes data in a columnar manner. Specifically, StarRocks stores data, organizes data in memory, and computes SQL operators all in a columnar manner. Columnar organization makes full use of CPU cache. Columnar computing reduces the number of virtual function calls and branch judgments, resulting in more sufficient CPU instruction flows.

The vectorized execution engine also makes full use of SIMD instructions. This engine can complete more data operations with fewer instructions. Tests against standard datasets show that this engine enhances the overall performance of operators by 3 to 10 times.

In addition to operator vectorization, StarRocks has implemented other optimizations for the query engine. For example, StarRocks uses the Operation on Encoded Data technology to directly execute operators on encoded strings, without the need for decoding. This noticeably reduces SQL complexity and increases the query speed by more than 2 times.

Cost-based optimizer



Performance of multi-table join queries is difficult to optimize. Execution engines alone cannot deliver superior performance because the complexity of execution plans may vary by several orders of magnitude in multi-table join query scenarios. The more the associated tables, the more the execution plans, which makes it NP-hard to choose an optimal plan. Only a query optimizer excellent enough can choose a relatively optimal query plan for efficient multi-table analytics.

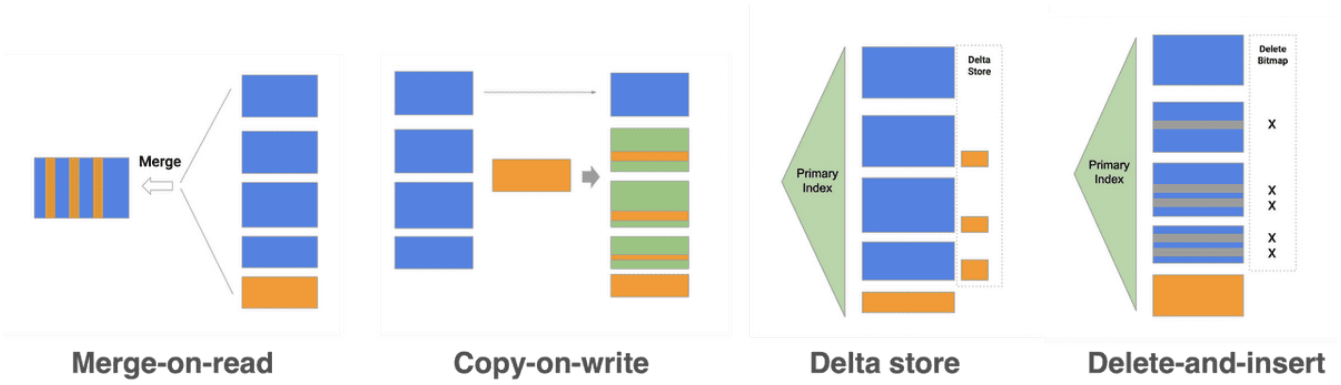
StarRocks designs a brand-new CBO from scratch. This CBO adopts the cascades-like framework and is deeply customized for the vectorized execution engine with a number of optimizations and innovations. These optimizations include the reuse of common table expressions (CTEs), rewriting of subqueries, Lateral Join, Join Reorder, strategy selection for distributed Join execution, and low-cardinality optimization. The CBO supports a total of 99 TPC-DS SQL statements.

The CBO enables StarRocks to deliver better multi-table join query performance than competitors, especially in complex multi-table join queries.

Real-time, updatable columnar storage engine

StarRocks is a columnar storage engine that allows data of the same type to be stored continuously. In columnar storage, data can be encoded in a more efficient way, increasing the compression ratio and lowering the storage cost. Columnar storage also reduces the total data read I/Os, improving query performance. In addition, in most OLAP scenarios, only specific columns are queried. Columnar storage enables users to query only part of the columns, significantly reducing disk I/Os.

StarRocks can load data within seconds for near-real-time analytics. StarRocks' storage engine guarantees the atomicity, consistency, isolation, and durability (ACID) of each data ingestion operation. For a data loading transaction, the entire transaction either succeeds or fails. Concurrent transactions do not affect each other, providing transaction-level isolation.



StarRocks' storage engine uses the Delete-and-insert pattern, which allows for efficient Partial Update and Upsert operations. The storage engine can quickly filter data using primary key indexes, eliminating the need for Sort and Merge operations at data reading. The engine can also make full use of secondary indexes. It delivers fast and predictable query performance even on huge volume of data updates.

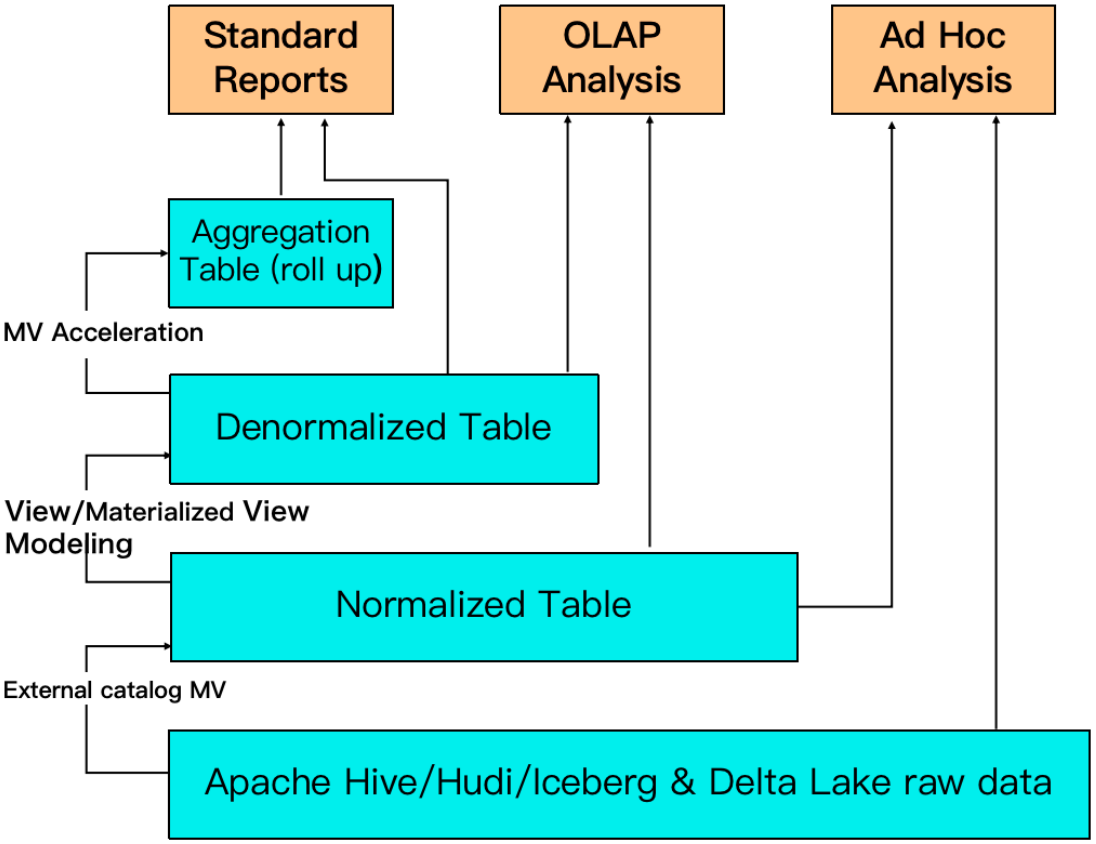
Intelligent materialized view

StarRocks uses intelligent materialized views to accelerate queries and data warehouse layering. Different from materialized views of other similar products that requires manual data synchronization with the base table, StarRocks' materialized views automatically update data according to the data changes in the base table without requiring additional maintenance operations. In addition, the selection of materialized views is also automatic. If StarRocks identifies a suitable materialized view (MV) to improve query performance, it will

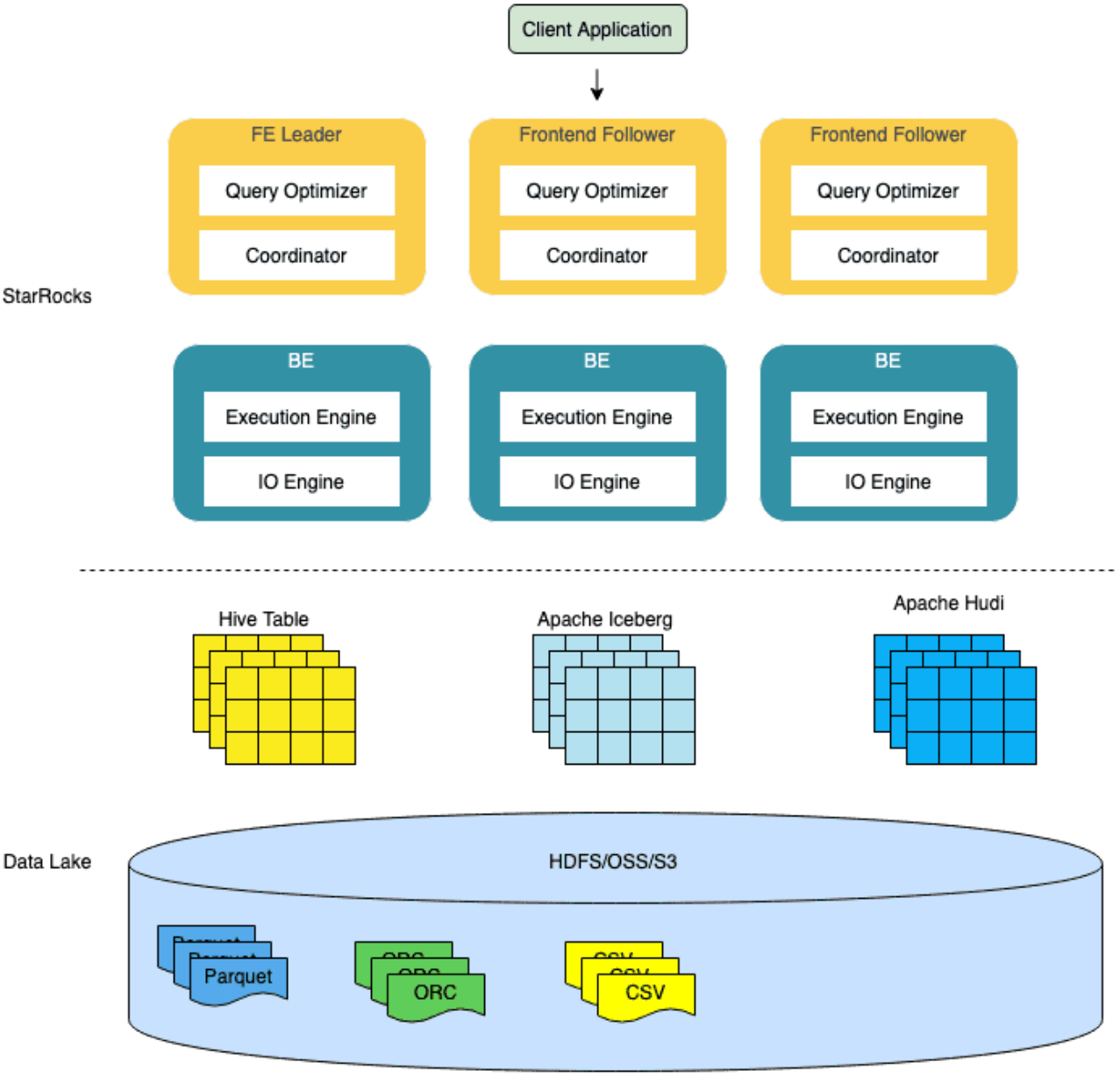
automatically rewrite the query to utilize the MV. This intelligent process significantly enhances query efficiency without requiring manual intervention.

StarRocks' MV can replace the traditional ETL data modeling process: Instead of transforming data in the upstream applications, you now have the option to transform data with MV within StarRocks, simplifying the data processing pipeline.

For example, in the figure, raw data on data lake can be used to create a normalized table based on an external MV. A denormalized table can be created from normalized tables through asynchronous materialized views. Another MV can be created from normalized tables to support high-concurrency queries and better query performance.



Data lake analytics



In addition to efficient analytics of local data, StarRocks can work as the compute engine to analyze data stored in data lakes such as Apache Hive, Apache Iceberg, Apache Hudi, and Delta Lake. One of the key features of StarRocks is its external catalog, which acts as a linkage to an externally maintained metastore. This functionality provides users with the capability to query external data sources seamlessly, eliminating the need for data migration. As such, users can analyze data from different systems such as HDFS and Amazon S3, in various file formats such as Parquet, ORC, and CSV, etc.

The preceding figure shows a data lake analytics scenario where StarRocks is responsible for data computing and analysis, and the data lake is responsible for data storage, organization, and maintenance. Data lakes allow users to store data in open storage formats and use flexible schemas to produce reports on "single source of truth" for various BI, AI, ad-hoc, and reporting use cases. StarRocks fully leverages the advantages of its

vectorization engine and CBO, significantly improving the performance of data lake analytics.

Quick Start

Deploy StarRocks with Docker

This QuickStart tutorial guides you through the procedures to deploy StarRocks on your local mach...

Create a table

This QuickStart tutorial will walk you through the necessary steps to create a table in StarRocks, an...

Load and query data

This QuickStart tutorial will lead you step by step in loading data into the table you created (see Cr...

Deploy StarRocks with Docker

This QuickStart tutorial guides you through the procedures to deploy StarRocks on your local machine with Docker. Before getting started, you can read StarRocks Architecture for more conceptual details.

By following these steps, you can deploy a simple StarRocks cluster with **one FE node** and **one BE node**. It can help you complete the upcoming QuickStart tutorials on creating a table and loading and querying data, and thereby acquaints you with the basic operations of StarRocks.

> **CAUTION**
>
> Deploying StarRocks with the Docker image used in this tutorial merely applies to the situation when you need to verify a DEMO with a small dataset. It is not recommended for a massive testing or production environment. To deploy a high-availability StarRocks cluster, see Deployment overview for other options that suit your scenarios.

Prerequisites

Before deploying StarRocks in Docker, make sure the following requirements are satisfied:

- **Hardware**

  We recommend deploying StarRocks on a machine with 8 CPU cores and 16 GB of memory or more.

- **Software**

  You must have the following software installed on your machine:

  - Docker Engine (17.06.0 or later)
  - MySQL client (5.5 or later)

Step 1: Download the StarRocks Docker image

Download a StarRocks Docker image from StarRocks Docker Hub. You can choose a specific version based on the tag of the image.

```
sudo docker run -p 9030:9030 -p 8030:8030 -p 8040:8040 \
    -itd registry.starrocks.io/starrocks/allin1-ubuntu
```

> **TROUBLESHOOTING**
>
> If any of the above ports on the host machine is occupied, the system prints "docker: Error response from daemon: driver failed programming external connectivity on endpoint tender_torvalds (): Bind for 0.0.0.0:xxxx failed: port is already allocated.". You can allocate available ports on the host machine by changing the ports preceding the colons (:) in the command.

You can check if the container is created and running properly by running the following command:

```
sudo docker ps
```

As shown below, if the `STATUS` of your StarRocks containers is `Up`, you have successfully deployed StarRocks in the Docker container.

```
CONTAINER ID   IMAGE                                                COMMAND
CREATED         STATUS                  PORTS                                                
```

```
mysql -P9030 -h127.0.0.1 -uroot --prompt="StarRocks > "
```

> **CAUTION**
>
> If you have allocated a different port for `9030` in the `docker run` command, you must replace `9030` in the above command with the port you have allocated.

You can check the status of the FE node by executing the following SQL:

```
SHOW PROC '/frontends'\G
```

Example:

```
StarRocks > SHOW PROC '/frontends'\G
*************************** 1. row ***************************
             Name: 8962368f9208_9010_1681370634632
               IP: 8962368f9208
      EditLogPort: 9010
         HttpPort: 8030
        QueryPort: 9030
          RpcPort: 9020
             Role: LEADER
        ClusterId: 555505802
             Join: true
            Alive: true
ReplayedJournalId: 99
    LastHeartbeat: 2023-04-13 07:28:50
         IsHelper: true
           ErrMsg:
        StartTime: 2023-04-13 07:24:11
          Version: BRANCH-3.0-0afb97bbf
1 row in set (0.02 sec)
```

- If the field `Alive` is `true`, this FE node is properly started and added to the cluster.
- If the field `Role` is `FOLLOWER`, this FE node is eligible to be elected as the Leader FE node.
- If the field `Role` is `LEADER`, this FE node is the Leader FE node.

You can check the status of the BE node by executing the following SQL:

```
SHOW PROC '/backends'\G
```

Example:

```
StarRocks > SHOW PROC '/backends'\G
*************************** 1. row ***************************
            BackendId: 10004
                   IP: 8962368f9208
        HeartbeatPort: 9050
               BePort: 9060
             HttpPort: 8040
             BrpcPort: 8060
        LastStartTime: 2023-04-13 07:24:25
        LastHeartbeat: 2023-04-13 07:29:05
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
            TabletNum: 30
     DataUsedCapacity: 0.000
        AvailCapacity: 527.437 GB
        TotalCapacity: 1.968 TB
              UsedPct: 73.83 %
       MaxDiskUsedPct: 73.83 %
               ErrMsg:
              Version: BRANCH-3.0-0afb97bbf
               Status: {"lastSuccessReportTabletsTime":"2023-04-13
07:28:26"}
    DataTotalCapacity: 527.437 GB
          DataUsedPct: 0.00 %
             CpuCores: 16
    NumRunningQueries: 0
           MemUsedPct: 0.02 %
           CpuUsedPct: 0.1 %
1 row in set (0.00 sec)
```

If the field `Alive` is `true`, this BE node is properly started and added to the cluster.

Stop and remove the Docker container

After completing the whole QuickStart tutorial, you can stop and remove the container that hosts your StarRocks cluster with its container ID.

> **NOTE**
>
> You can get the `container_id` of your Docker container by running `sudo docker ps`.

Run the following command to stop the container:

```
# Replace <container_id> with the container ID of your StarRocks cluster.
sudo docker stop <container_id>
```

If you do not need the container any longer, you can remove it by running the following command:

```
# Replace <container_id> with the container ID of your StarRocks cluster.
sudo docker rm <container_id>
```

> **CAUTION**
>
> The removal of the container is irreversible. Make sure you have a backup of the important data in the container before removing it.

What to do next

Having deployed StarRocks, you can continue the QuickStart tutorials on creating a table and loading and querying data.

Create a table

This QuickStart tutorial will walk you through the necessary steps to create a table in StarRocks, and introduce some basic features of StarRocks.

After the StarRocks instance is deployed (see Deploy StarRocks for details), you need to create a database and a table to load and query data. Creating databases and tables requires corresponding user privilege. In this QuickStart tutorial, you can perform the following steps with the default `root` user, which has the highest privileges on the StarRocks instance.

> **NOTE**
>
> You can complete this tutorial by using an existing StarRocks instance, database, table, and user privilege. However, for simplicity, we recommend that you use the schema and data the tutorial provides.

Step 1: Log in to StarRocks

Log in to StarRocks via your MySQL client. You can log in with the default user `root`, and the password is empty by default.

```
mysql -h <fe_ip> -P<fe_query_port> -uroot
```

> **NOTE**
>
> - Change the `-P` value accordingly if you have assigned a different FE MySQL server port (`query_port`, Default: `9030`).
> - Change the `-h` value accordingly if you have specified the configuration item `priority_networks` in the FE configuration file.

Step 2: Create a database

Create a database named `sr_hub` by referring to CREATE DATABASE.

```
CREATE DATABASE IF NOT EXISTS sr_hub;
```

You can view all databases in this StarRocks instance by executing SHOW DATABASES SQL.

Step 3: Create a table

Run `USE sr_hub` to switch to the `sr_hub` database and create a table named `sr_member` by referring to CREATE TABLE.

```
USE sr_hub;
CREATE TABLE IF NOT EXISTS sr_member (
    sr_id            INT,
    name             STRING,
    city_code        INT,
    reg_date         DATE,
    verified         BOOLEAN
)
PARTITION BY RANGE(reg_date)
(
    PARTITION p1 VALUES [('2022-03-13'), ('2022-03-14')),
    PARTITION p2 VALUES [('2022-03-14'), ('2022-03-15')),
    PARTITION p3 VALUES [('2022-03-15'), ('2022-03-16')),
    PARTITION p4 VALUES [('2022-03-16'), ('2022-03-17')),
    PARTITION p5 VALUES [('2022-03-17'), ('2022-03-18'))
)
DISTRIBUTED BY HASH(city_code);
```

> **NOTE**
>
> - To create a table in StarRocks, you MUST specify a bucket key in the `DISTRIBUTED BY HASH` clause to strategize the data distribution plan of the table. By default, the data is distributed to 10 tablets. See Data Distribution for more information.
> - You need to specify the table property `replication_num`, which represents the number of data replicas, as `1` because the StarRocks instance you deployed has only one BE node.

> - If no table type is specified, a Duplicate Key table is created by default. See Duplicate Key table
> - The columns of the table exactly correspond to the fields of data that you will be loading into StarRocks in the tutorial on loading and querying data.
> - To guarantee the high performance **in the production environment**, we strongly recommend that you strategize the data partitioning plan for the table by using the `PARTITION BY` clause. See Design partitioning and bucketing rules for more instructions.

After the table is created, you can check the details of the table using the DESC statement, and view all the tables in the database by executing SHOW TABLES. Tables in StarRocks support schema changes. You can see ALTER TABLE for more information.

What to do next

To learn more about conceptual details of StarRocks tables, see StarRocks Table Design.

In addition to the features this tutorial has demonstrated, StarRocks also supports:

- A variety of data types
- Multiple table types
- Flexible partitioning strategies
- Classic database query indexes, including bitmap index and bloom filter index
- Materialized view

Load and query data

This QuickStart tutorial will lead you step by step in loading data into the table you created (see Create a table for more instruction), and running a query on the data.

StarRocks supports loading data from a rich wealth of data sources, including some major cloud services, local files, or a streaming data system. You can see Ingestion Overview for more information. The following steps will show you how to insert data into StarRocks by using the INSERT INTO statement, and run queries on the data.

> **NOTE**
>
> You can complete this tutorial by using an existing StarRocks instance, database, table, user, and your own data. However, for simplicity, we recommend that you use the schema and data the tutorial provides.

Step 1: Load data with INSERT

You can insert additional rows of data using INSERT. See INSERT for detailed instruction.

Log in to StarRocks via your MySQL client, and execute the following statements to insert the following rows of data into the `sr_member` table you have created.

```
use sr_hub
INSERT INTO sr_member
WITH LABEL insertDemo
VALUES
    (001,"tom",100000,"2022-03-13",true),
    (002,"johndoe",210000,"2022-03-14",false),
    (003,"maruko",200000,"2022-03-14",true),
    (004,"ronaldo",100000,"2022-03-15",false),
    (005,"pavlov",210000,"2022-03-16",false),
    (006,"mohammed",300000,"2022-03-17",true);
```

If the loading transaction succeeds, the following message will be returned.

```
Query OK, 6 rows affected (0.07 sec)
{'label':'insertDemo', 'status':'VISIBLE', 'txnId':'5'}
```

> **NOTE**
>
> Loading data via INSERT INTO VALUES merely applies to the situation when you need to verify a DEMO with a small dataset. It is not recommended for a massive testing or production environment. To load mass data into StarRocks, see Ingestion Overview for other options that suit your scenarios.

Step 2: Query the data

StarRocks is compatible with SQL-92.

- Run a simple query to list all rows of data in the table.

  ```
  SELECT * FROM sr_member;
  ```

  The returned results are as follows:

  ```
  +-------+----------+-----------+------------+----------+
  | sr_id | name     | city_code | reg_date   | verified |
  +-------+----------+-----------+------------+----------+
  |     3 | maruko   |    200000 | 2022-03-14 |        1 |
  |     1 | tom      |    100000 | 2022-03-13 |        1 |
  |     4 | ronaldo  |    100000 | 2022-03-15 |        0 |
  |     6 | mohammed |    300000 | 2022-03-17 |        1 |
  |     5 | pavlov   |    210000 | 2022-03-16 |        0 |
  |     2 | johndoe  |    210000 | 2022-03-14 |        0 |
  +-------+----------+-----------+------------+----------+
  6 rows in set (0.05 sec)
  ```

- Run a standard query with a specified condition.

  ```
  SELECT sr_id, name
  FROM sr_member
  WHERE reg_date <= "2022-03-14";
  ```

The returned results are as follows:

```
+-------+----------+
| sr_id | name     |
+-------+----------+
|     1 | tom      |
|     3 | maruko   |
|     2 | johndoe  |
+-------+----------+
3 rows in set (0.01 sec)
```

- Run a query on a specified partition.

```
SELECT sr_id, name
FROM sr_member
PARTITION (p2);
```

The returned results are as follows:

```
+-------+---------+
| sr_id | name    |
+-------+---------+
|     3 | maruko  |
|     2 | johndoe |
+-------+---------+
2 rows in set (0.01 sec)
```

What to do next

To learn more about the data ingestion methods of StarRocks, see Ingestion Overview. In addition to a huge number of built-in functions, StarRocks also supports Java UDFs, which allows you to create your own data processing functions that suit your business scenarios.

You can also learn how to:

- Perform ETL when loading.
- Create an external table to access external data sources.
- Analyze the query plan to learn how to optimize the query performance.

Project Help

Community Chat

Slack with the community

Community Chat (via Slack)

The StarRocks project uses Slack for community chat. You can find our community at starrocks.slack.com.

Deployment overview

This chapter describes how to deploy, upgrade, and downgrade a StarRocks cluster in a production environment.

Deployment procedure

A summary of the deployment procedure is as follows and later topics provide the details.

The deployment of StarRocks generally follows the steps outlined here:

1. Confirm the hardware and software requirements for your StarRocks deployment.

   Check the prerequisites that your servers must meet before deploying StarRocks, including CPU, memory, storage, network, operating system, and dependencies.

2. Plan your cluster size.

   Plan the number of FE nodes and BE nodes in your cluster, and the hardware specifications of the servers.

3. Check environment configurations.

   When your servers are ready, you need to check and modify some environment configurations before deploying StarRocks on them.

4. Prepare deployment files.

   - If you want to deploy StarRocks on x86 architecture, you can directly download and extract the software package provided on our official website.
   - If you want to deploy StarRocks on ARM architecture, you need to prepare the deployment files from the StarRocks Docker image.
   - If you want to deploy StarRocks on Kubernetes, you can skip this step.

5. Deploy StarRocks.

   You have the following options:

    - Deploy StarRocks manually.
    - Deploy StarRocks on Kubernetes with operator.
    - Deploy StarRocks on Kubernetes with Helm.

6. Perform necessary post-deployment setup measures.

   Further setup measures are needed before your StarRocks cluster is put into production. These measures include securing the initial account and setting some performance-related system variables.

Upgrade and downgrade

If you plan to upgrade an existing StarRocks cluster to a later version rather than install StarRocks for the first time, see Upgrade StarRocks for information about upgrade procedures and issues that you should consider before upgrading.

For instructions to downgrade your StarRocks cluster, see Downgrade StarRocks.

Deployment prerequisites

This topic describes the hardware and software requirements that your servers must meet before ...

Plan StarRocks cluster

This topic describes how to plan resources for your StarRocks cluster in production from the persp...

Check environment configurations

This topic lists all environment and system configuration items that you must check and set before ...

Prepare deployment files

This topic describes how to prepare StarRocks deployment files.

Deployment prerequisites

This topic describes the hardware and software requirements that your servers must meet before deploying StarRocks. For recommended hardware specifications of your StarRocks cluster, see Plan your StarRocks cluster.

Hardware

CPU

StarRocks relies on AVX2 instruction sets to fully unleash its vectorization capability. Therefore, in a production environment, we highly recommend you deploy StarRocks on machines with x86 architecture CPUs.

You can run the following command in your terminal to check if the CPUs on your machines support the AVX2 instruction sets:

```
cat /proc/cpuinfo | grep avx2
```

Memory

No specific requirement is imposed on memory kits used for StarRocks. See Plan StarRocks cluster - CPU and Memory for the recommended memory size.

Storage

StarRocks supports both HDD and SSD as storage medium.

If your applications require real-time data analytics, intensive data scans, or random disk access, we strongly recommend you use SSD storage.

If your applications involve Primary Key tables with the persistent index, you must use SSD storage.

Network

We recommend that you use 10 Gigabit Ethernet networking to ensure stable data transmission across nodes within your StarRocks cluster.

Operating system

StarRocks supports deployments on CentOS Linux 7.9 or Ubuntu Linux 22.04.

Software

You must install JDK 8 on your servers to run StarRocks. For v2.5.10 and later versions, JDK 11 is recommended.

> **CAUTION**
>
> - StarRocks does not support JRE.
> - If you want to install StarRocks on Ubuntu 22.04, you must install JDK 11.

Follow these steps to install JDK 8:

1. Navigate to the path for the JDK installation.
2. Download JDK by running the following command:

```
wget --no-check-certificate --no-cookies \
    --header "Cookie: oraclelicense=accept-securebackup-cookie"
\
    http://download.oracle.com/otn-pub/java/jdk/8u131-
b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-x64.tar.gz
```

Plan StarRocks cluster

This topic describes how to plan resources for your StarRocks cluster in production from the perspectives of node count, CPU core count, memory size, and storage size.

Node count

StarRocks mainly consists of two types of components: FE nodes and BE nodes. Each node must be deployed separately on a physical or virtual machine.

FE node count

FE nodes are mainly responsible for metadata management, client connection management, query planning, and query scheduling.

In production, we recommend you deploy at least **THREE** Follower FE nodes in your StarRocks cluster to prevent single points of failure (SPOFs).

StarRocks uses the BDB JE protocol to manage the metadata across FE nodes. StarRocks elects a Leader FE node from all Follower FE nodes. Only the Leader FE node can write metadata. The other Follower FE nodes only update their metadata based on the logs from the Leader FE node. Each time the Leader FE node fails, StarRocks re-elects a new Leader FE node as long as more than half of the Follower FE nodes are alive.

If your application generates highly concurrent query requests, you can add Observer FE nodes to your cluster. Observer FE nodes only process the query requests and do not participate in the election for the Leader FE node.

BE node count

BE nodes are responsible for data storage and SQL execution.

In production, we recommend you deploy at least **THREE** BE nodes in your StarRocks cluster to ensure high data reliability and service availability. A high-availability cluster of BEs is automatically formed when at least three BE nodes are deployed and added to your

StarRocks cluster. The failure of one BE node will not affect the overall availability of the BE services.

You can increase the number of BE nodes to enable your StarRocks cluster to process highly concurrent queries.

CN node count

CN nodes are optional components of StarRocks, and are only responsible for SQL execution.

You can increase the number of CN nodes to elastically scale compute resources without changing the data distribution in your StarRocks cluster.

CPU and memory

Usually, the FE service does not consume a lot of CPU and memory resources. We recommend allocating 8 CPU cores and 16 GB RAM to each FE node.

Unlike the FE service, the BE service can be significantly CPU- and memory-intensive if your application works with highly concurrent or complex queries on a large dataset. Therefore, we recommend allocating 16 CPU cores and 64 GB RAM to each BE node.

Storage capacity

FE storage

Because FE nodes only maintain StarRocks' metadata in their storage, 100 GB of HDD storage is enough for each FE node in most scenarios.

BE storage

Estimate initial storage space for BE

The total storage space that your StarRocks cluster will need is simultaneously influenced by the size of your raw data, the data replica count, and the compression ratio of the data compression algorithm you use.

With the following formula, you can estimate the total storage space you will need for all BE nodes:

```
Total BE storage space = Raw data size * Replica count/Compression ratio

Raw data size = Sum of the space taken up by all fields in a row * Row count
```

In StarRocks, data in a table is first divided into multiple partitions and then into multiple tablets. Tablets are the basic logical units of data management in StarRocks. To ensure high data reliability, you can maintain multiple replicas of each tablet and store them across different BEs. By default, StarRocks maintains three replicas.

Currently, StarRocks supports four data compression algorithms, which are listed in order from higher to lower compression ratio: zlib, Zstandard (or zstd), LZ4, and Snappy. They can provide a compression ratio from 3:1 to 5:1.

After determining the total storage space, you can simply divide it by the number of BE nodes in your cluster to estimate the average storage space per BE node.

Add extra storage as you go

If the BE storage space runs out as your raw data grows, you can supplement it by scaling your cluster vertically or horizontally, or simply scaling up your cloud storage.

- Add new BE nodes to your StarRocks cluster

  You can add new BE nodes to your StarRocks cluster so that the data can be re-distributed evenly to more nodes. For detailed instructions, see Scale your StarRocks cluster - Scale BE out.

  After new BE nodes are added, StarRocks automatically re-balances the data among all BE nodes. Such auto-balancing is supported on all table types.

- Add extra storage volumes to your BE nodes

  You can also add extra storage volumes to existing BE nodes. For detailed instructions, see Scale your StarRocks cluster - Scale BE up.

After extra storage volumes are added, StarRocks automatically re-balances the data in all tables except Primary Key tables. In this situation, the distribution of data from Primary Key tables may be imbalanced across nodes, and consequently, queries against these tables are run slowly. Therefore, adding extra storage volumes to BE nodes is not recommended if your StarRocks cluster contains Primary Key tables.

- Add cloud storage

  If your StarRocks cluster is deployed on cloud, you can scale up your cloud storage on demand. For detailed instructions, contact your cloud provider.

Check environment configurations

This topic lists all environment and system configuration items that you must check and set before deploying StarRocks. Setting these configuration items properly allows your StarRocks cluster to work with high availability and performance.

Ports

StarRocks uses specific ports for different services. Check whether these ports are occupied on each instance if you have deployed other services on these instances.

FE ports

On the instances used for the FE deployment, you need to check the following ports:

- `8030`: FE HTTP server port (`http_port`)
- `9020`: FE Thrift server port (`rpc_port`)
- `9030`: FE MySQL server port (`query_port`)
- `9010`: FE internal communication port (`edit_log_port`)

Run the following commands on the FE instances to check whether these ports are occupied:

```
netstat -tunlp | grep 8030
netstat -tunlp | grep 9020
netstat -tunlp | grep 9030
netstat -tunlp | grep 9010
```

If any of the above ports are occupied, you must find alternatives and specify them later when you deploy FE nodes. For detailed instructions, see Deploy StarRocks - Start the Leader FE node.

BE ports

On the instances used for the BE deployment, you need to check the following ports:

- `9060`: BE Thrift server port (`be_port`)
- `8040`: BE HTTP server port (`webserver_port`)
- `9050`: BE heartbeat service port (`heartbeat_service_port`)
- `8060`: BE bRPC port (`brpc_port`)

Run the following commands on the BE instances to check whether these ports are occupied:

```
netstat -tunlp | grep 9060
netstat -tunlp | grep 8040
netstat -tunlp | grep 9050
netstat -tunlp | grep 8060
```

If any of the above ports are occupied, you must find alternatives and specify them later when you deploy BE nodes. For detailed instructions, see Deploy StarRocks - Start the BE service.

CN ports

On the instances used for the CN deployment, you need to check the following ports:

- `9060`: CN Thrift server port (`thrift_port`)
- `8040`: CN HTTP server port (`webserver_port`)
- `9050`: CN heartbeat service port (`heartbeat_service_port`)
- `8060`: CN bRPC port (`brpc_port`)

Run the following commands on the CN instances to check whether these ports are occupied:

```
netstat -tunlp | grep 9060
netstat -tunlp | grep 8040
netstat -tunlp | grep 9050
netstat -tunlp | grep 8060
```

If any of the above ports are occupied, you must find alternatives and specify them later when you deploy CN nodes. For detailed instructions, see Deploy StarRocks - Start the CN service.

Hostnames

If you want to enable FQDN access for your StarRocks cluster, you must assign a hostname to each instance.

In the file **/etc/hosts** on each instance, you must specify the IP addresses and corresponding hostnames of all the other instances in the cluster.

> **CAUTION**
>
> All IP addresses in the file **/etc/hosts** must be unique.

JDK configuration

StarRocks relies on the environment variable `JAVA_HOME` to locate the Java dependency on the instance.

Run the following command to check the environment variable `JAVA_HOME`:

```
echo $JAVA_HOME
```

Follow these steps to set `JAVA_HOME`:

1. Set `JAVA_HOME` in the file **/etc/profile**:

   ```
   sudo vi /etc/profile
   # Replace <path_to_JDK> with the path where JDK is installed.
   export JAVA_HOME=<path_to_JDK>
   export PATH=$PATH:$JAVA_HOME/bin
   ```

2. Bring the change into effect:

```
source /etc/profile
```

Run the following command to verify the change:

```
java -version
```

CPU scaling governor

This configuration item is **optional**. You can skip it if your CPU does not support the scaling governor.

The CPU scaling governor controls the CPU power mode. If your CPU supports it, we recommend you set it to `performance` for better CPU performance:

```
echo 'performance' | sudo tee
/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Memory configurations

Memory Overcommit

Memory Overcommit allows the operating system to overcommit memory resources to processes. We recommend you enable Memory Overcommit.

```
# Modify the configuration file.
cat >> /etc/sysctl.conf << EOF
vm.overcommit_memory=1
EOF
# Bring the change into effect.
sysctl -p
```

Transparent Huge Pages

Transparent Huge Pages are enabled by default. We recommend you disable this feature because it can interfere with the memory allocator, and thereby lead to a drop in

performance.

```
# Change the configuration temporarily.
echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
# Change the configuration permanently.
cat >> /etc/rc.d/rc.local << EOF
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
   echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
fi
if test -f /sys/kernel/mm/transparent_hugepage/defrag; then
   echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
fi
EOF
chmod +x /etc/rc.d/rc.local
```

Swap Space

We recommend you disable Swap Space.

Follow these steps to check and disable Swap Space:

1. Disable Swap Space.

   ```
   swapoff /<path_to_swap_space>
   swapoff -a
   ```

2. Delete the Swap Space information from the configuration file **/etc/fstab**.

   ```
   /<path_to_swap_space> swap swap defaults 0 0
   ```

3. Verify that Swap Space is disabled.

   ```
   free -m
   ```

Swappiness

We recommend you disable swappiness to eliminate its impact on performance.

```
# Modify the configuration file.
cat >> /etc/sysctl.conf << EOF
vm.swappiness=0
EOF
# Bring the change into effect.
sysctl -p
```

Storage configurations

We recommend that you choose your suitable scheduler algorithm in accordance with the storage medium you use.

You can run the following command to check the scheduler algorithm that you are using:

```
cat /sys/block/${disk}/queue/scheduler
# For example, run cat /sys/block/vdb/queue/scheduler
```

We recommend you use the mq-deadline scheduler for SATA disks and the kyber scheduler algorithm for SSD and NVMe disks.

SATA

The mq-deadline scheduler algorithm suits SATA disks.

```
# Change the configuration temporarily.
echo mq-deadline | sudo tee /sys/block/${disk}/queue/scheduler
# Change the configuration permanently.
cat >> /etc/rc.d/rc.local << EOF
echo mq-deadline | sudo tee /sys/block/${disk}/queue/scheduler
EOF
chmod +x /etc/rc.d/rc.local
```

SSD and NVMe

- If your NVMe or SSD disks support the kyber scheduler algorithm:

```
# Change the configuration temporarily.
echo kyber | sudo tee /sys/block/${disk}/queue/scheduler
# Change the configuration permanently.
cat >> /etc/rc.d/rc.local << EOF
echo kyber | sudo tee /sys/block/${disk}/queue/scheduler
EOF
chmod +x /etc/rc.d/rc.local
```

- If your NVMe or SSD disks support the none (or noop) scheduler.

```
# Change the configuration temporarily.
echo none | sudo tee /sys/block/vdb/queue/scheduler
# Change the configuration permanently.
cat >> /etc/rc.d/rc.local << EOF
echo none | sudo tee /sys/block/${disk}/queue/scheduler
EOF
chmod +x /etc/rc.d/rc.local
```

SELinux

We recommend you disable SELinux.

```
# Change the configuration temporarily.
setenforce 0
# Change the configuration permanently.
sed -i 's/SELINUX=.*/SELINUX=disabled/' /etc/selinux/config
sed -i 's/SELINUXTYPE/#SELINUXTYPE/' /etc/selinux/config
```

Firewall

Open the internal ports for FE nodes, BE nodes, and Broker if your firewall is enabled.

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

LANG variable

Run the following command to check and configure the LANG variable manually:

```
# Modify the configuration file.
echo "export LANG=en_US.UTF8" >> /etc/profile
# Bring the change into effect.
source /etc/profile
```

Time zone

Set this item in accordance with your actual time zone.

The following example sets the time zone to `/Asia/Shanghai`.

```
cp -f /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
hwclock
```

ulimit configurations

Problems can occur with StarRocks if the values of **max file descriptors** and **max user processes** are abnormally small. We recommend you enlarge these values.

```
cat >> /etc/security/limits.conf << EOF
* soft nproc 65535
* hard nproc 65535
* soft nofile 655350
* hard nofile 655350
* soft stack unlimited
* hard stack unlimited
* hard memlock unlimited
* soft memlock unlimited
EOF

cat >> /etc/security/limits.d/20-nproc.conf << EOF
*          soft    nproc     65535
root       soft    nproc     65535
EOF
```

File system configuration

We recommend you use the ext4 or xfs journaling file system. You can run the following command to check the mount type:

```
df -Th
```

Network configuration

tcp_abort_on_overflow

Allow the system to reset new connections if the system is currently overflowed with new connection attempts that the daemon(s) can not handle:

```
# Modify the configuration file.
cat >> /etc/sysctl.conf << EOF
net.ipv4.tcp_abort_on_overflow=1
EOF
# Bring the change into effect.
sysctl -p
```

somaxconn

Specify the maximum number of connection requests queued for any listening socket to `1024`:

```
# Modify the configuration file.
cat >> /etc/sysctl.conf << EOF
net.core.somaxconn=1024
EOF
# Bring the change into effect.
sysctl -p
```

NTP configuration

You must configure time synchronization between nodes within your StarRocks cluster to ensure linear consistency of transactions. You can either use the internet time service provided by pool.ntp.org, or use the NTP service built in an offline environment. For example, you can use the NTP service provided by your cloud service provider.

1. Check if the NTP time server or Chrony service exists.

```
rpm -qa | grep ntp
systemctl status chrony
```

2. Install the NTP service if there is not one.

```
sudo yum install ntp ntpdate && \
sudo systemctl start ntpd.service && \
sudo systemctl enable ntpd.service
```

3. Check the NTP service.

```
systemctl list-unit-files | grep ntp
```

4. Check the connectivity and monitoring status of the NTP service.

```
netstat -tlunp | grep ntp
```

5. Check if your application is synchronized with the NTP server.

```
ntpstat
```

6. Check the state of all the configured NTP servers in your network.

```
ntpq -p
```

High concurrency configurations

If your StarRocks cluster has a high load concurrency, we recommend you set the following configurations.

max_map_count

Specify the maximum number of memory map areas a process may have as `262144`:

```
# Modify the configuration file.
cat >> /etc/sysctl.conf << EOF
vm.max_map_count = 262144
EOF
# Bring the change into effect.
sysctl -p
```

Other

```
echo 120000 > /proc/sys/kernel/threads-max
echo 200000 > /proc/sys/kernel/pid_max
```

Prepare deployment files

This topic describes how to prepare StarRocks deployment files.

Currently, the binary distribution packages StarRocks provides on the StarRocks official website support deployments only on x86-based CentOS 7.9. If you want to deploy StarRocks with the ARM architecture CPUs or on Ubuntu 22.04, you need to prepare the deployment files using the StarRocks Docker image.

For x86-based CentOS 7.9

StarRocks binary distribution packages are named in the **StarRocks-version.tar.gz** format, where **version** is a number (for example, **2.5.2**) that indicates the version information of the binary distribution package. Make sure that you have chosen the correct version of the package.

Follow these steps to prepare deployment files for the x86-based CentOS 7.9 platform:

1. Obtain the StarRocks binary distribution package directly from the Download StarRocks page or by running the following command in your terminal:

```
# Replace <version> with the version of StarRocks you want to
download, for example, 2.5.2.
wget https://releases.starrocks.io/starrocks/StarRocks-
<version>.tar.gz
```

2. Extract the files in the package.

```
# Replace <version> with the version of StarRocks you have
downloaded.
tar -xzvf StarRocks-<version>.tar.gz
```

   The package includes the following directories and files:

Directory/File	Description
apache_hdfs_broker	The deployment directory of the Broker node. From StarRocks v2.5 onwards, you do not need to deploy Broker nodes in general scenarios. If you need to deploy Broker nodes in your StarRocks cluster, see Deploy Broker node for detailed instructions.
fe	The FE deployment directory.
be	The BE deployment directory.
LICENSE.txt	The StarRocks license file.
NOTICE.txt	The StarRocks notice file.

3. Dispatch the directory **fe** to all the FE instances and the directory **be** to all the BE or CN instances for manual deployment.

For ARM-based CPU or Ubuntu 22.04

Prerequisites

You must have Docker Engine (17.06.0 or later) installed on your machine.

Procedures

1. Download a StarRocks Docker image from StarRocks Docker Hub. You can choose a specific version based on the tag of the image.

   - If you use Ubuntu 22.04:

     ```
     # Replace <image_tag> with the tag of the image that you want to download, for example, 2.5.4.
     docker pull starrocks/artifacts-ubuntu:<image_tag>
     ```

   - If you use ARM-based CentOS 7.9:

```
# Replace <image_tag> with the tag of the image that you want
to download, for example, 2.5.4.
docker pull starrocks/artifacts-centos7:<image_tag>
```

2. Copy the StarRocks deployment files from the Docker image to your host machine by running the following command:

   - If you use Ubuntu 22.04:

   ```
   # Replace <image_tag> with the tag of the image that you have
   downloaded, for example, 2.5.4.
   docker run --rm starrocks/artifacts-ubuntu:<image_tag> \
       tar -cf - -C /release . | tar -xvf -
   ```

   - If you use ARM-based CentOS 7.9:

   ```
   # Replace <image_tag> with the tag of the image that you have
   downloaded, for example, 2.5.4.
   docker run --rm starrocks/artifacts-centos7:<image_tag> \
       tar -cf - -C /release . | tar -xvf -
   ```

   The deployment files include the following directories:

Directory	Description
be_artifacts	This directory includes the BE or CN deployment directory **be**, StarRocks license file **LICENSE.txt**, and StarRocks notice file **NOTICE.txt**.
broker_artifacts	This directory includes the Broker deployment directory **apache_hdfs_broker**. From StarRocks 2.5 onwards, you do not need to deploy Broker nodes in general scenarios. If you need to deploy Broker nodes in your StarRocks cluster, see Deploy Broker for detailed instructions.
fe_artifacts	This directory includes the FE deployment directory **fe**, StarRocks license file **LICENSE.txt**, and StarRocks notice file **NOTICE.txt**.

3. Dispatch the directory **fe** to all the FE instances and the directory **be** to all the BE or CN instances for manual deployment.

Deploy StarRocks manually

This topic describes how to manually deploy StarRocks. For other modes of installation, see Deplo...

Deploy StarRocks with Operator

This topic introduces how to use the StarRocks Operator to automate the deployment and manage...

Deploy StarRocks with Helm

Helm is a package manager for Kubernetes. A Helm Chart is a Helm package and contains all of th...

Deploy StarRocks manually

This topic describes how to manually deploy StarRocks. For other modes of installation, see Deployment Overview.

> **TIP**
>
> The preparations for manual deployment are outlined in the Deployment prerequisites and Check environment configurations documents. Please start there if you are planning a production deployment. If you are getting started with StarRocks and would like to follow one of the Quick Starts, please refer to Quick Starts.

Step 1: Start the Leader FE node

The following procedures are performed on an FE instance.

1. Create a dedicated directory for metadata storage. We recommend storing metadata in a separate directory from the FE deployment files. Make sure that this directory exists and that you have write access to it.

```
# Replace <meta_dir> with the metadata directory you want to
create.
mkdir -p <meta_dir>
```

2. Navigate to the directory that stores the StarRocks FE deployment files you prepared earlier, and modify the FE configuration file **fe/conf/fe.conf**.

   a. Specify the metadata directory in the configuration item `meta_dir`.

```
# Replace <meta_dir> with the metadata directory you have
created.
meta_dir = <meta_dir>
```

b. If any of the FE ports mentioned in the Environment Configuration Checklist are occupied, you must assign valid alternatives in the FE configuration file.

```
http_port = aaaa        # Default: 8030
rpc_port = bbbb         # Default: 9020
query_port = cccc       # Default: 9030
edit_log_port = dddd    # Default: 9010
```

> **CAUTION**
>
> If you want to deploy multiple FE nodes in a cluster, you must assign the same `http_port` to each FE node.

c. If you want to enable IP address access for your cluster, you must add the configuration item `priority_networks` in the configuration file and assign a dedicated IP address (in the CIDR format) to the FE node. You can ignore this configuration item if you want to enable FQDN access for your cluster.

```
priority_networks = x.x.x.x/x
```

> **NOTE**
>
> You can run `ifconfig` in your terminal to view the IP address(es) owned by the instance.

d. If you have multiple JDKs installed on the instance, and you want to use a specific JDK that is different from the one specified in the environment variable `JAVA_HOME`, you must specify the path where the chosen JDK is installed by adding the configuration item `JAVA_HOME` in the configuration file.

```
# Replace <path_to_JDK> with the path where the chosen JDK is
installed.
JAVA_HOME = <path_to_JDK>
```

f. For information about advanced configuration items, see Parameter Configuration - FE configuration items.

3. Start the FE node.

- To enable IP address access for your cluster, run the following command to start the FE node:

  ```
  ./fe/bin/start_fe.sh --daemon
  ```

- To enable FQDN access for your cluster, run the following command to start the FE node:

  ```
  ./fe/bin/start_fe.sh --host_type FQDN --daemon
  ```

  Note that you only need to specify the parameter `--host_type` ONCE when you start the node for the first time.

  > **CAUTION**
  >
  > Before starting the FE node with FQDN access enabled, make sure you have assigned hostnames for all instances in **/etc/hosts**. See Environment Configuration Checklist - Hostnames for more information.

4. Check the FE logs to verify if the FE node is started successfully.

   ```
   cat fe/log/fe.log | grep thrift
   ```

   A record of log like `2022-08-10 16:12:29,911 INFO (UNKNOWN x.x.x.x_9010_1660119137253(-1)|1) [FeServer.start():52] thrift server started with port 9020.` suggests that the FE node is started properly.

Step 2: Start the BE service

The following procedures are performed on the BE instances.

1. Create a dedicated directory for data storage. We recommend storing data in a separate directory from the BE deployment directory. Make sure that this directory exists and you have write access to it.

   ```
   # Replace <storage_root_path> with the data storage directory you want to create.
   ```

```
mkdir -p <storage_root_path>
```

2. Navigate to the directory that stores the StarRocks BE deployment files you prepared earlier, and modify the BE configuration file **be/conf/be.conf**.

   a. Specify the data directory in the configuration item `storage_root_path`.

```
# Replace <storage_root_path> with the data directory you have created.
storage_root_path = <storage_root_path>
```

   b. If any of the BE ports mentioned in the Environment Configuration Checklist are occupied, you must assign valid alternatives in the BE configuration file.

```
be_port = vvvv                    # Default: 9060
webserver_port = xxxx             # Default: 8040
heartbeat_service_port = yyyy     # Default: 9050
brpc_port = zzzz                  # Default: 8060
```

   c. If you want to enable IP address access for your cluster, you must add the configuration item `priority_networks` in the configuration file and assign a dedicated IP address (in the CIDR format) to the BE node. You can ignore this configuration item if you want to enable FQDN access for your cluster.

```
priority_networks = x.x.x.x/x
```

> **NOTE**
>
> You can run `ifconfig` in your terminal to view the IP address(es) owned by the instance.

   d. If you have multiple JDKs installed on the instance, and you want to use a specific JDK that is different from the one specified in the environment variable `JAVA_HOME`, you must specify the path where the chosen JDK is installed by adding the configuration item `JAVA_HOME` in the configuration file.

```
# Replace <path_to_JDK> with the path where the chosen JDK is installed.
```

```
JAVA_HOME = <path_to_JDK>
```

For information about advanced configuration items, see Parameter Configuration - BE configuration items.

3. Start the BE node.

```
./be/bin/start_be.sh --daemon
```

> **CAUTION**
>
> - Before starting the BE node with FQDN access enabled, make sure you have assigned hostnames for all instances in **/etc/hosts**. See Environment Configuration Checklist - Hostnames for more information.
> - You do not need to specify the parameter `--host_type` when you start BE nodes.

4. Check the BE logs to verify if the BE node is started successfully.

```
cat be/log/be.INFO | grep heartbeat
```

A record of log like "I0810 16:18:44.487284 3310141 task_worker_pool.cpp:1387] Waiting to receive first heartbeat from frontend" suggests that the BE node is started properly.

5. You can start new BE nodes by repeating the above procedures on other BE instances.

> **NOTE**
>
> A high-availability cluster of BEs is automatically formed when at least three BE nodes are deployed and added to a StarRocks cluster.

Step 3: (Optional) Start the CN service

A Compute Node (CN) is a stateless computing service that does not maintain data itself. You can optionally add CN nodes to your cluster to provide extra computing resources for queries. You can deploy CN nodes with the BE deployment files. Compute Nodes are supported since v2.4.

1. Navigate to the directory that stores the StarRocks BE deployment files you prepared earlier, and modify the CN configuration file **be/conf/cn.conf**.

   a. If any of the CN ports mentioned in the Environment Configuration Checklist are occupied, you must assign valid alternatives in the CN configuration file.

   ```
   thrift_port = vvvv                # Default: 9060
   webserver_port = xxxx             # Default: 8040
   heartbeat_service_port = yyyy     # Default: 9050
   brpc_port = zzzz                  # Default: 8060
   ```

   b. If you want to enable IP address access for your cluster, you must add the configuration item `priority_networks` in the configuration file and assign a dedicated IP address (in the CIDR format) to the CN node. You can ignore this configuration item if you want to enable FQDN access for your cluster.

   ```
   priority_networks = x.x.x.x/x
   ```

   > **NOTE**
   >
   > You can run `ifconfig` in your terminal to view the IP address(es) owned by the instance.

   c. If you have multiple JDKs installed on the instance, and you want to use a specific JDK that is different from the one specified in the environment variable `JAVA_HOME`, you must specify the path where the chosen JDK is installed by adding the configuration item `JAVA_HOME` in the configuration file.

   ```
   # Replace <path_to_JDK> with the path where the chosen JDK is installed.
   JAVA_HOME = <path_to_JDK>
   ```

   For information about advanced configuration items, see Parameter Configuration - BE configuration items because most of CN's parameters are inherited from BE.

2. Start the CN node.

   ```
   ./be/bin/start_cn.sh --daemon
   ```

> **CAUTION**
>
> - Before starting the CN node with FQDN access enabled, make sure you have assigned hostnames for all instances in **/etc/hosts**. See Environment Configuration Checklist - Hostnames for more information.
> - You do not need to specify the parameter `--host_type` when you start CN nodes.

3. Check the CN logs to verify if the CN node is started successfully.

```
cat be/log/cn.INFO | grep heartbeat
```

   A record of log like "I0313 15:03:45.820030 412450 thrift_server.cpp:375] heartbeat has started listening port on 9050" suggests that the CN node is started properly.

4. You can start new CN nodes by repeating the above procedures on other instances.

Step 4: Set up the cluster

After all FE, BE nodes, and CN nodes are started properly, you can set up the StarRocks cluster.

The following procedures are performed on a MySQL client. You must have MySQL client 5.5.0 or later installed.

1. Connect to StarRocks via your MySQL client. You need to log in with the initial account `root`, and the password is empty by default.

```
# Replace <fe_address> with the IP address (priority_networks) or
FQDN
# of the Leader FE node, and replace <query_port> (Default: 9030)
# with the query_port you specified in fe.conf.
mysql -h <fe_address> -P<query_port> -uroot
```

2. Check the status of the Leader FE node by executing the following SQL.

```
SHOW PROC '/frontends'\G
```

Example:

```
MySQL [(none)]> SHOW PROC '/frontends'\G
*************************** 1. row ***************************
             Name: x.x.x.x_9010_1686810741121
               IP: x.x.x.x
      EditLogPort: 9010
         HttpPort: 8030
        QueryPort: 9030
          RpcPort: 9020
             Role: LEADER
        ClusterId: 919351034
             Join: true
            Alive: true
ReplayedJournalId: 1220
    LastHeartbeat: 2023-06-15 15:39:04
         IsHelper: true
           ErrMsg:
        StartTime: 2023-06-15 14:32:28
          Version: 3.0.0-48f4d81
1 row in set (0.01 sec)
```

- If the field `Alive` is `true`, this FE node is properly started and added to the cluster.
- If the field `Role` is `FOLLOWER`, this FE node is eligible to be elected as the Leader FE node.
- If the field `Role` is `LEADER`, this FE node is the Leader FE node.

3. Add the BE nodes to the cluster.

```
-- Replace <be_address> with the IP address (priority_networks)
-- or FQDN of the BE nodes, and replace <heartbeat_service_port>
-- with the heartbeat_service_port (Default: 9050) you specified
in be.conf.
ALTER SYSTEM ADD BACKEND "<be_address>:<heartbeat_service_port>";
```

> **NOTE**
>
> You can use the preceding command to add multiple BE nodes at a time. Each `<be_address>:<heartbeat_service_port>` pair represents one BE node.

4. Check the status of the BE nodes by executing the following SQL.

```
SHOW PROC '/backends'\G
```

Example:

```
MySQL [(none)]> SHOW PROC '/backends'\G
*************************** 1. row ***************************
            BackendId: 10007
                   IP: 172.26.195.67
        HeartbeatPort: 9050
               BePort: 9060
             HttpPort: 8040
             BrpcPort: 8060
        LastStartTime: 2023-06-15 15:23:08
        LastHeartbeat: 2023-06-15 15:57:30
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
            TabletNum: 30
     DataUsedCapacity: 0.000
        AvailCapacity: 341.965 GB
        TotalCapacity: 1.968 TB
              UsedPct: 83.04 %
       MaxDiskUsedPct: 83.04 %
               ErrMsg:
              Version: 3.0.0-48f4d81
               Status: {"lastSuccessReportTabletsTime":"2023-06-
15 15:57:08"}
    DataTotalCapacity: 341.965 GB
          DataUsedPct: 0.00 %
             CpuCores: 16
    NumRunningQueries: 0
           MemUsedPct: 0.01 %
           CpuUsedPct: 0.0 %
```

If the field `Alive` is `true`, this BE node is properly started and added to the cluster.

5. (Optional) Add a CN node to the cluster.

```
-- Replace <cn_address> with the IP address (priority_networks)
-- or FQDN of the CN node, and replace <heartbeat_service_port>
-- with the heartbeat_service_port (Default: 9050) you specified
in cn.conf.
```

```
ALTER SYSTEM ADD COMPUTE NODE "<cn_address>:
<heartbeat_service_port>";
```

> **NOTE**
>
> You can add multiple CN nodes with one SQL. Each `<cn_address>:<heartbeat_service_port>` pair represents one CN node.

6. (Optional) Check the status of the CN nodes by executing the following SQL.

```
SHOW PROC '/compute_nodes'\G
```

Example:

```
MySQL [(none)]> SHOW PROC '/compute_nodes'\G
*************************** 1. row ***************************
        ComputeNodeId: 10003
                   IP: x.x.x.x
        HeartbeatPort: 9050
               BePort: 9060
             HttpPort: 8040
             BrpcPort: 8060
        LastStartTime: 2023-03-13 15:11:13
        LastHeartbeat: 2023-03-13 15:11:13
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
               ErrMsg:
              Version: 2.5.2-c3772fb
1 row in set (0.00 sec)
```

If the field `Alive` is `true`, this CN node is properly started and added to the cluster.

After CNs are properly started and you want to use CNs during queries, set the system variables `SET prefer_compute_node = true;` and `SET use_compute_nodes = -1;`. For more information, see System variables.

Step 5: (Optional) Deploy a high-availability FE cluster

A high-availability FE cluster requires at least THREE Follower FE nodes in the StarRocks cluster. After the Leader FE node is started successfully, you can then start two new FE nodes to deploy a high-availability FE cluster.

1. Connect to StarRocks via your MySQL client. You need to log in with the initial account `root`, and the password is empty by default.

```
# Replace <fe_address> with the IP address (priority_networks) or
FQDN
# of the Leader FE node, and replace <query_port> (Default: 9030)
# with the query_port you specified in fe.conf.
mysql -h <fe_address> -P<query_port> -uroot
```

2. Add the new Follower FE node to the cluster by executing the following SQL.

```
-- Replace <fe_address> with the IP address (priority_networks)
-- or FQDN of the new Follower FE node, and replace
<edit_log_port>
-- with the edit_log_port (Default: 9010) you specified in
fe.conf.
ALTER SYSTEM ADD FOLLOWER "<fe2_address>:<edit_log_port>";
```

> **NOTE**
>
> - You can use the preceding command to add a single Follower FE nodes each time.
> - If you want to add Observer FE nodes, execute `ALTER SYSTEM ADD OBSERVER "<fe_address>:<edit_log_port>"=`. For detailed instructions, see ALTER SYSTEM - FE.

3. Launch a terminal on the new FE instance, create a dedicated directory for metadata storage, navigate to the directory that stores the StarRocks FE deployment files, and modify the FE configuration file **fe/conf/fe.conf**. For more instructions, see Step 1: Start the Leader FE node. Basically, you can repeat the procedures in Step 1 **except for the command used to start the FE node**.

   After configuring the Follower FE node, execute the following SQL to assign a helper node for Follower FE node and start the Follower FE node.

   > **NOTE**

> When adding new Follower FE node to a cluster, you must assign a helper node (essentially an existing Follower FE node) to the new Follower FE node to synchronize the metadata.

- To start a new FE node with IP address access, run the following command to start the FE node:

  ```
  # Replace <helper_fe_ip> with the IP address
  (priority_networks)
  # of the Leader FE node, and replace <helper_edit_log_port>
  (Default: 9010) with
  # the Leader FE node's edit_log_port.
  ./fe/bin/start_fe.sh --helper <helper_fe_ip>:
  <helper_edit_log_port> --daemon
  ```

  Note that you only need to specify the parameter `--helper` ONCE when you start the node for the first time.

- To start a new FE node with FQDN access, run the following command to start the FE node:

  ```
  # Replace <helper_fqdn> with the FQDN of the Leader FE node,
  # and replace <helper_edit_log_port> (Default: 9010) with the
  Leader FE node's edit_log_port.
  ./fe/bin/start_fe.sh --helper <helper_fqdn>:
  <helper_edit_log_port> \
        --host_type FQDN --daemon
  ```

  Note that you only need to specify the parameters `--helper` and `--host_type` ONCE when you start the node for the first time.

4. Check the FE logs to verify if the FE node is started successfully.

   ```
   cat fe/log/fe.log | grep thrift
   ```

   A record of log like `2022-08-10 16:12:29,911 INFO (UNKNOWN x.x.x.x_9010_1660119137253(-1)|1) [FeServer.start():52] thrift server started with port 9020.` suggests that the FE node is started properly.

5. Repeat the preceding procedure 2, 3, and 4 until you have start all the new Follower FE nodes properly, and then check the status of the FE nodes by executing the following SQL from your MySQL client:

```
SHOW PROC '/frontends'\G
```

Example:

```
MySQL [(none)]> SHOW PROC '/frontends'\G
*************************** 1. row ***************************
             Name: x.x.x.x_9010_1686810741121
               IP: x.x.x.x
      EditLogPort: 9010
         HttpPort: 8030
        QueryPort: 9030
          RpcPort: 9020
             Role: LEADER
        ClusterId: 919351034
             Join: true
            Alive: true
ReplayedJournalId: 1220
    LastHeartbeat: 2023-06-15 15:39:04
         IsHelper: true
           ErrMsg:
        StartTime: 2023-06-15 14:32:28
          Version: 3.0.0-48f4d81
*************************** 2. row ***************************
             Name: x.x.x.x_9010_1686814080597
               IP: x.x.x.x
      EditLogPort: 9010
         HttpPort: 8030
        QueryPort: 9030
          RpcPort: 9020
             Role: FOLLOWER
        ClusterId: 919351034
             Join: true
            Alive: true
ReplayedJournalId: 1219
    LastHeartbeat: 2023-06-15 15:39:04
         IsHelper: true
           ErrMsg:
        StartTime: 2023-06-15 15:38:53
          Version: 3.0.0-48f4d81
```

```
*************************** 3. row ***************************
             Name: x.x.x.x_9010_1686814090833
               IP: x.x.x.x
      EditLogPort: 9010
         HttpPort: 8030
        QueryPort: 9030
          RpcPort: 9020
             Role: FOLLOWER
        ClusterId: 919351034
             Join: true
            Alive: true
ReplayedJournalId: 1219
    LastHeartbeat: 2023-06-15 15:39:04
         IsHelper: true
           ErrMsg:
        StartTime: 2023-06-15 15:37:52
          Version: 3.0.0-48f4d81
3 rows in set (0.02 sec)
```

- If the field `Alive` is `true`, this FE node is properly started and added to the cluster.
- If the field `Role` is `FOLLOWER`, this FE node is eligible to be elected as the Leader FE node.
- If the field `Role` is `LEADER`, this FE node is the Leader FE node.

Stop the StarRocks cluster

You can stop the StarRocks cluster by running the following commands on the corresponding instances.

- Stop an FE node.

```
./fe/bin/stop_fe.sh --daemon
```

- Stop a BE node.

```
./be/bin/stop_be.sh --daemon
```

- Stop a CN node.

```
./be/bin/stop_cn.sh --daemon
```

Troubleshooting

Try the following steps to identify the errors that occur when you start the FE, BE, or CN nodes:

- If an FE node is not started properly, you can identify the problem by checking its log in **fe/log/fe.warn.log**.

  ```
  cat fe/log/fe.warn.log
  ```

  Having identified and resolved the problem, you must first terminate the current FE process, delete the existing **meta** directory, create a new metadata storage directory, and then restart the FE node with the correct configuration.

- If a BE node is not started properly, you can identify the problem by checking its log in **be/log/be.WARNING**.

  ```
  cat be/log/be.WARNING
  ```

  Having identified and resolved the problem, you must first terminate the existing BE process, delete the existing **storage** directory, create a new data storage directory, and then restart the BE node with the correct configuration.

- If a CN node is not started properly, you can identify the problem by checking its log in **be/log/cn.WARNING**.

  ```
  cat be/log/cn.WARNING
  ```

  Having identified and resolved the problem, you must first terminate the existing CN process, and then restart the CN node with the correct configuration.

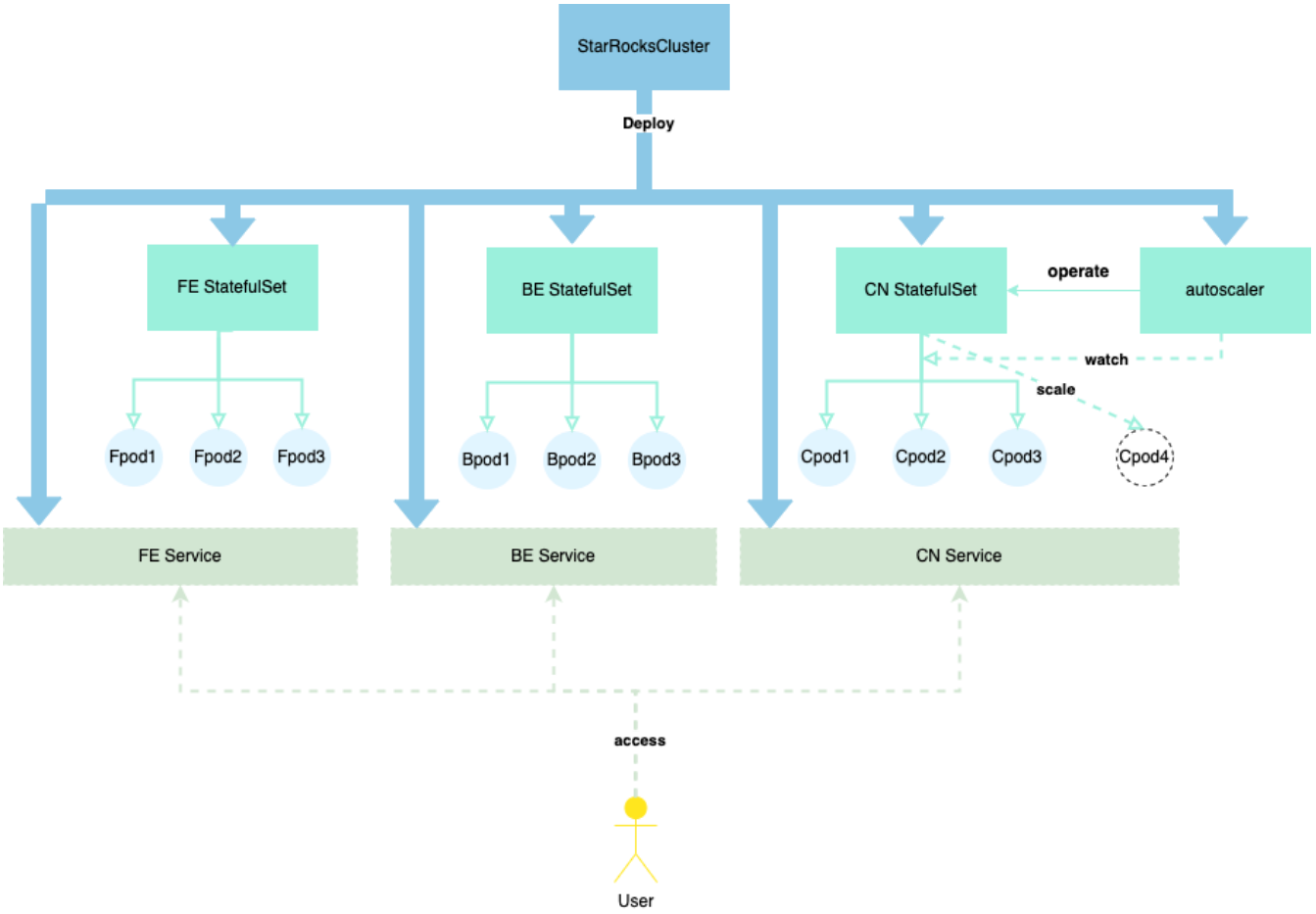
What to do next

Having deployed your StarRocks cluster, you can move on to Post-deployment Setup for instructions on initial management measures.

Deploy StarRocks with Operator

This topic introduces how to use the StarRocks Operator to automate the deployment and management of a StarRocks cluster on a Kubernetes cluster.

How it works



Before you begin

Create Kubernetes cluster

You can use the cloud-managed Kubernetes service, such as an Amazon Elastic Kubernetes Service (EKS) or Google Kubernetes Engine (GKE) cluster, or a self-managed Kubernetes cluster.

- Create an Amazon EKS cluster

  i. Check that the following command-line tools are installed in your environment:
     a. Install and configure AWS command-line tool AWS CLI.
     b. Install EKS cluster command-line tool eksctl.
     c. Install Kubernetes cluster command-line tool kubectl.
  ii. Use one of the following methods to create an EKS cluster:
     a. Use eksctl to quickly create an EKS cluster.
     b. Manually create an EKS cluster with the AWS console and AWS CLI.

- Create a GKE cluster

  Before you start to create a GKE cluster, make sure that you complete all the prerequisites. Then follow the instructions provided in Create a GKE cluster to create a GKE cluster.

- Create a self-managed Kubernetes cluster

  Follow the instructions provided in Bootstrapping clusters with kubeadm to create a self-managed Kubernetes cluster. You can use Minikube and Docker Desktop to create a single-node private Kubernetes cluster with minimum steps.

Deploy StarRocks Operator

1. Add the custom resource StarRocksCluster.

```
kubectl apply -f
https://raw.githubusercontent.com/StarRocks/starrocks-kubernetes-
operator/main/deploy/starrocks.com_starrocksclusters.yaml
```

2. Deploy the StarRocks Operator. You can choose to deploy the StarRocks Operator by using a default configuration file or a custom configuration file.

   i. Deploy the StarRocks Operator by using a default configuration file.

```
kubectl apply -f
https://raw.githubusercontent.com/StarRocks/starrocks-
kubernetes-operator/main/deploy/operator.yaml
```

The StarRocks Operator is deployed to the namespace `starrocks` and manages all StarRocks clusters under all namespaces.

ii. Deploy the StarRocks Operator by using a custom configuration file.

- Download the configuration file **operator.yaml**, which is used to deploy the StarRocks Operator.

```
curl -O
https://raw.githubusercontent.com/StarRocks/starrocks-
kubernetes-operator/main/deploy/operator.yaml
```

- Modify the configuration file **operator.yaml** to suit your needs.
- Deploy the StarRocks Operator.

```
kubectl apply -f operator.yaml
```

3. Check the running status of the StarRocks Operator. If the pod is in the `Running` state and all containers inside the pod are `READY`, the StarRocks Operator is running as expected.

```
$ kubectl -n starrocks get pods
NAME                                  READY   STATUS    RESTARTS
AGE
starrocks-controller-65bb8679-jkbtg   1/1     Running   0
5m6s
```

> **NOTE**
>
> If you customize the namespace in which the StarRocks Operator is located, you need to replace `starrocks` with the name of your customized namespace.

Deploy StarRocks Cluster

You can directly use the sample configuration files provided by StarRocks to deploy a StarRocks cluster (an object instantiated by using the custom resource StarRocks Cluster). For example, you can use **starrocks-fe-and-be.yaml** to deploy a StarRocks cluster that contains three FE nodes and three BE nodes.

```
kubectl apply -f
https://raw.githubusercontent.com/StarRocks/starrocks-kubernetes-
operator/main/examples/starrocks/starrocks-fe-and-be.yaml
```

The following table describes a few important fields in the **starrocks-fe-and-be.yaml** file.

Field	Description
Kind	The resource type of the object. The value must be `StarRocksCluster`.
Metadata	Metadata, in which the following sub-fields are nested: • `name`: the name of the object. Each object name uniquely identifies an object of the same resource type. • `namespace`: the namespace to which the object belongs.
Spec	The expected status of the object. Valid values are `starRocksFeSpec`, `starRocksBeSpec`, and `starRocksCnSpec`.

You can also deploy the StarRocks cluster by using a modified configuration file. For supported fields and detailed descriptions, see api.md.

Deploying the StarRocks cluster takes a while. During this period, you can use the command `kubectl -n starrocks get pods` to check the starting status of the StarRocks cluster. If all the pods are in the `Running` state and all containers inside the pods are `READY`, the StarRocks cluster is running as expected.

> **NOTE**
>
> If you customize the namespace in which the StarRocks cluster is located, you need to replace `starrocks` with the name of your customized namespace.

```
$ kubectl -n starrocks get pods
NAME                                   READY   STATUS    RESTARTS
AGE
starrocks-controller-65bb8679-jkbtg    1/1     Running   0
22h
starrockscluster-sample-be-0           1/1     Running   0
23h
starrockscluster-sample-be-1           1/1     Running   0
23h
starrockscluster-sample-be-2           1/1     Running   0
22h
starrockscluster-sample-fe-0           1/1     Running   0
21h
starrockscluster-sample-fe-1           1/1     Running   0
21h
starrockscluster-sample-fe-2           1/1     Running   0
22h
```

> **Note**
>
> If some pods cannot start after a long period of time, you can use `kubectl logs -n starrocks <pod_name>` to view the log information or use `kubectl -n starrocks describe pod <pod_name>` to view the event information to locate the problem.

Manage StarRocks Cluster

Access StarRocks Cluster

The components of the StarRocks cluster can be accessed through their associated Services, such as the FE Service. For detailed descriptions of Services and their access addresses, see api.md and Services.

> **NOTE**
>
> - Only the FE Service is deployed by default. If you need to deploy the BE Service and CN Service, you need to configure `starRocksBeSpec` and `starRocksCnSpec` in the StarRocks cluster configuration file.
> - The name of a Service is `<cluster name>-<component name>-service` by default, for example, `starrockscluster-sample-fe-service`. You can also specify the Service name in the spec of each component.

Access StarRocks Cluster from within Kubernetes Cluster

From within the Kubernetes cluster, the StarRocks cluster can be accessed through the FE Service's ClusterIP.

1. Obtain the internal virtual IP address `CLUSTER-IP` and port `PORT(S)` of the FE Service.

```
$ kubectl -n starrocks get svc
NAME                                    TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)                                    AGE
be-domain-search                        ClusterIP   None
<none>        9050/TCP                                   23m
fe-domain-search                        ClusterIP   None
<none>        9030/TCP                                   25m
starrockscluster-sample-fe-service   ClusterIP   10.100.162.xxx
<none>        8030/TCP,9020/TCP,9030/TCP,9010/TCP   25m
```

2. Access the StarRocks cluster by using the MySQL client from within the Kubernetes cluster.

```
mysql -h 10.100.162.xxx -P 9030 -uroot
```

Access StarRocks Cluster from outside Kubernetes Cluster

From outside the Kubernetes cluster, you can access the StarRocks cluster through the FE Service's LoadBalancer or NodePort. This topic uses LoadBalancer as an example:

1. Run the command `kubectl -n starrocks edit src starrockscluster-sample` to update the StarRocks cluster configuration file, and change the Service type of `starRocksFeSpec` to `LoadBalancer`.

```
starRocksFeSpec:
  image: starrocks/fe-ubuntu:3.0-latest
  replicas: 3
  requests:
    cpu: 4
    memory: 16Gi
  service:
    type: LoadBalancer # specified as LoadBalancer
```

2. Obtain the IP address `EXTERNAL-IP` and port `PORT(S)` that the FE Service exposes to the outside.

```
$ kubectl -n starrocks get svc
NAME                                   TYPE           CLUSTER-IP
EXTERNAL-IP
PORT(S)                                                          AGE
be-domain-search                       ClusterIP      None
<none>
9050/TCP
127m
fe-domain-search                       ClusterIP      None
<none>
9030/TCP
129m
starrockscluster-sample-fe-service   LoadBalancer
10.100.162.xxx   a7509284bf3784983a596c6eec7fc212-618xxxxxx.us-
west-2.elb.amazonaws.com
8030:30629/TCP,9020:32544/TCP,9030:32244/TCP,9010:32024/TCP
129m               ClusterIP      None            <none>
9030/TCP                                                         23h
```

3. Log in to your machine host and access the StarRocks cluster by using the MySQL client.

```
mysql -h a7509284bf3784983a596c6eec7fc212-618xxxxxx.us-west-
2.elb.amazonaws.com -P9030 -uroot
```

Upgrade StarRocks Cluster

Upgrade BE nodes

Run the following command to specify a new BE image file, such as `starrocks/be-ubuntu:latest`:

```
kubectl -n starrocks patch starrockscluster starrockscluster-sample --type='merge' -p '{"spec":{"starRocksBeSpec":{"image":"starrocks/be-ubuntu:latest"}}}'
```

Upgrade FE nodes

Run the following command to specify a new FE image file, such as `starrocks/fe-ubuntu:latest`:

```
kubectl -n starrocks patch starrockscluster starrockscluster-sample --type='merge' -p '{"spec":{"starRocksFeSpec":{"image":"starrocks/fe-ubuntu:latest"}}}'
```

The upgrade process lasts for a while. You can run the command `kubectl -n starrocks get pods` to view the upgrade progress.

Scale StarRocks cluster

This topic takes scaling out the BE and FE clusters as examples.

Scale out BE cluster

Run the following command to scale out the BE cluster to 9 nodes:

```
kubectl -n starrocks patch starrockscluster starrockscluster-sample --type='merge' -p '{"spec":{"starRocksBeSpec":{"replicas":9}}}'
```

Scale out FE cluster

Run the following command to scale out the FE cluster to 4 nodes:

```
kubectl -n starrocks patch starrockscluster starrockscluster-sample --type='merge' -p '{"spec":{"starRocksFeSpec":{"replicas":4}}}'
```

The scaling process lasts for a while. You can use the command `kubectl -n starrocks get pods` to view the scaling progress.

Automatic scaling for CN cluster

Run the command `kubectl -n starrocks edit src starrockscluster-sample` to configure the automatic scaling policy for the CN cluster. You can specify the resource metrics for CNs as the average CPU utilization, average memory usage, elastic scaling threshold, upper elastic scaling limit, and lower elastic scaling limit. The upper elastic

scaling limit and lower elastic scaling limit specify the maximum number and minimum number of CNs allowed for elastic scaling.

> **NOTE**
>
> If the automatic scaling policy for the CN cluster is configured, delete the `replicas` field from the `starRocksCnSpec` in the StarRocks cluster configuration file.

Kubernetes also supports using `behavior` to customize scaling behaviors according to business scenarios, helping you achieve rapid or slow scaling or disable scaling. For more information about automatic scaling policies, see Horizontal Pod Scaling.

The following is a template provided by StarRocks to help you configure automatic scaling policies:

```
  starRocksCnSpec:
    image: starrocks/cn-ubuntu:latest
    limits:
      cpu: 16
      memory: 64Gi
    requests:
      cpu: 16
      memory: 64Gi
    autoScalingPolicy: # Automatic scaling policy of the CN cluster.
      maxReplicas: 10 # The maximum number of CNs is set to 10.
      minReplicas: 1 # The minimum number of CNs is set to 1.
      # operator creates an HPA resource based on the following
field.
      # see https://kubernetes.io/docs/tasks/run-
application/horizontal-pod-autoscale/ for more information.
      hpaPolicy:
        metrics: # Resource metrics
          - type: Resource
            resource:
              name: memory  # The average memory usage of CNs is
specified as a resource metric.
              target:
                # The elastic scaling threshold is 60%.
                # When the average memory utilization of CNs exceeds
60%, the number of CNs increases for scale-out.
                # When the average memory utilization of CNs is below
60%, the number of CNs decreases for scale-in.
                averageUtilization: 60
                type: Utilization
```

```
          - type: Resource
            resource:
              name: cpu # The average CPU utilization of CNs is
specified as a resource metric.
              target:
                # The elastic scaling threshold is 60%.
                # When the average CPU utilization of CNs exceeds
60%, the number of CNs increases for scale-out.
                # When the average CPU utilization of CNs is below
60%, the number of CNs decreases for scale-in.
                averageUtilization: 60
                type: Utilization
        behavior: #  The scaling behavior is customized according to
business scenarios, helping you achieve rapid or slow scaling or
disable scaling.
          scaleUp:
            policies:
              - type: Pods
                value: 1
                periodSeconds: 10
          scaleDown:
            selectPolicy: Disabled
```

The following table describes a few important fields:

- The upper and lower elastic scaling limit.

```
maxReplicas: 10 # The maximum number of CNs is set to 10.
minReplicas: 1 # The minimum number of CNs is set to 1.
```

- The elastic scaling threshold.

```
# For example, the average CPU utilization of CNs is specified as a
resource metric.
# The elastic scaling threshold is 60%.
# When the average CPU utilization of CNs exceeds 60%, the number of
CNs increases for scale-out.
# When the average CPU utilization of CNs is below 60%, the number of
CNs decreases for scale-in.
- type: Resource
  resource:
    name: cpu
```

```
    target:
      averageUtilization: 60
```

FAQ

Issue description: When a custom resource StarRocksCluster is installed using `kubectl apply -f xxx`, an error is returned `The CustomResourceDefinition 'starrocksclusters.starrocks.com' is invalid: metadata.annotations: Too long: must have at most 262144 bytes`.

Cause analysis: Whenever `kubectl apply -f xxx` is used to create or update resources, a metadata annotation `kubectl.kubernetes.io/last-applied-configuration` is added. This metadata annotation is in JSON format and records the *last-applied-configuration*. `kubectl apply -f xxx`" is suitable for most cases, but in rare situations , such as when the configuration file for the custom resource is too large, it may cause the size of the metadata annotation to exceed the limit.

Solution: If you install the custom resource StarRocksCluster for the first time, it is recommended to use `kubectl create -f xxx`. If the custom resource StarRocksCluster is already installed in the environment, and you need to update its configuration, it is recommended to use `kubectl replace -f xxx`.

Deploy StarRocks with Helm

Helm is a package manager for Kubernetes. A Helm Chart is a Helm package and contains all of the resource definitions necessary to run an application on a Kubernetes cluster. This topic describes how to use Helm to automatically deploy a StarRocks cluster on a Kubernetes cluster.

Before you begin

- Create a Kubernetes cluster.
- Install Helm.

Procedure

1. Add the Helm Chart Repo for StarRocks. The Helm Chart contains the definitions of the StarRocks Operator and the custom resource StarRocksCluster.

   i. Add the Helm Chart Repo.

   ```
   helm repo add starrocks-community
   https://starrocks.github.io/starrocks-kubernetes-operator
   ```

   ii. Update the Helm Chart Repo to the latest version.

   ```
   helm repo update
   ```

   iii. View the Helm Chart Repo that you added.

   ```
   $ helm search repo starrocks-community
   NAME                                    CHART VERSION    APP
   VERSION  DESCRIPTION
   starrocks-community/kube-starrocks      1.8.0            3.1-
   latest   kube-starrocks includes two subcharts, starrock...
   ```

```
starrocks-community/operator            1.8.0           1.8.0
A Helm chart for StarRocks operator
starrocks-community/starrocks           1.8.0           3.1-
latest   A Helm chart for StarRocks cluster
```

2. Use the default **values.yaml** of the Helm Chart to deploy the StarRocks Operator and StarRocks cluster, or create a YAML file to customize your deployment configurations.

   i. Deployment with default configurations

   Run the following command to deploy the StarRocks Operator and the StarRocks cluster which consists of one FE and one BE:

   ```
   $ helm install starrocks starrocks-community/kube-starrocks
   # If the following result is returned, the StarRocks Operator
   and StarRocks cluster are being deployed.
   NAME: starrocks
   LAST DEPLOYED: Tue Aug 15 15:12:00 2023
   NAMESPACE: starrocks
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   ```

   ii. Deployment with custom configurations

   - Create a YAML file, for example, **my-values.yaml**, and customize the configurations for the StarRocks Operator and StarRocks cluster in the YAML file. For the supported parameters and descriptions, see the comments in the default **values.yaml** of the Helm Chart.
   - Run the following command to deploy the StarRocks Operator and StarRocks cluster with the custom configurations in **my-values.yaml**.

     ```
     helm install -f my-values.yaml starrocks starrocks-
     community/kube-starrocks
     ```

Deployment takes a while. During this period, you can check the deployment status by using the prompt command in the returned result of the deployment command above. The default prompt command is as follows:

```
$ kubectl --namespace default get starrockscluster -l
"cluster=kube-starrocks"
# If the following result is returned, the deployment has been
successfully completed.
NAME             FESTATUS   CNSTATUS   BESTATUS
kube-starrocks   running               running
```

You can also run `kubectl get pods` to check the deployment status. If all Pods are in the `Running` state and all containers within the Pods are `READY`, the deployment has been successfully completed.

```
$ kubectl get pods
NAME                                      READY   STATUS
RESTARTS   AGE
kube-starrocks-be-0                       1/1     Running   0
2m50s
kube-starrocks-fe-0                       1/1     Running   0
4m31s
kube-starrocks-operator-69c5c64595-pc7fv  1/1     Running   0
4m50s
```

Next steps

- Access StarRocks cluster

  You can access the StarRocks cluster from inside and outside the Kubernetes cluster. For detailed instructions, see Access StarRocks Cluster.

- Manage StarRocks operator and StarRocks cluster

  - If you need to update the configurations of the StarRocks operator and StarRocks cluster, see Helm Upgrade.

  - If you need to uninstall the StarRocks Operator and StarRocks cluster, run the following command:

    ```
    helm uninstall starrocks
    ```

More information

- The address of the GitHub repository: starrocks-kubernetes-operator and kube-starrocks Helm Chart.
- The docs in the GitHub repository provide more information, for example:
  - If you need to manage objects like the StarRocks cluster via the Kubernetes API, see API reference.
  - If you need to mount persistent volumes to FE and BE pods to store FE metadata and logs, as well as BE data and logs, see Mount Persistent Volumes by Helm Chart.

    > **DANGER**
    >
    > If persistent volumes are not mounted, the StarRocks Operator will use emptyDir to store FE metadata and logs, as well as BE data and logs. When containers restart, data will be lost.

  - If you need to set the root user password:
    - Manually set the root user's password after deploying the StarRocks cluster, see Change root user password HOWTO.
    - Automatically set the root user's password when deploying the StarRocks cluster, see Initialize root user password.
- How to resolve the following error that occurs after a CREATE TABLE statement is executed in a StarRocks shared-data cluster.
  - **Error message**

    ```
    ERROR 1064 (HY000): Table replication num should be less than
    or equal to the number of available BE nodes. You can change
    this default by setting the replication_num table properties.
    Current alive backend is [10001]. , table=orders1,
    default_replication_num=3
    ```

  - **Solution**

This may be because only one BE exists in that StarRocks shared-data cluster, which supports only one replica. However, the default number of replicas is 3. You can modify the number of replicas to 1 in PROPERTIES, such as, `PROPERTIES( "replication_num" = "1" )`.

- The address of Helm Chart maintained by StarRocks on Artifact Hub: kube-starrocks.

Post-deployment setup

This topic describes tasks that you should perform after deploying StarRocks.

Upgrade StarRocks

This topic describes how to upgrade your StarRocks cluster.

Downgrade StarRocks

This topic describes how to downgrade your StarRocks cluster.

Post-deployment setup

This topic describes tasks that you should perform after deploying StarRocks.

Before getting your new StarRocks cluster into production, you must secure the initial account and set the necessary variables and properties to allow your cluster to run properly.

Secure initial account

Upon the creation of a StarRocks cluster, the initial `root` user of the cluster is generated automatically. The `root` user is granted the `root` privileges, which are the collection of all privileges within the cluster. We recommend you secure this user account and avoid using it in production to prevent misuse.

StarRocks automatically assigns an empty password to the `root` user when the cluster is created. Follow these procedures to set a new password for the `root` user:

1. Connect to StarRocks via your MySQL client with the username `root` and an empty password.

```
# Replace <fe_address> with the IP address (priority_networks) or
FQDN
# of the FE node you connect to, and replace <query_port>
# with the query_port (Default: 9030) you specified in fe.conf.
mysql -h <fe_address> -P<query_port> -uroot
```

2. Reset the password of the `root` user by executing the following SQL:

```
-- Replace <password> with the password you want to assign to the
root user.
SET PASSWORD = PASSWORD('<password>')
```

> **NOTE**

- Keep the password properly after resetting it. If you forgot the password, see Reset lost root password for detailed instructions.
- After completing the post-deployment setup, you can create new users and roles to manage the privileges within your team. See Manage user privileges for detailed instructions.

Set necessary system variables

To allow your StarRocks cluster to work properly in production, you need to set the following system variables:

Variable name	StarRocks Version	Recommended value	Description
is_report_success	v2.4 or earlier	false	The boolean switch that controls whether to send the profile of a query for analysis. The default value is `false`, which means no profile is required. Setting this variable to `true` can affect the concurrency of StarRocks.
enable_profile	v2.5 or later	false	The boolean switch that

Variable name	StarRocks Version	Recommended value	Description
			controls whether to send the profile of a query for analysis. The default value is `false`, which means no profile is required. Setting this variable to `true` can affect the concurrency of StarRocks.
enable_pipeline_engine	v2.3 or later	true	The boolean switch that controls whether to enable the pipeline execution engine. `true` indicates enabled and `false` indicates the opposite. Default value: `true`.

Variable name	StarRocks Version	Recommended value	Description
parallel_fragment_exec_instance_num	v2.3 or later	If you have enabled the pipeline engine, you can set this variable to `1`.If you have not enabled the pipeline engine, you should set it to half the number of CPU cores.	The number of instances used to scan nodes on each BE. The default value is `1`.
pipeline_dop	v2.3, v2.4, and v2.5	0	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. From v3.0 onwards, StarRocks

Variable name	StarRocks Version	Recommended value	Description
			adaptively adjusts this parameter based on query parallelism.

- Set `is_report_success` to `false` globally:

```
SET GLOBAL is_report_success = false;
```

- Set `enable_profile` to `false` globally:

```
SET GLOBAL enable_profile = false;
```

- Set `enable_pipeline_engine` to `true` globally:

```
SET GLOBAL enable_pipeline_engine = true;
```

- Set `parallel_fragment_exec_instance_num` to `1` globally:

```
SET GLOBAL parallel_fragment_exec_instance_num = 1;
```

- Set `pipeline_dop` to `0` globally:

```
SET GLOBAL pipeline_dop = 0;
```

For more information about system variables, see System variables.

Set user property

If you have created new users in your cluster, you need to enlarge their maximum connection number (to `1000`, for example):

```
-- Replace <username> with the username you want to enlarge the
maximum connection number for.
SET PROPERTY FOR '<username>' 'max_user_connections' = '1000';
```

What to do next

After deploying and setting up your StarRocks cluster, you can then proceed to design tables that best work for your scenarios. See Understand StarRocks table design for detailed instructions on designing a table.

Upgrade StarRocks

This topic describes how to upgrade your StarRocks cluster.

Overview

Review the information in this section before upgrading, and perform any recommended actions.

StarRocks versions

The version of StarRocks is represented by three numbers in the form **Major.Minor.Patch**, for example, `2.5.4`. The first number represents the major version of StarRocks, the second number represents the minor version, and the third number represents the patch version.

Upgrade paths

- **For patch version upgrade**

  You can upgrade your StarRocks cluster across patch versions, for example, from v2.2.6 directly to v2.2.11.

- **For minor version upgrade**

  From StarRocks v2.0 onwards, you can upgrade a StarRocks cluster across minor versions, for example, from v2.2.x directly to v2.5.x. However, for compatibility and safety reasons, we strongly recommend you upgrade your StarRocks cluster **consecutively from one minor version to another**. For example, to upgrade a StarRocks v2.2 cluster to v2.5, you need to upgrade it in the following order: v2.2.x --> v2.3.x --> v2.4.x --> v2.5.x.

- **For major version upgrade**

  To upgrade your StarRocks cluster to v3.0, you must first upgrade it to v2.5.

> **CAUTION**
>
> Suppose you need to perform consecutive minor version upgrades, for example, 2.4->2.5->3.0->3.1->3.2, or you have downgraded your cluster after a failed upgrade and you want to upgrade the cluster again, for example, 2.5->3.0->2.5->3.0. To prevent metadata upgrade failure for some Follower FEs, perform the following steps between two consecutive upgrades or after the downgrade before the second trial of upgrade:
>
> 1. Run ALTER SYSTEM CREATE IMAGE to create a new image.
> 2. Wait for the new image to be synchronized to all Follower FEs.
>
> You can check whether the image file has been synchronized by viewing the log file **fe.log** of the Leader FE. A record of log like "push image.* from subdir [] to other nodes. totally xx nodes, push successful xx nodes" suggests that the image file has been successfully synchronized.

Upgrade procedure

StarRocks supports **rolling upgrades**, which allow you to upgrade your cluster without stopping the service. By design, BEs and CNs are backward compatible with the FEs. Therefore, you need to **upgrade BEs and CNs first and then FEs** to allow your cluster to run properly while being upgraded. Upgrading them in an inverted order may lead to incompatibility between FEs and BEs/CNs, and thereby cause the service to crash. For FE nodes, you must first upgrade all Follower FE nodes before upgrading the Leader FE node.

Before you begin

During preparation, you must perform the compatibility configuration if you are up for a minor or major version upgrade. You also need to perform the upgrade availability test on one of the FEs and BEs before upgrading all nodes in the cluster.

Perform compatibility configuration

If you want to upgrade your StarRocks cluster to a later minor or major version, you must perform the compatibility configuration. In addition to the universal compatibility configuration, detailed configurations vary depending on the version of the StarRocks cluster you upgrade from.

- **Universal compatibility configuration**

Before upgrading your StarRocks cluster, you must disable tablet clone. You can skip this step if you have disabled the balancer.

```
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_scheduling_tablets" =
"0");
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_balancing_tablets" =
"0");
ADMIN SET FRONTEND CONFIG ("disable_balance"="true");
ADMIN SET FRONTEND CONFIG ("disable_colocate_balance"="true");
```

After the upgrade, and the status of all BE nodes is `Alive`, you can re-enable tablet clone.

```
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_scheduling_tablets" =
"10000");
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_balancing_tablets" =
"500");
ADMIN SET FRONTEND CONFIG ("disable_balance"="false");
ADMIN SET FRONTEND CONFIG ("disable_colocate_balance"="false");
```

- **If you upgrade from v2.0 to later versions**

Before upgrading your StarRocks v2.0 cluster, you must set the following BE configuration and system variable.

1. If you have modified the BE configuration item `vector_chunk_size`, you must set it to `4096` before upgrading. Because it is a static parameter, you must modify it in the BE configuration file **be.conf** and restart the node to allow the modification to take effect.

2. Set the system variable `batch_size` to less than or equal to `4096` globally.

   ```
   SET GLOBAL batch_size = 4096;
   ```

Upgrade BE

Having passed the upgrade availability test, you can first upgrade the BE nodes in the cluster.

1. Navigate to the working directory of the BE node and stop the node.

```
# Replace <be_dir> with the deployment directory of the BE node.
cd <be_dir>/be
./bin/stop_be.sh
```

2. Replace the original deployment files under **bin** and **lib** with the ones of the new version.

```
mv lib lib.bak
mv bin bin.bak
cp -r /tmp/StarRocks-x.x.x/be/lib  .
cp -r /tmp/StarRocks-x.x.x/be/bin  .
```

3. Start the BE node.

```
sh bin/start_be.sh --daemon
```

4. Check if the BE node is started successfully.

```
ps aux | grep starrocks_be
```

5. Repeat the above procedures to upgrade other BE nodes.

Upgrade CN

1. Navigate to the working directory of the CN node and stop the node gracefully.

```
# Replace <cn_dir> with the deployment directory of the CN node.
cd <cn_dir>/be
./bin/stop_cn.sh --graceful
```

2. Replace the original deployment files under **bin** and **lib** with the ones of the new version.

```
mv lib lib.bak
mv bin bin.bak
```

```
cp -r /tmp/StarRocks-x.x.x/be/lib  .
cp -r /tmp/StarRocks-x.x.x/be/bin  .
```

3. Start the CN node.

```
sh bin/start_cn.sh --daemon
```

4. Check if the CN node is started successfully.

```
ps aux | grep starrocks_be
```

5. Repeat the above procedures to upgrade other CN nodes.

Upgrade FE

After upgrading all BE and CN nodes, you can then upgrade the FE nodes. You must first upgrade the Follower FE nodes and then the Leader FE node.

1. Navigate to the working directory of the FE node and stop the node.

```
# Replace <fe_dir> with the deployment directory of the FE node.
cd <fe_dir>/fe
./bin/stop_fe.sh
```

2. Replace the original deployment files under **bin**, **lib**, and **spark-dpp** with the ones of the new version.

```
mv lib lib.bak
mv bin bin.bak
mv spark-dpp spark-dpp.bak
cp -r /tmp/StarRocks-x.x.x/fe/lib  .
cp -r /tmp/StarRocks-x.x.x/fe/bin  .
cp -r /tmp/StarRocks-x.x.x/fe/spark-dpp  .
```

3. Start the FE node.

```
```

```
sh bin/start_fe.sh --daemon
```

4. Check if the FE node is started successfully.

```
ps aux | grep StarRocksFE
```

5. Repeat the above procedures to upgrade other Follower FE nodes, and finally the Leader FE node.

Downgrade StarRocks

This topic describes how to downgrade your StarRocks cluster.

If an exception occurs after you upgrade a StarRocks cluster, you can downgrade it to the earlier version to quickly recover the cluster.

Overview

Review the information in this section before downgrading. Perform any recommended actions.

Downgrade paths

- **For patch version downgrade**

  You can downgrade your StarRocks cluster across patch versions, for example, from v2.2.11 directly to v2.2.6.

- **For minor version downgrade**

  For compatibility and safety reasons, we strongly recommend you downgrade your StarRocks cluster **consecutively from one minor version to another**. For example, to downgrade a StarRocks v2.5 cluster to v2.2, you need to downgrade it in the following order: v2.5.x --> v2.4.x --> v2.3.x --> v2.2.x.

- **For major version downgrade**

  You can only downgrade your StarRocks v3.0 cluster to v2.5.3 and later versions.

  - StarRocks upgrades the BDB library in v3.0. However, BDBJE cannot be rolled back. You must use BDB library of v3.0 after a downgrade.
  - The new RBAC privilege system is used by default after you upgrade to v3.0. You can only use the RBAC privilege system after a downgrade.

> **CAUTION**

> Suppose you have downgraded your cluster after a failed upgrade and you want to upgrade the cluster again, for example, 2.5->3.0->2.5->3.0. To prevent metadata upgrade failure for some Follower FEs, perform the following steps after the downgrade:
>
> 1. Run ALTER SYSTEM CREATE IMAGE to create a new image.
> 2. Wait for the new image to be synchronized to all Follower FEs.
>
> You can check whether the image file has been synchronized by viewing the log file **fe.log** of the Leader FE. A record of log like "push image.* from subdir [] to other nodes. totally xx nodes, push successful xx nodes" suggests that the image file has been successfully synchronized.

Downgrade procedure

StarRocks' downgrade procedure is the reverse order of the upgrade procedure. Therefore, you need to **downgrade FEs first and then BEs and CNs**. Downgrading them in the wrong order may lead to incompatibility between FEs and BEs/CNs, and thereby cause the service to crash. For FE nodes, you must first downgrade all Follower FE nodes before downgrading the Leader FE node.

Before you begin

During preparation, you must perform the compatibility configuration if you are up for a minor or major version downgrade. You also need to perform the downgrade availability test on one of the FEs or BEs before downgrading all nodes in the cluster.

Perform compatibility configuration

If you want to downgrade your StarRocks cluster to an earlier minor or major version, you must perform the compatibility configuration. In addition to the universal compatibility configuration, detailed configurations vary depending on the version of the StarRocks cluster you downgrade from.

- **Universal compatibility configuration**

Before downgrading your StarRocks cluster, you must disable tablet clone. You can skip this step if you have disabled the balancer.

```
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_scheduling_tablets" =
"0");
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_balancing_tablets" =
"0");
ADMIN SET FRONTEND CONFIG ("disable_balance"="true");
ADMIN SET FRONTEND CONFIG ("disable_colocate_balance"="true");
```

After the downgrade, you can enable tablet clone again if the status of all BE nodes becomes `Alive`.

```
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_scheduling_tablets" =
"10000");
ADMIN SET FRONTEND CONFIG ("tablet_sched_max_balancing_tablets" =
"500");
ADMIN SET FRONTEND CONFIG ("disable_balance"="false");
ADMIN SET FRONTEND CONFIG ("disable_colocate_balance"="false");
```

- **If you downgrade from v2.2 and later versions**

Set the FE configuration item `ignore_unknown_log_id` to `true`. Because it is a static parameter, you must modify it in the FE configuration file **fe.conf** and restart the node to allow the modification to take effect. After the downgrade and the first checkpoint are completed, you can reset it to `false` and restart the node.

- **If you have enabled FQDN access**

If you have enabled FQDN access (supported from v2.4 onwards) and need to downgrade to versions earlier than v2.4, you must switch to IP address access before downgrading. See Rollback FQDN for detailed instructions.

Downgrade FE

After the compatibility configuration and the availability test, you can downgrade the FE nodes. You must first downgrade the Follower FE nodes and then the Leader FE node.

1. Navigate to the working directory of the FE node and stop the node.

```
# Replace <fe_dir> with the deployment directory of the FE node.
cd <fe_dir>/fe
```

```
./bin/stop_fe.sh
```

2. Replace the original deployment files under **bin**, **lib**, and **spark-dpp** with the ones of the earlier version.

```
mv lib lib.bak
mv bin bin.bak
mv spark-dpp spark-dpp.bak
cp -r /tmp/StarRocks-x.x.x/fe/lib  .
cp -r /tmp/StarRocks-x.x.x/fe/bin  .
cp -r /tmp/StarRocks-x.x.x/fe/spark-dpp  .
```

> **CAUTION**
>
> If you are downgrading StarRocks v3.0 to v2.5, you must follow these steps after you replace the deployment files:
>
> i. Copy the file **fe/lib/starrocks-bdb-je-18.3.13.jar** of the v3.0 deployment to the directory **fe/lib** of the v2.5 deployment.
> ii. Delete the file **fe/lib/je-7.*.jar**.

3. Start the FE node.

```
sh bin/start_fe.sh --daemon
```

4. Check if the FE node is started successfully.

```
ps aux | grep StarRocksFE
```

5. Repeat the above procedures to downgrade other Follower FE nodes, and finally the Leader FE node.

> **CAUTION**
>
> If you are downgrading StarRocks v3.0 to v2.5, you must follow these steps after the downgrade:
>
> i. Run ALTER SYSTEM CREATE IMAGE to create a new image.
> ii. Wait for the new image to be synchronized to all Follower FEs.

> If you do not run this command, some of the downgrade operations may fail.
> ALTER SYSTEM CREATE IMAGE is supported from v2.5.3 and later.

Downgrade BE

Having downgraded the FE nodes, you can then downgrade the BE nodes in the cluster.

1. Navigate to the working directory of the BE node and stop the node.

```
# Replace <be_dir> with the deployment directory of the BE node.
cd <be_dir>/be
./bin/stop_be.sh
```

2. Replace the original deployment files under **bin** and **lib** with the ones of the earlier version.

```
mv lib lib.bak
mv bin bin.bak
cp -r /tmp/StarRocks-x.x.x/be/lib  .
cp -r /tmp/StarRocks-x.x.x/be/bin  .
```

3. Start the BE node.

```
sh bin/start_be.sh --daemon
```

4. Check if the BE node is started successfully.

```
ps aux | grep starrocks_be
```

5. Repeat the above procedures to downgrade other BE nodes.

Downgrade CN

1. Navigate to the working directory of the CN node and stop the node gracefully.

```
# Replace <cn_dir> with the deployment directory of the CN node.
cd <cn_dir>/be
./bin/stop_cn.sh --graceful
```

2. Replace the original deployment files under **bin** and **lib** with the ones of the earlier version.

```
mv lib lib.bak
mv bin bin.bak
cp -r /tmp/StarRocks-x.x.x/be/lib  .
cp -r /tmp/StarRocks-x.x.x/be/bin  .
```

3. Start the CN node.

```
sh bin/start_cn.sh --daemon
```

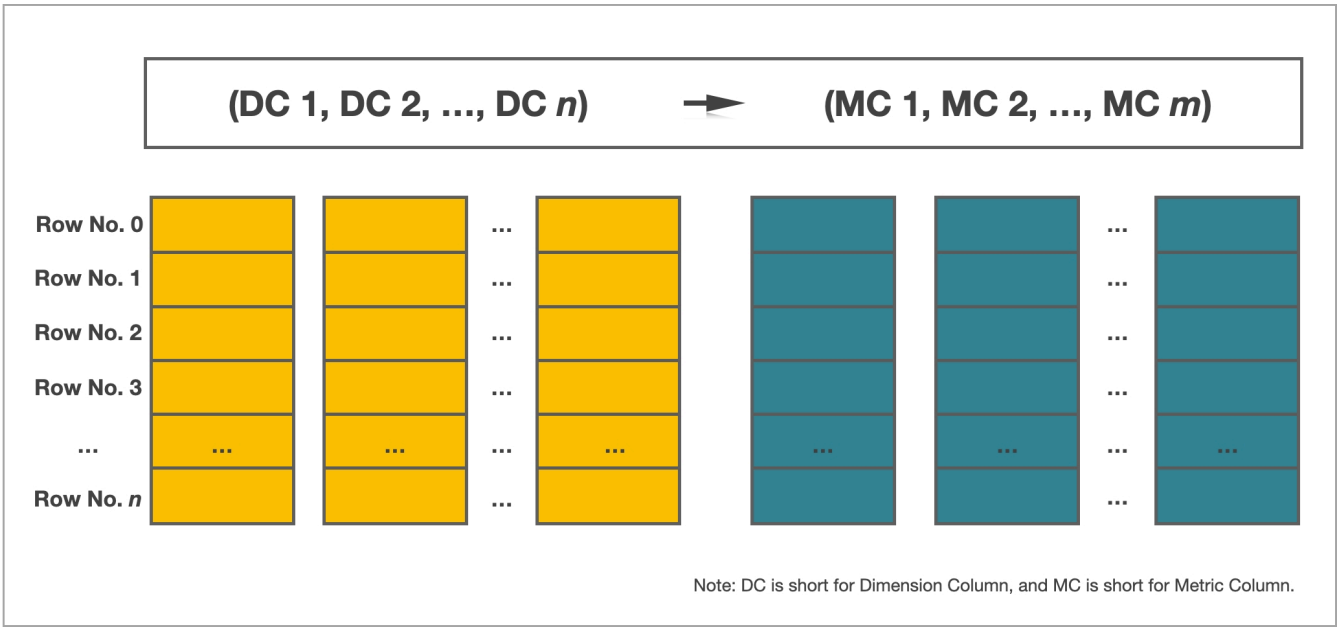
4. Check if the CN node is started successfully.

```
ps aux | grep  starrocks_be
```

5. Repeat the above procedures to downgrade other CN nodes.

Understand table design

Columnar storage



Like in other relational databases, tables in StarRocks consist of rows and columns. Each row holds a record of user data, and data in each column has the same type. All rows in a table have the same number of columns. You can dynamically add columns to or delete columns from a table. The columns of a table can be categorized as dimension columns and metric columns. Dimension columns are also called key columns, and metric columns are also called value columns. Values in dimension columns are used to group and sort data, and values in metric columns can be accumulated by using functions such as sum, count, min, max, hll_union_agg, and bitmap_union.

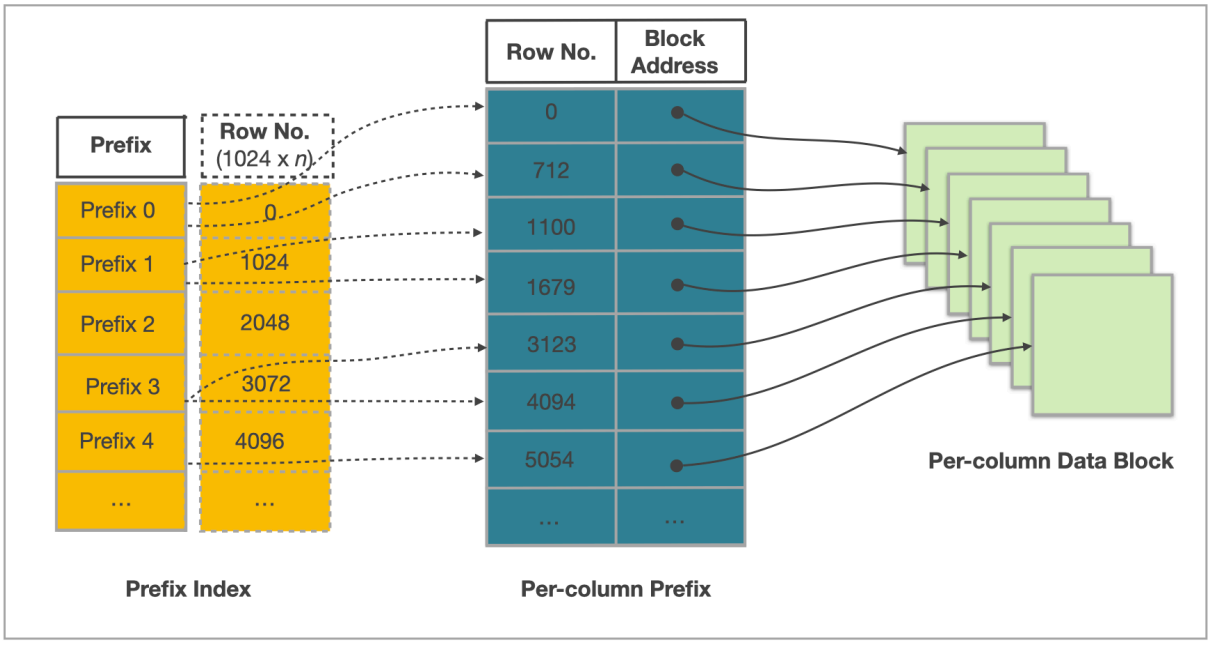
StarRocks uses columnar storage for tables. Physically, the data in a column is segregated into data blocks, encoded, compressed, and then persistently stored on disk. Logically, the data in a column can be compared to an array that consists of elements of the same data type. The column values held in a row keep are listed as elements in the column order in their respective arrays. This means the column values held in a row have the same array index. Array indexes are implicit and do not need to be stored. All rows in a table are sorted in the order specified by one or more dimension columns. The location of a row in the sorted table is represented by the sequence number of that row.

For a query on a table, if you specify equality or range conditions on specific dimension columns that can comprise a dimension column prefix, StarRocks can run binary searches to quickly locate the rows of interest among sorted data. For example, you want to query data from a table named `table1`, and the table consists of four columns: `event_day`, `siteid`, `citycode`, and `username`, among which `event_day` and `siteid` are dimension columns. If you specify `event_day = 2020-09-18` and `siteid = 2` as query conditions, StarRocks can run binary searches and only needs to process the data within the specified range, because `event_day` and `siteid` can comprise a dimension column prefix. If you specify `citycode = 4` and `username = Andy` as query conditions, StarRocks cannot run binary searches and needs to process the data of the entire table, because `citycode` and `username` cannot comprise a dimension column prefix.

Indexing

StarRocks uses prefix indexes and per-column indexes to quickly locate the starting rows of the data blocks taken by rows of interest.

The following figure shows how the StarRocks table design works to accelerate queries on a table in StarRocks.



The data of a table in StarRocks is organized into the following three parts:

- Prefix index

StarRocks stores the data of every 1024 rows as a data block, for which an entry is maintained in the prefix index table. The content of the prefix index entry for each data block is the prefix composed of the dimension columns for the starting row in the data block and cannot exceed 36 bytes in length. The prefix index is a sparse index. When you query a row, StarRocks searches the prefix index table to retrieve the prefix that is composed of the dimension columns for the row. Then, StarRocks can quickly locate the sequence number of the starting row in the data block taken by the row of interest.

- Per-column data block

  StarRocks segregates the data of each column into multiple 64-KB data blocks. Each data block is independently encoded and compressed as the minimal I/O unit and is read from or written to disk as a whole.

- Per-column index

  StarRocks maintains a row number index for each column. In the row number index table, data blocks for the column are mapped one by one onto the sequence numbers of the rows held in the column. Additionally, each entry in the row number index table consists of the starting row number, address, and length of the data block that is mapped onto a specific row number. When you query a row, StarRocks searches the row number index table to retrieve the address of the data block mapped onto the sequence number of the row. Then, StarRocks reads the data block to locate the row.

In summary, StarRocks performs the following five steps to locate a row of a table by using the prefix that is composed of the dimension columns for the row:

1. Search the prefix index table to locate the sequence number of the starting row in the data block taken by the row of interest.

2. Search the row number index table of each dimension column to locate the data blocks for the dimension column.

3. Read the data blocks.

4. Decompress and decode the data blocks.

5. Search the data blocks to locate the row onto which the dimension column index is mapped.

Accelerated processing

This section introduces the mechanisms that help StarRocks process data at higher speeds.

Pre-aggregation

StarRocks provides the Aggregate table. For an Aggregate table, the rows that have the same values in the table's dimension columns can be aggregated into a single row. The value in each dimension column remains unchanged for the new row generated from the aggregation, and the values in each metric column are aggregated by the aggregate function you specify to produce the resulting value for the new row in the metric column. Pre-aggregation helps accelerate aggregate operations.

Partitioning and bucketing

Each table in StarRocks is divided into multiple tablets. Each tablet is stored in multiple replicas on BEs. The number of BEs and the number of tablets can scale flexibly in line with changes in computing resources and data sizes. When you initiate a query, multiple BEs can search tablets in parallel to quickly locate the data of interest. Additionally, tablet replicas can be replicated and migrated, which increases data reliability and prevents data skews. Partitioning and bucketing help ensure the efficiency and stability of data retrieval.

Materialized view

The prefix index of a table helps accelerate queries on the table but rely on the sequence of the table's dimension columns. If you construct query predicates by using dimension columns that are not included in a dimension column prefix, the prefix index does not work. In this case, you can create a materialized view for the table. The data of the materialized view is organized and stored in the same way as the data of the table. However, the materialized view can have its own prefix index. When you create a prefix index for the materialized view, you can specify appropriate aggregation granularity, column count, and dimension column ordering to ensure that frequently used query conditions can hit expected entries in the prefix index table for the materialized view.

Per-column index

StarRocks supports per-column indexes such as Bloom filters, zone maps, and bitmap indexes:

- A Bloom filter is used to determine whether data blocks contain the values you want to query.
- A zone map is used to locate the values within a specified range.
- A bitmap index is used to locate the rows that meet specified query conditions in a column of the ENUM data type.

Table Types

You must specify a table type and define one or more columns as a sort key at table creation. This way, when data is initially loaded into the table that you created, StarRocks can sort, process, and store the data based on the sort key. This topic describes the table types that StarRocks provides to meet your varying business requirements.

Basic concepts

Table types

StarRocks provides four table types: Duplicate Key table, Aggregate table, Unique Key table, and Primary Key table. These four table types are well suited to a wide range of data analytics scenarios such as log analysis, data aggregation and analysis, and real-time data analysis.

Sort keys

When data is loaded into a table created by using a certain table type, data is sorted and stored according to one or more columns defined as the sort key when the table is created. The sort key is usually one or more columns that are frequently used as filter conditions in queries, thereby accelerating queries.

- In the Duplicate Key table, the sort key specified by `DUPLICATE KEY` is used to sort data and is not assigned a UNIQUE constraint.
- In the Aggregate table, the sort key specified by `AGGREGATE KEY` is used to sort data and is assigned a UNIQUE constraint.
- In the Unique Key table, the sort key specified by `UNIQUE KEY` is used to sort data and is assigned a UNIQUE constraint.
- In the Primary Key table, the sort key specified by `PRIMARY KEY` is used to sort data and is assigned UNIQUE and NOT NULL constraints.

> **NOTE**

> For more descriptions of sort keys, see Sort keys and prefix indexes.

Precautions

- After a table is created, you cannot modify its table types. For example, you cannot change a Duplicate Key table to a Primary Key table. You must create a new Primary Key table.
- Sort key columns must be defined prior to the other columns in the statement for table creation.
- The order of sort key columns in the statement for table creation specifies the order of the conditions based on which the rows in the table are sorted.
- The length of the prefix index for a table is limited to 36 bytes. If the total length of the sort key columns exceeds 36 bytes, StarRocks stores only the first few sort key columns within the length limit for the prefix index.
- If the records to be loaded into a table have the same primary key, StarRocks processes and stores the records based on the table type:
    - Duplicate Key table

      StarRocks loads each of the records as a separate row into the table. After the data load is complete, the table contains rows that have the same primary key, and the rows map the source records in a one-to-one relationship. You can recall all historical data that you loaded.
    - Aggregate table

      StarRocks aggregates the records into one record and loads the aggregated record as a row into the table. After the loading is complete, the table does not contain rows that have the same primary key. You can recall the aggregation results of all historical data that you loaded. However, you cannot recall all historical data.
    - Unique Key table and Primary Key table

      StarRocks replaces each newly loaded record with the previously loaded record and retains only the most recently loaded record as a row in the table. After the loading is complete, the table does not contain rows that have the same primary

key. The Unique Key table and the Primary Key table can be considered a special Aggregate table in which the REPLACE aggregate function is specified for metric columns to return the most recent record among a group of records that have the same primary key.

Duplicate Key table

The Duplicate Key table is the default model in StarRocks. If you did not specify a model when you create a table, a Duplicate Key table is created by default.

When you create a Duplicate Key table, you can define a sort key for that table. If the filter conditions contain the sort key columns, StarRocks can quickly filter data from the table to accelerate queries. The Duplicate Key table allows you to append new data to the table. However, it does not allow you to modify existing data in the table.

Scenarios

The Duplicate Key table is suitable for the following scenarios:

- Analyze raw data, such as raw logs and raw operation records.
- Query data by using a variety of methods without being limited by the pre-aggregation method.
- Load log data or time-series data. New data is written in append-only mode, and existing data is not updated.

Create a table

Suppose that you want to analyze the event data over a specific time range. In this example, create a table named `detail` and define `event_time` and `event_type` as sort key columns.

Statement for creating the table:

```
CREATE TABLE IF NOT EXISTS detail (
    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)
DISTRIBUTED BY HASH(user_id) BUCKETS 8;
```

> You must specify `DISTRIBUTED BY HASH`. Otherwise, the table creation fails.

Usage notes

- Take note of the following points about the sort key of a table:
  - You can use the `DUPLICATE KEY` keyword to explicitly define the columns that are used in the sort key.

    > Note: By default, if you do not specify sort key columns, StarRocks uses the **first three** columns as sort key columns.

  - In the Duplicate Key table, the sort key can consist of some or all of the dimension columns.
- You can create indexes such as BITMAP indexes and Bloomfilter indexes at table creation.
- If two identical records are loaded, the Duplicate Key table retains them as two records, rather than one.

What to do next

After a table is created, you can use various data ingestion methods to load data into StarRocks. For information about the data ingestion methods that are supported by StarRocks, see Overview of data loading.

> Note: When you load data into a table that uses the Duplicate Key table, you can only append data to the table. You cannot modify the existing data in the table.

Aggregate table

When you create a table that uses the Aggregate table, you can define sort key columns and metric columns and can specify an aggregate function for the metric columns. If the records to be loaded have the same sort key, the metric columns are aggregated. The Aggregate table helps reduce the amount of data that needs to be processed for queries, thereby accelerating queries.

Scenarios

The Aggregate table is well suited to data statistics and analytics scenarios. A few examples are as follows:

- Help website or app providers analyze the amount of traffic and time that their users spend on a specific website or app and the total number of visits to the website or app.
- Help advertising agencies analyze the total clicks, total views, and consumption statistics of an advertisement that they provide for their customers.
- Help e-commerce companies analyze their annual trading data to identify the geographic bestsellers within individual quarters or months.

The data querying and ingestion in the preceding scenarios have the following characteristics:

- Most queries are aggregate queries, such as SUM, MAX, and MIN.
- Raw detailed data does not need to be retrieved.
- Historical data is not frequently updated. Only new data is appended.

Principle

Starting from data ingestion to data querying, data with the same sort key in a table that uses the Aggregate table is aggregated multiple times as follows:

1. In the data ingestion phase, when data is loaded as batches into the table, each batch comprises a data version. After a data version is generated, StarRocks aggregates the data that has the same sort key in the data version.
2. In the background compaction phase, when the files of multiple data versions that are generated at data ingestion are periodically compacted into a large file, StarRocks aggregates the data that has the same sort key in the large file.
3. In the data query phase, StarRocks aggregates the data that has the same sort key among all data versions before it returns the query result.

The aggregate operations help reduce the amount of data that needs to be processed, thereby accelerating queries.

Suppose that you have a table that uses the Aggregate table and want to load the following four raw records into the table.

Date	Country	PV
2020.05.01	CHN	1
2020.05.01	CHN	2
2020.05.01	USA	3
2020.05.01	USA	4

StarRocks aggregates the four raw records into the following two records at data ingestion.

Date	Country	PV
2020.05.01	CHN	3
2020.05.01	USA	7

Create a table

Suppose that you want to analyze the numbers of visits by users from different cities to different web pages. In this example, create a table named `example_db.aggregate_tbl`,

define `site_id`, `date`, and `city_code` as sort key columns, define `pv` as a metric column, and specify the SUM function for the `pv` column.

The statement for creating the table is as follows:

```
CREATE TABLE IF NOT EXISTS example_db.aggregate_tbl (
    site_id LARGEINT NOT NULL COMMENT "id of site",
    date DATE NOT NULL COMMENT "time of event",
    city_code VARCHAR(20) COMMENT "city_code of user",
    pv BIGINT SUM DEFAULT "0" COMMENT "total page views"
)
AGGREGATE KEY(site_id, date, city_code)
DISTRIBUTED BY HASH(site_id) BUCKETS 8
PROPERTIES (
"replication_num" = "3"
);
```

Usage notes

- Take note of the following points about the sort key of a table:
    - You can use the `AGGREGATE KEY` keyword to explicitly define the columns that are used in the sort key.
        - If the `AGGREGATE KEY` keyword does not include all the dimension columns, the table cannot be created.
        - By default, if you do not explicitly define sort key columns by using the `AGGREGATE KEY` keyword, StarRocks selects all columns except metric columns as the sort key columns.
    - The sort key must be created on columns on which unique constraints are enforced. It must be composed of all the dimension columns whose names cannot be changed.
- You can specify an aggregate function following the name of a column to define the column as a metric column. In most cases, metric columns hold data that needs to be aggregated and analyzed.
- For information about the aggregate functions that are supported by the Aggregate table, see CREATE TABLE.

- When queries are run, sort key columns are filtered before the aggregation of multiple data versions, whereas metric columns are filtered after the aggregation of multiple data versions. Therefore, we recommend that you identify the columns that are frequently used as filter conditions and define these columns as the sort key. This way, data filtering can start before the aggregation of multiple data versions to improve query performance.

- When you create a table, you cannot create BITMAP indexes or Bloom Filter indexes on the metric columns of the table.

What to do next

After a table is created, you can use various data ingestion methods to load data into StarRocks. For information about the data ingestion methods that are supported by StarRocks, see Data import.

> Note: When you load data into a table that uses the Aggregate table, you can only update all columns of the table. For example, when you update the preceding `example_db.aggregate_tbl` table, you must update all its columns, which are `site_id`, `date`, `city_code`, and `pv`.

Unique Key table

When you create a table, you can define primary key columns and metric columns. This way, queries return the most recent record among a group of records that have the same primary key. Compared with the Duplicate Key table, the Unique Key table simplifies the data loading process to better support real-time and frequent data updates.

Scenarios

The Unique Key table is suitable for business scenarios in which data needs to be frequently updated in real time. For example, in e-commerce scenarios, hundreds of millions of orders can be placed per day, and the statuses of the orders frequently change.

Principle

The Unique Key table can be considered a special Aggregate Key table in which the REPLACE aggregate function is specified for metric columns to return the most recent record among a group of records that have the same primary key.

When you load data into a table that uses the Unique Key table, the data is split into multiple batches. Each batch is assigned a version number. Therefore, records with the same primary key may come in multiple versions, of which the most recent version (namely, the record with the largest version number) is retrieved for queries.

As shown in the following table, `ID` is the primary key column, `value` is a metric column, and `_version` holds the data version numbers generated within StarRocks. In this example, the record with an `ID` of 1 is loaded by two batches whose version numbers are `1` and `2`, and the record with an `ID` of `2` is loaded by three batches whose version numbers are `3`, `4`, and `5`.

ID	value	_version
1	100	1

ID	value	_version
1	101	2
2	100	3
2	101	4
2	102	5

When you query the record with an `ID` of `1`, the most recent record with the largest version number, which is `2` in this case, is returned. When you query the record with an `ID` of `2`, the most recent record with the largest version number, which is `5` in this case, is returned. The following table shows the records returned by the two queries:

ID	value
1	101
2	102

Create a table

In e-commerce scenarios, you often need to collect and analyze the statuses of orders by date. In this example, create a table named `orders` to hold the orders, define `create_time` and `order_id`, which are frequently used as conditions to filter the orders, as primary key columns, and define the other two columns, `order_state` and `total_price`, as metric columns. This way, the orders can be updated in real time as their statuses change, and can be quickly filtered to accelerate queries.

The statement for creating the table is as follows:

```
CREATE TABLE IF NOT EXISTS orders (
    create_time DATE NOT NULL COMMENT "create time of an order",
    order_id BIGINT NOT NULL COMMENT "id of an order",
    order_state INT COMMENT "state of an order",
    total_price BIGINT COMMENT "price of an order"
```

```
)
UNIQUE KEY(create_time, order_id)
DISTRIBUTED BY HASH(order_id) BUCKETS 8;
```

Usage notes

- Take note of the following points about the primary key of a table:
    - The primary key is defined by using the `UNIQUE KEY` keyword.
    - The primary key must be created on columns on which unique constraints are enforced and whose names cannot be changed.
    - The primary key must be properly designed:
        - When queries are run, primary key columns are filtered before the aggregation of multiple data versions, whereas metric columns are filtered after the aggregation of multiple data versions. Therefore, we recommend that you identify the columns that are frequently used as filter conditions and define these columns as primary key columns. This way, data filtering can start before the aggregation of multiple data versions to improve query performance.
        - During the aggregation process, StarRocks compares all primary key columns. This is time-consuming and may decrease query performance. Therefore, do not define a large number of primary key columns. If a column is rarely used as a filter condition for queries, we recommend that you do not define the column as a primary key column.
- When you create a table, you cannot create BITMAP indexes or Bloom Filter indexes on the metric columns of the table.
- The Unique Key table does not support materialized views.

What to do next

After a table is created, you can use various data ingestion methods to load data into StarRocks. For information about the data ingestion methods that are supported by StarRocks, see Data import.

> - When you load data into a table that uses the Unique Key table, you can only update all columns of the table. For example, when you update the preceding

> `orders` table, you must update all its columns, which are `create_time`, `order_id`, `order_state`, and `total_price`.

- When you query data from a table that uses the Unique Key table, StarRocks needs to aggregate records of multiple data versions. In this situation, a large number of data versions decreases query performance. Therefore, we recommend that you specify a proper frequency at which data is loaded into the table to meet meet your requirements for real-time data analytics while preventing a large number of data versions. If you require minute-level data, you can specify a loading frequency of 1 minute instead of a loading frequency of 1 second.

Primary Key table

StarRocks has started to support the Primary Key table since v1.19. When you create a table that uses the Primary Key table, you can define primary key columns and metric columns. Queries return the most recent record among a group of records that have the same primary key. Unlike the Unique Key table, the Primary Key table does not require aggregate operations during queries and supports the pushdown of predicates and indexes. As such, the Primary Key table can deliver high query performance despite real-time and frequent data updates.

Scenarios

- The Primary Key table is suitable for the following scenarios in which data needs to be frequently updated in real time:
    - **Stream data in real time from transaction processing systems into StarRocks.** In normal cases, transaction processing systems involve a large number of update and delete operations in addition to insert operations. If you need to synchronize data from a transaction processing system to StarRocks, we recommend that you create a table that uses the Primary Key table. Then, you can use tools, such as CDC Connectors for Apache Flink®, to synchronize the binary logs of the transaction processing system to StarRocks. StarRocks uses the binary logs to add, delete, and update the data in the table in real time. This simplifies data synchronization and delivers 3 to 10 times higher query performance than when a Merge on Read (MoR) table of the Unique Key table is used. For example, you can use flink-connector-starrocks to load data. For more information, see Load data by using flink-connector-starrocks.
    - **Join multiple streams by performing update operations on individual columns**. In business scenarios such as user profiling, flat tables are preferably used to improve multi-dimensional analysis performance and simplify the analytics model that is used by data analysts. Upstream data in these scenarios may come from various apps, such as shopping apps, delivery apps, and banking apps, or from systems, such as machine learning systems that perform computations to obtain the distinct tags and properties of users. The Primary Key table is well suited in

these scenarios, because it supports updates to individual columns. Each app or system can update only the columns that hold the data within its own service scope while benefiting from real-time data additions, deletions, and updates at high query performance.

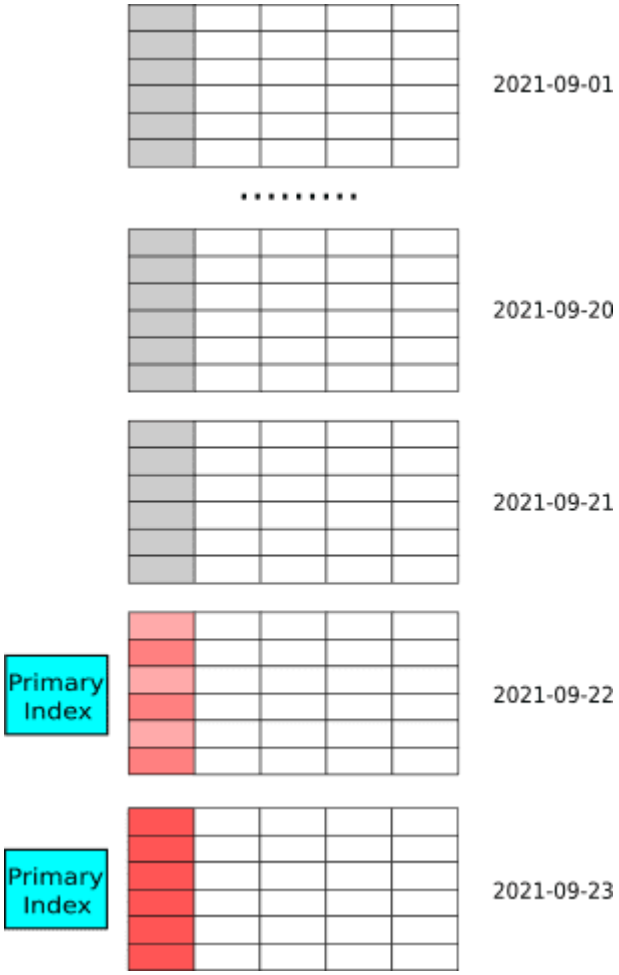
- The Primary Key table is suitable for scenarios in which the memory occupied by the primary key is controllable.

  The storage engine of StarRocks creates an index for the primary key of each table that uses the Primary Key table. Additionally, when you load data into a table, StarRocks loads the primary key index into the memory. Therefore, the Primary Key table requires a larger memory capacity than the other three table types. **StarRocks limits the total length of the fields that comprise the primary key to 127 bytes after encoding.**

  Consider using the Primary Key table if a table has the following characteristics:

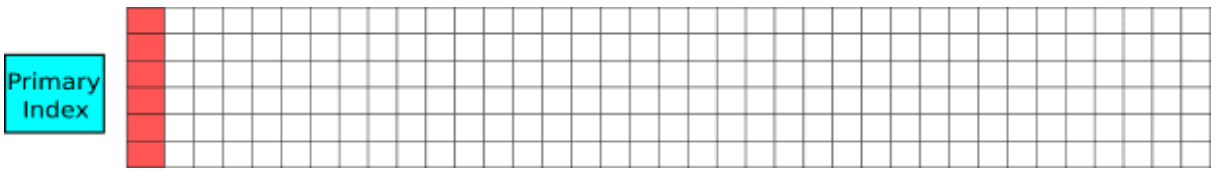
  - The table contains both fast-changing data and slow-changing data. Fast-changing data is frequently updated over the most recent days, whereas slow-changing data is rarely updated. Suppose that you need to synchronize a MySQL order table to StarRocks in real time for analytics and queries. In this example, the data of the table is partitioned by day, and most updates are performed on orders that are created within the most recent days. Historical orders are no longer updated after they are completed. When you run a data load job, the primary key index is not loaded into the memory and only the index entries of the recently updated orders are loaded into the memory.

    As shown in the following figure, the data in the table is partitioned by day, and the data in the most recent two partitions is frequently updated.



- The table is a flat table that is composed of hundreds or thousands of columns. The primary key comprises only a small portion of the table data and consumes only a small amount of memory. For example, a user status or profile table consists of a large number of columns but only tens to hundreds of millions of users. In this situation, the amount of memory consumed by the primary key is controllable.

  As shown in the following figure, the table contains only a few rows, and the primary key of the table comprises only a small portion of the table.



Principle

The Primary Key table is designed based on a new storage engine that is provided by StarRocks. The metadata structure and the read/write mechanism in the Primary Key table differ from those in the Duplicate Key table. As such, the Primary Key table does not require

aggregate operations and supports the pushdown of predicates and indexes. These significantly increase query performance.

The Duplicate Key table adopts the MoR policy. MoR streamlines data writes but requires online aggregation of multiple data versions. Additionally, the Merge operator does not support the pushdown of predicates and indexes. As a result, query performance deteriorates.

The Primary Key table adopts the Delete+Insert policy to ensure that each record has a unique primary key. This way, the Primary Key table does not require merge operations. Details are as follows:

- When StarRocks receives a request for an update operation on a record, it locates the record by searching the primary key index, marks the record as deleted, and inserts a new record. In other words, StarRocks converts an update operation to a delete operation plus an insert operation.
- When StarRocks receives a delete operation on a record, it locates the record by searching the primary key index and marks the record as deleted.

Create a table

Example 1: Suppose that you need to analyze orders on a daily basis. In this example, create a table named `orders`, define `dt` and `order_id` as the primary key, and define the other columns as metric columns.

```
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) BUCKETS 4
PROPERTIES("replication_num" = "3",
"enable_persistent_index" = "true");
```

Example 2: Suppose that you need to analyze user behavior in real time. In this example, create a table named `users`, define `user_id` as the primary key, and define the other columns as metric columns.

```
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) BUCKETS 4
PROPERTIES("replication_num" = "3",
"enable_persistent_index" = "true");
```

Usage notes

- Take note of the following points about the primary key of a table:
    - The primary key is defined by using the `PRIMARY KEY` keyword.
    - The primary key must be created on columns on which unique constraints are enforced, and the names of the primary key columns cannot be changed.
    - The primary key columns can be any of the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, STRING, VARCHAR, DATE, and DATETIME. However, the primary key columns cannot be defined as `NULL`.

    - The partition column and the bucket column must participate in the primary key.
    - The number and total length of primary key columns must be properly designed to save memory. We recommend that you identify columns whose data types occupy less memory and define those columns as the primary key. Such data types include INT and BIGINT. We recommend that you do not let a column of the VARCHAR data type to participate in the primary key.
    - Before you create the table, we recommend that you estimate the memory occupied by the primary key index based on the data types of the primary key columns and the number of rows in the table. This way, you can prevent the table from running out of memory. The following example explains how to calculate the memory occupied by the primary key index:
        - Suppose that the `dt` column, which is of the DATE data type that occupies 4 bytes, and the `id` column, which is of the BIGINT data type that occupies 8 bytes, are defined as the primary key. In this case, the primary key is 12 bytes in length.
        - Suppose that the table contains 10,000,000 rows of hot data and is stored in three replicas.
        - Given the preceding information, the memory occupied by the primary key index is 945 MB based on the following formula:

          (12 + 9) x 10,000,000 x 3 x 1.5 = 945 (MB)

          In the preceding formula, `9` is the immutable overhead per row, and `1.5` is the average extra overhead per hash table.
- `enable_persistent_index`: the primary key index can be persisted to disk and stored in memory to avoid it taking up too much memory. Generally, the primary key index can only take up 1/10 of the memory it does before. You can set this property in `PROPERTIES` when you create a table. Valid values are true or false. Default value is false.

  > - If you want to modify this parameter after the table is created, please see the part Modify the properties of table in ALTER TABLE.
  > - It is recommended to set this property to true if the disk is SSD.
  > - As of version 2.3.0, StarRocks supports to set this property.

- Since version 2.3.0, the indicator column now supports BITMAP, HLL data types.
- When you create a table, you cannot create BITMAP indexes or Bloom Filter indexes on the metric columns of the table.
- Since version 2.4.0, you can create asynchronous materialized views based on Primary Key tables.
- The Primary Key table does not support materialized views.
- You cannot use the ALTER TABLE statement to change the data types of the primary key columns and reorder metric columns. For the syntax and examples of using the ALTER TABLE statement, see ALTER TABLE.

What to do next

After table creation, you can run load jobs to load data into the Primary Key table. For more information about supported loading methods, see Overview of data loading.

If you need to update data in the Primary Key table, you can run a load job or execute a DML statement (UPDATE or DELETE). Also, these update operations guarantee atomicity.

Data distribution

When you create a table, you must specify the data distribution method by configuring the number of partitions and the number of tablets in a table. An appropriate data distribution method helps evenly distribute data across the nodes of your StarRocks cluster, reduces table scans, makes full use of the concurrency of the cluster, thereby improving query performance.

Basic concepts

Before you dive into the details of designing and managing data distribution, familiarize yourself with the following concepts:

- Partitioning

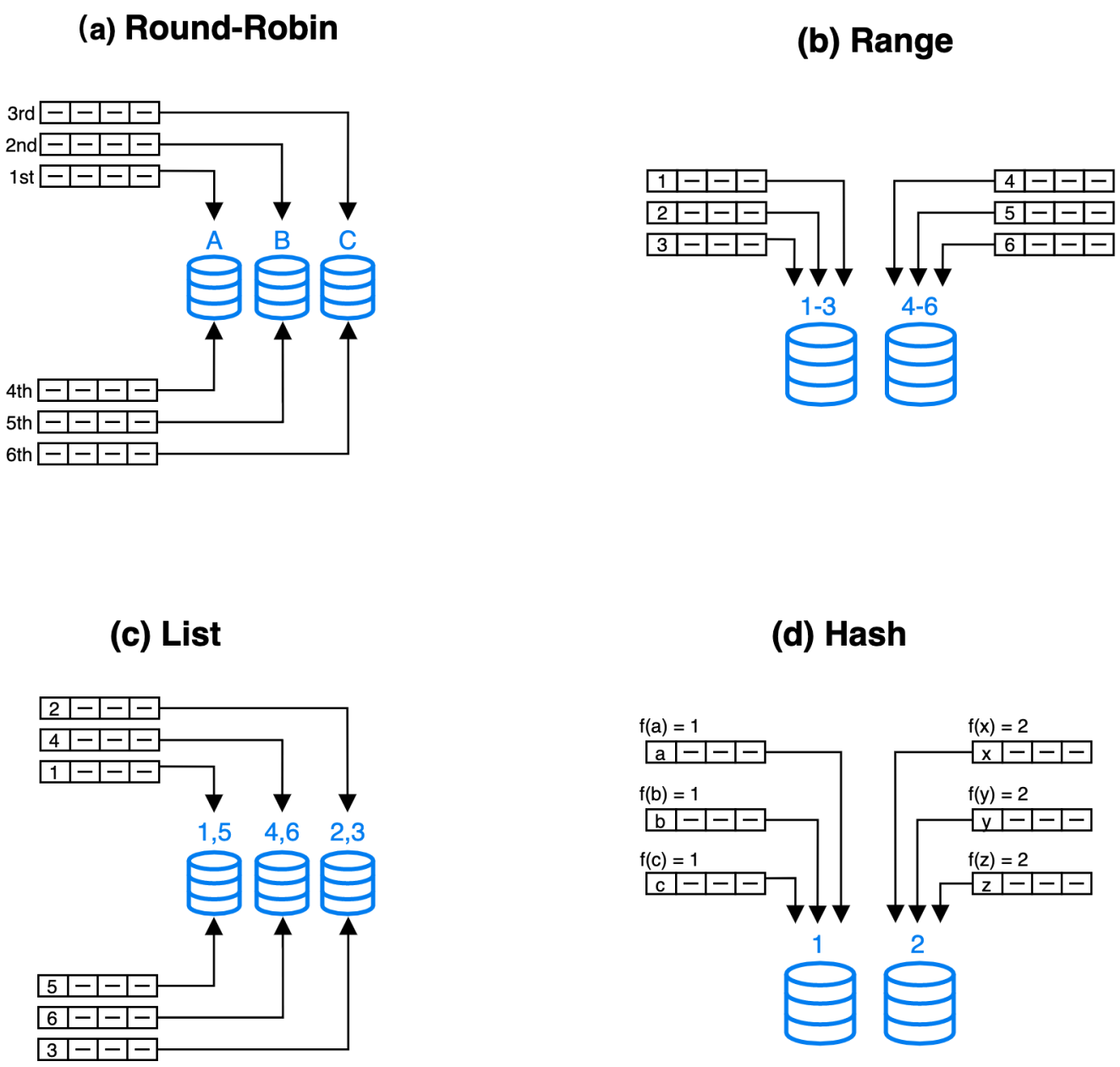
  Partitioning divides a table into multiple segments called partitions based on the partitioning column you specified. You can set a storage strategy for partitions, including the number of replicas, strategy of storing hot or cold data, and storage medium. StarRocks allows you to use multiple storage mediums within a cluster. For example, you can store the latest data on solid-state drives (SSD) to improve query performance, and historical data on SATA hard drives to reduce storage costs.

- Bucketing

  Bucketing divides a partition into multiple more manageable parts called tablets, which is the smallest unit of storage that you can use and allocate. StarRocks uses a hash algorithm to bucket data. Data with the same hash value of the bucketing column is distributed to the same tablet. StarRocks creates multiple replicas (three by default) for each tablet to prevent data loss. These replicas are managed by a separate local storage engine. You must specify the bucketing column when you create a table.

Partitioning methods

Modern distributed database systems generally use four basic partitioning methods: round-robin, range, list, and hash.



- **Round-robin**: distributes data across different nodes in a cyclic.
- **Range**: distributes data across different nodes based on the value range of partitioning columns.
- **List**: distributes data across different nodes based on the discrete values of partitioning columns, such as age.
- **Hash**: distributes data across different nodes based on a hash algorithm.

To achieve more flexible data distribution, you can combine the preceding four partitioning methods based on your business requirements, such as hash-hash, range-hash, and hash-list. **StarRocks offers the following two partitioning methods:**

- **Hash**: A hash-partitioned table has only one partition (the entire table is considered a partition). The partition is divided into tablets based on the bucketing column and the number of tablets that you specified.

  For example, the following statement creates a table `site_access`. The table is divided into 10 tablets based on the `site_id` column.

```
CREATE TABLE site_access(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(site_id, city_code, user_name)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

- **Range-hash**: A range-hash partitioned table divides data into multiple partitions based on the partitioning column. Each partition is further divided into tablets based on the bucketing column and the number of tablets that you specified.

  For example, the following statement creates a table `site_access` that is partitioned by the `event_day` column. The table contains 3 partitions: `p1`, `p2`, and `p3`. Each partition is divided into 10 tablets based on the `site_id` column.

```
CREATE TABLE site_access(
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY RANGE(event_day)
(
    PARTITION p1 VALUES LESS THAN ("2020-01-31"),
    PARTITION p2 VALUES LESS THAN ("2020-02-29"),
    PARTITION p3 VALUES LESS THAN ("2020-03-31")
)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

Design partitioning and bucketing rules

Before you partition a table, determine the partitioning column, bucketing column, and the number of tablets.

Choose partitioning columns

Data in a partitioned table is divided based on partitioning columns, also called partition keys. Generally, date or area is used as the partitioning column. In StarRocks, only the column of the DATE, DATETIME or INT type can be used as the partitioning column. We recommend that you follow the suggestions below to decide the partitioning column:

- The column with low cardinality.
- The column that is often used as a filter in queries.
- The amount of data in each partition must be less than 100 GB.

Choose bucketing columns

Data in partitions can be subdivided into tablets based on the hash values of the bucketing columns and the number of buckets. We recommend that you choose the column that satisfy the following two requirements as the bucketing column.

- high cardinality column such as ID
- column that often used as a filter in queries

But if the column that satisfies both requirements does not exist, you need to determine the bucketing column according to the complexity of queries.

- If the query is complex, it is recommended that you select the high cardinality column as the bucketing column to ensure that the data is as balanced as possible in each bucket and improve the cluster resource utilization.
- If the query is relatively simple, then it is recommended to select the column that is often used as in the query condition as the bucketing column to improve the query efficiency.

If partition data cannot be evenly distributed into each tablet by using one bucketing column, you can choose multiple bucketing columns. You can decide on the number of bucketing columns based on the following scenarios:

- One bucketing column: This method can reduce data transmission between nodes. It improves the performance of short-running query because short-running query only runs on one server and scan a small amount of data.
- Multiple bucketing columns: This method makes the most of the concurrency performance of a distributed cluster. It improves the performance of long-running query because long-running query runs across multiple servers and scan a large amount of data by using multiple servers in parallel. We recommend that you choose three bucketing columns at most.

Precautions

- **When a table is created, you must specify the bucketing columns**.
- The data types of bucketing columns must be INTEGER, DECIMAL, DATE/DATETIME, or CHAR/VARCHAR/STRING.
- Bucketing columns cannot be modified after they are specified.
- The values of bucketing columns cannot be updated.

Examples

The following statement creates a table named `site_access`.

```
CREATE TABLE site_access(
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY RANGE(event_day)
(
    PARTITION p1 VALUES LESS THAN ("2020-01-31"),
    PARTITION p2 VALUES LESS THAN ("2020-02-29"),
    PARTITION p3 VALUES LESS THAN ("2020-03-31")
)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

In this example, `site_access` uses `site_id` as the bucketing column because this column is always used as a filter in queries. When the bucketing column is used as a filter in queries, StarRocks only scans the relevant tablets, which greatly improves query performance.

Suppose you execute the following statement to query `site_access`. After you send the query, StarRocks only scans data in one tablet of `site_access`. The overall query speed is much faster than scanning 10 tablets.

```
select sum(pv)
from site_access
where site_id = 54321;
```

However, if `site_id` is unevenly distributed and a large number of queries are destined for only a few sites, the preceding bucketing method will cause severe data skew, causing system performance bottlenecks. In this case, you can use a combination of bucketing columns. For example, the following statement uses `site_id` and `city_code` as bucketing columns.

```
CREATE TABLE site_access
(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(site_id, city_code, user_name)
DISTRIBUTED BY HASH(site_id,city_code) BUCKETS 10;
```

Determine the number of tablets

Tablets reflect how data files are organized in StarRocks. Since StarRocks 2.5, you do not need to set the number of buckets when a table is created, and StarRocks sets the number of buckets automatically.

```
CREATE TABLE site_access(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(site_id, city_code, user_name)
DISTRIBUTED BY HASH(site_id,city_code); --do not need to set the
number of buckets
```

If you intend to set the number of buckets, StarRocks 2.4 and later versions support using multiple threads to scan a tablet in parallel during a query, thereby reducing the dependency of scanning performance on the tablet count. We recommend that each tablet contains about 10 GB of raw data. You can estimate the amount of data in each partition of a table and then decide the number of tablets. To enable the parallel scanning on tablets, make sure the `GLOBAL enable_tablet_internal_parallel` is enabled.

> Note: You cannot modify the number of tablets for an existing partition. You can only modify the number of tablets when you add a partition.

Manage partitions

Create partitions

You can partition a table in threes ways by using `PARTITION BY RANGE` clause:

- Partition a table with the LESS THAN clause. For more information, see CREATE TABLE.

```
PARTITION BY RANGE (k1, k2, ...)
(
    PARTITION partition_name1 VALUES LESS THAN ("value1",
"value2", ...),
    PARTITION partition_name2 VALUES LESS THAN ("value1",
"value2", ...),
    PARTITION partition_name3 VALUES LESS THAN (MAXVALUE)
)
```

- Partition a table by specifying values of a fixed range. For more information, see CREATE TABLE.

```
PARTITION BY RANGE (k1, k2, k3, ...)
(
    PARTITION partition_name1 VALUES [("k1-lower1", "k2-lower1",
"k3-lower1",...), ("k1-upper1", "k2-upper1", "k3-upper1", ...)],
    PARTITION partition_name2 VALUES [("k1-lower1-2", "k2-lower1-
2", ...), ("k1-upper1-2", MAXVALUE, )],
    "k3-upper1-2", ...
)
```

- Partition a table by specifying START, END, and EVERY. You can create multiple partitions at a time by using this method. For more information, see CREATE TABLE.

```
PARTITION BY RANGE (k1, k2, ...)
(
    START ("value1") END ("value2") EVERY (INTERVAL value3 day)
)
```

Examples

The following examples demonstrate how to partition a table by specifying START, END, and EVERY.

- The data type of the partitioning column is DATE and you specify the time range of partitioning via START and END and define the time range via EVERY. Example:

```
CREATE TABLE site_access (
   datekey DATE,
   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 ("2021-01-01") END ("2021-01-04") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(site_id) BUCKETS 10
PROPERTIES ("replication_num" = "3"
);
```

The PARTITION BY RANGE clause in this example is equal to the following:

```
PARTITION BY RANGE (datekey) (
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'))
)
```

- The data type of the partitioning column is DATE and you specify different EVERY clauses for different time ranges (which cannot overlap with each other). Example:

```
CREATE TABLE site_access(
    datekey DATE,
    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 ("2019-01-01") END ("2021-01-01") EVERY (INTERVAL 1
YEAR),
    START ("2021-01-01") END ("2021-05-01") EVERY (INTERVAL 1
MONTH),
    START ("2021-05-01") END ("2021-05-04") EVERY (INTERVAL 1
DAY)
)
DISTRIBUTED BY HASH(site_id) BUCKETS 10
PROPERTIES(
    "replication_num" = "1"
);
```

  The PARTITION BY RANGE clause in this example is equal to the following:

```
PARTITION BY RANGE (datekey) (
PARTITION p2019 VALUES [('2019-01-01'), ('2020-01-01')),
PARTITION p2020 VALUES [('2020-01-01'), ('2021-01-01')),
PARTITION p202101 VALUES [('2021-01-01'), ('2021-02-01')),
PARTITION p202102 VALUES [('2021-02-01'), ('2021-03-01')),
PARTITION p202103 VALUES [('2021-03-01'), ('2021-04-01')),
PARTITION p202104 VALUES [('2021-04-01'), ('2021-05-01')),
PARTITION p20210501 VALUES [('2021-05-01'), ('2021-05-02')),
PARTITION p20210502 VALUES [('2021-05-02'), ('2021-05-03')),
PARTITION p20210503 VALUES [('2021-05-03'), ('2021-05-04'))
)
```

- The data type of the partitioning column is INT and you specify the value range of partitioning by using START and END and define the incremental value via EVERY.

Example:

> Note: Do not double quote the incremental value defined by EVERY.

```
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) BUCKETS 10
PROPERTIES ("replication_num" = "1"
);
```

The PARTITION BY RANGE clause in this example is equal to the following:

```
PARTITION BY RANGE (datekey) (
PARTITION p2019 VALUES [('2019-01-01'), ('2020-01-01')),
PARTITION p2020 VALUES [('2020-01-01'), ('2021-01-01')),
PARTITION p202101 VALUES [('2021-01-01'), ('2021-02-01')),
PARTITION p202102 VALUES [('2021-02-01'), ('2021-03-01')),
PARTITION p202103 VALUES [('2021-03-01'), ('2021-04-01')),
PARTITION p202104 VALUES [('2021-04-01'), ('2021-05-01')),
PARTITION p20210501 VALUES [('2021-05-01'), ('2021-05-02')),
PARTITION p20210502 VALUES [('2021-05-02'), ('2021-05-03')),
PARTITION p20210503 VALUES [('2021-05-03'), ('2021-05-04'))
)
```

- After a table is created, you can use the ALTER TABLE statement to add partitions for the table.

```
ALTER TABLE site_access
ADD PARTITIONS START ("2021-01-04") END ("2021-01-06") EVERY
(INTERVAL 1 DAY);
```

Add a partition

You can add new partitions to store input data in a table. By default, the number of tablets in the new partition is equal to the number of tablets in the existing partitions. You can also adjust the number of tablets according to the amount of data in the new partition.

The following statement adds a new partition `p4` to table `site_access` to store data generated in April.

```
ALTER TABLE site_access
ADD PARTITION p4 VALUES LESS THAN ("2020-04-30")
DISTRIBUTED BY HASH(site_id) BUCKETS 20;
```

Delete a partition

The following statement deletes partition `p1` from table `site_access`.

```
ALTER TABLE site_access
DROP PARTITION p1;
```

> Note: If a partition is deleted within 24 hours, the data of the partition remains in Trash. During the period, you can execute the RECOVER statement to restore this data.

Restore a partition

The following statement restores partition `p1` to table `site_access`.

```
RECOVER PARTITION p1 FROM site_access;
```

View partitions

The following statement displays all partitions in table `site_access`.

```
SHOW PARTITIONS FROM site_access;
```

Dynamic partitioning

StarRocks supports dynamic partitioning, which can automatically manage the time to life (TTL) of partitions, such as partitioning new input data in tables and deleting expired partitions. This feature significantly reduces maintenance costs.

Enable dynamic partitioning

Take table `site_access` as an example. To enable dynamic partitioning, you need to configure the PROPERTIES parameter. For information about the configuration items, see CREATE TABLE.

```
CREATE TABLE site_access(
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY RANGE(event_day)(
    PARTITION p20200321 VALUES LESS THAN ("2020-03-22"),
    PARTITION p20200322 VALUES LESS THAN ("2020-03-23"),
    PARTITION p20200323 VALUES LESS THAN ("2020-03-24"),
    PARTITION p20200324 VALUES LESS THAN ("2020-03-25")
)
DISTRIBUTED BY HASH(event_day, site_id) BUCKETS 32
PROPERTIES(
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-3",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32",
    "dynamic_partition.history_partition_num" = "0"
);
```

`PROPERTIES`:

parameter	required	description
dynamic_partition.enable	No	Enables dynamic partitioning. Vali values are `TRUE` and `FALSE`. The default value is `TRUE`.
dynamic_partition.time_unit	Yes	The time granularity for dynamically created partitions. It i a required parameter. Valid values are `HOUR`, `DAY`, `WEEK`, `MONTH`, an `YEAR`. The time granularity determines the suffix format for dynamically created partitions. • If the value is `DAY`, the suffix format for dynamically create partitions is yyyyMMdd. An example partition name suffix is `20200321`. • If the value is `WEEK`, the suffi format for dynamically create partitions is yyyy_ww, for example `2020_13` for the 13t week of 2020. • If the value is `MONTH`, the suffix format for dynamically created partitions is yyyyMM, for example `202003`. • If the value is `YEAR`, the suffi format for dynamically create partitions is yyyy, for example `2020`.

parameter	required	description
dynamic_partition.time_zone	No	The time zone for dynamic partitions, which is by default the same as the system time zone.
dynamic_partition.start	No	The starting offset of dynamic partitioning. The value of this parameter must be a negative integer. The partitions before this offset will be deleted based on the current day, week, or month which is determined by the value of the parameter `dynamic_partition.time_unit`. The default value is `Integer.MIN_VALUE`, namely, -2147483648, which means that the history partitions will not be deleted.
dynamic_partition.end	Yes	The end offset of dynamic partitioning. The value of this parameter must be a positive integer. The partitions from the current day, week, or month to the end offset will be created in advance.
dynamic_partition.prefix	No	The prefix added to the names of dynamic partitions. The default value is `p`.
dynamic_partition.buckets	No	The number of buckets per dynamic partition. The default value is the same as the number of buckets determined by the

parameter	required	description
		reserved word BUCKETS or automatically set by StarRocks.
dynamic_partition.history_partition_num	No	The number of historical partitions created by the dynamic partitioning mechanism, with a default value of `0`. When the value is greater than 0, historical partitions are created in advance. From v2.5.2, StarRocks supports this parameter.
dynamic_partition.start_day_of_week	No	When `dynamic_partition.time_unit` is `WEEK`, this parameter is used to specify the first day of each week. Valid values: `1` to `7`. `1` means Monday and `7` means Sunday. The default value is `1`, which means that every week starts on Monday.
dynamic_partition.start_day_of_month	No	When `dynamic_partition.time_unit` is `MONTH`, this parameter is used to specify the first day of each month. Valid values: `1` to `28`. `1` means the 1st of every month and `28` means the 28th of every month. The default value is `1`, which means that every month starts on the 1st. The first day can not be the 29th, 30th, or 31st.
dynamic_partition.replication_num	No	The number of replicas for tablets in dynamically created partitions. The default value is the same as

parameter	required	description
		the number of replicas configured at table creation.

FE configuration:

`dynamic_partition_check_interval_seconds`: the interval for scheduling dynamic partitioning. The default value is 600s, which means that the partition situation is checked every 10 minutes to see whether the partitions meet the dynamic partitioning conditions specified in `PROPERTIES`. If not, the partitions will be created and deleted automatically.

View partitions

After you enable dynamic partitions for a table, the input data is continuously and automatically partitioned. You can view the current partitions by using the following statement. For example, if the current date is 2020-03-25, you can only see partitions in the time range from 2020-03-22 to 2020-03-28.

```
SHOW PARTITIONS FROM site_access;

[types: [DATE]; keys: [2020-03-22]; ..types: [DATE]; keys: [2020-03-
23]; )
[types: [DATE]; keys: [2020-03-23]; ..types: [DATE]; keys: [2020-03-
24]; )
[types: [DATE]; keys: [2020-03-24]; ..types: [DATE]; keys: [2020-03-
25]; )
[types: [DATE]; keys: [2020-03-25]; ..types: [DATE]; keys: [2020-03-
26]; )
[types: [DATE]; keys: [2020-03-26]; ..types: [DATE]; keys: [2020-03-
27]; )
[types: [DATE]; keys: [2020-03-27]; ..types: [DATE]; keys: [2020-03-
28]; )
[types: [DATE]; keys: [2020-03-28]; ..types: [DATE]; keys: [2020-03-
29]; )
```

Modify properties of dynamic partitioning

You can use the ALTER TABLE statement to modify properties of dynamic partitioning, such as disabling dynamic partitioning. Take the following statement as an example.

```
ALTER TABLE site_access
SET("dynamic_partition.enable"="false");
```

> Note:
>
> - To check the properties of dynamic partitioning of a table, execute the SHOW CREATE TABLE statement.
> - You can also use the ALTER TABLE statement to modify other properties of a table.

Temporary partition

This topic describes how to use the temporary partition feature.

You can create temporary partitions on a partitioned table that already has defined partitioning rules and define new data distribution strategies for these temporary partitions. Temporary partitions can serve as temporary data carriers when you atomically overwrite data in a partition or when you adjust partitioning and bucketing strategies. For temporary partitions, you can reset data distribution strategies such as partition ranges, number of buckets, and properties such as the number of replicas, and the storage medium, to meet specific requirements.

You can use the temporary partition feature in the following scenarios:

- Atomic overwrite operation

  If you need to rewrite the data in a partition while ensuring that the data can be queried during the rewriting process, you can first create a temporary partition based on the original formal partition, and load the new data into the temporary partition. Then you can use the replace operation to atomically replace the original formal partition with the temporary partition. For atomic overwrite operations on non-partitioned tables, see ALTER TABLE - SWAP.

- Adjust partition data query concurrency

  If you need to modify the number of buckets for a partition, you can first create a temporary partition with the same partition range as the original formal partition and specify the new number of buckets. Then, you can use the `INSERT INTO` command to load the data of the original formal partition into the temporary partition. Finally, you can use the replace operation to atomically replace the original formal partition with the temporary partition.

- Modify partitioning rules

  If you want to modify a partitioning strategy, such as merging partitions or splitting a large partition into multiple smaller partitions, you can first create temporary partitions with the expected merged or split ranges. Then, you can use the `INSERT INTO`

command to load the data of the original formal partitions into the temporary partitions. Finally, you can use the replace operation to atomically replace the original formal partitions with the temporary partitions.

Create temporary partitions

You can create one or more partitions at a time by using the ALTER TABLE command.

Syntax

Create a single temporary partition

```
ALTER TABLE <table_name>
ADD TEMPORARY PARTITION <temporary_partition_name> VALUES
[("value1"), {MAXVALUE|("value2")})]
[(partition_desc)]
[DISTRIBUTED BY HASH(<bucket_key>) BUCKETS <bucket_number>];
ALTER TABLE <table_name>
ADD TEMPORARY PARTITION <temporary_partition_name> VALUES LESS THAN
{MAXVALUE|(<"value">)}
[(partition_desc)]
[DISTRIBUTED BY HASH(<bucket_key>) BUCKETS <bucket_number>];
```

Create multiple partitions at a time

```
ALTER TABLE <table_name>
ADD TEMPORARY PARTITIONS START ("value1") END ("value2") EVERY
{(INTERVAL <num> <time_unit>)|<num>}
[(partition_desc)]
[DISTRIBUTED BY HASH(<bucket_key>) BUCKETS <bucket_number>];
```

Parameters

`partition_desc`: specifies the number of buckets and properties for temporary partitions, such as the number of replicas and the storage medium.

Examples

Create a temporary partition `tp1` in the table `site_access` and specify its range as `[2020-01-01, 2020-02-01)` by using the `VALUES [(...), (...)]` syntax.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITION tp1 VALUES [("2020-01-01"), ("2020-02-01"));
```

Create a temporary partition `tp2` in the table `site_access` and specify its upper bound as `2020-03-01` by using the `VALUES LESS THAN (...)` syntax. StarRocks uses the upper bound of the previous temporary partition as the lower bound of this temporary partition, generating a temporary partition with the left-closed and right-open range of `[2020-02-01, 2020-03-01)`.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITION tp2 VALUES LESS THAN ("2020-03-01");
```

Create a temporary partition `tp3` in the table `site_access`, specify its upper bound as `2020-04-01` by using the `VALUES LESS THAN (...)` syntax, and specify the number of replicas as `1` and the number of buckets as `5` .

```
ALTER TABLE site_access
ADD TEMPORARY PARTITION tp3 VALUES LESS THAN ("2020-04-01")
 ("replication_num" = "1")
DISTRIBUTED BY HASH (site_id) BUCKETS 5;
```

Create multiple partitions at a time in the table `site_access` by using the `START (...) END (...) EVERY (...)` syntax, and specify the range of these partitions as `[2020-04-01, 2021-01-01)` with a monthly partition granularity.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITIONS START ("2020-04-01") END ("2021-01-01")
EVERY (INTERVAL 1 MONTH);
```

Usage notes

- The partition column for a temporary partition must be the same as the partition column for the original formal partition based on which you create the temporary partition and cannot be changed.

- The name of a temporary partition cannot be the same as the name of any formal partition or other temporary partition.
- The ranges of all temporary partitions in a table cannot overlap, but the ranges of a temporary partition and a formal partition can overlap.

Show temporary partitions

You can view the temporary partitions by using the SHOW TEMPORARY PARTITIONS command.

```
SHOW TEMPORARY PARTITIONS FROM [db_name.]table_name [WHERE] [ORDER BY] [LIMIT]
```

Load data into temporary partitions

You can load data into one or more temporary partitions by using the `INSERT INTO` command, STREAM LOAD, or BROKER LOAD.

Load data by using `INSERT INTO` command

Example:

```
INSERT INTO site_access TEMPORARY PARTITION (tp1) VALUES ("2020-01-01",1,"ca","lily",4);
INSERT INTO site_access TEMPORARY PARTITION (tp2) SELECT * FROM site_access_copy PARTITION p2;
INSERT INTO site_access TEMPORARY PARTITION (tp3, tp4,...) SELECT * FROM site_access_copy PARTITION (p3, p4,...);
```

For detailed syntax and parameter descriptions, see INSERT INTO.

Load data by using STREAM LOAD

Example:

```
curl --location-trusted -u root: -H "label:123" -H "Expect:100-
continue" -H "temporary_partitions: tp1, tp2, ..." -T testData \
    http://host:port/api/example_db/site_access/_stream_load
```

For detailed syntax and parameter descriptions, see STREAM LOAD.

Load data by using BROKER LOAD

Example:

```
LOAD LABEL example_db.label1
(
    DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/starrocks/data/input/file")
    INTO TABLE my_table
    TEMPORARY PARTITION (tp1, tp2, ...)
    ...
)
WITH BROKER
(
    StorageCredentialParams
);
```

Note that `StorageCredentialParams` represents a group of authentication parameters which vary depending on the authentication method you choose. For detailed syntax and parameter descriptions, see Broker Load.

Load data by using ROUTINE LOAD

Example:

```
CREATE ROUTINE LOAD example_db.site_access ON example_tbl
COLUMNS(col, col2,...),
TEMPORARY PARTITIONS(tp1, tp2, ...)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest"
);
```

For detailed syntax and parameter descriptions, see CREATE ROUTINE LOAD.

Query data in temporary partitions

You can use the SELECT statement to query data in specified temporary partitions.

```
SELECT * FROM
site_access TEMPORARY PARTITION (tp1);

SELECT * FROM
site_access TEMPORARY PARTITION (tp1, tp2, ...);

SELECT event_day,site_id,pv FROM
site_access TEMPORARY PARTITION (tp1, tp2, ...);
```

You can use the JOIN clause to query data in temporary partitions from two tables.

```
SELECT * FROM
site_access TEMPORARY PARTITION (tp1, tp2, ...)
JOIN
site_access_copy TEMPORARY PARTITION (tp1, tp2, ...)
ON site_access.site_id=site_access1.site_id and
site_access.event_day=site_access1.event_day;
```

Replace original formal partitions with temporary partitions

You can use the ALTER TABLE statement to replace the original formal partition with a temporary partition, thereby creating a new formal partition.

> **NOTE**
>
> The original formal partition and temporary partition that you operated in the ALTER TABLE statement are deleted and cannot be recovered.

Syntax

```
ALTER TABLE table_name REPLACE PARTITION (partition_name) WITH
TEMPORARY PARTITION (temporary_partition_name1, ...)
PROPERTIES ("key" = "value");
```

Parameters

- **strict_range**

  Default value: `true`.

  When this parameter is set to `true`, the unions of ranges of all original formal partitions must be exactly the same as the unions of ranges of the temporary partitions that are used for replacement. When this parameter is set to `false`, you only need to make sure that the ranges of the new formal partitions do not overlap with other formal partitions after replacement.

  - Example 1:

    In the following example, the unions of the original formal partitions `p1`, `p2`, and `p3` are the same as the unions of the temporary partitions `tp1` and `tp2`, and you can use `tp1` and `tp2` to replace `p1`, `p2`, and `p3`.

    ```
    # Ranges of original formal partitions p1, p2, and p3 =>
    Unions of these ranges
    [10, 20), [20, 30), [40, 50) => [10, 30), [40, 50)

    # Ranges of temporary partitions tp1 and tp2 => Unions of
    these ranges
    [10, 30), [40, 45), [45, 50) => [10, 30), [40, 50)
    ```

  - Example 2:

    In the following example, the union of ranges of the original formal partition differs from the union of ranges of the temporary partitions. If the value of the parameter `strict_range` is set to `true`, the temporary partitions `tp1` and `tp2` cannot replace the original formal partition `p1`. If the value is set to `false`, and the ranges [10, 30) and [40, 50) of the temporary partitions do not overlap with other formal partitions, the temporary partitions can replace the original formal partition.

```
# Range of original formal partition p1 => Union of the range
[10, 50) => [10, 50)

# Ranges of temporary partitions tp1 and tp2 => Unions of
these ranges
[10, 30), [40, 50) => [10, 30), [40, 50)
```

- **use_temp_partition_name**

  Default value: `false`.

  If the number of original formal partitions is the same as the number of temporary partitions used for replacement, the names of the new formal partitions remain unchanged after replacement when this parameter is set to `false`. When this parameter is set to `true`, the names of the temporary partitions are used as the names of the new formal partitions after replacement.

  In the following example, the partition name of the new formal partition remains `p1` after replacement when this parameter is set to `false`. However, its related data and properties are replaced with the data and properties of the temporary partition `tp1`. When this parameter is set to `true`, the partition name of the new formal partition is changed to `tp1` after replacement. The original formal partition `p1` no longer exists.

  ```
  ALTER TABLE tbl1 REPLACE PARTITION (p1) WITH TEMPORARY PARTITION
  (tp1);
  ```

  If the number of formal partitions to be replaced is different from the number of temporary partitions used for replacement, and this parameter remains the default value `false`, the value `false` of this parameter is invalid.

  In the following example, after replacement, the name for the new formal partition is changed to `tp1`, and the original formal partitions `p1` and `p2` no longer exist.

  ```
  ALTER TABLE site_access REPLACE PARTITION (p1, p2) WITH TEMPORARY
  PARTITION (tp1);
  ```

Examples

Replace the original formal partition `p1` with the temporary partition `tp1`.

```
ALTER TABLE site_access REPLACE PARTITION (p1) WITH TEMPORARY
PARTITION (tp1);
```

Replace the original formal partitions `p2` and `p3` with the temporary partitions `tp2` and `tp3`.

```
ALTER TABLE site_access REPLACE PARTITION (p2, p3) WITH TEMPORARY
PARTITION (tp2, tp3);
```

Replace the original formal partitions `p4` and `p5` with the temporary partitions `tp4` and `tp5`, and specify the parameters `strict_range` as `false` and `use_temp_partition_name` as `true`.

```
ALTER TABLE site_access REPLACE PARTITION (p4, p5) WITH TEMPORARY
PARTITION (tp4, tp5)
PROPERTIES (
    "strict_range" = "false",
    "use_temp_partition_name" = "true"
);
```

Usage notes

- When a table has temporary partitions, you cannot use the `ALTER` command to perform Schema Change operations on the table.
- When performing Schema Change operations on a table, you cannot add temporary partitions to the table.

Delete temporary partitions

Use the following command to delete the temporary partition `tp1`.

```
ALTER TABLE site_access DROP TEMPORARY PARTITION tp1;
```

Take note of the following limits:

- If you use the `DROP` command to directly delete a database or table, you can recover the database or table within a limited time period by using the `RECOVER` command. However, temporary partitions cannot be recovered.
- After using the `ALTER` command to delete a formal partition, you can recover it within a limited time period by using the `RECOVER` command. Temporary partitions are not bound with formal partitions, so operations on temporary partitions do not affect formal partitions.
- After using the `ALTER` command to delete a temporary partition, you cannot recover it by using the `RECOVER` command.
- When using the `TRUNCATE` command to delete data in a table, the temporary partitions of the table are deleted and cannot be recovered.
- When using the `TRUNCATE` command to delete data in a formal partition, the temporary partitions are not affected.
- The TRUNCATE command cannot be used to delete data in a temporary partition.

Data compression

StarRocks supports data compression for table and index storage. Data compression not only helps save storage space, but also improves performance of I/O intensive tasks because StarRocks can read fewer pages from disk for each request. Note that extra CPU resource is required to compress and decompress the data.

Choose a data compression algorithm

StarRocks supports four data compression algorithms: LZ4, Zstandard (or zstd), zlib, and Snappy. These data compression algorithms differ in compression ratio and compression/decompression performance. Generally, the compression ratios of these algorithms rank as follows: zlib > Zstandard > LZ4 > Snappy. Among them, zlib has shown relatively high compression ratios. As a result of the highly compressed data, the loading and query performances on tables with the zlib compression algorithm are also affected. LZ4 and Zstandard, especially, have well-balanced compression ratios and decompression performances. You can choose among these compression algorithms to cater to your business needs for less storage space or better performance. We recommend LZ4 or Zstandard if you do not have specific demands for less storage space.

> **NOTE**
>
> Different data types can affect the compression ratio.

Specify a data compression algorithm for a table

You can specify a data compression algorithm for a table only when creating the table, and you cannot change it afterwards.

The following example creates a table `data_compression` with the Zstandard algorithm. For detailed instructions, see CREATE TABLE.

```
CREATE TABLE `data_compression` (
  `id`      INT(11)     NOT NULL     COMMENT "",
  `name`    CHAR(200)   NULL         COMMENT ""
)
ENGINE=OLAP
UNIQUE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`) BUCKETS 7
PROPERTIES (
"compression" = "ZSTD"
);
```

> **NOTE**
>
> If no data compression algorithm is specified, StarRocks uses LZ4 by default.

You can view the compression algorithm of a table using SHOW CREATE TABLE.

Sort keys and prefix indexes

When you create a table, you can select one or more of its columns to comprise a sort key. The sort key determines the order in which the data of the table is sorted before the data is stored on disk. You can use the sort key columns as filter conditions for queries. As such, StarRocks can quickly locate the data of interest, saving it from scanning the entire table to find the data that it needs to process. This reduces search complexity and therefore accelerates queries.

Additionally, to reduce memory consumption, StarRocks supports creating a prefix index on a table. Prefix indexes are a type of spare index. StarRocks stores every 1024 rows of the table in a block, for which an index entry is generated and stored in the prefix index table. The prefix index entry for a block cannot exceed 36 bytes in length, and its content is the prefix composed of the table's sort key columns in the first row of that block. This helps StarRocks quickly locate the starting column number of the block that stores the data of that row when a search on the prefix index table is run. The prefix index of a table is 1024 times less than the table itself in size. Therefore, the entire prefix index can be cached in memory to help accelerate queries.

Principles

In the Duplicate Key table, sort key columns are defined by using the `DUPLICATE KEY` keyword.

In the Aggregate table, sort key columns are defined by using the `AGGREGATE KEY` keyword.

In the Unique Key table, sort key columns are defined by using the `UNIQUE KEY` keyword.

In the Primary Key table, sort key columns are primary key columns, which are defined by using the `PRIMARY KEY` keyword.

When you define sort key columns for a table at table creation, take note of the following points:

- Sort key columns must be continuously defined columns, of which the first defined column must be the beginning sort key column.
- The columns that you plan to select as sort key columns must be defined prior to the other common columns.
- The sequence in which you list sort key columns must comply with the sequence in which you define the columns of the table.

The following examples show allowed sort key columns and unallowed sort key columns of a table that consists of four columns, which are `site_id`, `city_code`, `user_id`, and `pv`:

- Examples of allowed sort key columns
  - `site_id` and `city_code`
  - `site_id`, `city_code`, and `user_id`
- Examples of unallowed sort key columns
  - `city_code` and `site_id`
  - `city_code` and `user_id`
  - `site_id`, `city_code`, and `pv`

The following sections provide examples of how to define sort key columns when you create tables of different types. These examples are suitable for StarRocks clusters that have at least three BEs.

Duplicate Key

Create a table named `site_access_duplicate`. The table consists of four columns: `site_id`, `city_code`, `user_id`, and `pv`, of which `site_id` and `city_code` are selected as sort key columns.

The statement for creating the table is as follows:

```
CREATE TABLE site_access_duplicate
(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_id INT,
    pv BIGINT DEFAULT '0'
```

```
)
DUPLICATE KEY(site_id, city_code)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

Aggregate Key

Create a table named `site_access_aggregate`. The table consists of four columns: `site_id`, `city_code`, `user_id`, and `pv`, of which `site_id` and `city_code` are selected as sort key columns.

The statement for creating the table is as follows:

```
CREATE TABLE site_access_aggregate
(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_id BITMAP BITMAP_UNION,
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(site_id, city_code)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

> **NOTICE**
>
> For an Aggregate table, columns for which `agg_type` is not specified are key columns, and those for which `agg_type` is specified are value columns. See CREATE TABLE. In the preceding example, only `site_id` and `city_code` are specified as sort key columns, and therefore `agg_type` must be specified for `user_id` and `pv`.

Unique Key

Create a table named `site_access_unique`. The table consists of four columns: `site_id`, `city_code`, `user_id`, and `pv`, of which `site_id` and `city_code` are selected as sort key columns.

The statement for creating the table is as follows:

```
CREATE TABLE site_access_unique
(
    site_id INT DEFAULT '10',
```

```
    city_code SMALLINT,
    user_id INT,
    pv BIGINT DEFAULT '0'
)
UNIQUE KEY(site_id, city_code)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

Primary Key

Create a table named `site_access_primary`. The table consists of four columns: `site_id`, `city_code`, `user_id`, and `pv`, of which `site_id` and `city_code` are selected as sort key columns.

The statement for creating the table is as follows:

```
CREATE TABLE site_access_primary
(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_id INT,
    pv BIGINT DEFAULT '0'
)
PRIMARY KEY(site_id, city_code)
DISTRIBUTED BY HASH(site_id) BUCKETS 10;
```

Sorting effect

Use the preceding tables as examples. The sorting effect varies in the following three situations:

- If your query filters on both `site_id` and `city_code`, the number of rows that StarRocks needs to scan during the query is significantly reduced:

  ```
  select sum(pv) from site_access_duplicate where site_id = 123 and
  city_code = 2;
  ```

- If your query filters only on `site_id`, StarRocks can narrow the query range down to the rows that contain `site_id` values:

```
select sum(pv) from site_access_duplicate where site_id = 123;
```

- If your query filters only on `city_code`, StarRocks needs to scan the entire table:

```
select sum(pv) from site_access_duplicate where city_code = 2;
```

> **NOTE**
>
> In this situation, the sort key columns do not yield the expected sorting effect.

As described above, when your query filters on both `site_id` and `city_code`, StarRocks runs a binary search on the table to narrow the query range down to a specific location. If the table consists of a large number of rows, StarRocks runs binary searches on the `site_id` and `city_code` columns instead. This requires StarRocks to load the data of the two columns into memory and therefore increases memory consumption. In this case, you can use a prefix index to reduce the amount of data cached in memory, thereby accelerating your query.

Additionally, note that a large number of sort key columns also increase memory consumption. To reduce memory consumption, StarRocks imposes the following limits on the usage of prefix indexes:

- The prefix index entry of a block must be composed of the prefix of the table's sort key columns in the first row of that block.
- A prefix index can be created on a maximum of 3 columns.
- A prefix index entry cannot exceed 36 bytes in length.
- A prefix index cannot be created on columns of the FLOAT or DOUBLE data type.
- Of all the columns on which a prefix index is created, only one column of the VARCHAR data type is allowed, and that column must be the end column for the prefix index.
- If the end column for a prefix index is of the CHAR or VARCHAR data type, no entries in the prefix index can exceed 36 bytes.

How to select sort key columns

This section uses the `site_access_duplicate` table as an example to describe how to select sort key columns.

- We recommend that you identify the columns on which your queries frequently filter and select these columns as sort key columns.
- If you select more than one sort key column, we recommend that you list frequently filtered columns of high discrimination levels prior to the other columns.

  A column has a high discrimination level if the number of values in the column is large and continuously grows. For example, the number of cities in the `site_access_duplicate` table is fixed, which means that the number of values in the `city_code` column of the table is fixed. However, the number of values in the `site_id` column is much greater than the number of values in the `city_code` column and continuously grows. Therefore, the `site_id` column has a higher discrimination level than the `city_code` column.
- We recommend that you do not select a large number of sort key columns. A large number of sort key columns cannot help improve query performance but increase the overheads for sorting and data loading.

In summary, take note of the following points when you select sort key columns for the `site_access_duplicate` table:

- If your queries frequently filter on both `site_id` and `city_code`, we recommend that you select `site_id` as the beginning sort key column.
- If your queries frequently filter only on `city_code` and occasionally filter on both `site_id` and `city_code`, we recommend that you select `city_code` as the beginning sort key column.
- If the number of times that your queries filter on both `site_id` and `city_code` is roughly equal to the number of times that your queries filter only on `city_code`, we recommend that you create a materialized view, for which the first column is `city_code`. As such, StarRocks creates a sort index on the `city_code` column of the materialized view.

Overview of data loading

Data loading is the process of cleansing and transforming raw data from various data sources based on your business requirements and loading the resulting data into StarRocks to facilitate blazing-fast data analytics.

You can load data into StarRocks by running load jobs. Each load job has a unique label that is specified by the user or automatically generated by StarRocks to identify the job. Each label can be used only for one load job. After a load job is complete, its label cannot be reused for any other load jobs. Only the labels of failed load jobs can be reused. This mechanism helps ensure that the data associated with a specific label can be loaded only once, thus implementing At-Most-Once semantics.

All the loading methods provided by StarRocks can guarantee atomicity. Atomicity means that the qualified data within a load job must be all successfully loaded or none of the qualified data is successfully loaded. It never happens that some of the qualified data is loaded while the other data is not. Note that the qualified data does not include the data that is filtered out due to quality issues such as data type conversion errors.

StarRocks supports two communication protocols that can be used to submit load jobs: MySQL and HTTP. For more information about the protocol supported by each loading method, see the "Loading methods" section of this topic.

Supported data types

StarRocks supports loading data of all data types. You only need to take note of the limits on the loading of a few specific data types. For more information, see Data types.

Loading modes

StarRocks supports two loading modes: synchronous loading mode and asynchronous loading mode.

> **NOTE**

> If you load data by using external programs, you must choose a loading mode that best suits your business requirements before you decide the loading method of your choice.

Synchronous loading

In synchronous loading mode, after you submit a load job, StarRocks synchronously runs the job to load data, and returns the result of the job after the job finishes. You can check whether the job is successful based on the job result.

StarRocks provides two loading methods that support synchronous loading: Stream Load and INSERT.

The process of synchronous loading is as follows:

1. Create a load job.
2. View the job result returned by StarRocks.
3. Check whether the job is successful based on the job result. If the job result indicates a load failure, you can retry the job.

Asynchronous loading

In asynchronous loading mode, after you submit a load job, StarRocks immediately returns the job creation result.

- If the result indicates a job creation success, StarRocks asynchronously runs the job. However, that does not mean that the data has been successfully loaded. You must use statements or commands to check the status of the job. Then, you can determine whether the data is successfully loaded based on the job status.
- If the result indicates a job creation failure, you can determine whether you need to retry the job based on the failure information.

StarRocks provides three loading methods that support asynchronous loading: Broker Load, Routine Load, and Spark Load.

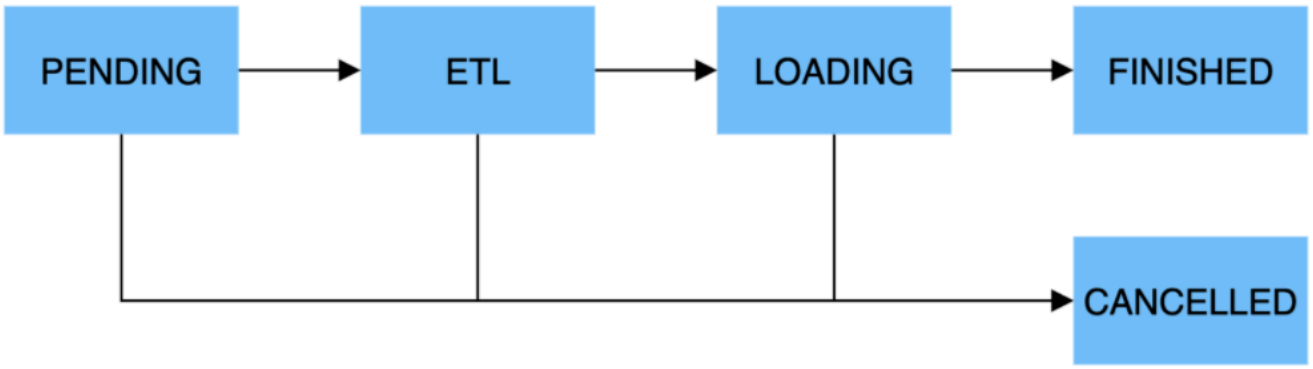
The process of asynchronous loading is as follows:

1. Create a load job.

2. View the job creation result returned by StarRocks and determine whether the job is successfully created. a. If the job creation succeeds, go to Step 3. b. If the job creation fails, return to Step 1.

3. Use statements or commands to check the status of the job until the job status shows **FINISHED** or **CANCELLED**.

The workflow of a Broker Load or Spark Load job consists of five stages, as shown in the following figure.



The workflow is described as follows:

1. **PENDING**

   The job is in queue waiting to be scheduled by an FE.

2. **ETL**

   The FE pre-processes the data, including cleansing, partitioning, sorting, and aggregation.

   > **NOTE**
   >
   > If the job is a Broker Load job, this stage is directly finished.

3. **LOADING**

   The FE cleanses and transforms the data, and then sends the data to the BEs. After all data is loaded, the data is in queue waiting to take effect. At this time, the status of the job remains **LOADING**.

4. **FINISHED**

When all data takes effect, the status of the job becomes **FINISHED**. At this time, the data can be queried. **FINISHED** is a final job state.

5. **CANCELLED**

   Before the status of the job becomes **FINISHED**, you can cancel the job at any time. Additionally, StarRocks can automatically cancel the job in case of load errors. After the job is canceled, the status of the job becomes **CANCELLED**. **CANCELLED** is also a final job state.

The workflow of a Routine job is described as follows:

1. The job is submitted to an FE from a MySQL client.
2. The FE splits the job into multiple tasks. Each task is engineered to load data from multiple partitions.
3. The FE distributes the tasks to specified BEs.
4. The BEs execute the tasks, and report to the FE after they finish the tasks.
5. The FE generates subsequent tasks, retries failed tasks if there are any, or suspends task scheduling based on the reports from the BEs.

Loading methods

StarRocks provides five loading methods to help you load data in various business scenarios: Stream Load, Broker Load, Routine Load, Spark Load, and INSERT.

Loading method	Data source	Business scenario	Data volume per load job	Data file format	Lo n
Stream Load	• Local files • Data streams	Load data files from local file systems or load data streams by using programs.	10 GB or less	• CSV • JSON	Sync

Loading method	Data source	Business scenario	Data volume per load job	Data file format	Lo n
Broker Load	• HDFS • Amazon S3 • Google GCS • Alibaba Cloud OSS • Tencent Cloud COS • Huawei Cloud OBS • Other S3-compatible storage system (such as MinIO)	Load data from HDFS or cloud storage.	Dozens of GB to hundreds of GB	• CSV • Parquet • ORC	Asyn
Routine Load	Apache Kafka®	Load data in real time from Kafka.	MBs to GBs of data as mini-batches	• CSV • JSON • Avro (supported since v3.0.1)	Asyn
Spark Load	• HDFS • Hive	• Migrate large amounts of data from HDFS or Hive by using Apache	Dozens of GB to TBs	• CSV • ORC (supported since v2.0)	Asyn

Loading method	Data source	Business scenario	Data volume per load job	Data file format	Lo n
		Spark™ clusters. • Load data while using a global data dictionary for deduplication.		• Parquet (supported since v2.0)	
INSERT INTO SELECT	• StarRocks tables • External tables	• Load data from external tables. • Load data between StarRocks tables.	Not fixed (The data volume varies based on the memory size.)	StarRocks tables	Sync
INSERT INTO VALUES	• Programs • ETL tools	• Insert small amounts of data as individual records. • Load data by using APIs such as JDBC.	In small quantities	SQL	Sync

You can determine the loading method of your choice based on your business scenario, data volume, data source, data file format, and loading frequency. Additionally, take note of the following points when you select a loading method:

- When you load data from Kafka, we recommend that you use Routine Load. However, if the data requires multi-table joins and extract, transform and load (ETL) operations, you can use Apache Flink® to read and pre-process the data from Kafka and then use flink-connector-starrocks to load the data into StarRocks.

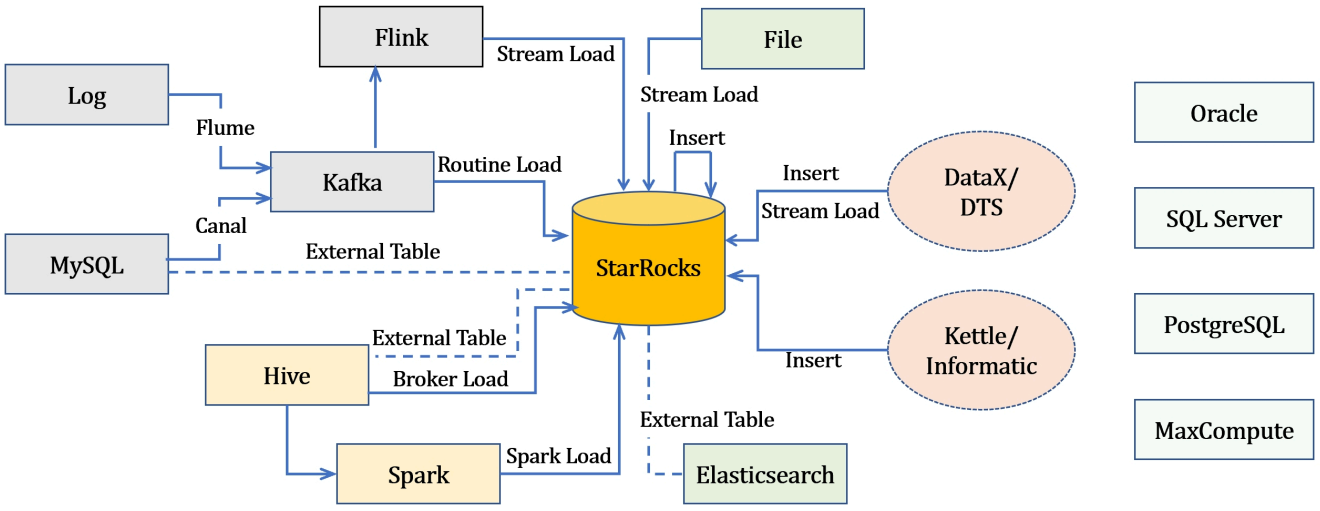
- When you load data from Hive, Iceberg, Hudi, or Delta Lake, we recommend that you create a Hive catalog, Iceberg catalog, Hudi Catalog, or Delta Lake Catalog and then use INSERT to load the data.

- When you load data from another StarRocks cluster or from an Elasticsearch cluster, we recommend that you create a StarRocks external table or an Elasticsearch external table and then use INSERT to load the data.

  > **NOTICE**
  >
  > StarRocks external tables only support data writes. They do not support data reads.

- When you load data from MySQL databases, we recommend that you create a MySQL external table and then use INSERT to load the data. If you want to load data in real time, we recommend that you load the data by following the instructions provided in Realtime synchronization from MySQL.

The following figure provides an overview of various data sources supported by StarRocks and the loading methods that you can use to load data from these data sources.



Memory limits

StarRocks provides parameters for you to limit the memory usage for each load job, thereby reducing memory consumption, especially in high concurrency scenarios. However, do not specify an excessively low memory usage limit. If the memory usage limit is excessively low, data may be frequently flushed from memory to disk because the memory usage for load jobs reaches the specified limit. We recommend that you specify a proper memory usage limit based on your business scenario.

The parameters that are used to limit memory usage vary for each loading method. For more information, see Stream Load, Broker Load, Routine Load, Spark Load, and INSERT. Note that a load job usually runs on multiple BEs. Therefore, the parameters limit the memory usage of each load job on each involved BE rather than the total memory usage of the load job on all involved BEs.

StarRocks also provides parameters for you to limit the total memory usage of all load jobs that run on each individual BE. For more information, see the "System configurations" section of this topic.

Usage notes

Automatically fill in the destination column while loading

When you load data, you can choose not to load the data from a specific field of your data file:

- If you have specified the `DEFAULT` keyword for the destination StarRocks table column mapping the source field when you create the StarRocks table, StarRocks automatically fills the specified default value into the destination column.

  Stream Load, Broker Load, Routine Load, and INSERT supports `DEFAULT current_timestamp`, `DEFAULT <default_value>`, and `DEFAULT (<expression>)`. Spark Load supports only `DEFAULT current_timestamp` and `DEFAULT <default_value>`.

  > **NOTE**

> `DEFAULT (<expression>)` supports only the functions `uuid()` and `uuid_numeric()`.

- If you did not specify the `DEFAULT` keyword for the destination StarRocks table column mapping the source field when you create the StarRocks table, StarRocks automatically fills `NULL` into the destination column.

  > **NOTE**
  >
  > If the destination column is defined as `NOT NULL`, the load fails.

  For Stream Load, Broker Load, Routine Load, and Spark Load, you can also specify the value you want to fill in the destination column by using the parameter that is used to specify column mapping.

For information about the usage of `NOT NULL` and `DEFAULT`, see CREATE TABLE.

Set write quorum for data loading

If your StarRocks cluster has multiple data replicas, you can set different write quorum for tables, that is, how many replicas are required to return loading success before StarRocks can determine the loading task is successful. You can specify write quorum by adding the property `write_quorum` when you CREATE TABLE, or add this property to an existing table using ALTER TABLE. This property is supported from v2.5.

System configurations

This section describes some parameter configurations that are applicable to all of the loading methods provided by StarRocks.

FE configurations

You can configure the following parameters in the configuration file **fe.conf** of each FE:

- `max_load_timeout_second` and `min_load_timeout_second`

  These parameters specify the maximum timeout period and minimum timeout period of each load job. The timeout periods are measured in seconds. The default maximum timeout period spans 3 days, and the default minimum timeout period spans 1 second.

The maximum timeout period and minimum timeout period that you specify must fall within the range of 1 second to 3 days. These parameters are valid for both synchronous load jobs and asynchronous load jobs.

- `desired_max_waiting_jobs`

  This parameter specifies the maximum number of load jobs that can be held waiting in queue. The default value is **1024** (100 in v2.4 and earlier, and 1024 in v2.5 and later). When the number of load jobs in the **PENDING** state on an FE reaches the maximum number that you specify, the FE rejects new load requests. This parameter is valid only for asynchronous load jobs.

- `max_running_txn_num_per_db`

  This parameter specifies the maximum number of ongoing load transactions that are allowed in each database of your StarRocks cluster. A load job can contain one or more transactions. The default value is **100**. When the number of load transactions running in a database reaches the maximum number that you specify, the subsequent load jobs that you submit are not scheduled. In this situation, if you submit a synchronous load job, the job is rejected. If you submit an asynchronous load job, the job is held waiting in queue.

  > **NOTE**
  >
  > StarRocks counts all load jobs together and does not distinguish between synchronous load jobs and asynchronous load jobs.

- `label_keep_max_second`

  This parameter specifies the retention period of the history records for load jobs that have finished and are in the **FINISHED** or **CANCELLED** state. The default retention period spans 3 days. This parameter is valid for both synchronous load jobs and asynchronous load jobs.

BE configurations

You can configure the following parameters in the configuration file **be.conf** of each BE:

- `write_buffer_size`

This parameter specifies the maximum memory block size. The default size is 100 MB. The loaded data is first written to a memory block on the BE. When the amount of data that is loaded reaches the maximum memory block size that you specify, the data is flushed to disk. You must specify a proper maximum memory block size based on your business scenario.

  - If the maximum memory block size is exceedingly small, a large number of small files may be generated on the BE. In this case, query performance degrades. You can increase the maximum memory block size to reduce the number of files generated.
  - If the maximum memory block size is exceedingly large, remote procedure calls (RPCs) may time out. In this case, you can adjust the value of this parameter based on your business needs.

- `streaming_load_rpc_max_alive_time_sec`

  The waiting timeout period for each Writer process. The default value is 600 seconds. During the data loading process, StarRocks starts a Writer process to receive data from and write data to each tablet. If a Writer process does not receive any data within the waiting timeout period that you specify, StarRocks stops the Writer process. When your StarRocks cluster processes data at low speeds, a Writer process may not receive the next batch of data within a long period of time and therefore reports a "TabletWriter add batch with unknown id" error. In this case, you can increase the value of this parameter.

- `load_process_max_memory_limit_bytes` and `load_process_max_memory_limit_percent`

  These parameters specify the maximum amount of memory that can be consumed for all load jobs on each individual BE. StarRocks identifies the smaller memory consumption among the values of the two parameters as the final memory consumption that is allowed.

  - `load_process_max_memory_limit_bytes`: specifies the maximum memory size. The default maximum memory size is 100 GB.

  - `load_process_max_memory_limit_percent`: specifies the maximum memory usage. The default value is 30%. This parameter differs from the `mem_limit` parameter. The `mem_limit` parameter specifies the total maximum memory usage of your StarRocks cluster, and the default value is 90% x 90%.

If the memory capacity of the machine on which the BE resides is M, the maximum amount of memory that can be consumed for load jobs is calculated as follows: `M x 90% x 90% x 30%`.

System variable configurations

You can configure the following system variable:

- `query_timeout`

  The query timeout duration. Unit: seconds. Value range: `1` to `259200`. Default value: `300`. This variable will act on all query statements in the current connection, as well as INSERT statements.

Troubleshooting

For more information, see FAQ about data loading.

Load data from a local file system

StarRocks provides two methods of loading data from a local file system:

- Synchronous loading using Stream Load
- Asynchronous loading using Broker Load

Each of these options has its own advantages:

- Stream Load supports CSV and JSON file formats. This method is recommended if you want to load data from a small number of files whose individual sizes do not exceed 10 GB.
- Broker Load supports Parquet, ORC, CSV, and JSON file formats (JSON file format is supported from v3.2.3 onwards). This method is recommended if you want to load data from a large number of files whose individual sizes exceed 10 GB, or if the files are stored in a network attached storage (NAS) device. **Using Broker Load to load data from a local file system is supported from v2.5 onwards. Note that before you can use Broker Load to load data from a local file system, you must deploy a broker on the machine on which your data files are located.**

For CSV data, take note of the following points:

- You can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
- Null values are denoted by using `\N`. For example, a data file consists of three columns, and a record from that data file holds data in the first and third columns but no data in the second column. In this situation, you need to use `\N` in the second column to denote a null value. This means the record must be compiled as `a,\N,b` instead of `a,,b`. `a,,b` denotes that the second column of the record holds an empty string.

Stream Load and Broker Load both support data transformation at data loading and support data changes made by UPSERT and DELETE operations during data loading. For more information, see Transform data at loading and Change data through loading.

Before you begin

Check privileges

You can load data into StarRocks tables only as a user who has the INSERT privilege on those StarRocks tables. If you do not have the INSERT privilege, follow the instructions provided in GRANT to grant the INSERT privilege to the user that you use to connect to your StarRocks cluster.

Check network configuration

Make sure that the machine on which the data you want to load resides can access the FE and BE nodes of the StarRocks cluster via the `http_port` (default: `8030`) and `be_http_port` (default: `8040`) , respectively.

Loading from a local file system via Stream Load

Stream Load is an HTTP PUT-based synchronous loading method. After you submit a load job, StarRocks synchronously runs the job, and returns the result of the job after the job finishes. You can determine whether the job is successful based on the job result.

> **NOTE**
>
> After you load data into a StarRocks table by using Stream Load, the data of the materialized views that are created on that table is also updated.

How it works

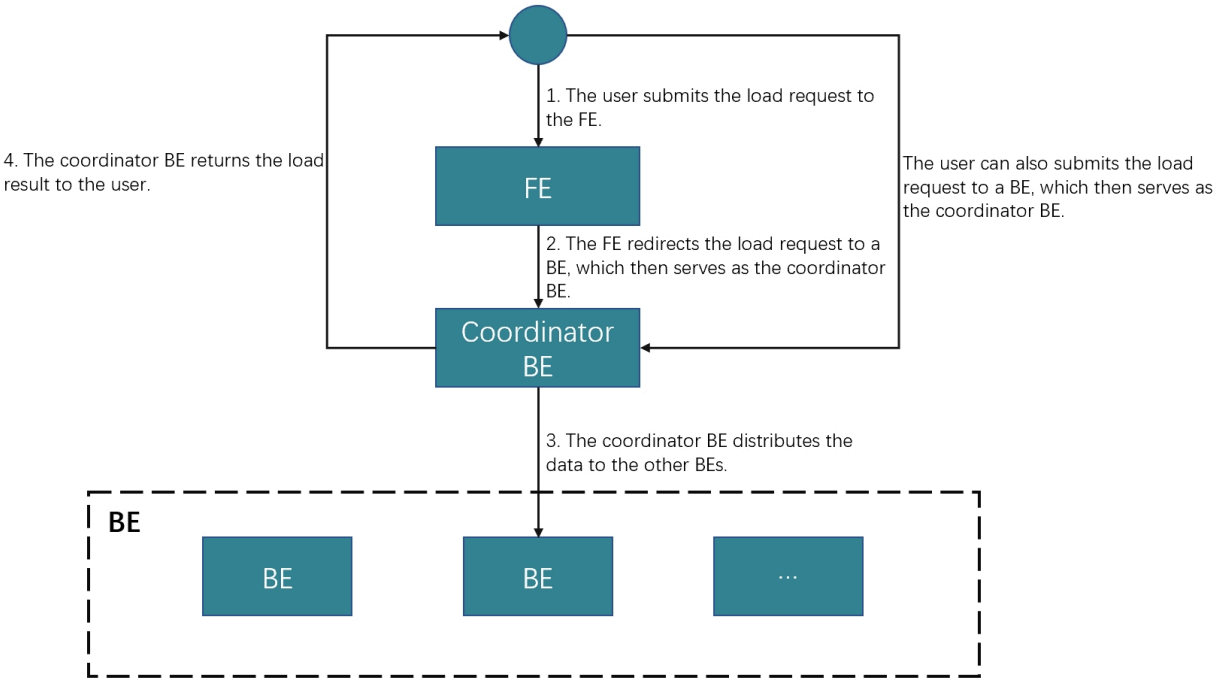
You can submit a load request on your client to an FE according to HTTP, and the FE then uses an HTTP redirect to forward the load request to a specific BE. You can also directly submit a load request on your client to a BE of your choice.

> **NOTE**
>
> If you submit load requests to an FE, the FE uses a polling mechanism to decide which BE will serve as a coordinator to receive and process the load requests. The polling

> mechanism helps achieve load balancing within your StarRocks cluster. Therefore, we recommend that you send load requests to an FE.

The BE that receives the load request runs as the Coordinator BE to split data based on the used schema into portions and assign each portion of the data to the other involved BEs. After the load finishes, the Coordinator BE returns the result of the load job to your client. Note that if you stop the Coordinator BE during the load, the load job fails.

The following figure shows the workflow of a Stream Load job.



Limits

Stream Load does not support loading the data of a CSV file that contains a JSON-formatted column.

Typical example

This section uses curl as an example to describe how to load the data of a CSV or JSON file from your local file system into StarRocks. For detailed syntax and parameter descriptions, see STREAM LOAD.

Note that in StarRocks some literals are used as reserved keywords by the SQL language. Do not directly use these keywords in SQL statements. If you want to use such a keyword in an SQL statement, enclose it in a pair of backticks (`). See Keywords.

Load CSV data

Prepare datasets

In your local file system, create a CSV file named `example1.csv`. The file consists of three columns, which represent the user ID, user name, and user score in sequence.

```
1,Lily,23
2,Rose,23
3,Alice,24
4,Julia,25
```

Create a database and a table

Create a database and switch to it:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Create a Primary Key table named `table1`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

```
CREATE TABLE `table1`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of

buckets. For detailed information, see Determine the number of buckets.

Start a Stream Load

Run the following command to load the data of `example1.csv` into `table1`:

```
curl --location-trusted -u <username>:<password> -H "label:123" \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: id, name, score" \
    -T example1.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/mydatabase/table1/_stream_load
```

> **NOTE**
>
> - If you use an account for which no password is set, you need to input only `<username>:`.
> - You can use SHOW FRONTENDS to view the IP address and HTTP port of the FE node.

`example1.csv` consists of three columns, which are separated by commas (,) and can be mapped in sequence onto the `id`, `name`, and `score` columns of `table1`. Therefore, you need to use the `column_separator` parameter to specify the comma (,) as the column separator. You also need to use the `columns` parameter to temporarily name the three columns of `example1.csv` as `id`, `name`, and `score`, which are mapped in sequence onto the three columns of `table1`.

After the load is complete, you can query `table1` to verify that the load is successful:

```
SELECT * FROM table1;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    1 | Lily  |    23 |
|    2 | Rose  |    23 |
|    3 | Alice |    24 |
|    4 | Julia |    25 |
```

```
+------+-------+-------+
4 rows in set (0.00 sec)
```

Load JSON data

Prepare datasets

In your local file system, create a JSON file named `example2.json`. The file consists of two columns, which represent city ID and city name in sequence.

```
{"name": "Beijing", "code": 2}
```

Create a database and a table

Create a database and switch to it:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Create a Primary Key table named `table2`. The table consists of two columns: `id` and `city`, of which `id` is the primary key.

```
CREATE TABLE `table2`
(
    `id` int(11) NOT NULL COMMENT "city ID",
    `city` varchar(65533) NULL COMMENT "city name"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **NOTE**
>
> Since v2.5.7, StarRocks can set the number of(BUCKETS) automatically when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see Determine the number of buckets.

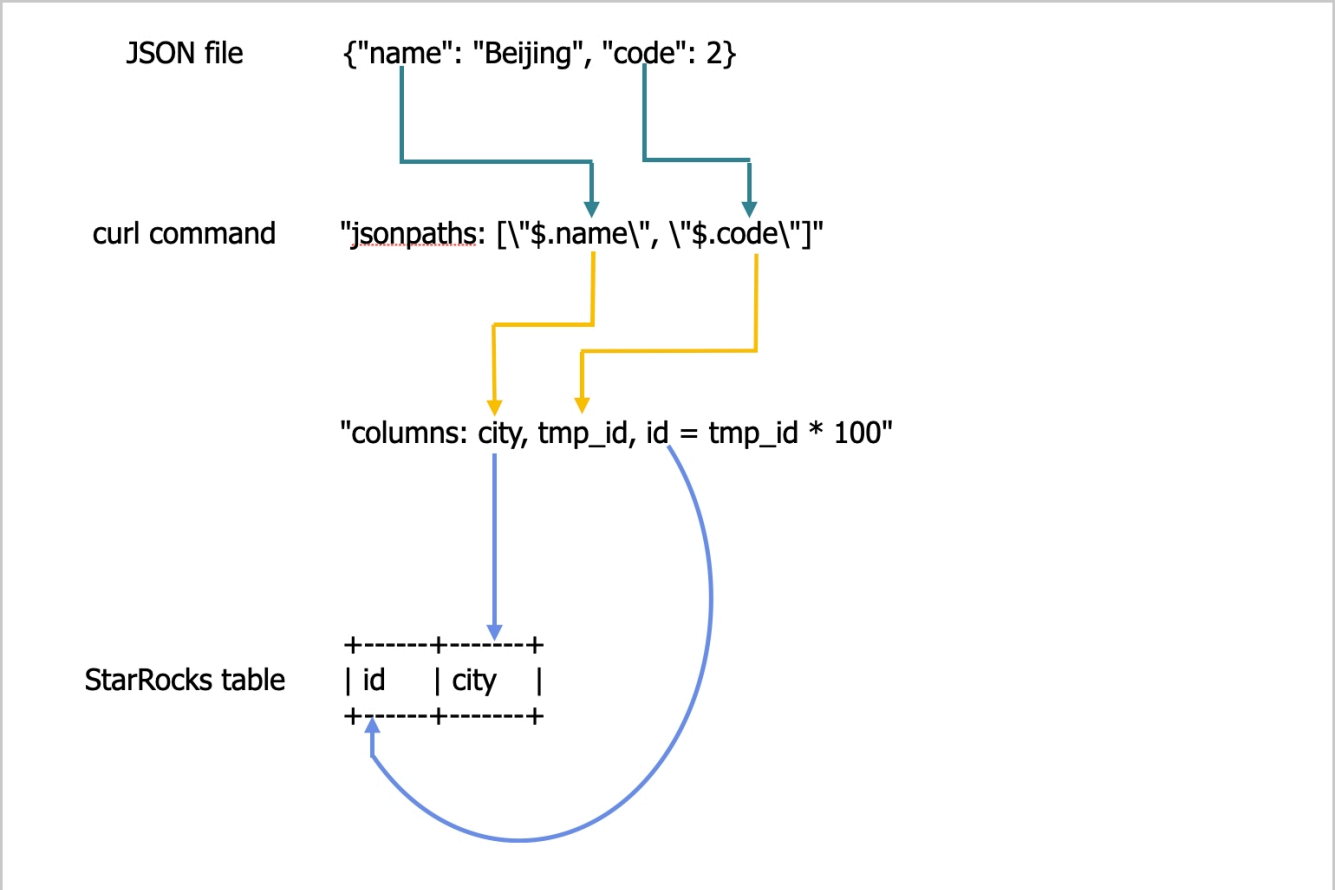
Start a Stream Load

Run the following command to load the data of `example2.json` into `table2`:

```
curl -v --location-trusted -u <username>:<password> -H "strict_mode:
true" \
    -H "Expect:100-continue" \
    -H "format: json" -H "jsonpaths: [\"$.name\", \"$.code\"]" \
    -H "columns: city,tmp_id, id = tmp_id * 100" \
    -T example2.json -XPUT \
    http://<fe_host>:
<fe_http_port>/api/mydatabase/table2/_stream_load
```

> **NOTE**
>
> - If you use an account for which no password is set, you need to input only `<username>:`.
> - You can use SHOW FRONTENDS to view the IP address and HTTP port of the FE node.

`example2.json` consists of two keys, `name` and `code`, which are mapped onto the `id` and `city` columns of `table2`, as shown in the following figure.



The mappings shown in the preceding figure are described as follows:

- StarRocks extracts the `name` and `code` keys of `example2.json` and maps them onto the `name` and `code` fields declared in the `jsonpaths` parameter.
- StarRocks extracts the `name` and `code` fields declared in the `jsonpaths` parameter and **maps them in sequence** onto the `city` and `tmp_id` fields declared in the `columns` parameter.
- StarRocks extracts the `city` and `tmp_id` fields declared in the `columns` parameter and **maps them by name** onto the `city` and `id` columns of `table2`.

> **NOTE**
>
> In the preceding example, the value of `code` in `example2.json` is multiplied by 100 before it is loaded into the `id` column of `table2`.

For detailed mappings between `jsonpaths`, `columns`, and the columns of the StarRocks table, see the "Column mappings" section in STREAM LOAD.

After the load is complete, you can query `table2` to verify that the load is successful:

```
SELECT * FROM table2;
+------+--------+
| id   | city   |
+------+--------+
| 200  | Beijing|
+------+--------+
4 rows in set (0.01 sec)
```

Check Stream Load progress

After a load job is complete, StarRocks returns the result of the job in JSON format. For more information, see the "Return value" section in STREAM LOAD.

Stream Load does not allow you to query the result of a load job by using the SHOW LOAD statement.

Cancel a Stream Load job

Stream Load does not allow you to cancel a load job. If a load job times out or encounters errors, StarRocks automatically cancels the job.

Parameter configurations

This section describes some system parameters that you need to configure if you choose the loading method Stream Load. These parameter configurations take effect on all Stream Load jobs.

- `streaming_load_max_mb`: the maximum size of each data file you want to load. The default maximum size is 10 GB. For more information, see Configure BE dynamic parameters.

  We recommend that you do not load more than 10 GB of data at a time. If the size of a data file exceeds 10 GB, we recommend that you split the data file into small files that each are less than 10 GB in size and then load these files one by one. If you cannot split a data file greater than 10 GB, you can increase the value of this parameter based on the file size.

  After you increase the value of this parameter, the new value can take effect only after you restart the BEs of your StarRocks cluster. Additionally, system performance may deteriorate, and the costs of retries in the event of load failures also increase.

  > **NOTE**
  >
  > When you load the data of a JSON file, take note of the following points:
  >
  > - The size of each JSON object in the file cannot exceed 4 GB. If any JSON object in the file exceeds 4 GB, StarRocks throws an error "This parser can't support a document that big."
  > - By default, the JSON body in an HTTP request cannot exceed 100 MB. If the JSON body exceeds 100 MB, StarRocks throws an error "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." To prevent this error, you can add `"ignore_json_size:true"` in the HTTP request header to ignore the check on the JSON body size.

- `stream_load_default_timeout_second`: the timeout period of each load job. The default timeout period is 600 seconds. For more information, see Configure FE dynamic parameters.

  If many of the load jobs that you create time out, you can increase the value of this parameter based on the calculation result that you obtain from the following formula:

  Timeout period of each load job > Amount of data to be loaded/Average loading speed

  For example, if the size of the data file that you want to load is 10 GB and the average loading speed of your StarRocks cluster is 100 MB/s, set the timeout period to more than 100 seconds.

  > **NOTE**
  >
  > **Average loading speed** in the preceding formula is the average loading speed of your StarRocks cluster. It varies depending on the disk I/O and the number of BEs in your StarRocks cluster.

  Stream Load also provides the `timeout` parameter, which allows you to specify the timeout period of an individual load job. For more information, see STREAM LOAD.

Usage notes

If a field is missing for a record in the data file you want to load and the column onto which the field is mapped in your StarRocks table is defined as `NOT NULL`, StarRocks automatically fills a `NULL` value in the mapping column of your StarRocks table during the load of the record. You can also use the `ifnull()` function to specify the default value that you want to fill.

For example, if the field that represents city ID in the preceding `example2.json` file is missing and you want to fill an `x` value in the mapping column of `table2`, you can specify `"columns: city, tmp_id, id = ifnull(tmp_id, 'x')"`.

Loading from a local file system via Broker Load

In addition to Stream Load, you can also use Broker Load to load data from a local file system. This feature is supported from v2.5 onwards.

Broker Load is an asynchronous loading method. After you submit a load job, StarRocks asynchronously runs the job and does not immediately return the job result. You need to query the job result by hand. See Check Broker Load progress.

Limits

- Currently Broker Load supports loading from a local file system only through a single broker whose version is v2.5 or later.
- Highly concurrent queries against a single broker may cause issues such as timeout and OOM. To mitigate the impact, you can use the `pipeline_dop` variable (see System variable) to set the query parallelism for Broker Load. For queries against a single broker, we recommend that you set `pipeline_dop` to a value smaller than `16`.

Before you begin

Before you can use Broker Load to load data from a local file system, finish the following preparations:

1. Configure the machine on which your local files are located as instructed in "Deployment prerequisites", "Check environment configurations", and "Prepare deployment files. Then, deploy a broker on that machine. The operations are the same as you deploy brokers on BE nodes. For detailed operations, see Deploy and manage Broker node.

   > **NOTICE**
   >
   > Deploy only a single broker and make sure that the broker version is v2.5 or later.

2. Execute ALTER SYSTEM to add the broker you have deployed in the previous step to your StarRocks cluster, and define a new name for the broker. The following example adds a broker `172.26.199.40:8000` to the StarRocks cluster and defines the broker name as `sole_broker`:

   ```
   ALTER SYSTEM ADD BROKER sole_broker "172.26.199.40:8000";
   ```

Typical example

Broker Load supports loading from a single data file to a single table, loading from multiple data files to a single table, and loading from multiple data files to multiple tables. This section uses loading from multiple data files to a single table as an example.

Note that in StarRocks some literals are used as reserved keywords by the SQL language. Do not directly use these keywords in SQL statements. If you want to use such a keyword in an SQL statement, enclose it in a pair of backticks (`). See Keywords.

Prepare datasets

Use the CSV file format as an example. Log in to your local file system, and create two CSV files, `file1.csv` and `file2.csv`, in a specific storage location (for example, `/home/disk1/business/`). Both files consist of three columns, which represent the user ID, user name, and user score in sequence.

- `file1.csv`

```
1,Lily,21
2,Rose,22
3,Alice,23
4,Julia,24
```

- `file2.csv`

```
5,Tony,25
6,Adam,26
7,Allen,27
8,Jacky,28
```

Create a database and a table

Create a database and switch to it:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Create a Primary Key table named `mytable`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

```
CREATE TABLE `mytable`
(
    `id` int(11) NOT NULL COMMENT "User ID",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "User name",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT "User score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`)
PROPERTIES("replication_num"="1");
```

Start a Broker Load

Run the following command to start a Broker Load job that loads data from all data files (`file1.csv` and `file2.csv`) stored in the `/home/disk1/business/` path of your local file system to the StarRocks table `mytable`:

```
LOAD LABEL mydatabase.label_local
(
    DATA INFILE("file:///home/disk1/business/csv/*")
    INTO TABLE mytable
    COLUMNS TERMINATED BY ","
    (id, name, score)
)
WITH BROKER "sole_broker"
PROPERTIES
(
    "timeout" = "3600"
);
```

This job has four main sections:

- `LABEL`: A string used when querying the state of the load job.
- `LOAD` declaration: The source URI, source data format, and destination table name.
- `BROKER`: The name of the broker.
- `PROPERTIES`: The timeout value and any other properties to apply to the load job.

For detailed syntax and parameter descriptions, see BROKER LOAD.

Check Broker Load progress

Use the SHOW LOAD statement or the curl command to view the progress of Broker Load jobs.

For usage of the SHOW LOAD statement, see SHOW LOAD.

The syntax for curl is as follows:

```
curl --location-trusted -u <username>:<password> \
    'http://<fe_host>:<fe_http_port>/api/<database_name>/_load_info?
label=<label_name>'
```

> **NOTE**
>
> If you use an account for which no password is set, you need to input only `<username>:`.

For example, you can run the following command to view the information about a load job, whose label is `label1`, in the `test_db` database:

```
curl --location-trusted -u <username>:<password> \
    'http://<fe_host>:<fe_http_port>/api/test_db/_load_info?
label=label1'
```

The `curl` command returns the information about the load job as a JSON object `jobInfo`:

```
{"jobInfo":{"dbName":"default_cluster:test_db","tblNames":
["table1_simple"],"label":"label1","state":"FINISHED","failMsg":"","tra
```

The following table describes the parameters in `jobInfo`.

Parameter	Description
dbName	The name of the database into which data is loaded
tblNames	The name of the table into which data is loaded.

Parameter	Description
label	The label of the load job.
state	The status of the load job. Valid values: • `PENDING`: The load job is in queue waiting to be scheduled. • `QUEUEING`: The load job is in the queue waiting to be scheduled. • `LOADING`: The load job is running. • `PREPARED`: The transaction has been committed. • `FINISHED`: The load job succeeded. • `CANCELLED`: The load job failed. For more information, see the "Asynchronous loading" section in Overview of data loading.

failMsg	The reason why the load job failed. If the `state` value for the load job is `PENDING`, `LOADING`, or `FINISHED`, `NULL` is returned for the `failMsg` parameter. If the `state` value for the load job is `CANCELLED`, the value returned for the `failMsg` parameter consists of two parts: `type` and `msg`.
	• The `type` part can be any of the following values:
	  ◦ `USER_CANCEL`: The load job was manually canceled.
	  ◦ `ETL_SUBMIT_FAIL`: The load job failed to be submitted.
	  ◦ `ETL-QUALITY-UNSATISFIED`: The load job failed because the percentage of unqualified data exceeds the value of the `max-filter-ratio` parameter.
	  ◦ `LOAD-RUN-FAIL`: The load job failed in the `LOADING` stage.
	  ◦ `TIMEOUT`: The load job failed to finish within the specified timeout period.
	  ◦ `UNKNOWN`: The load job failed due to an unknown error.
	• The `msg` part provides the detailed cause of the load failure.

Parameter	Description
trackingUrl	The URL that is used to access the unqualified data detected in the load job. You can use the `curl` or `wget` command to access the URL and obtain the unqualified data. If no unqualified data is detected, `NULL` is returned for the `trackingUrl` parameter.
status	The status of the HTTP request for the load job. Valid values: `OK` and `Fail`.
msg	The error information of the HTTP request for the load job.

After you confirm that the load job has finished, you can query table to see if the data has been successfully loaded. Example:

```
SELECT * FROM mytable;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    3 | Alice |    23 |
|    5 | Tony  |    25 |
|    6 | Adam  |    26 |
|    1 | Lily  |    21 |
|    2 | Rose  |    22 |
|    4 | Julia |    24 |
|    7 | Allen |    27 |
|    8 | Jacky |    28 |
+------+-------+-------+
8 rows in set (0.07 sec)
```

Cancel a Broker Load job

When a load job is not in the **CANCELLED** or **FINISHED** stage, you can use the CANCEL LOAD statement to cancel the job.

For example, you can execute the following statement to cancel a load job, whose label is `label_local`, in the database `mydatabase`:

```
CANCEL LOAD
FROM mydatabase
```

```
WHERE LABEL = "label_local";
```

Loading from NAS via Broker Load

There are two ways to load data from NAS by using Broker Load:

- Consider NAS as a local file system, and run a load job with a broker. See the previous section "Loading from a local system via Broker Load".
- (Recommended) Consider NAS as a cloud storage system, and run a load job without a broker.

This section introduces the second way. Detailed operations are as follows:

1. Mount your NAS device to the same path on all the BE nodes and FE nodes of your StarRocks cluster. As such, all BEs can access the NAS device like they access their own locally stored files.

2. Use Broker Load to load data from the NAS device to the destination StarRocks table. Example:

   ```
   LOAD LABEL test_db.label_nas
   (
       DATA INFILE("file:///home/disk1/sr/*")
       INTO TABLE mytable
       COLUMNS TERMINATED BY ","
   )
   WITH BROKER
   PROPERTIES
   (
       "timeout" = "3600"
   );
   ```

   This job has four main sections:

   - `LABEL`: A string used when querying the state of the load job.
   - `LOAD` declaration: The source URI, source data format, and destination table name. Note that `DATA INFILE` in the declaration is used to specify the mount point folder path of the NAS device, as shown in the above example in which `file:///` is the prefix and `/home/disk1/sr` is the mount point folder path.

- `BROKER`: You do not need to specify the broker name.
- `PROPERTIES`: The timeout value and any other properties to apply to the load job.

For detailed syntax and parameter descriptions, see BROKER LOAD.

After you submit a job, you can view the load progress or cancel the job as needed. For detailed operations, see "Check Broker Load progress" and "Cancel a Broker Load job in this topic.

Load data from HDFS or cloud storage

StarRocks provides the loading method MySQL-based Broker Load to help you load a large amount of data from HDFS or cloud storage into StarRocks.

Broker Load runs in asynchronous loading mode. After you submit a load job, StarRocks asynchronously runs the job. You need to use the SHOW LOAD statement or the `curl` command to check the result of the job.

Broker Load supports single-table loads and multi-table loads. You can load one or multiple data files into one or multiple destination tables by running one Broker Load job. Broker Load ensures the transactional atomicity of each load job that is run to load multiple data files. Atomicity means that the loading of multiple data files in one load job must all succeed or fail. It never happens that the loading of some data files succeeds while the loading of the other files fails.

Broker Load supports data transformation at data loading and supports data changes made by UPSERT and DELETE operations during data loading. For more information, see Transform data at loading and Change data through loading.

Background information

In v2.4 and earlier, StarRocks depends on brokers to set up connections between your StarRocks cluster and your external storage system when it runs a Broker Load job. Therefore, you need to input `WITH BROKER "<broker_name>"` to specify the broker you want to use in the load statement. This is called "broker-based loading." A broker is an independent, stateless service that is integrated with a file-system interface. With brokers, StarRocks can access and read data files that are stored in your external storage system, and can use its own computing resources to pre-process and load the data of these data files.

From v2.5 onwards, StarRocks no longer depends on brokers to set up connections between your StarRocks cluster and your external storage system when it runs a Broker

Load job. Therefore, you no longer need to specify a broker in the load statement, but you still need to retain the `WITH BROKER` keyword. This is called "broker-free loading."

When your data is stored in HDFS, you may encounter situations where broker-free loading does not work. This can happen when your data is stored across multiple HDFS clusters or when you have configured multiple Kerberos users. In these situations, you can resort to using broker-based loading instead. To do this successfully, make sure that at least one independent broker group is deployed. For information about how to specify authentication configuration and HA configuration in these situations, see HDFS.

> **NOTE**
>
> You can use the SHOW BROKER statement to check for brokers that are deployed in your StarRocks cluster. If no brokers are deployed, you can deploy brokers by following the instructions provided in Deploy a broker.

Supported data file formats

Broker Load supports the following data file formats:

- CSV
- Parquet
- ORC

> **NOTE**
>
> For CSV data, take note of the following points:
>
> - You can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
> - Null values are denoted by using `\N`. For example, a data file consists of three columns, and a record from that data file holds data in the first and third columns but no data in the second column. In this situation, you need to use `\N` in the second column to denote a null value. This means the record must be compiled as `a,\N,b` instead of `a,,b`. `a,,b` denotes that the second column of the record holds an empty string.

Supported storage systems

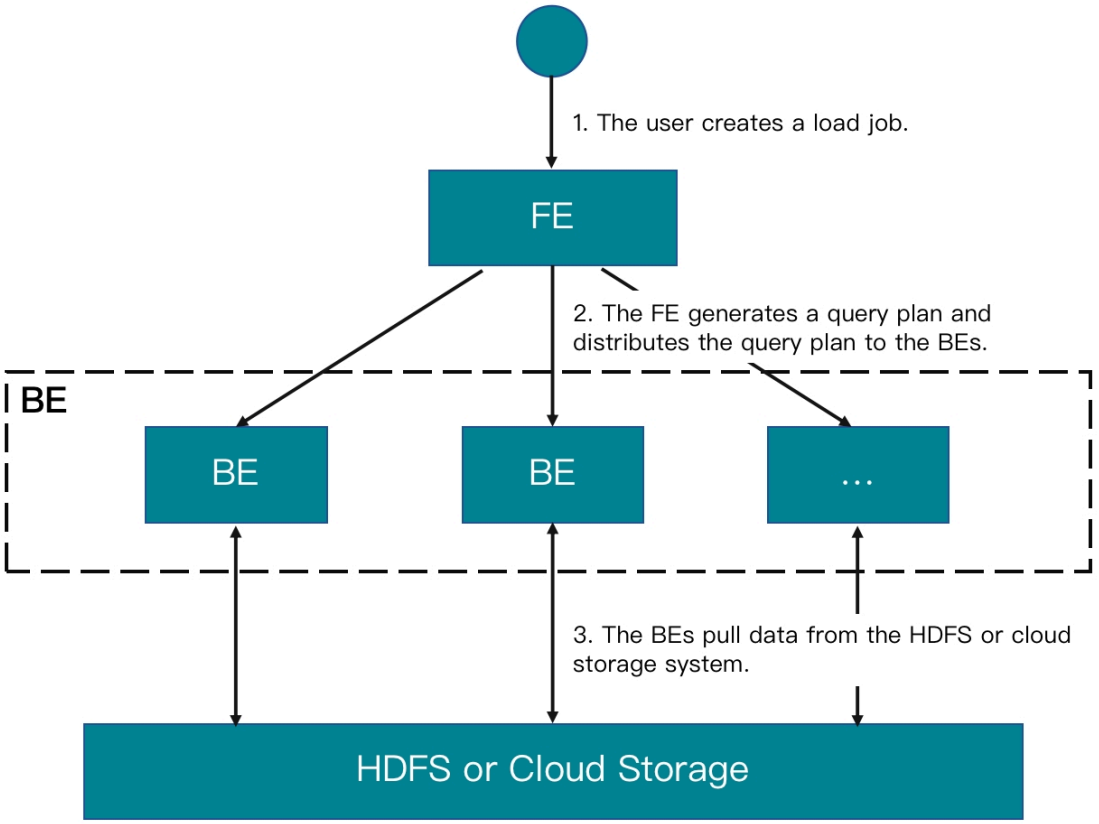
Broker Load supports the following storage systems:

- HDFS
- AWS S3
- Google GCS
- Other S3-compatible storage system such as MinIO

How it works

After you submit a load job to an FE, the FE generates a query plan, splits the query plan into portions based on the number of available BEs and the size of the data file you want to load, and then assigns each portion of the query plan to an available BE. During the load, each involved BE pulls the data of the data file from your HDFS or cloud storage system, pre-processes the data, and then loads the data into your StarRocks cluster. After all BEs finish their portions of the query plan, the FE determines whether the load job is successful.

The following figure shows the workflow of a Broker Load job.



Basic operations

Create a multi-table load job

This topic uses CSV as an example to describe how to load multiple data files into multiple tables. For information about how to load data in other file formats and about the syntax and parameter descriptions for Broker Load, see BROKER LOAD.

Note that in StarRocks some literals are used as reserved keywords by the SQL language. Do not directly use these keywords in SQL statements. If you want to use such a keyword in an SQL statement, enclose it in a pair of backticks (`). See Keywords.

Data examples

1. Create CSV files in your local file system.

   a. Create a CSV file named `file1.csv`. The file consists of three columns, which represent user ID, user name, and user score in sequence.

```
1,Lily,23
2,Rose,23
3,Alice,24
4,Julia,25
```

b. Create a CSV file named `file2.csv`. The file consists of two columns, which represent city ID and city name in sequence.

```
200,'Beijing'
```

2. Upload `file1.csv` and `file2.csv` to the `/user/starrocks/` path of your HDFS cluster, to the `input` folder of your AWS S3 bucket `bucket_s3`, to the `input` folder of your Google GCS bucket `bucket_gcs`, and to the `input` folder of your MinIO bucket `bucket_minio`.

3. Create StarRocks tables in your StarRocks database `test_db`.

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

a. Create a Primary Key table named `table1`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

```
CREATE TABLE `table1`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "user name",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

b. Create a Primary Key table named `table2`. The table consists of two columns: `id` and `city`, of which `id` is the primary key.

```
CREATE TABLE `table2`
(
    `id` int(11) NOT NULL COMMENT "city ID",
    `city` varchar(65533) NULL DEFAULT "" COMMENT "city name"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

Load data from HDFS

Execute the following statement to load `file1.csv` and `file2.csv` from the `/user/starrocks` path of your HDFS cluster into `table1` and `table2`, respectively:

```
LOAD LABEL test_db.label1
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/file1.csv")
    INTO TABLE table1
    COLUMNS TERMINATED BY ","
    (id, name, score)
    ,
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/file2.csv")
    INTO TABLE table2
    COLUMNS TERMINATED BY ","
    (id, city)
)
WITH BROKER
(
    StorageCredentialParams
)
PROPERTIES
(
    "timeout" = "3600"
);
```

In the preceding example, `StorageCredentialParams` represents a group of authentication parameters which vary depending on the authentication method you choose. For more information, see BROKER LOAD.

Load data from AWS S3

Execute the following statement to load `file1.csv` and `file2.csv` from the `input` folder of your AWS S3 bucket `bucket_s3` into `table1` and `table2`, respectively:

```
LOAD LABEL test_db.label2
(
    DATA INFILE("s3a://bucket_s3/input/file1.csv")
    INTO TABLE table1
    (id, name, score)
    ,
    DATA INFILE("s3a://bucket_s3/input/file2.csv")
    INTO TABLE table2
    (id, city)
)
WITH BROKER
(
    StorageCredentialParams
);
```

> **NOTE**
>
> Broker Load supports accessing AWS S3 only according to the S3A protocol. Therefore, when you load data from AWS S3, you must replace `s3://` in the S3 URI you pass as a file path into `DATA INFILE` with `s3a://`.

In the preceding example, `StorageCredentialParams` represents a group of authentication parameters which vary depending on the authentication method you choose. For more information, see BROKER LOAD.

Load data from Google GCS

Execute the following statement to load `file1.csv` and `file2.csv` from the `/input/` folder of your Google GCS bucket `bucket_gcs` into `table1` and `table2`, respectively:

```
LOAD LABEL test_db.label3
(
    DATA INFILE("s3a://bucket_gcs/input/file1.csv")
    INTO TABLE table1
    (id, name, score)
    ,
    DATA INFILE("s3a://bucket_gcs/input/file2.csv")
    INTO TABLE table2
    (id, city)
```

```
)
WITH BROKER
(
    StorageCredentialParams
);
```

> **NOTE**
>
> Broker Load supports accessing Google GCS only according to the S3A protocol. Therefore, when you load data from Google GCS, you must replace the prefix in the GCS URI you pass as a file path into `DATA INFILE` with `s3a://`.

In the preceding example, `StorageCredentialParams` represents a group of authentication parameters which vary depending on the authentication method you choose. For more information, see BROKER LOAD.

Load data from other S3-compatible storage system

Use MinIO as an example. You can execute the following statement to load `file1.csv` and `file2.csv` from the `input` folder of your MinIO bucket `bucket_gcs` into `table1` and `table2`, respectively:

```
LOAD LABEL test_db.label7
(
    DATA INFILE("s3://bucket_minio/input/file1.csv")
    INTO TABLE table1
    COLUMNS TERMINATED BY ","
    (id, name, score)
    ,
    DATA INFILE("s3://bucket_minio/input/file2.csv")
    INTO TABLE table2
    COLUMNS TERMINATED BY ","
    (id, city)
)
WITH BROKER
(
    StorageCredentialParams
);
```

In the preceding example, `StorageCredentialParams` represents a group of authentication parameters which vary depending on the authentication method you choose. For more information, see BROKER LOAD.

Query data

After the load of data from your HDFS cluster, AWS S3 bucket, or Google GCS bucket is complete, you can use the SELECT statement to query the data of the StarRocks tables to verify that the load is successful.

1. Execute the following statement to query the data of `table1`:

```
MySQL [test_db]> SELECT * FROM table1;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    1 | Lily  |    23 |
|    2 | Rose  |    23 |
|    3 | Alice |    24 |
|    4 | Julia |    25 |
+------+-------+-------+
4 rows in set (0.00 sec)
```

2. Execute the following statement to query the data of `table2`:

```
MySQL [test_db]> SELECT * FROM table2;
+------+--------+
| id   | city   |
+------+--------+
| 200  | Beijing|
+------+--------+
4 rows in set (0.01 sec)
```

Create a single-table load job

You can also load a single data file or all data files from a specified path into a single destination table. Suppose your AWS S3 bucket `bucket_s3` contains a folder named `input`. The `input` folder contains multiple data files, one of which is named `file1.csv`. These data files consist of the same number of columns as `table1` and the columns from each of these data files can be mapped one on one in sequence to the columns from `table1`.

To load `file1.csv` into `table1`, execute the following statement:

```
LOAD LABEL test_db.label_7
(
    DATA INFILE("s3a://bucket_s3/input/file1.csv")
    INTO TABLE table1
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
)
WITH BROKER
(
    StorageCredentialParams
);
```

To load all data files from the `input` folder into `table1`, execute the following statement:

```
LOAD LABEL test_db.label_8
(
    DATA INFILE("s3a://bucket_s3/input/*")
    INTO TABLE table1
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
)
WITH BROKER
(
    StorageCredentialParams
);
```

In the preceding examples, `StorageCredentialParams` represents a group of authentication parameters which vary depending on the authentication method you choose. For more information, see BROKER LOAD.

View a load job

Broker Load allows you to view a lob job by using the SHOW LOAD statement or the `curl` command.

Use SHOW LOAD

For more information, see SHOW LOAD.

Use curl

The syntax is as follows:

```
curl --location-trusted -u <username>:<password> \
    'http://<fe_host>:<fe_http_port>/api/<database_name>/_load_info?
label=<label_name>'
```

> **NOTE**
>
> If you use an account for which no password is set, you need to input only `<username>:`.

For example, you can run the following command to view the information about a load job, whose label is `label1`, in the `test_db` database:

```
curl --location-trusted -u <username>:<password> \
    'http://<fe_host>:<fe_http_port>/api/test_db/_load_info?
label=label1'
```

The `curl` command returns the information about the most recently executed load job with the specified label as a JSON object `jobInfo`:

```
{"jobInfo":{"dbName":"default_cluster:test_db","tblNames":
["table1_simple"],"label":"label1","state":"FINISHED","failMsg":"","tra
```

The following table describes the parameters in `jobInfo`.

Parameter	Description
dbName	The name of the database into which data is loaded
tblNames	The name of the table into which data is loaded.
label	The label of the load job.
state	The status of the load job. Valid values: • `PENDING`: The load job is in queue waiting to be scheduled. • `QUEUEING`: The load job is in the queue waiting to be scheduled. • `LOADING`: The load job is running.

Parameter	Description
	• `PREPARED`: The transaction has been committed. • `FINISHED`: The load job succeeded. • `CANCELLED`: The load job failed. For more information, see the "Asynchronous loading" section in Overview of data loading.

failMsg	The reason why the load job failed. If the `state` value for the load job is `PENDING`, `LOADING`, or `FINISHED`, `NULL` is returned for the `failMsg` parameter. If the `state` value for the load job is `CANCELLED`, the value returned for the `failMsg` parameter consists of two parts: `type` and `msg`.
	• The `type` part can be any of the following values:
	  ◦ `USER_CANCEL`: The load job was manually canceled.
	  ◦ `ETL_SUBMIT_FAIL`: The load job failed to be submitted.
	  ◦ `ETL-QUALITY-UNSATISFIED`: The load job failed because the percentage of unqualified data exceeds the value of the `max-filter-ratio` parameter.
	  ◦ `LOAD-RUN-FAIL`: The load job failed in the `LOADING` stage.
	  ◦ `TIMEOUT`: The load job failed to finish within the specified timeout period.
	  ◦ `UNKNOWN`: The load job failed due to an unknown error.
	• The `msg` part provides the detailed cause of the load failure.

trackingUrl	The URL that is used to access the unqualified data detected in the load job. You can use the `curl` or `wget` command to access the URL and obtain the unqualified data. If no unqualified data is detected, `NULL` is returned for the `trackingUrl` parameter.
status	The status of the HTTP request for the load job. Valid values: `OK` and `Fail`.
msg	The error information of the HTTP request for the load job.

Cancel a load job

When a load job is not in the **CANCELLED** or **FINISHED** stage, you can use the CANCEL LOAD statement to cancel the job.

For example, you can execute the following statement to cancel a load job, whose label is `label1`, in the database `test_db`:

```
CANCEL LOAD
FROM test_db
WHERE LABEL = "label";
```

Job splitting and concurrent running

A Broker Load job can be split into one or more tasks that concurrently run. The tasks within a load job are run within a single transaction. They must all succeed or fail. StarRocks splits each load job based on how you declare `data_desc` in the `LOAD` statement:

- If you declare multiple `data_desc` parameters, each of which specifies a distinct table, a task is generated to load the data of each table.
- If you declare multiple `data_desc` parameters, each of which specifies a distinct partition for the same table, a task is generated to load the data of each partition.

Additionally, each task can be further split into one or more instances, which are evenly distributed to and concurrently run on the BEs of your StarRocks cluster. StarRocks splits each task based on the following FE configurations:

- `min_bytes_per_broker_scanner`: the minimum amount of data processed by each instance. The default amount is 64 MB.
- `max_broker_concurrency`: the maximum number of concurrent instances allowed in each task. The default maximum number is 100.
- `load_parallel_instance_num`: the number of concurrent instances allowed in each load job on an individual BE. The default number is 1.

  You can use the following formula to calculate the number of instances in an individual task:

Number of instances in an individual task = min(Amount of data to be loaded by an individual task/`min_bytes_per_broker_scanner`,`max_broker_concurrency`,`load_parallel_instance_num` x Number of BEs)

In most cases, only one `data_desc` is declared for each load job, each load job is split into only one task, and the task is split into the same number of instances as the number of BEs.

Usage notes

The FE configuration item `max_broker_load_job_concurrency` specifies the maximum number of Broker Load jobs that can be concurrently run within your StarRocks cluster.

In StarRocks v2.4 and earlier, if the total number of Broker Load jobs that are submitted within a specific period of time exceeds the maximum number, excessive jobs are queued and scheduled based on their submission time.

Since StarRocks v2.5, if the total number of Broker Load jobs that are submitted within a specific period of time exceeds the maximum number, excessive jobs are queued and scheduled based on their priorities. You can specify a priority for a job by using the `priority` parameter at job creation. See BROKER LOAD. You can also use ALTER LOAD to modify the priority of an existing job that is in the **QUEUEING** or **LOADING** state.

Load data using Kafka connector

StarRocks provides a self-developed connector named Apache Kafka® connector (StarRocks Conn...

Load data using Routine Load

This topic introduces how to create a Routine Load job to stream Kafka messages (events) into Sta...

Load data using Kafka connector

StarRocks provides a self-developed connector named Apache Kafka® connector (StarRocks Connector for Apache Kafka®, Kafka connector for short), as a sink connector, that continuously consumes messages from Kafka and loads them into StarRocks. The Kafka connector guarantees at-least-once semantics.

The Kafka connector can seamlessly integrate with Kafka Connect, which allows StarRocks better integrated with the Kafka ecosystem. It is a wise choice if you want to load real-time data into StarRocks. Compared with Routine Load, it is recommended to use the Kafka connector in the following scenarios:

- Compared with Routine Load which only supports loading data in CSV, JSON, and Avro formats, Kafka connector can load data in more formats, such as Protobuf. As long as data can be converted into JSON and CSV formats using Kafka Connect's converters, data can be loaded into StarRocks via the Kafka connector.
- Customize data transformation, such as Debezium-formatted CDC data.
- Load data from multiple Kafka topics.
- Load data from Confluent Cloud.
- Need finer control over load batch sizes, parallelism, and other parameters to achieve a balance between load speed and resource utilization.

Preparations

Set up Kafka environment

Both self-managed Apache Kafka clusters and Confluent Cloud are supported.

- For a self-managed Apache Kafka cluster, you can refer to Apache Kafka quickstart to quickly deploy a Kafka cluster. Kafka Connect is already integrated into Kafka.
- For Confluent Cloud, make sure that you have a Confluent account and have created a cluster.

Download Kafka connector

Submit the Kafka connector into Kafka Connect:

- Self-managed Kafka cluster:

  Download and extract starrocks-kafka-connector-xxx.tar.gz.

- Confluent Cloud:

  Currently, the Kafka connector is not uploaded to Confluent Hub. You need to download and extract starrocks-kafka-connector-xxx.tar.gz, package it into a ZIP file and upload the ZIP file to Confluent Cloud.

Usage

This section uses a self-managed Kafka cluster as an example to explain how to configure the Kafka connector and the Kafka Connect, and then run the Kafka Connect to load data into StarRocks.

Prepare a dataset

Suppose that JSON-format data exists in the topic `test` in a Kafka cluster.

```
{"id":1,"city":"New York"}
{"id":2,"city":"Los Angeles"}
{"id":3,"city":"Chicago"}
```

Create a table

Create the table `test_tbl` in the database `example_db` in the StarRocks cluster according to the keys of the JSON-format data.

```
CREATE DATABASE example_db;
USE example_db;
CREATE TABLE test_tbl (id INT, city STRING);
```

Configure Kafka connector and Kafka Connect, and then run Kafka Connect to load data

Run Kafka Connect in standalone mode

1. Configure the Kafka connector. In the **config** directory under the Kafka installation directory, create the configuration file **connect-StarRocks-sink.properties** for the Kafka connector, and configure the following parameters. For more parameters and dsescriptions, see Parameters.

   > **INFO**
   >
   > - In this example, the Kafka connector provided by StarRocks is a sink connector that can continuously consume data from Kafka and load data into StarRocks.
   > - If the source data is CDC data, such as data in Debezium format, and the StarRocks table is a Primary Key table, you also need to configure `transform` in the configuration file **connect-StarRocks-sink.properties** for the Kafka connector provided by StarRocks, to synchronize the source data changes to the Primary Key table.

   ```
   name=starrocks-kafka-connector
   connector.class=com.starrocks.connector.kafka.StarRocksSinkConnector
   topics=test
   key.converter=org.apache.kafka.connect.json.JsonConverter
   value.converter=org.apache.kafka.connect.json.JsonConverter
   key.converter.schemas.enable=true
   value.converter.schemas.enable=false
   # The HTTP URL of the FE in your StarRocks cluster. The default port
   is 8030.
   starrocks.http.url=192.168.xxx.xxx:8030
   # If the Kafka topic name is different from the StarRocks table
   name, you need to configure the mapping relationship between them.
   starrocks.topic2table.map=test:test_tbl
   # Enter the StarRocks username.
   starrocks.username=user1
   # Enter the StarRocks password.
   starrocks.password=123456
   starrocks.database.name=example_db
   sink.properties.strip_outer_array=true
   ```

2. Configure and run the Kafka Connect.

    i. Configure the Kafka Connect. In the configuration file **config/connect-standalone.properties** in the **config** directory, configure the following parameters. For more parameters and descriptions, see Running Kafka Connect.

```
# The addresses of Kafka brokers. Multiple addresses of Kafka brokers need to be separated by commas (,).
# Note that this example uses PLAINTEXT as the security protocol to access the Kafka cluster. If you are using other security protocol to access the Kafka cluster, you need to configure the relevant information in this file.
bootstrap.servers=<kafka_broker_ip>:9092
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=false
# The absolute path of the Kafka connector after extraction. For example:
plugin.path=/home/kafka-connect/starrocks-kafka-connector-1.0.3
```

    ii. Run the Kafka Connect.

```
CLASSPATH=/home/kafka-connect/starrocks-kafka-connector-1.0.3/* bin/connect-standalone.sh config/connect-standalone.properties config/connect-starrocks-sink.properties
```

Run Kafka Connect in distributed mode

1. Configure and run the Kafka Connect.

    i. Configure the Kafka Connect. In the configuration file `config/connect-distributed.properties` in the **config** directory, configure the following parameters. For more parameters and descriptions, refer to Running Kafka Connect.

```
# The addresses of Kafka brokers. Multiple addresses of Kafka brokers need to be separated by commas (,).
```

```
# Note that this example uses PLAINTEXT as the security
protocol to access the Kafka cluster. If you are using other
security protocol to access the Kafka cluster, you need to
configure the relevant information in this file.
bootstrap.servers=<kafka_broker_ip>:9092
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=false
# The absolute path of the Kafka connector after extraction.
For example:
plugin.path=/home/kafka-connect/starrocks-kafka-connector-
1.0.3
```

ii. Run the Kafka Connect.

```
CLASSPATH=/home/kafka-connect/starrocks-kafka-connector-
1.0.3/* bin/connect-distributed.sh config/connect-
distributed.properties
```

2. Configure and create the Kafka connector. Note that in distributed mode, you need to configure and create the Kafka connector through the REST API. For parameters and descriptions, see Parameters.

> **INFO**
>
> - In this example, the Kafka connector provided by StarRocks is a sink connector that can continuously consume data from Kafka and load data into StarRocks.
> - If the source data is CDC data, such as data in Debezium format, and the StarRocks table is a Primary Key table, you also need to configure `transform` in the configuration file **connect-StarRocks-sink.properties** for the Kafka connector provided by StarRocks, to synchronize the source data changes to the Primary Key table.

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type:
application/json" -X POST -d '{
  "name":"starrocks-kafka-connector",
  "config":{
```

```
"connector.class":"com.starrocks.connector.kafka.StarRocksSinkConnec
    "topics":"test",
    "key.converter":"org.apache.kafka.connect.json.JsonConverter",
    "value.converter":"org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable":"true",
    "value.converter.schemas.enable":"false",
    "starrocks.http.url":"192.168.xxx.xxx:8030",
    "starrocks.topic2table.map":"test:test_tbl",
    "starrocks.username":"user1",
    "starrocks.password":"123456",
    "starrocks.database.name":"example_db",
    "sink.properties.strip_outer_array":"true"
  }
}'
```

Query StarRocks table

Query the target StarRocks table `test_tbl`.

```
MySQL [example_db]> select * from test_tbl;

+------+-------------+
| id   | city        |
+------+-------------+
|    1 | New York    |
|    2 | Los Angeles |
|    3 | Chicago     |
+------+-------------+
3 rows in set (0.01 sec)
```

The data is successfully loaded when the above result is returned.

Parameters

name

Required: YES
Default value:
Description: Name for this Kafka connector. It must be globally unique among all Kafka connectors within this Kafka Connect cluster. For example, starrocks-kafka-connector.

connector.class

Required: YES
Default value:
Description: Class used by this Kafka connector's sink. Set the value to `com.starrocks.connector.kafka.StarRocksSinkConnector`.

topics

Required:
Default value:
Description: One or more topics to subscribe to, where each topic corresponds to a StarRocks table. By default, StarRocks assumes that the topic name matches the name of the StarRocks table. So StarRocks determines the target StarRocks table by using the topic name. Please choose either to fill in `topics` or `topics.regex` (below), but not both. However, if the StarRocks table name is not the same as the topic name, then use the optional `starrocks.topic2table.map` parameter (below) to specify the mapping from topic name to table name.

topics.regex

Required:
Default value: **Description**: Regular expression to match the one or more topics to subscribe to. For more description, see `topics`. Please choose either to fill in `topics.regex` or `topics` (above), but not both.

starrocks.topic2table.map

Required: NO
Default value:
Description: The mapping of the StarRocks table name and the topic name when the topic name is different from the StarRocks table name. The format is `<topic-1>:<table-1>,<topic-2>:<table-2>,...`.

starrocks.http.url

Required: YES
Default value:

Description: The HTTP URL of the FE in your StarRocks cluster. The format is `<fe_host1>:<fe_http_port1>,<fe_host2>:<fe_http_port2>,...`. Multiple addresses are separated by commas (,). For example, `192.168.xxx.xxx:8030,192.168.xxx.xxx:8030`.

starrocks.database.name

Required: YES
Default value:
Description: The name of StarRocks database.

starrocks.username

Required: YES
Default value:
Description: The username of your StarRocks cluster account. The user needs the INSERT privilege on the StarRocks table.

starrocks.password

Required: YES
Default value:
Description: The password of your StarRocks cluster account.

key.converter

Required: NO
Default value: Key converter used by Kafka Connect cluster
Description: This parameter specifies the key converter for the sink connector (Kafka-connector-starrocks), which is used to deserialize the keys of Kafka data. The default key converter is the one used by Kafka Connect cluster.

value.converter

Required: NO
Default value: Value converter used by Kafka Connect cluster
Description: This parameter specifies the value converter for the sink connector (Kafka-

connector-starrocks), which is used to deserialize the values of Kafka data. The default value converter is the one used by Kafka Connect cluster.

key.converter.schema.registry.url

Required: NO
Default value:
Description: Schema registry URL for the key converter.

value.converter.schema.registry.url

Required: NO
Default value:
Description: Schema registry URL for the value converter.

tasks.max

Required: NO
Default value: 1
Description: The upper limit for the number of task threads that the Kafka connector can create, which is usually the same as the number of CPU cores on the worker nodes in the Kafka Connect cluster. You can tune this parameter to control load performance.

bufferflush.maxbytes

Required: NO
Default value: 94371840(90M)
Description: The maximum size of data that can be accumulated in memory before being sent to StarRocks at a time. The maximum value ranges from 64 MB to 10 GB. Keep in mind that the Stream Load SDK buffer may create multiple Stream Load jobs to buffer data. Therefore, the threshold mentioned here refers to the total data size.

bufferflush.intervalms

Required: NO
Default value: 300000
Description: Interval for sending a batch of data which controls the load latency. Range: [1000, 3600000].

connect.timeoutms

Required: NO
Default value: 1000
Description: Timeout for connecting to the HTTP URL. Range: [100, 60000].

sink.properties.*

Required:
Default value:
Description: Stream Load parameters o control load behavior. For example, the parameter `sink.properties.format` specifies the format used for Stream Load, such as CSV or JSON. For a list of supported parameters and their descriptions, see [STREAM LOAD](../sql-reference/sql-statements/data-manipulation/STREAM LOAD.md).

sink.properties.format

Required: NO
Default value: json
Description: The format used for Stream Load. The Kafka connector will transform each batch of data to the format before sending them to StarRocks. Valid values: `csv` and `json`. For more information, see CSV parameters**和** JSON parameters.

Limits

- It is not supported to flatten a single message from a Kafka topic into multiple data rows and load into StarRocks.
- The sink of the Kafka connector provided by StarRocks guarantees at-least-once semantics.

Best practices

Load Debezium-formatted CDC data

If the Kafka data is in Debezium CDC format and the StarRocks table is a Primary Key table, you also need to configure the `transforms` parameter and other related parameters in the

configuration file **connect-StarRocks-sink.properties** for the Kafka connector provided by StarRocks.

> **INFO**
>
> In this example, the Kafka connector provided by StarRocks is a sink connector that can continuously consume data from Kafka and load data into StarRocks.

```
transforms=addfield,unwrap
transforms.addfield.type=com.starrocks.connector.kafka.transforms.AddOp
transforms.unwrap.type=io.debezium.transforms.ExtractNewRecordState
transforms.unwrap.drop.tombstones=true
transforms.unwrap.delete.handling.mode
```

In the above configurations, we specify `transforms=addfield,unwrap`.

- The `op` field of the Debezium-formatted CDC data records the SQL operation on each data row from the upstream database. The values `c`, `u`, and `d` represent create, update, and delete, respectively. If the StarRocks table is a Primary Key table, you need to specify the addfield transform. The addfield transform adds a `__op` field for each data row to mark the SQL operation on each data row. To form a complete data row, the addfield transform also retrieves the values of other columns from the `before` or `after` fields based on the value of the `op` field in the Debezium-formatted CDC data. Finally, the data will be converted into JSON or CSV format and written into StarRocks. The addfield transform class is `com.Starrocks.Kafka.Transforms.AddOpFieldForDebeziumRecord`. It is included in the Kafka connector JAR file, so you do not need to manually install it.

  If the StarRocks table is not a Primary Key table, you do not need to specify the addfield transform.

- The unwrap transform is provided by Debezium and is used to unwrap Debezium's complex data structure based on the operation type. For more information, see New Record State Extraction.

Load data using Routine Load

This topic introduces how to create a Routine Load job to stream Kafka messages (events) into StarRocks, and familiarizes you with some basic concepts about Routine Load.

To continuously load messages of a stream into StarRocks, you can store the message stream in a Kafka topic, and create a Routine Load job to consume the messages. The Routine Load job persists in StarRocks, generates a series of load tasks to consume the messages in all or part of the partitions in the topic, and loads the messages into StarRocks.

A Routine Load job supports exactly-once delivery semantics to guarantee the data loaded into StarRocks is neither lost nor duplicated.

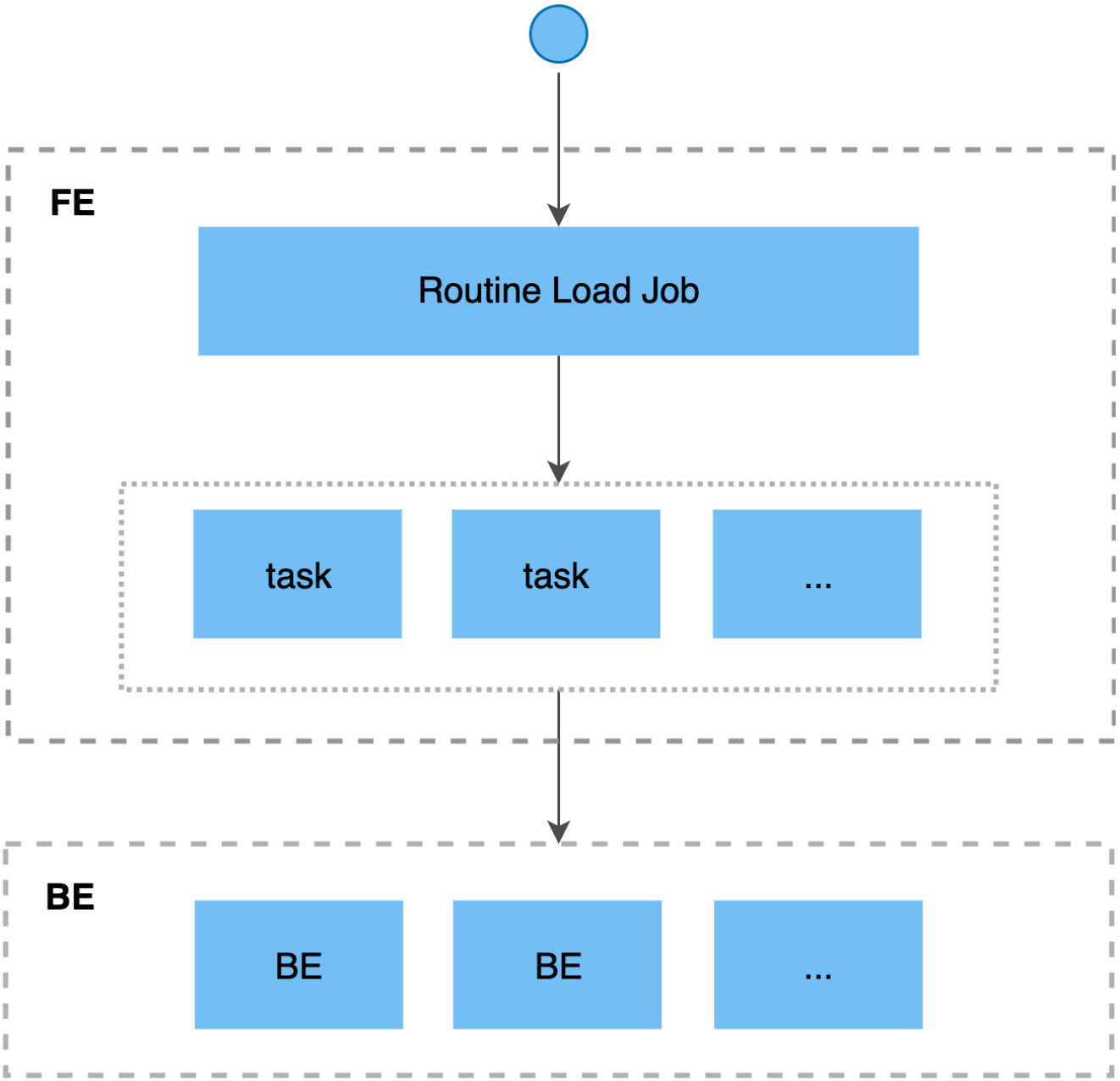
Routine Load supports data transformation at data loading and supports data changes made by UPSERT and DELETE operations during data loading. For more information, see Transform data at loading and Change data through loading.

Supported data formats

Routine Load now supports consuming CSV and JSON format data from a Kafka cluster.

> **NOTE**
>
> For CSV data, take note of the following points:
>
> - You can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
> - Null values are denoted by using `\N`. For example, a data file consists of three columns, and a record from that data file holds data in the first and third columns but no data in the second column. In this situation, you need to use `\N` in the second column to denote a null value. This means the record must be compiled as `a,\N,b` instead of `a,,b`. `a,,b` denotes that the second column of the record holds an empty string.

Basic concepts



Terminology

- **Load job**

  A Routine Load job is a long-running job. As long as its status is RUNNING, a load job continuously generates one or multiple concurrent load tasks which consume the messages in a topic of a Kafka cluster and load the data into StarRocks.

- **Load task**

A load job is split into multiple load tasks by certain rules. A load task is the basic unit of data loading. As an individual event, a load task implements the load mechanism based on Stream Load. Multiple load tasks concurrently consume the messages from different partitions of a topic, and load the data into StarRocks.

Workflow

1. **Create a Routine Load job.** To load data from Kafka, you need to create a Routine Load job by running the CREATE ROUTINE LOAD statement. The FE parses the statement, and creates the job according to the properties you have specified.

2. **The FE splits the job into multiple load tasks.**

   The FE split the job into multiple load tasks based on certain rules. Each load task is an individual transaction. The splitting rules are as follows:

   - The FE calculates the actual concurrent number of the load tasks according to the desired concurrent number `desired_concurrent_number`, the partition number in the Kafka topic, and the number of the BE nodes that are alive.
   - The FE splits the job into load tasks based on the actual concurrent number calculated, and arranges the tasks in the task queue.

   Each Kafka topic consists of multiple partitions. The relation between the topic partition and the load task is as follows:

   - A partition is uniquely assigned to a load task, and all messages from the partition are consumed by the load task.
   - A load task can consume messages from one or more partitions.
   - All partitions are distributed evenly among load tasks.

3. **Multiple load tasks run concurrently to consume the messages from multiple Kafka topic partitions, and load the data into StarRocks**

   i. **The FE schedules and submits load tasks**: the FE schedules the load tasks in the queue on a timely basis, and assigns them to selected Coordinator BE nodes. The interval between load tasks is defined by the configuration item `max_batch_interval`. The FE distributes the load tasks evenly to all BE nodes. See CREATE ROUTINE LOAD for more information about `max_batch_interval`.

ii. The Coordinator BE starts the load task, consumes messages in partitions, parses and filters the data. A load task lasts until the pre-defined amount of messages are consumed or the pre-defined time limit is reached. The message batch size and time limit are defined in the FE configurations `max_routine_load_batch_size` and `routine_load_task_consume_second`. For detailed information, see Configuration. The Coordinator BE then distributes the messages to the Executor BEs. The Executor BEs write the messages to disks.

> **NOTE**
>
> StarRocks supports access to Kafka via security protocols including SASL_SSL, SAS_PLAINTEXT, SSL, and PLAINTEXT. This topic uses connecting to Kafka via PLAINTEXT as an example. If you need to connect to Kafka via other security protocols, see CREATE ROUTINE LOAD.

4. **The FE generates new load tasks to load data continuously.** After the Executor BEs has written the data to disks, the Coordinator BE reports the result of the load task to the FE. Based on the result, the FE then generates new load tasks to load the data continuously. Or the FE retries the failed tasks to make sure the data loaded into StarRocks is neither lost nor duplicated.

Create a Routine Load job

The following two examples describe how to consume CSV-format and JSON-format data in Kafka, and load the data into StarRocks by creating a Routine Load job. For detailed instruction and reference, see CREATE ROUTINE LOAD.

Load CSV-format data

This section describes how to create a Routine Load job to consume CSV-format data in a Kafka cluster, and load the data into StarRocks.

Prepare a dataset

Suppose there is a CSV-format dataset in the topic `ordertest1` in a Kafka cluster. Every message in the dataset includes six fields: order ID, payment date, customer name, nationality, gender, and price.

```
2020050802,2020-05-08,Johann Georg Faust,Deutschland,male,895
2020050802,2020-05-08,Julien Sorel,France,male,893
2020050803,2020-05-08,Dorian Grey,UK,male,1262
2020050901,2020-05-09,Anna Karenina",Russia,female,175
2020051001,2020-05-10,Tess Durbeyfield,US,female,986
2020051101,2020-05-11,Edogawa Conan,japan,male,8924
```

Create a table

According to the fields of CSV-format data, create the table `example_tbl1` in the database `example_db`. The following example creates a table with 5 fields excluding the field of customer gender in the CSV-format data.

```
CREATE TABLE example_db.example_tbl1 (
    `order_id` bigint NOT NULL COMMENT "Order ID",
    `pay_dt` date NOT NULL COMMENT "Payment date",
    `customer_name` varchar(26) NULL COMMENT "Customer name",
    `nationality` varchar(26) NULL COMMENT "Nationality",
    `price`double NULL COMMENT "Price"
)
ENGINE=OLAP
DUPLICATE KEY (order_id,pay_dt)
DISTRIBUTED BY HASH(`order_id`) BUCKETS 5;
```

Submit a Routine Load job

Execute the following statement to submit a Routine Load job named `example_tbl1_ordertest1` to consume the messages in the topic `ordertest1` and load the data into the table `example_tbl1`. The load task consumes the messages from the initial offset in the specified partitions of the topic.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON
example_tbl1
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, temp_gender,
price)
PROPERTIES
(
    "desired_concurrent_number" = "5"
)
FROM KAFKA
```

```
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1",
    "kafka_partitions" ="0,1,2,3,4",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

After submitting the load job, you can execute the SHOW ROUTINE LOAD statement to check the status of the load job.

- **load job name**

  There could be multiple load job on a table. Therefore, we recommend you name a load job with the corresponding Kafka topic and the time when the load job is submitted. It helps you distinguish the load job on each table.

- **Column separator**

  The property `COLUMN TERMINATED BY` defines the column separator of the CSV-format data. The default is `\t`.

- **Kafka topic partition and offset**

  You can specify the properties `kafka_partitions` and `kafka_offsets` to specify the partitions and offsets to consume the messages. For example, if you want the load job to consume messages from the Kafka partitions `"0,1,2,3,4"` of the topic `ordertest1` all with the initial offsets, you can specify the properties as follows: If you want the load job to consume messages from the Kafka partitions `"0,1,2,3,4"` and you need to specify a separate starting offset for each partition, you can configure as follows:

  ```
  "kafka_partitions" ="0,1,2,3,4",
  "kafka_offsets" = "OFFSET_BEGINNING, OFFSET_END, 1000, 2000,
  3000"
  ```

  You can also set the default offsets of all partitions with the property `property.kafka_default_offsets`.

```
"kafka_partitions" ="0,1,2,3,4",
"property.kafka_default_offsets" = "OFFSET_BEGINNING"
```

For detailed information, see CREATE ROUTINE LOAD.

- **Data mapping and transformation**

  To specify the mapping and transformation relationship between the CSV-format data, and the StarRocks table, you need to use the `COLUMNS` parameter.

  Data mapping:

  - StarRocks extracts the columns in the CSV-format data and maps them **in sequence** onto the fields declared in the `COLUMNS` parameter.
  - StarRocks extracts the fields declared in the `COLUMNS` parameter and maps them **by name** onto the columns of StarRocks table.

  Data transformation:

  And because the example excludes the column of customer gender from the CSV-format data, the field `temp_gender` in `COLUMNS` parameter is used as a placeholder for this field. The other fields are mapped to columns of the StarRocks table `example_tbl1` directly.

  For more information about data transformation, see Transform data at loading.

  > **NOTE**
  >
  > You do not need to specify the `COLUMNS` parameter if the names, number, and order of the columns in the CSV-format data completely correspond to those of the StarRocks table.

- **Task concurrency**

  When there are many Kafka topic partitions and enough BE nodes, you can accelerate the loading by increasing the task concurrency.

  To increase the actual load task concurrency, you can increase the desired load task concurrency `desired_concurrent_number` when you create a routine load job. You can also set the dynamic configuration item of FE

`max_routine_load_task_concurrent_num` (default maximum load task currency) to a larger value. For more information about `max_routine_load_task_concurrent_num`, please see FE configuration items.

The actual task concurrency is defined by the minimum value among the number of BE nodes that are alive, the number of the pre-specified Kafka topic partitions, and the values of `desired_concurrent_number` and `max_routine_load_task_concurrent_num`.

In the example, the number of BE nodes that are alive is `5`, the number of the pre-specified Kafka topic partitions is `5`, and the value of `max_routine_load_task_concurrent_num` is `5`. To increase the actual load task concurrency, you can increase the `desired_concurrent_number` from the default value `3` to `5`.

For more about the properties, see CREATE ROUTINE LOAD. For detailed instructions on accelerating the loading, see Routine Load FAQ.

Load JSON-format data

This section describes how to create a Routine Load job to consume JSON-format data in a Kafka cluster, and load the data into StarRocks.

Prepare a dataset

Suppose there is a JSON-format dataset in the topic `ordertest2` in a Kafka cluster. The dataset includes six keys: commodity ID, customer name, nationality, payment time, and price. Besides, you want to transform the payment time column into the DATE type, and load it into the `pay_dt` column in the StarRocks table.

```
{"commodity_id": "1", "customer_name": "Mark Twain", "country":
"US","pay_time": 1589191487,"price": 875}
{"commodity_id": "2", "customer_name": "Oscar Wilde", "country":
"UK","pay_time": 1589191487,"price": 895}
{"commodity_id": "3", "customer_name": "Antoine de Saint-
Exupéry","country": "France","pay_time": 1589191487,"price": 895}
```

> **CAUTION** Each JSON object in a row must be in one Kafka message, otherwise a JSON parsing error is returned.

Create a table

According to the keys of the JSON-format data, create the table `example_tbl2` in the database `example_db`.

```
CREATE TABLE `example_tbl2` (
    `commodity_id` varchar(26) NULL COMMENT "Commodity ID",
    `customer_name` varchar(26) NULL COMMENT "Customer name",
    `country` varchar(26) NULL COMMENT "Country",
    `pay_time` bigint(20) NULL COMMENT "Payment time",
    `pay_dt` date NULL COMMENT "Payment date",
    `price`double SUM NULL COMMENT "Price"
)
ENGINE=OLAP
AGGREGATE
KEY(`commodity_id`,`customer_name`,`country`,`pay_time`,`pay_dt`)
DISTRIBUTED BY HASH(`commodity_id`) BUCKETS 5;
```

Submit a Routine Load job

Execute the following statement to submit a Routine Load job named `example_tbl2_ordertest2` to consume the messages in the topic `ordertest2` and load the data into the table `example_tbl2`. The load task consumes the messages from the initial offset in the specified partitions of the topic.

```
CREATE ROUTINE LOAD example_db.example_tbl2_ordertest2 ON example_tbl2
COLUMNS(commodity_id, customer_name, country, pay_time, price,
pay_dt=from_unixtime(pay_time, '%Y%m%d'))
PROPERTIES
(
    "desired_concurrent_number"="5",
    "format" ="json",
    "jsonpaths" ="
[\"$.commodity_id\",\"$.customer_name\",\"$.country\",\"$.pay_time\",\"
 )
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest2",
    "kafka_partitions" ="0,1,2,3,4",
```

```
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

After submitting the load job, you can execute the SHOW ROUTINE LOAD statement to check the status of the load job.

- **Data format**

  You need to specify `"format"="json"` in the clause `PROPERTIES` to define that the data format is JSON.

- **Data mapping and transformation**

  To specify the mapping and transformation relationship between the JSON-format data, and the StarRocks table, you need to specify the paramter `COLUMNS` and property `jsonpaths`. The order of fields specified in the `COLUMNS` parameter must match that of the JSON-format data, and the name of fields must match that of the StarRocks table. The property `jsonpaths` is used to extract the required fields from the JSON data. These fields are then named by the property `COLUMNS`.

  Because the example needs to transform the payment time field to the DATE data type, and load the data into the `pay_dt` column in the StarRocks table, you need to use the from_unixtime function. The other fields are mapped to fields of the table `example_tbl2` directly.

  Data mapping:

  - StarRocks extracts the `name` and `code` keys of JSON-format data and maps them onto the keys declared in the `jsonpaths` property.
  - StarRocks extracts the keys declared in the `jsonpaths` property and maps them **in sequence** onto the fields declared in the `COLUMNS` parameter.
  - StarRocks extracts the fields declared in the `COLUMNS` parameter and maps them **by name** onto the columns of StarRocks table.

  Data transformation:

  - Because the example needs to transform the key `pay_time` to the DATE data type, and load the data into the `pay_dt` column in the StarRocks table, you need to use

the from_unixtime function in `COLUMNS` parameter. The other fields are mapped to fields of the table `example_tbl2` directly.

- And because the example excludes the column of customer gender from the JSON-format data, the field `temp_gender` in `COLUMNS` parameter is used as a placeholder for this field. The other fields are mapped to columns of the StarRocks table `example_tbl1` directly.

  For more information about data transformation, see Transform data at loading.

  > **NOTE**
  >
  > You do not need to specify the `COLUMNS` parameter if the names and number of the keys in the JSON object completely match those of fields in the StarRocks table.

Check a load job and task

Check a load job

Execute the SHOW ROUTINE LOAD statement to check the status of the load job `example_tbl2_ordertest2`. StarRocks returns the execution state `State`, the statistical information (including the total rows consumed and the total rows loaded) `Statistics`, and the progress of the load job `progress`.

If the state of the load job is automatically changed to **PAUSED**, it is possibly because the number of error rows has exceeded the threshold. For detailed instructions on setting this threshold, see CREATE ROUTINE LOAD. You can check the files `ReasonOfStateChanged` and `ErrorLogUrls` to identify and troubleshoot the problem. Having fixed the problem, you can then execute the RESUME ROUTINE LOAD statement to resume the **PAUSED** load job.

If the state of the load job is **CANCELLED**, it is possibly because the load job encounters an exception (such as the table has been dropped). You can check the files `ReasonOfStateChanged` and `ErrorLogUrls` to identify and troubleshoot the problem. However, you cannot resume a **CANCELLED** load job.

```
MySQL [example_db]> SHOW ROUTINE LOAD FOR example_tbl2_ordertest2 \G
*************************** 1. row ***************************
```

```
                 Id: 63013
               Name: example_tbl2_ordertest2
         CreateTime: 2022-08-10 17:09:00
          PauseTime: NULL
            EndTime: NULL
             DbName: default_cluster:example_db
          TableName: example_tbl2
              State: RUNNING
     DataSourceType: KAFKA
     CurrentTaskNum: 3
      JobProperties: {"partitions":"*","partial_update":"false","colum
'%Y%m%d'),price","maxBatchIntervalS":"20","whereExpr":"*","dataFormat":
[\"$.commodity_id\",\"$.customer_name\",\"$.country\",\"$.pay_time\",\"
DataSourceProperties: {"topic":"ordertest2","currentKafkaPartitions":"0
    CustomProperties: {"kafka_default_offsets":"OFFSET_BEGINNING"}
           Statistic: {"receivedBytes":230,"errorRows":0,"committedTaskN
            Progress: {"0":"1","1":"OFFSET_ZERO","2":"OFFSET_ZERO","3":
ReasonOfStateChanged:
        ErrorLogUrls:
            OtherMsg:
```

> **CAUTION**
>
> You cannot check a load job that has stopped or has not yet started.

Check a load task

Execute the SHOW ROUTINE LOAD TASK statement to check the load tasks of the load job `example_tbl2_ordertest2`, such as how many tasks are currently running, the Kafka topic partitions that are consumed and the consumption progress `DataSourceProperties`, and the corresponding Coordinator BE node `BeId`.

```
MySQL [example_db]> SHOW ROUTINE LOAD TASK WHERE JobName =
"example_tbl2_ordertest2" \G
*************************** 1. row ***************************
              TaskId: 18c3a823-d73e-4a64-b9cb-b9eced026753
                TxnId: -1
           TxnStatus: UNKNOWN
               JobId: 63013
          CreateTime: 2022-08-10 17:09:05
   LastScheduledTime: 2022-08-10 17:47:27
    ExecuteStartTime: NULL
             Timeout: 60
```

```
                BeId: -1
DataSourceProperties: {"1":0,"4":0}
             Message: there is no new data in kafka, wait for 20
seconds to schedule again
*************************** 2. row ***************************
              TaskId: f76c97ac-26aa-4b41-8194-a8ba2063eb00
                TxnId: -1
           TxnStatus: UNKNOWN
               JobId: 63013
          CreateTime: 2022-08-10 17:09:05
   LastScheduledTime: 2022-08-10 17:47:26
    ExecuteStartTime: NULL
             Timeout: 60
                BeId: -1
DataSourceProperties: {"2":0}
             Message: there is no new data in kafka, wait for 20
seconds to schedule again
*************************** 3. row ***************************
              TaskId: 1a327a34-99f4-4f8d-8014-3cd38db99ec6
                TxnId: -1
           TxnStatus: UNKNOWN
               JobId: 63013
          CreateTime: 2022-08-10 17:09:26
   LastScheduledTime: 2022-08-10 17:47:27
    ExecuteStartTime: NULL
             Timeout: 60
                BeId: -1
DataSourceProperties: {"0":2,"3":0}
             Message: there is no new data in kafka, wait for 20
seconds to schedule again
```

Pause a load job

You can execute the PAUSE ROUTINE LOAD statement to pause a load job. The state of the load job will be **PAUSED** after the statement is executed. However, it has not stopped. You can execute the RESUME ROUTINE LOAD statement to resume it. You can also check its status with the SHOW ROUTINE LOAD statement.

The following example pauses the load job `example_tbl2_ordertest2`:

```
PAUSE ROUTINE LOAD FOR example_tbl2_ordertest2;
```

Resume a load job

You can execute the RESUME ROUTINE LOAD statement to resume a paused load job. The state of the load job will be **NEED_SCHEDULE** temporarily (because the load job is being re-scheduled), and then become **RUNNING**. You can check its status with the SHOW ROUTINE LOAD statement.

The following example resumes the paused load job `example_tbl2_ordertest2`:

```
RESUME ROUTINE LOAD FOR example_tbl2_ordertest2;
```

Alter a load job

Before altering a load job, you must pause it with the PAUSE ROUTINE LOAD statement. Then you can execute the ALTER ROUTINE LOAD. After altering it, you can execute the RESUME ROUTINE LOAD statement to resume it, and check its status with the SHOW ROUTINE LOAD statement.

Suppose the number of the BE nodes that are alive increases to `6` and the Kafka topic partitions to be consumed is `"0,1,2,3,4,5,6,7"`. If you want to increase the actual load task concurrency, you can execute the following statement to increase the number of desired task concurrency `desired_concurrent_number` to `6` (greater than or equal to the number of BE nodes that are alive), and specify the Kafka topic partitions and initial offsets.

> **NOTE**
>
> Because the actual task concurrency is determined by the minimum value of multiple parameters, you must make sure that the value of the FE dynamic parameter `max_routine_load_task_concurrent_num` is greater than or equal to `6`.

```
ALTER ROUTINE LOAD FOR example_tbl2_ordertest2
PROPERTIES
(
    "desired_concurrent_number" = "6"
)
FROM kafka
(
    "kafka_partitions" = "0,1,2,3,4,5,6,7",
    "kafka_offsets" =
```

```
"OFFSET_BEGINNING,OFFSET_BEGINNING,OFFSET_BEGINNING,OFFSET_BEGINNING,OF
);
```

Stop a load job

You can execute the STOP ROUTINE LOAD statement to stop a load job. The state of the load job will be **STOPPED** after the statement is executed, and you cannot resume a stopped load job. You cannot check the status of a stopped load job with the SHOW ROUTINE LOAD statement.

The following example stops the load job `example_tbl2_ordertest2`:

```
STOP ROUTINE LOAD FOR example_tbl2_ordertest2;
```

FAQ

Please see Routine Load FAQ.

Load data using Spark connector (recommended)

StarRocks provides a self-developed connector named StarRocks Connector for Apache Spark™ (...

Load data in bulk using Spark Load

This load uses external Apache Spark™ resources to pre-process imported data, which improves i...

Load data using Spark connector (recommended)

StarRocks provides a self-developed connector named StarRocks Connector for Apache Spark™ (Spark connector for short) to help you load data into a StarRocks table by using Spark. The basic principle is to accumulate the data and then load it all at a time into StarRocks through STREAM LOAD. The Spark connector is implemented based on Spark DataSource V2. A DataSource can be created by using Spark DataFrames or Spark SQL. And both batch and structured streaming modes are supported.

> **NOTICE**
>
> Only users with the SELECT and INSERT privileges on a StarRocks table can load data into this table. You can follow the instructions provided in GRANT to grant these privileges to a user.

Version requirements

Spark connector	Spark	StarRocks	Java	Scala
1.1.2	3.2, 3.3, 3.4, 3.5	2.5 and later	8	2.12
1.1.1	3.2, 3.3, or 3.4	2.5 and later	8	2.12
1.1.0	3.2, 3.3, or 3.4	2.5 and later	8	2.12

> **NOTICE**
>
> - Please see Upgrade Spark connector for behaviour changes among different versions of the Spark connector.
> - The Spark connector does not provide MySQL JDBC driver since version 1.1.1, and you need import the driver to the spark classpath manually. You can find the driver on MySQL site or Maven Central.

Obtain Spark connector

You can obtain the Spark connector JAR file in the following ways:

- Directly download the compiled Spark Connector JAR file.
- Add the Spark connector as a dependency in your Maven project and then download the JAR file.
- Compile the source code of the Spark Connector into a JAR file by yourself.

The naming format of the Spark connector JAR file is `starrocks-spark-connector-${spark_version}_${scala_version}-${connector_version}.jar`.

For example, if you install Spark 3.2 and Scala 2.12 in your environment and you want to use Spark connector 1.1.0, you can use `starrocks-spark-connector-3.2_2.12-1.1.0.jar`.

> **NOTICE**
>
> In general, the latest version of the Spark connector only maintains compatibility with the three most recent versions of Spark.

Download the compiled Jar file

Directly download the corresponding version of the Spark connector JAR from the Maven Central Repository.

Maven Dependency

1. In your Maven project's `pom.xml` file, add the Spark connector as a dependency according to the following format. Replace `spark_version`, `scala_version`, and `connector_version` with the respective versions.

```
<dependency>
<groupId>com.starrocks</groupId>
<artifactId>starrocks-spark-connector-${spark_version}_${scala_version}</artifactId>
<version>${connector_version}</version>
</dependency>
```

2. For example, if the version of Spark in your environment is 3.2, the version of Scala is 2.12, and you choose Spark connector 1.1.0, you need to add the following dependency:

```
<dependency>
<groupId>com.starrocks</groupId>
<artifactId>starrocks-spark-connector-3.2_2.12</artifactId>
<version>1.1.0</version>
</dependency>
```

Compile by yourself

1. Download the Spark connector package.

2. Execute the following command to compile the source code of Spark connector into a JAR file. Note that `spark_version` is replaced with the corresponding Spark version.

```
sh build.sh <spark_version>
```

For example, if the Spark version in your environment is 3.2, you need to execute the following command:

```
sh build.sh 3.2
```

3. Go to the `target/` directory to find the Spark connector JAR file, such as `starrocks-spark-connector-3.2_2.12-1.1.0-SNAPSHOT.jar` , generated upon compilation.

> **NOTE**
>
> The name of Spark connector which is not formally released contains the `SNAPSHOT` suffix.

Parameters

Parameter	Required	Default value	
starrocks.fe.http.url	YES	None	The H cluste which Forma <fe_h versio prefix http <fe_h <fe_h
starrocks.fe.jdbc.url	YES	None	The a MySQ jdbc <fe_c
starrocks.table.identifier	YES	None	The n <data
starrocks.user	YES	None	The u accou INSER
starrocks.password	YES	None	The p accou
starrocks.write.label.prefix	NO	spark-	The la
starrocks.write.enable.transaction-stream-load	NO	TRUE	Wheth interfa v2.5 o data i usage **NOTI** effect star

Parameter	Required	Default value	
			positi interfa
starrocks.write.buffer.size	NO	104857600	The m accun to Sta param loadin loadin
starrocks.write.buffer.rows	NO	Integer.MAX_VALUE	Suppo numb memo time.
starrocks.write.flush.interval.ms	NO	300000	The in StarR the loa

starrocks.write.max.retries	NO	3	Suppo
			of tim
			perfor
			of dat
			NOTI
			interfa
			param
			use St
			value
			`star`
			`strea`

starrocks.write.retry.interval.ms	NO	10000	Suppo retry t data it
starrocks.columns	NO	None	The S want t

Parameter	Required	Default value	
			colum comm `"col`
starrocks.column.types	NO	None	Suppo colum using StarR The p forma Struct `col1` you o custo into c
starrocks.write.properties.*	NO	None	The p Strea param `star` specif loade suppo descr
starrocks.write.properties.format	NO	CSV	The fi conne before values
starrocks.write.properties.row_delimiter	NO	\n	The r
starrocks.write.properties.column_separator	NO	\t	The c data.
starrocks.write.num.partitions	NO	None	The n can w

Parameter	Required	Default value	
			volum of par concu value Spark Spark
starrocks.write.partition.columns	NO	None	The p param `star` specif all col partiti
starrocks.timezone	NO	Default timezone of JVM	Suppo conve StarR timez `Zone` can b `Asia/` `+08:0`

Data type mapping between Spark and StarRocks

- The default data type mapping is as follows:

Spark data type	StarRocks data type
BooleanType	BOOLEAN
ByteType	TINYINT

Spark data type	StarRocks data type
ShortType	SMALLINT
IntegerType	INT
LongType	BIGINT
StringType	LARGEINT
FloatType	FLOAT
DoubleType	DOUBLE
DecimalType	DECIMAL
StringType	CHAR
StringType	VARCHAR
StringType	STRING
StringType	JSON
DateType	DATE
TimestampType	DATETIME
ArrayType	ARRAY **NOTE:** **Supported since version 1.1.1**. For detailed steps, see Load data into columns of ARRAY type.

- You can also customize the data type mapping.

  For example, a StarRocks table consists of the BITMAP and HLL data types, but Spark does not support the two data types. You need to customize the corresponding data

types in Spark. For detailed steps, see load data into columns of BITMAP and HLL types. **BITMAP and HLL are supported since version 1.1.1**.

Upgrade Spark connector

Upgrade from version 1.1.0 to 1.1.1

- Since 1.1.1, the Spark connector does not provide `mysql-connector-java` which is the official JDBC driver for MySQL, because of the limitations of the GPL license used by `mysql-connector-java`. However, the Spark connector still needs the MySQL JDBC driver to connect to StarRocks for the table metadata, so you need to add the driver to the Spark classpath manually. You can find the driver on MySQL site or Maven Central.
- Since 1.1.1, the connector uses Stream Load interface by default rather than Stream Load transaction interface in version 1.1.0. If you still want to use Stream Load transaction interface, you can set the option `starrocks.write.max.retries` to `0`. Please see the description of `starrocks.write.enable.transaction-stream-load` and `starrocks.write.max.retries` for details.

Examples

The following examples show how to use the Spark connector to load data into a StarRocks table with Spark DataFrames or Spark SQL. The Spark DataFrames supports both Batch and Structured Streaming modes.

For more examples, see Spark Connector Examples.

Preparations

Create a StarRocks table

Create a database `test` and create a Primary Key table `score_board`.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`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`);
```

Set up your Spark environment

Note that the following examples are run in Spark 3.2.4 and use `spark-shell`, `pyspark` and `spark-sql`. Before running the examples, make sure to place the Spark connector JAR file in the `$SPARK_HOME/jars` directory.

Load data with Spark DataFrames

The following two examples explain how to load data with Spark DataFrames Batch or Structured Streaming mode.

Batch

Construct data in memory and load data into the StarRocks table.

1. You can write the spark application using Scala or Python.

For Scala, run the following code snippet in `spark-shell`:

```
// 1. Create a DataFrame from a sequence.
val data = Seq((1, "starrocks", 100), (2, "spark", 100))
val df = data.toDF("id", "name", "score")

// 2. Write to StarRocks by configuring the format as "starrocks" and
the following options.
// You need to modify the options according your own environment.
df.write.format("starrocks")
    .option("starrocks.fe.http.url", "127.0.0.1:8030")
    .option("starrocks.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030")
    .option("starrocks.table.identifier", "test.score_board")
    .option("starrocks.user", "root")
    .option("starrocks.password", "")
```

```
    .mode("append")
    .save()
```

For Python, run the following code snippet in `pyspark`:

```
from pyspark.sql import SparkSession

spark = SparkSession \
     .builder \
     .appName("StarRocks Example") \
     .getOrCreate()

 # 1. Create a DataFrame from a sequence.
 data = [(1, "starrocks", 100), (2, "spark", 100)]
 df = spark.sparkContext.parallelize(data) \
         .toDF(["id", "name", "score"])

 # 2. Write to StarRocks by configuring the format as "starrocks" and
the following options.
 # You need to modify the options according your own environment.
 df.write.format("starrocks") \
     .option("starrocks.fe.http.url", "127.0.0.1:8030") \
     .option("starrocks.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030")
\
     .option("starrocks.table.identifier", "test.score_board") \
     .option("starrocks.user", "root") \
     .option("starrocks.password", "") \
     .mode("append") \
     .save()
```

2. Query data in the StarRocks table.

```
MySQL [test]> SELECT * FROM `score_board`;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | starrocks |   100 |
|    2 | spark     |   100 |
+------+-----------+-------+
2 rows in set (0.00 sec)
```

Structured Streaming

Construct a streaming read of data from a CSV file and load data into the StarRocks table.

1. In the directory `csv-data`, create a CSV file `test.csv` with the following data:

```
3,starrocks,100
4,spark,100
```

2. You can write the Spark application using Scala or Python.

For Scala, run the following code snippet in `spark-shell`:

```
import org.apache.spark.sql.types.StructType

// 1. Create a DataFrame from CSV.
val schema = (new StructType()
        .add("id", "integer")
        .add("name", "string")
        .add("score", "integer")
    )
val df = (spark.readStream
        .option("sep", ",")
        .schema(schema)
        .format("csv")
        // Replace it with your path to the directory "csv-data".
        .load("/path/to/csv-data")
    )

// 2. Write to StarRocks by configuring the format as "starrocks" and
the following options.
// You need to modify the options according your own environment.
val query = (df.writeStream.format("starrocks")
        .option("starrocks.fe.http.url", "127.0.0.1:8030")
        .option("starrocks.fe.jdbc.url",
"jdbc:mysql://127.0.0.1:9030")
        .option("starrocks.table.identifier", "test.score_board")
        .option("starrocks.user", "root")
        .option("starrocks.password", "")
        // replace it with your checkpoint directory
        .option("checkpointLocation", "/path/to/checkpoint")
        .outputMode("append")
        .start()
    )
```

For Python, run the following code snippet in `pyspark`:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType, StringType, StructType,
StructField

spark = SparkSession \
     .builder \
     .appName("StarRocks SS Example") \
     .getOrCreate()

 # 1. Create a DataFrame from CSV.
 schema = StructType([ \
         StructField("id", IntegerType()), \
         StructField("name", StringType()), \
         StructField("score", IntegerType()) \
     ])
 df = spark.readStream \
         .option("sep", ",") \
         .schema(schema) \
         .format("csv") \
         # Replace it with your path to the directory "csv-data".
         .load("/path/to/csv-data")

 # 2. Write to StarRocks by configuring the format as "starrocks" and
the following options.
 # You need to modify the options according your own environment.
 query = df.writeStream.format("starrocks") \
         .option("starrocks.fe.http.url", "127.0.0.1:8030") \
         .option("starrocks.fe.jdbc.url",
"jdbc:mysql://127.0.0.1:9030") \
         .option("starrocks.table.identifier", "test.score_board") \
         .option("starrocks.user", "root") \
         .option("starrocks.password", "") \
         # replace it with your checkpoint directory
         .option("checkpointLocation", "/path/to/checkpoint") \
         .outputMode("append") \
         .start()
     )
```

3. Query data in the StarRocks table.

```
MySQL [test]> select * from score_board;
+------+-----------+-------+
```

```
| id   | name      | score |
+------+-----------+-------+
|    4 | spark     |   100 |
|    3 | starrocks |   100 |
+------+-----------+-------+
2 rows in set (0.67 sec)
```

Load data with Spark SQL

The following example explains how to load data with Spark SQL by using the `INSERT INTO` statement in the Spark SQL CLI.

1. Execute the following SQL statement in the `spark-sql`:

```
-- 1. Create a table by configuring the data source as
`starrocks` and the following options.
-- You need to modify the options according your own environment.
CREATE TABLE `score_board`
USING starrocks
OPTIONS(
"starrocks.fe.http.url"="127.0.0.1:8030",
"starrocks.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
"starrocks.table.identifier"="test.score_board",
"starrocks.user"="root",
"starrocks.password"=""
);

-- 2. Insert two rows into the table.
INSERT INTO `score_board` VALUES (5, "starrocks", 100), (6,
"spark", 100);
```

2. Query data in the StarRocks table.

```
MySQL [test]> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    6 | spark     |   100 |
|    5 | starrocks |   100 |
+------+-----------+-------+
2 rows in set (0.00 sec)
```

Best Practices

Load data to Primary Key table

This section will show how to load data to StarRocks Primary Key table to achieve partial updates, and conditional updates. You can see Change data through loading for the detailed introduction of these features. These examples use Spark SQL.

Preparations

Create a database `test` and create a Primary Key table `score_board` in StarRocks.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`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`);
```

Partial updates

This example will show how to only update data in the column `name` through loading:

1. Insert initial data to StarRocks table in MySQL client.

```
mysql> INSERT INTO `score_board` VALUES (1, 'starrocks', 100),
(2, 'spark', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | starrocks |   100 |
|    2 | spark     |   100 |
```

```
+------+----------+-------+
2 rows in set (0.02 sec)
```

2. Create a Spark table `score_board` in Spark SQL client.

   - Set the option `starrocks.write.properties.partial_update` to `true` which tells the connector to do partial update.
   - Set the option `starrocks.columns` to `"id,name"` to tell the connector which columns to write.

```
CREATE TABLE `score_board`
USING starrocks
OPTIONS(
    "starrocks.fe.http.url"="127.0.0.1:8030",
    "starrocks.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
    "starrocks.table.identifier"="test.score_board",
    "starrocks.user"="root",
    "starrocks.password"="",
    "starrocks.write.properties.partial_update"="true",
    "starrocks.columns"="id,name"
 );
```

3. Insert data into the table in Spark SQL client, and only update the column `name`.

```
INSERT INTO `score_board` VALUES (1, 'starrocks-update'), (2,
'spark-update');
```

4. Query the StarRocks table in MySQL client.

   You can see that only values for `name` change, and the values for `score` does not change.

```
mysql> select * from score_board;
+------+-----------------+-------+
| id   | name            | score |
+------+-----------------+-------+
|    1 | starrocks-update |   100 |
|    2 | spark-update     |   100 |
+------+-----------------+-------+
2 rows in set (0.02 sec)
```

Conditional updates

This example will show how to do conditional updates according to the values of column `score`. The update for an `id` takes effect only when the new value for `score` is has a greater or equal to the old value.

1. Insert initial data to StarRocks table in MySQL client.

```
mysql> INSERT INTO `score_board` VALUES (1, 'starrocks', 100),
(2, 'spark', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | starrocks |   100 |
|    2 | spark     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Create a Spark table `score_board` in the following ways.

   - Set the option `starrocks.write.properties.merge_condition` to `score` which tells the connector to use the column `score` as the condition.
   - Make sure that the Spark connector use Stream Load interface to load data, rather than Stream Load transaction interface, because the latter does not support this feature.

```
CREATE TABLE `score_board`
USING starrocks
OPTIONS(
    "starrocks.fe.http.url"="127.0.0.1:8030",
    "starrocks.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
    "starrocks.table.identifier"="test.score_board",
    "starrocks.user"="root",
    "starrocks.password"="",
    "starrocks.write.properties.merge_condition"="score"
 );
```

3. Insert data to the table in Spark SQL client, and update the row whose `id` is 1 with a smaller score value, and the row whose `id` is 2 with a larger score value.

```
INSERT INTO `score_board` VALUES (1, 'starrocks-update', 99), (2,
'spark-update', 101);
```

4. Query the StarRocks table in MySQL client.

   You can see that only the row whose `id` is 2 changes, and the row whose `id` is 1 does not change.

```
mysql> select * from score_board;
+------+--------------+-------+
| id   | name         | score |
+------+--------------+-------+
|    1 | starrocks    |   100 |
|    2 | spark-update |   101 |
+------+--------------+-------+
2 rows in set (0.03 sec)
```

Load data into columns of BITMAP type

`BITMAP` is often used to accelerate count distinct, such as counting UV, see Use Bitmap for exact Count Distinct. Here we take the counting of UV as an example to show how to load data into columns of the `BITMAP` type. **`BITMAP` is supported since version 1.1.1**.

1. Create a StarRocks Aggregate table.

   In the database `test`, create an Aggregate table `page_uv` where the column `visit_users` is defined as the `BITMAP` type and configured with the aggregate function `BITMAP_UNION`.

```
CREATE TABLE `test`.`page_uv` (
  `page_id` INT NOT NULL COMMENT 'page ID',
  `visit_date` datetime NOT NULL COMMENT 'access time',
  `visit_users` BITMAP BITMAP_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`);
```

2. Create a Spark table.

The schema of the Spark table is inferred from the StarRocks table, and the Spark does not support the `BITMAP` type. So you need to customize the corresponding column data type in Spark, for example as `BIGINT`, by configuring the option `"starrocks.column.types"="visit_users BIGINT"`. When using Stream Load to ingest data, the connector uses the `to_bitmap` function to convert the data of `BIGINT` type into `BITMAP` type.

Run the following DDL in `spark-sql`:

```
CREATE TABLE `page_uv`
USING starrocks
OPTIONS(
   "starrocks.fe.http.url"="127.0.0.1:8030",
   "starrocks.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
   "starrocks.table.identifier"="test.page_uv",
   "starrocks.user"="root",
   "starrocks.password"="",
   "starrocks.column.types"="visit_users BIGINT"
);
```

3. Load data into StarRocks table.

   Run the following DML in `spark-sql`:

```
INSERT INTO `page_uv` VALUES
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 13),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 33),
   (1, CAST('2020-06-23 02:30:30' AS TIMESTAMP), 13),
   (2, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23);
```

4. Calculate page UVs from the StarRocks table.

```
MySQL [test]> SELECT `page_id`, COUNT(DISTINCT `visit_users`)
FROM `page_uv` GROUP BY `page_id`;
+---------+----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+----------------------------+
|       2 |                          1 |
|       1 |                          3 |
```

```
+--------+----------------------------+
2 rows in set (0.01 sec)
```

> **NOTICE:**
>
> The connector uses `to_bitmap` function to convert data of the `TINYINT`, `SMALLINT`, `INTEGER`, and `BIGINT` types in Spark to the `BITMAP` type in StarRocks, and uses `bitmap_hash` function for other Spark data types.

Load data into columns of HLL type

`HLL` can be used for approximate count distinct, see Use HLL for approximate count distinct.

Here we take the counting of UV as an example to show how to load data into columns of the `HLL` type. **`HLL` is supported since version 1.1.1**.

1. Create a StarRocks Aggregate table.

   In the database `test`, create an Aggregate table `hll_uv` where the column `visit_users` is defined as the `HLL` type and configured with the aggregate function `HLL_UNION`.

   ```
   CREATE TABLE `hll_uv` (
   `page_id` INT NOT NULL COMMENT 'page ID',
   `visit_date` datetime NOT NULL COMMENT 'access time',
   `visit_users` HLL HLL_UNION NOT NULL COMMENT 'user ID'
   ) ENGINE=OLAP
   AGGREGATE KEY(`page_id`, `visit_date`)
   DISTRIBUTED BY HASH(`page_id`);
   ```

2. Create a Spark table.

   The schema of the Spark table is inferred from the StarRocks table, and the Spark does not support the `HLL` type. So you need to customize the corresponding column data type in Spark, for example as `BIGINT`, by configuring the option `"starrocks.column.types"="visit_users BIGINT"`. When using Stream Load to ingest data, the connector uses the `hll_hash` function to convert the data of `BIGINT` type into `HLL` type.

Run the following DDL in `spark-sql`:

```
CREATE TABLE `hll_uv`
USING starrocks
OPTIONS(
   "starrocks.fe.http.url"="127.0.0.1:8030",
   "starrocks.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
   "starrocks.table.identifier"="test.hll_uv",
   "starrocks.user"="root",
   "starrocks.password"="",
   "starrocks.column.types"="visit_users BIGINT"
);
```

3. Load data into StarRocks table.

   Run the following DML in `spark-sql`:

```
INSERT INTO `hll_uv` VALUES
   (3, CAST('2023-07-24 12:00:00' AS TIMESTAMP), 78),
   (4, CAST('2023-07-24 13:20:10' AS TIMESTAMP), 2),
   (3, CAST('2023-07-24 12:30:00' AS TIMESTAMP), 674);
```

4. Calculate page UVs from the StarRocks table.

```
MySQL [test]> SELECT `page_id`, COUNT(DISTINCT `visit_users`)
FROM `hll_uv` GROUP BY `page_id`;
+---------+----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+----------------------------+
|       4 |                          1 |
|       3 |                          2 |
+---------+----------------------------+
2 rows in set (0.01 sec)
```

Load data into columns of ARRAY type

The following example explains how to load data into columns of the `ARRAY` type.

1. Create a StarRocks table.

In the database `test`, create a Primary Key table `array_tbl` that includes one `INT` column and two `ARRAY` columns.

```
CREATE TABLE `array_tbl` (
    `id` INT NOT NULL,
    `a0` ARRAY<STRING>,
    `a1` ARRAY<ARRAY<INT>>
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`)
;
```

2. Write data to StarRocks.

   Because some versions of StarRocks does not provide the metadata of `ARRAY` column, the connector can not infer the corresponding Spark data type for this column. However, you can explicitly specify the corresponding Spark data type of the column in the option `starrocks.column.types`. In this example, you can configure the option as `a0 ARRAY<STRING>,a1 ARRAY<ARRAY<INT>>`.

   Run the following codes in `spark-shell`:

```
val data = Seq(
   |  (1, Seq("hello", "starrocks"), Seq(Seq(1, 2), Seq(3, 4))),
   |  (2, Seq("hello", "spark"), Seq(Seq(5, 6, 7), Seq(8, 9,
10)))
   | )
val df = data.toDF("id", "a0", "a1")
df.write
     .format("starrocks")
     .option("starrocks.fe.http.url", "127.0.0.1:8030")
     .option("starrocks.fe.jdbc.url",
"jdbc:mysql://127.0.0.1:9030")
     .option("starrocks.table.identifier", "test.array_tbl")
     .option("starrocks.user", "root")
     .option("starrocks.password", "")
     .option("starrocks.column.types", "a0 ARRAY<STRING>,a1
ARRAY<ARRAY<INT>>")
     .mode("append")
     .save()
```

3. Query data in the StarRocks table.

```
MySQL [test]> SELECT * FROM `array_tbl`;
+------+----------------------+-------------------+
| id   | a0                   | a1                |
+------+----------------------+-------------------+
|    1 | ["hello","starrocks"] | [[1,2],[3,4]]     |
|    2 | ["hello","spark"]     | [[5,6,7],[8,9,10]] |
+------+----------------------+-------------------+
2 rows in set (0.01 sec)
```

Load data in bulk using Spark Load

This load uses external Apache Spark™ resources to pre-process imported data, which improves import performance and saves compute resources. It is mainly used for **initial migration** and **large data import** into StarRocks (data volume up to TB level).

Spark load is an **asynchronous** import method that requires users to create Spark-type import jobs via the MySQL protocol and view the import results using `SHOW LOAD`.

> **NOTICE**
>
> Spark Load can not be used to load data into a Primary Key table.
>
> In versions earlier than v2.5.9, the bucketing column of a StarRocks table cannot be of DATE, DATETIME, or DECIMAL type when Spark Load is used to load data into this table.

Terminology explanation

- **Spark ETL**: Mainly responsible for ETL of data in the import process, including global dictionary construction (BITMAP type), partitioning, sorting, aggregation, etc.
- **Broker**: Broker is an independent stateless process. It encapsulates the file system interface and provides StarRocks with the ability to read files from remote storage systems.
- **Global Dictionary**: Saves the data structure that maps data from the original value to the encoded value. The original value can be any data type, while the encoded value is an integer. The global dictionary is mainly used in scenarios where exact count distinct is precomputed.

Background information

In StarRocks v2.4 and earlier, Spark Load depends on Broker process to set up connections between your StarRocks cluster and your storage system. When you create a Spark Load job, you need to input `WITH BROKER "<broker_name>"` to specify the Broker you want to use. A Broker is an independent, stateless process that is integrated with a file-system interface. With Broker process, StarRocks can access and read data files that are stored in your storage system, and can use its own computing resources to pre-process and load the data of these data files.

From StarRocks v2.5 onwards, Spark Load no longer needs to depend on Broker process to set up connections between your StarRocks cluster and your storage system. When you create a Spark Load job, you no longer need to specify the Broker, but you still need to retain the `WITH BROKER` keyword.

> **NOTE**
>
> Loading without Broker process may not work in certain circumstances, such as when you have multiple HDFS clusters or multiple Kerberos users. In this situation, you can still load data by using Broker process.

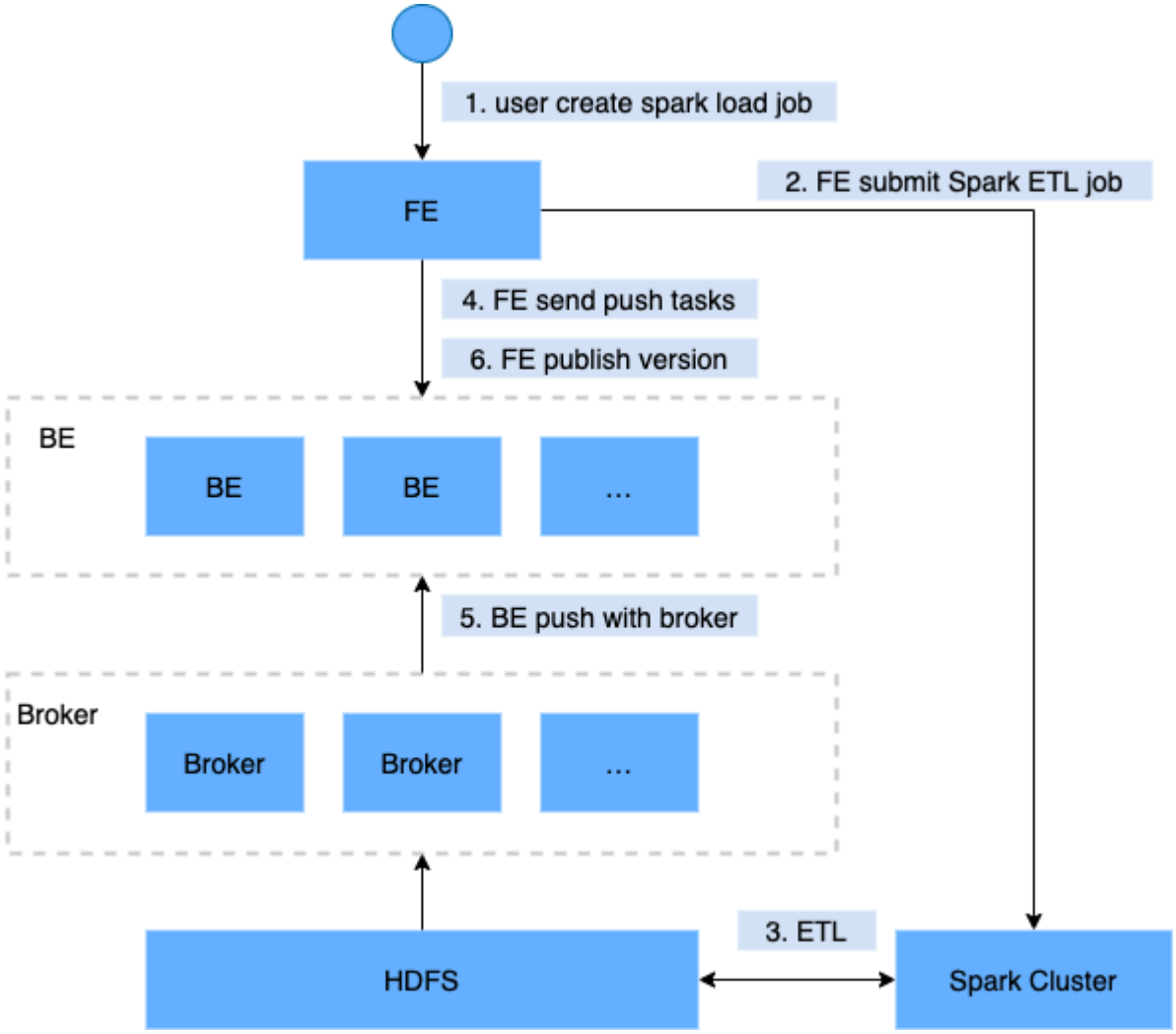
Fundamentals

The user submits a Spark type import job through the MySQL client;the FE records the metadata and returns the submission result.

The execution of the spark load task is divided into the following main phases.

1. The user submits the spark load job to the FE.
2. The FE schedules the submission of the ETL task to the Apache Spark™ cluster for execution.
3. The Apache Spark™ cluster executes the ETL task that includes global dictionary construction (BITMAP type), partitioning, sorting, aggregation, etc.
4. After the ETL task is completed, the FE gets the data path of each preprocessed slice and schedules the relevant BE to execute the Push task.
5. The BE reads data through Broker process from HDFS and converts it into StarRocks storage format.
   > If you choose not to use Broker process, the BE reads data from HDFS directly.
6. The FE schedules the effective version and completes the import job.

The following diagram illustrates the main flow of spark load.



Global Dictionary

Applicable Scenarios

Currently, the BITMAP column in StarRocks is implemented using the Roaringbitmap, which only has integer to be the input data type. So if you want to implement precomputation for the BITMAP column in the import process, then you need to convert the input data type to integer.

In the existing import process of StarRocks, the data structure of the global dictionary is implemented based on the Hive table, which saves the mapping from the original value to the encoded value.

Build Process

1. Read the data from the upstream data source and generate a temporary Hive table, named `hive-table`.
2. Extract the values of the de-emphasized fields of `hive-table` to generate a new Hive table named `distinct-value-table`.
3. Create a new global dictionary table named `dict-table` with one column for the original values and one column for the encoded values.
4. Left join between `distinct-value-table` and `dict-table`, and then use the window function to encode this set. Finally both the original value and the encoded value of the de-duplicated column are written back to `dict-table`.
5. Join between `dict-table` and `hive-table` to finish the job of replacing the original value in `hive-table` with the integer encoded value.
6. `hive-table` will be read by the next time data pre-processing, and then imported into StarRocks after calculation.

Data Pre-processing

The basic process of data pre-processing is as follows:

1. Read data from the upstream data source (HDFS file or Hive table).
2. Complete field mapping and calculation for the read data, then generate `bucket-id` based on the partition information.
3. Generate RollupTree based on the Rollup metadata of StarRocks table.
4. Iterate through the RollupTree and perform hierarchical aggregation operations. The Rollup of the next hierarchy can be calculated from the Rollup of the previous hierarchy.
5. Each time the aggregation calculation is completed, the data is bucketed according to `bucket-id` and then written to HDFS.
6. The subsequent Broker process will pull the files from HDFS and import them into the StarRocks BE node.

Basic Operations

Prerequisites

If you continue to load data through Broker process, you must ensure that Broker process are deployed in your StarRocks cluster.

You can use the SHOW BROKER statement to check for Broker that are deployed in your StarRocks cluster. If no Broker are deployed, you must deploy Broker by following the instructions provided in Deploy Broker.

Configuring ETL Clusters

Apache Spark™ is used as an external computational resource in StarRocks for ETL work. There may be other external resources added to StarRocks, such as Spark/GPU for query, HDFS/S3 for external storage, MapReduce for ETL, etc. Therefore, we introduce `Resource Management` to manage these external resources used by StarRocks.

Before submitting a Apache Spark™ import job, configure the Apache Spark™ cluster for performing ETL tasks. The syntax for operation is as follows:

```
-- create Apache Spark™ resource
CREATE EXTERNAL RESOURCE resource_name
PROPERTIES
(
 type = spark,
 spark_conf_key = spark_conf_value,
 working_dir = path,
 broker = broker_name,
 broker.property_key = property_value
);

-- drop Apache Spark™ resource
DROP RESOURCE resource_name;

-- show resources
SHOW RESOURCES
SHOW PROC "/resources";

-- privileges
GRANT USAGE_PRIV ON RESOURCE resource_name TO user_identityGRANT
USAGE_PRIV ON RESOURCE resource_name TO ROLE role_name;
REVOKE USAGE_PRIV ON RESOURCE resource_name FROM user_identityREVOKE
USAGE_PRIV ON RESOURCE resource_name FROM ROLE role_name;
```

- Create resource

For example:

```
-- yarn cluster mode
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/starrocks",
    "broker" = "broker0",
    "broker.username" = "user0",
    "broker.password" = "password0"
);

-- yarn HA cluster mode
CREATE EXTERNAL RESOURCE "spark1"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.submit.deployMode" = "cluster",
    "spark.hadoop.yarn.resourcemanager.ha.enabled" = "true",
    "spark.hadoop.yarn.resourcemanager.ha.rm-ids" = "rm1,rm2",
    "spark.hadoop.yarn.resourcemanager.hostname.rm1" = "host1",
    "spark.hadoop.yarn.resourcemanager.hostname.rm2" = "host2",
    "spark.hadoop.fs.defaultFS" = "hdfs://127.0.0.1:10000",
    "working_dir" = "hdfs://127.0.0.1:10000/tmp/starrocks",
    "broker" = "broker1"
);
```

`resource-name` is the name of the Apache Spark™ resource configured in StarRocks.

`PROPERTIES` are parameters relating to the Apache Spark™ resource, as follows:

- **type**: Resource type, required, currently only supports Spark.
- **spark** Related parameters are as follows.
- `spark.master`: Required, currently supports yarn.

- `spark.submit.deployMode`: The deployment mode of the Apache Spark™ program, required, currently supports both cluster and client.
- `spark.hadoop.fs.defaultFS`: Required if master is yarn.
- Parameters related to yarn resource manager, required if master is yarn.
- Single point resource manager required
- `spark.hadoop.yarn.resourcemanager.address`: Address of the single point resource manager.
- HA resource manager needs to be configured, where either hostname or address is configured.
- `spark.hadoop.yarn.resourcemanager.ha.enabled`: Enable the resource manager HA, set to true.
- `spark.hadoop.yarn.resourcemanager.ha.rm-ids`: list of resource manager logical ids.
- `spark.hadoop.yarn.resourcemanager.hostname.rm-id`: For each rm-id, specify the hostname corresponding to the resource manager.
- `spark.hadoop.yarn.resourcemanager.address.rm-id`: For each rm-id, specify `host:port` for the client to submit jobs to.
- Other parameters are optional, refer to Spark Configuration
- **working_dir**: The directory used by ETL. Required if Apache Spark™ is used as an ETL resource. For example: `hdfs://host:port/tmp/starrocks`.
- **broker**: Broker name. Required if Apache Spark™ is used as an ETL resource. You need to use the `ALTER SYSTEM ADD BROKER` command to complete the configuration in advance.
- `broker.property_key`: Information (e.g.authentication information) to be specified when Broker process reads the intermediate file generated by the ETL.

Precaution:

The above is a description of parameters for loading through Broker process. If you intend to load data without Broker process, the following should be noted.

- You do not need to specify `broker`.
- If you need to configure user authentication, and HA for NameNode nodes, you need to configure the parameters in the hdfs-site.xml file in the HDFS cluster, see broker_properties for descriptions of parameters. and you need to move the **hdfs-site.xml** file under **$FE_HOME/conf** for each FE and **$BE_HOME/conf** for each BE.

> Note
>
> If the HDFS file can only be accessed by a specific user, you still need to specify the HDFS username in `broker.name` and the user password in `broker.password`.

- View resources

Regular accounts can only view resources to which they have `USAGE-PRIV` access. The root and admin accounts can view all resources.

- Resource Permissions

Resource permissions are managed through `GRANT REVOKE`, which currently only supports `USAGE-PRIV` permissions. You can give `USAGE-PRIV` permissions to a user or a role.

```
-- Grant access to spark0 resources to user0
GRANT USAGE_PRIV ON RESOURCE "spark0" TO "user0"@"%";

-- Grant access to spark0 resources to role0
GRANT USAGE_PRIV ON RESOURCE "spark0" TO ROLE "role0";

-- Grant access to all resources to user0
GRANT USAGE_PRIV ON RESOURCE* TO "user0"@"%";

-- Grant access to all resources to role0
GRANT USAGE_PRIV ON RESOURCE* TO ROLE "role0";

-- Revoke the use privileges of spark0 resources from user user0
REVOKE USAGE_PRIV ON RESOURCE "spark0" FROM "user0"@"%";
```

Configuring Spark Client

Configure the Spark client for FE so that the latter can submit Spark tasks by executing the `spark-submit` command. It is recommended to use the official version of Spark2 2.4.5 or above spark download address. After downloading, please use the following steps to complete the configuration.

- Configure `SPARK-HOME`

Place the Spark client in a directory on the same machine as the FE, and configure `spark_home_default_dir` in the FE configuration file to this directory, which by default is

the `lib/spark2x` path in the FE root directory, and cannot be empty.

- **Configure SPARK dependency package**

To configure the dependency package, zip and archive all jar files in the jars folder under the Spark client, and configure the `spark_resource_path` item in the FE configuration to this zip file. If this configuration is empty, the FE will try to find the `lib/spark2x/jars/spark-2x.zip` file in the FE root directory. If the FE fails to find it, it will report an error.

When the spark load job is submitted, the archived dependency files will be uploaded to the remote repository. The default repository path is under the `working_dir/{cluster_id}` directory named with `--spark-repository--{resource-name}`, which means that a resource in the cluster corresponds to a remote repository. The directory structure is referenced as follows:

```
---spark-repository--spark0/

   |---archive-1.0.0/

   |        |\---lib-990325d2c0d1d5e45bf675e54e44fb16-spark-dpp-
1.0.0\-jar-with-dependencies.jar

   |        |\---lib-7670c29daf535efe3c9b923f778f61fc-spark-2x.zip

   |---archive-1.1.0/

   |        |\---lib-64d5696f99c379af2bee28c1c84271d5-spark-dpp-
1.1.0\-jar-with-dependencies.jar

   |        |\---lib-1bbb74bb6b264a270bc7fca3e964160f-spark-2x.zip

   |---archive-1.2.0/

   |        |-...
```

In addition to the spark dependencies (named `spark-2x.zip` by default), the FE also uploads the DPP dependencies to the remote repository. If all the dependencies submitted by the spark load already exist in the remote repository, then there is no need to upload the dependencies again, saving the time of repeatedly uploading a large number of files each time.

Configuring YARN Client

Configure the yarn client for FE so that the FE can execute yarn commands to get the status of the running application or kill it.It is recommended to use the official version of H adoop2 2.5.2 or above (hadoop download address). After downloading, please use the following steps to complete the configuration:

- **Configure the YARN executable path**

Place the downloaded yarn client in a directory on the same machine as the FE, and configure the `yarn_client_path` item in the FE configuration file to the binary executable file of yarn, which by default is the `lib/yarn-client/hadoop/bin/yarn` path in the FE root directory.

- **Configure the path to the configuration file needed to generate YARN (optional)**

When the FE goes through the yarn client to get the status of the application, or to kill the application, by default StarRocks generates the configuration file required to execute the yarn command in the `lib/yarn-config` path of the FE root directory This path can be modified by configuring the `yarn_config_dir` entry in the FE configuration file, which currently includes `core-site.xml` and `yarn-site.xml`.

Create Import Job

Syntax:

```
LOAD LABEL load_label
    (data_desc, ...)
WITH RESOURCE resource_name
[resource_properties]
[PROPERTIES (key1=value1, ... )]

* load_label:
    db_name.label_name

* data_desc:
    DATA INFILE ('file_path', ...)
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [COLUMNS TERMINATED BY separator ]
    [(col1, ...)]
```

```
    [COLUMNS FROM PATH AS (col2, ...)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [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]

* resource_properties:
 (key2=value2, ...)
```

Example 1: The case where the upstream data source is HDFS

```
LOAD LABEL db1.label1
(
    DATA INFILE("hdfs://abc.com:8888/user/starrocks/test/ml/file1")
    INTO TABLE tbl1
    COLUMNS TERMINATED BY ","
    (tmp_c1,tmp_c2)
    SET
    (
        id=tmp_c2,
        name=tmp_c1
    ),
    DATA INFILE("hdfs://abc.com:8888/user/starrocks/test/ml/file2")
    INTO TABLE tbl2
    COLUMNS TERMINATED BY ","
    (col1, col2)
    where col1 > 1
)
WITH RESOURCE 'spark0'
(
    "spark.executor.memory" = "2g",
    "spark.shuffle.compress" = "true"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Example 2: The case where the upstream data source is Hive.

- Step 1: Create a new hive resource

```
CREATE EXTERNAL RESOURCE hive0
properties
(
    "type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:8080"
);
```

- Step 2: Create a new hive external table

```
CREATE EXTERNAL TABLE hive_t1
(
    k1 INT,
    K2 SMALLINT,
    k3 varchar(50),
    uuid varchar(100)
)
ENGINE=hive
properties
(
    "resource" = "hive0",
    "database" = "tmp",
    "table" = "t1"
);
```

- Step 3: Submit the load command, requiring that the columns in the imported StarRocks table exist in the hive external table.

```
LOAD LABEL db1.label1
(
    DATA FROM TABLE hive_t1
    INTO TABLE tbl1
    SET
    (
        uuid=bitmap_dict(uuid)
    )
)
WITH RESOURCE 'spark0'
(
    "spark.executor.memory" = "2g",
    "spark.shuffle.compress" = "true"
```

```
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Introduction to the parameters in the Spark load:

- **Label**

Label of the import job. Each import job has a Label that is unique within the database, following the same rules as broker load.

- **Data description class parameters**

Currently, supported data sources are CSV and Hive table. Other rules are the same as broker load.

- **Import Job Parameters**

Import job parameters refer to the parameters belonging to the `opt_properties` section of the import statement. These parameters are applicable to the entire import job. The rules are the same as broker load.

- **Spark Resource Parameters**

Spark resources need to be configured into StarRocks in advance and users need to be given USAGE-PRIV permissions before they can apply the resources to Spark load. Spark resource parameters can be set when the user has a temporary need, such as adding resources for a job and modifying Spark configs. The setting only takes effect on this job and does not affect the existing configurations in the StarRocks cluster.

```
WITH RESOURCE 'spark0'
(
    "spark.driver.memory" = "1g",
    "spark.executor.memory" = "3g"
)
```

- **Import when the data source is Hive**

Currently, to use a Hive table in the import process, you need to create an external table of the `Hive` type and then specify its name when submitting the import command.

- **Import process to build a global dictionary**

In the load command, you can specify the required fields for building the global dictionary in the following format: `StarRocks field name=bitmap_dict(hive table field name)` Note that currently **the global dictionary is only supported when the upstream data source is a Hive table**.

- **Load binary type data**

Since v2.5.17, Spark Load supports the bitmap_from_binary function, which can convert binary data into bitmap data. If the column type of the Hive table or HDFS file is binary and the corresponding column in the StarRocks table is a bitmap-type aggregate column, you can specify the fields in the load command in the following format, `StarRocks field name=bitmap_from_binary(Hive table field name)`. This eliminates the need for building a global dictionary.

Viewing Import Jobs

The Spark load import is asynchronous, as is the broker load. The user must record the label of the import job and use it in the `SHOW LOAD` command to view the import results. The command to view the import is common to all import methods. The example is as follows.

Refer to Broker Load for a detailed explanation of returned parameters.The differences are as follows.

```
mysql> show load order by createtime desc limit 1\G
*************************** 1. row ***************************
  JobId: 76391
  Label: label1
  State: FINISHED
 Progress: ETL:100%; LOAD:100%
  Type: SPARK
 EtlInfo: unselected.rows=4; dpp.abnorm.ALL=15; dpp.norm.ALL=28133376
 TaskInfo: cluster:cluster0; timeout(s):10800; max_filter_ratio:5.0E-5
 ErrorMsg: N/A
 CreateTime: 2019-07-27 11:46:42
 EtlStartTime: 2019-07-27 11:46:44
```

```
 EtlFinishTime: 2019-07-27 11:49:44
 LoadStartTime: 2019-07-27 11:49:44
LoadFinishTime: 2019-07-27 11:50:16
  URL: http://1.1.1.1:8089/proxy/application_1586619723848_0035/
 JobDetails:
{"ScannedRows":28133395,"TaskNumber":1,"FileNumber":1,"FileSize":200000
```

- **State**

The current stage of the imported job. PENDING: The job is committed. ETL: Spark ETL is committed. LOADING: The FE schedule an BE to execute push operation. FINISHED: The push is completed and the version is effective.

There are two final stages of the import job – `CANCELLED` and `FINISHED`, both indicating the load job is completed. `CANCELLED` indicates import failure and `FINISHED` indicates import success.

- **Progress**

Description of the import job progress. There are two types of progress –ETL and LOAD, which correspond to the two phases of the import process, ETL and LOADING.

- The range of progress for LOAD is 0~100%.

`LOAD progress = the number of currently completed tablets of all replications imports / the total number of tablets of this import job * 100%`.

- If all tables have been imported, the LOAD progress is 99%, and changes to 100% when the import enters the final validation phase.

- The import progress is not linear. If there is no change in progress for a period of time, it does not mean that the import is not executing.

- **Type**

The type of the import job. SPARK for spark load.

- **CreateTime/EtlStartTime/EtlFinishTime/LoadStartTime/LoadFinishTime**

These values represent the time when the import was created, when the ETL phase started, when the ETL phase completed, when the LOADING phase started, and when the entire

import job was completed.

- **JobDetails**

Displays the detailed running status of the job, including the number of imported files, total size (in bytes), number of subtasks, number of raw rows being processed, etc. For example:

```
{"ScannedRows":139264,"TaskNumber":1,"FileNumber":1,"FileSize":94075406
```

- **URL**

You can copy the input to your browser to access the web interface of the corresponding application.

View Apache Spark™ Launcher commit logs

Sometimes users need to view the detailed logs generated during a Apache Spark™ job commit. By default, the logs are saved in the path `log/spark_launcher_log` in the FE root directory named as `spark-launcher-{load-job-id}-{label}.log`. The logs are saved in this directory for a period of time and will be erased when the import information in FE metadata is cleaned up. The default retention time is 3 days.

Cancel Import

When the Spark load job status is not `CANCELLED` or `FINISHED`, it can be cancelled manually by the user by specifying the Label of the import job.

Related System Configurations

FE Configuration: The following configuration is the system-level configuration of Spark load, which applies to all Spark load import jobs. The configuration values can be adjusted mainly by modifying `fe.conf`.

- enable-spark-load: Enable Spark load and resource creation with a default value of false.

- spark-load-default-timeout-second: The default timeout for the job is 259200 seconds (3 days).
- spark-home-default-dir: The Spark client path (`fe/lib/spark2x`).
- spark-resource-path: The path to the packaged S park dependency file (empty by default).
- spark-launcher-log-dir: The directory where the commit log of the Spark client is stored (`fe/log/spark-launcher-log`).
- yarn-client-path: The path to the yarn binary executable (`fe/lib/yarn-client/hadoop/bin/yarn`).
- yarn-config-dir: Yarn's configuration file path (`fe/lib/yarn-config`).

Best Practices

The most suitable scenario for using Spark load is when the raw data is in the file system (HDFS) and the data volume is in the tens of GB to TB level. Use Stream Load or Broker Load for smaller data volumes.

For the full spark load import example, refer to the demo on github: https://github.com/StarRocks/demo/blob/master/docs/03_sparkLoad2StarRocks.md

FAQs

- `Error: When running with master 'yarn' either HADOOP-CONF-DIR or YARN-CONF-DIR must be set in the environment.`

Using Spark Load without configuring the `HADOOP-CONF-DIR` environment variable in `spark-env.sh` of the Spark client.

- `Error: Cannot run program "xxx/bin/spark-submit": error=2, No such file or directory`

The `spark_home_default_dir` configuration item does not specify the Spark client root directory when using Spark Load.

- `Error: File xxx/jars/spark-2x.zip does not exist.`

The `spark-resource-path` configuration item does not point to the packed zip file when using Spark load.

- `Error: yarn client does not exist in path: xxx/yarn-client/hadoop/bin/yarn`

The yarn-client-path configuration item does not specify the yarn executable when using Spark load.

- `ERROR: Cannot execute hadoop-yarn/bin/... /libexec/yarn-config.sh`

When using Hadoop with CDH, you need to configure the `HADOOP_LIBEXEC_DIR` environment variable. Since `hadoop-yarn` and hadoop directories are different, the default `libexec` directory will look for `hadoop-yarn/bin/... /libexec`, while `libexec` is in the hadoop directory. The ```yarn application status`` command to get the Spark task status reported an error causing the import job to fail.

Load data using INSERT

This topic describes how to load data into StarRocks by using a SQL statement - INSERT.

Similar to MySQL and many other database management systems, StarRocks supports loading data to an internal table with INSERT. You can insert one or more rows directly with the VALUES clause to test a function or a DEMO. You can also insert data defined by the results of a query into an internal table from an external table.

StarRocks v2.4 further supports overwriting data into a table by using INSERT OVERWRITE. The INSERT OVERWRITE statement integrates the following operations to implement the overwriting function:

1. Creates temporary partitions according to the partitions that store the original data.
2. Inserts data into the temporary partitions.
3. Swaps the original partitions with the temporary partitions.

> **NOTE**
>
> If you need to verify the data before overwriting it, instead of using INSERT OVERWRITE, you can follow the above procedures to overwrite your data and verify it before swapping the partitions.

Precautions

- Because INSERT transaction is synchronous, you can cancel it only by pressing the **Ctrl** and **C** keys from your MySQL client.
- As for the current version of StarRocks, the INSERT transaction fails by default if the data of any rows does not comply with the schema of the table. For example, the INSERT transaction fails if the length of a field in any row exceeds the length limit for the mapping field in the table. You can set the session variable `enable_insert_strict` to `false` to allow the transaction to continue by filtering out the rows that mismatch the table.
- If you execute the INSERT statement frequently to load small batches of data into StarRocks, excessive data versions are generated. It severely affects query

performance. We recommend that, in production, you should not load data with the INSERT command too often or use it as a routine for data loading on a daily basis. If your application or analytic scenario demand solutions to loading streaming data or small data batches separately, we recommend you use Apache Kafka® as your data source and load the data via Routine Load.

- If you execute the INSERT OVERWRITE statement, StarRocks creates temporary partitions for the partitions which store the original data, inserts new data into the temporary partitions, and swaps the original partitions with the temporary partitions. All these operations are executed in the FE Leader node. Hence, if the FE Leader node crashes while executing INSERT OVERWRITE command, the whole load transaction will fail, and the temporary partitions will be truncated.

Preparation

Create a database named `load_test`, and create a table `insert_wiki_edit` as the destination table and a table `source_wiki_edit` as the source table.

> **NOTE**
>
> Examples demonstrated in this topic are based on the table `insert_wiki_edit` and the table `source_wiki_edit`. If you prefer working with your own tables and data, you can skip the preparation and move on to the next step.

```
CREATE DATABASE IF NOT EXISTS load_test;
USE load_test;
CREATE TABLE insert_wiki_edit
(
    event_time DATETIME,
    channel VARCHAR(32) DEFAULT '',
    user VARCHAR(128) DEFAULT '',
    is_anonymous TINYINT DEFAULT '0',
    is_minor TINYINT DEFAULT '0',
    is_new TINYINT DEFAULT '0',
    is_robot TINYINT DEFAULT '0',
    is_unpatrolled TINYINT DEFAULT '0',
    delta INT DEFAULT '0',
    added INT DEFAULT '0',
    deleted INT DEFAULT '0'
)
DUPLICATE KEY
(
```

```
    event_time,
    channel,
    user,
    is_anonymous,
    is_minor,
    is_new,
    is_robot,
    is_unpatrolled
)
PARTITION BY RANGE(event_time)
(
    PARTITION p06 VALUES LESS THAN ('2015-09-12 06:00:00'),
    PARTITION p12 VALUES LESS THAN ('2015-09-12 12:00:00'),
    PARTITION p18 VALUES LESS THAN ('2015-09-12 18:00:00'),
    PARTITION p24 VALUES LESS THAN ('2015-09-13 00:00:00')
)
DISTRIBUTED BY HASH(user) BUCKETS 3;

CREATE TABLE source_wiki_edit
(
    event_time DATETIME,
    channel VARCHAR(32) DEFAULT '',
    user VARCHAR(128) DEFAULT '',
    is_anonymous TINYINT DEFAULT '0',
    is_minor TINYINT DEFAULT '0',
    is_new TINYINT DEFAULT '0',
    is_robot TINYINT DEFAULT '0',
    is_unpatrolled TINYINT DEFAULT '0',
    delta INT DEFAULT '0',
    added INT DEFAULT '0',
    deleted INT DEFAULT '0'
)
DUPLICATE KEY
(
    event_time,
    channel,
    user,
    is_anonymous,
    is_minor,
    is_new,
    is_robot,
    is_unpatrolled
)
PARTITION BY RANGE(event_time)
(
    PARTITION p06 VALUES LESS THAN ('2015-09-12 06:00:00'),
```

```
    PARTITION p12 VALUES LESS THAN ('2015-09-12 12:00:00'),
    PARTITION p18 VALUES LESS THAN ('2015-09-12 18:00:00'),
    PARTITION p24 VALUES LESS THAN ('2015-09-13 00:00:00')
)
DISTRIBUTED BY HASH(user) BUCKETS 3;
```

Insert data via INSERT INTO VALUES

You can append one or more rows to a specific table by using INSERT INTO VALUES command. Multiple rows are separated by comma (,). For detailed instructions and parameter references, see SQL Reference - INSERT.

> **CAUTION**
>
> Inserting data via INSERT INTO VALUES merely applies to the situation when you need to verify a DEMO with a small dataset. It is not recommended for a massive testing or production environment. To load mass data into StarRocks, see Ingestion Overview for other options that suit your scenarios.

The following example inserts two rows into the data source table `source_wiki_edit` with the label `insert_load_wikipedia`. Label is the unique identification label for each data load transaction within the database.

```
INSERT INTO source_wiki_edit
WITH LABEL insert_load_wikipedia
VALUES
    ("2015-09-12
00:00:00","#en.wikipedia","AustinFF",0,0,0,0,0,21,5,0),
    ("2015-09-12
00:00:00","#ca.wikipedia","helloSR",0,1,0,1,0,3,23,0);
```

Insert data via INSERT INTO SELECT

You can load the result of a query on a data source table into the target table via INSERT INTO SELECT command. INSERT INTO SELECT command performs ETL operations on the data from the data source table, and loads the data into an internal table in StarRocks. The data source can be one or more internal or external tables. The target table MUST be an

internal table in StarRocks. For detailed instructions and parameter references, see SQL Reference - INSERT.

> **NOTE**
>
> Inserting data from an external table is identical to inserting data from an internal table. For simplicity, we only demonstrate how to insert data from an internal table in the following examples.

- The following example inserts the data from the source table to the target table `insert_wiki_edit`.

```
INSERT INTO insert_wiki_edit
WITH LABEL insert_load_wikipedia_1
SELECT * FROM source_wiki_edit;
```

- The following example inserts the data from the source table to the `p06` and `p12` partitions of the target table `insert_wiki_edit`. If no partition is specified, the data will be inserted into all partitions. Otherwise, the data will be inserted only into the specified partition(s).

```
INSERT INTO insert_wiki_edit PARTITION(p06, p12)
WITH LABEL insert_load_wikipedia_2
SELECT * FROM source_wiki_edit;
```

Query the target table to make sure there is data in them.

```
MySQL > select * from insert_wiki_edit;
+---------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+---------
+
| event_time          | channel      | user     | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added |
deleted |
+---------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+---------
+
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |           0 |
0 |      0 |        0 |             0 |    21 |     5 |       0 |
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |           0 |
1 |      0 |        1 |             0 |     3 |    23 |       0 |
```

```
+--------------------+--------------+---------+-------------+----
------+--------+---------+---------------+-------+-------+--------
+
2 rows in set (0.00 sec)
```

If you truncate the `p06` and `p12` partitions, the data will not be returned in a query.

```
MySQL > TRUNCATE TABLE insert_wiki_edit PARTITION(p06, p12);
Query OK, 0 rows affected (0.01 sec)

MySQL > select * from insert_wiki_edit;
Empty set (0.00 sec)
```

- The following example inserts the `event_time` and `channel` columns from the source table to the target table `insert_wiki_edit`. Default values are used in the columns that are not specified here.

```
INSERT INTO insert_wiki_edit
WITH LABEL insert_load_wikipedia_3
(
    event_time,
    channel
)
SELECT event_time, channel FROM source_wiki_edit;
```

Overwrite data via INSERT OVERWRITE VALUES

You can overwrite a specific table with one or more rows by using INSERT OVERWRITE VALUES command. Multiple rows are separated by comma (,). For detailed instructions and parameter references, see SQL Reference - INSERT.

> **CAUTION**
>
> Overwriting data via INSERT OVERWRITE VALUES merely applies to the situation when you need to verify a DEMO with a small dataset. It is not recommended for a massive testing or production environment. To load mass data into StarRocks, see Ingestion Overview for other options that suit your scenarios.

Query the source table and the target table to make sure there is data in them.

```
MySQL > SELECT * FROM source_wiki_edit;
+--------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
| event_time          | channel       | user     | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added |
deleted |
+--------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |
1 |      0 |        1 |             0 |     3 |    23 |       0 |
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |
0 |      0 |        0 |             0 |    21 |     5 |       0 |
+--------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
2 rows in set (0.02 sec)

MySQL > SELECT * FROM insert_wiki_edit;
+--------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
| event_time          | channel       | user     | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added |
deleted |
+--------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |
1 |      0 |        1 |             0 |     3 |    23 |       0 |
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |
0 |      0 |        0 |             0 |    21 |     5 |       0 |
+--------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
2 rows in set (0.01 sec)
```

The following example overwrites the source table `source_wiki_edit` with two new rows.

```
INSERT OVERWRITE source_wiki_edit
WITH LABEL insert_load_wikipedia_ow
```

```
VALUES
    ("2015-09-12
00:00:00","#cn.wikipedia","GELongstreet",0,0,0,0,0,36,36,0),
    ("2015-09-12
00:00:00","#fr.wikipedia","PereBot",0,1,0,1,0,17,17,0);
```

Overwrite data via INSERT OVERWRITE SELECT

You can overwrite a table with the result of a query on a data source table via INSERT OVERWRITE SELECT command. INSERT OVERWRITE SELECT statement performs ETL operations on the data from one or more internal or external tables, and overwrites an internal table with the data For detailed instructions and parameter references, see SQL Reference - INSERT.

> **NOTE**
>
> Loading data from an external table is identical to loading data from an internal table. For simplicity, we only demonstrate how to overwrite the target table with the data from an internal table in the following examples.

Query the source table and the target table to make sure that they hold different rows of data.

```
MySQL > SELECT * FROM source_wiki_edit;
+---------------------+--------------+-------------+-------------
+---------+--------+---------+---------------+-------+-------+----
-----+
| event_time          | channel      | user        | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added |
deleted |
+---------------------+--------------+-------------+-------------
+---------+--------+---------+---------------+-------+-------+----
-----+
| 2015-09-12 00:00:00 | #cn.wikipedia | GELongstreet |            0 |
0 |      0 |        0 |             0 |    36 |    36 |       0 |
| 2015-09-12 00:00:00 | #fr.wikipedia | PereBot      |            0 |
1 |      0 |        1 |             0 |    17 |    17 |       0 |
+---------------------+--------------+-------------+-------------
+---------+--------+---------+---------------+-------+-------+----
-----+
```

```
2 rows in set (0.02 sec)

MySQL > SELECT * FROM insert_wiki_edit;
+---------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
| event_time          | channel      | user     | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added |
deleted |
+---------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |
0 |      0 |        0 |             0 |    21 |     5 |       0 |
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |
1 |      0 |        1 |             0 |     3 |    23 |       0 |
+---------------------+--------------+----------+-------------+----
------+--------+----------+---------------+-------+-------+--------
+
2 rows in set (0.01 sec)
```

- The following example overwrites the table `insert_wiki_edit` with the data from the source table.

```
INSERT OVERWRITE insert_wiki_edit
WITH LABEL insert_load_wikipedia_ow_1
SELECT * FROM source_wiki_edit;
```

- The following example overwrites the `p06` and `p12` partitions of the table `insert_wiki_edit` with the data from the source table.

```
INSERT OVERWRITE insert_wiki_edit PARTITION(p06, p12)
WITH LABEL insert_load_wikipedia_ow_2
SELECT * FROM source_wiki_edit;
```

Query the target table to make sure there is data in them.

```
MySQL > select * from insert_wiki_edit;
+---------------------+--------------+-------------+-------------
+----------+--------+----------+---------------+-------+-------+----
-----+
```

```
| event_time          | channel       | user         | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added |
deleted |
+---------------------+---------------+--------------+--------------
+----------+--------+----------+----------------+-------+-------+----
-----+
| 2015-09-12 00:00:00 | #fr.wikipedia | PereBot      |            0 |
1 |      0 |        1 |              0 |    17 |    17 |       0 |
| 2015-09-12 00:00:00 | #cn.wikipedia | GELongstreet |            0 |
0 |      0 |        0 |              0 |    36 |    36 |       0 |
+---------------------+---------------+--------------+--------------
+----------+--------+----------+----------------+-------+-------+----
-----+
2 rows in set (0.01 sec)
```

If you truncate the `p06` and `p12` partitions, the data will not be returned in a query.

```
MySQL > TRUNCATE TABLE insert_wiki_edit PARTITION(p06, p12);
Query OK, 0 rows affected (0.01 sec)

MySQL > select * from insert_wiki_edit;
Empty set (0.00 sec)
```

- The following example overwrites the target table `insert_wiki_edit` with the `event_time` and `channel` columns from the source table. The default value is assigned to the columns into which no data is overwritten.

```
INSERT OVERWRITE insert_wiki_edit
WITH LABEL insert_load_wikipedia_ow_3
(
    event_time,
    channel
)
SELECT event_time, channel FROM source_wiki_edit;
```

Check the INSERT transaction status

Check via the result

The INSERT transaction returns different status in accordance with the result of the transaction.

- **Transaction succeeds**

StarRocks returns the following if the transaction succeeds:

```
Query OK, 2 rows affected (0.05 sec)
{'label':'insert_load_wikipedia', 'status':'VISIBLE', 'txnId':'1006'}
```

- **Transaction fails**

If all rows of data fail to be loaded into the target table, the INSERT transaction fails. StarRocks returns the following if the transaction fails:

```
ERROR 1064 (HY000): Insert has filtered data in strict mode,
tracking_url=http://x.x.x.x:yyyy/api/_load_error_log?
file=error_log_9f0a4fd0b64e11ec_906bbede076e9d08
```

You can locate the problem by checking the log with `tracking_url`.

Check via SHOW LOAD

You can check the INSERT transaction status by using SHOW LOAD command.

The following example checks the status of the transaction with label `insert_load_wikipedia`.

```
SHOW LOAD WHERE label="insert_load_wikipedia"\G
```

The return is as follows:

```
*************************** 1. row ***************************
         JobId: 10278
         Label: insert_load_wikipedia
         State: FINISHED
      Progress: ETL:100%; LOAD:100%
          Type: INSERT
      Priority: NORMAL
```

```
      ScanRows: 0
  FilteredRows: 0
UnselectedRows: 0
      SinkRows: 2
       EtlInfo: NULL
      TaskInfo: resource:N/A; timeout(s):300; max_filter_ratio:0.0
      ErrorMsg: NULL
    CreateTime: 2023-06-12 18:31:07
  EtlStartTime: 2023-06-12 18:31:07
 EtlFinishTime: 2023-06-12 18:31:07
 LoadStartTime: 2023-06-12 18:31:07
LoadFinishTime: 2023-06-12 18:31:08
   TrackingSQL:
    JobDetails: {"All backends":{"3d96e21a-090c-11ee-9083-00163e0e2cf9"
[10142]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":175,"Inte
backends":{"3d96e21a-090c-11ee-9083-00163e0e2cf9":[]}}
1 row in set (0.00 sec)
```

Check via curl command

You can check the INSERT transaction status by using curl command.

Launch a terminal, and execute the following command:

```
curl --location-trusted -u <username>:<password> \
  http://<fe_address>:<fe_http_port>/api/<db_name>/_load_info?label=
<label_name>
```

The following example checks the status of the transaction with label `insert_load_wikipedia`.

```
curl --location-trusted -u <username>:<password> \
  http://x.x.x.x:8030/api/load_test/_load_info?
label=insert_load_wikipedia
```

> **NOTE**
>
> If you use an account for which no password is set, you need to input only `<username>:`.

The return is as follows:

```
{
   "jobInfo":{
      "dbName":"load_test",
      "tblNames":[
         "source_wiki_edit"
      ],
      "label":"insert_load_wikipedia",
      "state":"FINISHED",
      "failMsg":"",
      "trackingUrl":""
   },
   "status":"OK",
   "msg":"Success"
}
```


Configuration

You can set the following configuration items for INSERT transaction:

- **FE configuration**

FE configuration	Description
insert_load_default_timeout_second	Default timeout for INSERT transaction. Unit: second. If the current INSERT transaction is not completed within the time set by this parameter, it will be canceled by the system and the status will be CANCELLED. As for current version of StarRocks, you can only specify a uniform timeout for all INSERT transactions using this parameter, and you cannot set a different timeout for a specific INSERT transaction. The default is 3600 seconds (1 hour). If the INSERT transaction cannot be completed within the specified time, you can extend the timeout by adjusting this parameter.

- **Session variables**

Session variable	Description
enable_insert_strict	Switch value to control if the INSERT transaction is tolerant of invalid data rows. When it is set to `true`, the transaction fails if any of the data rows is invalid. When it is set to `false`, the transaction succeeds when at least one row of data has been loaded correctly, and the label will be returned. The default is `true`. You can set this variable with `SET enable_insert_strict = {true or false};` command.
query_timeout	Timeout for SQL commands. Unit: second. INSERT, as a SQL command, is also restricted by this session variable. You can set this variable with the `SET query_timeout = xxx;` command.

Load data using Stream Load transaction interface

From v2.4 onwards, StarRocks provides a Stream Load transaction interface to implement two-phase commit (2PC) for transactions that are run to load data from external systems such as Apache Flink® and Apache Kafka®. The Stream Load transaction interface helps improve the performance of highly concurrent stream loads.

This topic describes the Stream Load transaction interface and how to load data into StarRocks by using this interface.

Description

The Stream Load transaction interface supports using an HTTP protocol-compatible tool or language to call API operations. This topic uses curl as an example to explain how to use this interface. This interface provides various features, such as transaction management, data write, transaction pre-commit, transaction deduplication, and transaction timeout management.

> **NOTE**
>
> Stream Load supports CSV and JSON file formats. This method is recommended if you want to load data from a small number of files whose individual sizes do not exceed 10 GB. Stream Load does not support Parquet file format. If you need to load data from Parquet files, use INSERT+files().

Transaction management

The Stream Load transaction interface provides the following API operations, which are used to manage transactions:

- `/api/transaction/begin`: starts a new transaction.

- `/api/transaction/commit`: commits the current transaction to make data changes persistent.
- `/api/transaction/rollback`: rolls back the current transaction to abort data changes.

Transaction pre-commit

The Stream Load transaction interface provides the `/api/transaction/prepare` operation, which is used to pre-commit the current transaction and make data changes temporarily persistent. After you pre-commit a transaction, you can proceed to commit or roll back the transaction. If your StarRocks cluster breaks down after a transaction is pre-committed, you can still proceed to commit the transaction after your StarRocks cluster is restored to normal.

> **NOTE**
>
> After the transaction is pre-committed, do not continue to write data using the transaction. If you continue to write data using the transaction, your write request returns errors.

Data write

The Stream Load transaction interface provides the `/api/transaction/load` operation, which is used to write data. You can call this operation multiple times within one transaction.

Transaction deduplication

The Stream Load transaction interface carries over the labeling mechanism of StarRocks. You can bind a unique label to each transaction to achieve at-most-once guarantees for transactions.

Transaction timeout management

You can use the `stream_load_default_timeout_second` parameter in the configuration file of each FE to specify a default transaction timeout period for that FE.

When you create a transaction, you can use the `timeout` field in the HTTP request header to specify a timeout period for the transaction.

When you create a transaction, you can also use the `idle_transaction_timeout` field in the HTTP request header to specify a timeout period within which the transaction can stay idle. If no data is written within the timeout period, the transaction automatically rolls back.

Benefits

The Stream Load transaction interface brings the following benefits:

- **Exactly-once semantics**

  A transaction is split into two phases, pre-commit and commit, which make it easy to load data across systems. For example, this interface can guarantee exactly-once semantics for data loads from Flink.

- **Improved load performance**

  If you run a load job by using a program, the Stream Load transaction interface allows you to merge multiple mini-batches of data on demand and then send them all at once within one transaction by calling the `/api/transaction/commit` operation. As such, fewer data versions need to be loaded, and load performance is improved.

Limits

The Stream Load transaction interface has the following limits:

- Only **single-database single-table** transactions are supported. Support for **multi-database multi-table** transactions is in development.

- Only **concurrent data writes from one client** are supported. Support for **concurrent data writes from multiple clients** is in development.

- The `/api/transaction/load` operation can be called multiple times within one transaction. In this case, the parameter settings specified for all of the `/api/transaction/load` operations that are called must be the same.

- When you load CSV-formatted data by using the Stream Load transaction interface, make sure that each data record in your data file ends with a row delimiter.

Precautions

- If the `/api/transaction/begin`, `/api/transaction/load`, or `/api/transaction/prepare` operation that you have called returns errors, the transaction fails and is automatically rolled back.
- When calling the `/api/transaction/begin` operation to start a new transaction, you must specify a label. Note that the subsequent `/api/transaction/load`, `/api/transaction/prepare`, and `/api/transaction/commit` operations must use the same label as the `/api/transaction/begin` operation.
- If you the label of a previous transaction to call the `/api/transaction/begin` operation to start a new transaction, the previous transaction will fail and be rolled back.
- The default column separator and row delimiter that StarRocks supports for CSV-formatted data are `\t` and `\n`. If your data file does not use the default column separator or row delimiter, you must use `"column_separator: <column_separator>"` or `"row_delimiter: <row_delimiter>"` to specify the column separator or row delimiter that is actually used in your data file when calling the `/api/transaction/load` operation.

Before you begin

Check privileges

You can load data into StarRocks tables only as a user who has the INSERT privilege on those StarRocks tables. If you do not have the INSERT privilege, follow the instructions provided in GRANT to grant the INSERT privilege to the user that you use to connect to your StarRocks cluster.

Check network configuration

Make sure that the machine on which the data you want to load resides can access the FE and BE nodes of the StarRocks cluster via the `http_port` (default: `8030`) and `be_http_port` (default: `8040`) , respectively.

Basic operations

Prepare sample data

This topic uses CSV-formatted data as an example.

1. In the `/home/disk1/` path of your local file system, create a CSV file named `example1.csv`. The file consists of three columns, which represent the user ID, user name, and user score in sequence.

```
1,Lily,23
2,Rose,23
3,Alice,24
4,Julia,25
```

2. In your StarRocks database `test_db`, create a Primary Key table named `table1`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

```
CREATE TABLE `table1`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10;
```

Start a transaction

Syntax

```
curl --location-trusted -u <username>:<password> -H "label:
<label_name>" \
    -H "Expect:100-continue" \
    -H "db:<database_name>" -H "table:<table_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/begin
```

Example

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" -H "table:table1" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/begin
```

> **NOTE**
>
> For this example, `streamload_txn_example1_table1` is specified as the label of the transaction.

Return result

- If the transaction is successfully started, the following result is returned:

```
{
    "Status": "OK",
    "Message": "",
    "Label": "streamload_txn_example1_table1",
    "TxnId": 9032,
    "BeginTxnTimeMs": 0
}
```

- If the transaction is bound to a duplicate label, the following result is returned:

```
{
    "Status": "LABEL_ALREADY_EXISTS",
    "ExistingJobStatus": "RUNNING",
    "Message": "Label [streamload_txn_example1_table1] has
already been used."
}
```

- If errors other than duplicate label occur, the following result is returned:

```
{
    "Status": "FAILED",
    "Message": ""
}
```

Write data

Syntax

```
curl --location-trusted -u <username>:<password> -H "label:
<label_name>" \
    -H "Expect:100-continue" \
    -H "db:<database_name>" -H "table:<table_name>" \
    -T <file_path> \
    -XPUT http://<fe_host>:<fe_http_port>/api/transaction/load
```

> **NOTE**
>
> When calling the `/api/transaction/load` operation, you must use `<file_path>` to specify the save path of the data file you want to load.

Example

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" -H "table:table1" \
    -T /home/disk1/example1.csv \
    -H "column_separator: ," \
    -XPUT http://<fe_host>:<fe_http_port>/api/transaction/load
```

> **NOTE**
>
> For this example, the column separator used in the data file `example1.csv` is commas (`,`) instead of StarRocks's default column separator (`\t`). Therefore, when calling the `/api/transaction/load` operation, you must use `"column_separator: <column_separator>"` to specify commas (`,`) as the column separator.

Return result

- If the data write is successful, the following result is returned:

```
{
    "TxnId": 1,
    "Seq": 0,
    "Label": "streamload_txn_example1_table1",
```

```
    "Status": "OK",
    "Message": "",
    "NumberTotalRows": 5265644,
    "NumberLoadedRows": 5265644,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 10737418067,
    "LoadTimeMs": 418778,
    "StreamLoadPutTimeMs": 68,
    "ReceivedDataTimeMs": 38964,
}
```

- If the transaction is considered unknown, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "TXN_NOT_EXISTS"
}
```

- If the transaction is considered in an invalid state, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation State Invalid"
}
```

- If errors other than unknown transaction and invalid status occur, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": ""
}
```

Pre-commit a transaction

Syntax

```
curl --location-trusted -u <username>:<password> -H "label:
<label_name>" \
    -H "Expect:100-continue" \
    -H "db:<database_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/prepare
```

Example

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/prepare
```

Return result

- If the pre-commit is successful, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "",
    "NumberTotalRows": 5265644,
    "NumberLoadedRows": 5265644,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 10737418067,
    "LoadTimeMs": 418778,
    "StreamLoadPutTimeMs": 68,
    "ReceivedDataTimeMs": 38964,
    "WriteDataTimeMs": 417851
    "CommitAndPublishTimeMs": 1393
}
```

- If the transaction is considered not existent, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation Not Exist"
}
```

- If the pre-commit times out, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "commit timeout",
}
```

- If errors other than non-existent transaction and pre-commit timeout occur, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "publish timeout"
}
```

Commit a transaction

Syntax

```
curl --location-trusted -u <username>:<password> -H "label:
<label_name>" \
    -H "Expect:100-continue" \
    -H "db:<database_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/commit
```

Example

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/commit
```

Return result

- If the commit is successful, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "",
    "NumberTotalRows": 5265644,
    "NumberLoadedRows": 5265644,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 10737418067,
    "LoadTimeMs": 418778,
    "StreamLoadPutTimeMs": 68,
    "ReceivedDataTimeMs": 38964,
    "WriteDataTimeMs": 417851
    "CommitAndPublishTimeMs": 1393
}
```

- If the transaction has already been committed, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "Transaction already commited",
}
```

- If the transaction is considered not existent, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
```

```
    "Status": "FAILED",
    "Message": "Transcation Not Exist"
}
```

- If the commit times out, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "commit timeout",
}
```

- If the data publish times out, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "publish timeout",
    "CommitAndPublishTimeMs": 1393
}
```

- If errors other than non-existent transaction and timeout occur, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": ""
}
```

Roll back a transaction

Syntax

```
curl --location-trusted -u <username>:<password> -H "label:
<label_name>" \
    -H "Expect:100-continue" \
```

```
    -H "db:<database_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/rollback
```

Example

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/rollback
```

Return result

- If the rollback is successful, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": ""
}
```

- If the transaction is considered not existent, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation Not Exist"
}
```

- If errors other than not existent transaction occur, the following result is returned:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": ""
}
```

References

For information about the suitable application scenarios and supported data file formats of Stream Load and about how Stream Load works, see Loading from a local file system via Stream Load.

For information about the syntax and parameters for creating Stream Load jobs, see STREAM LOAD.

Realtime synchronization from MySQL

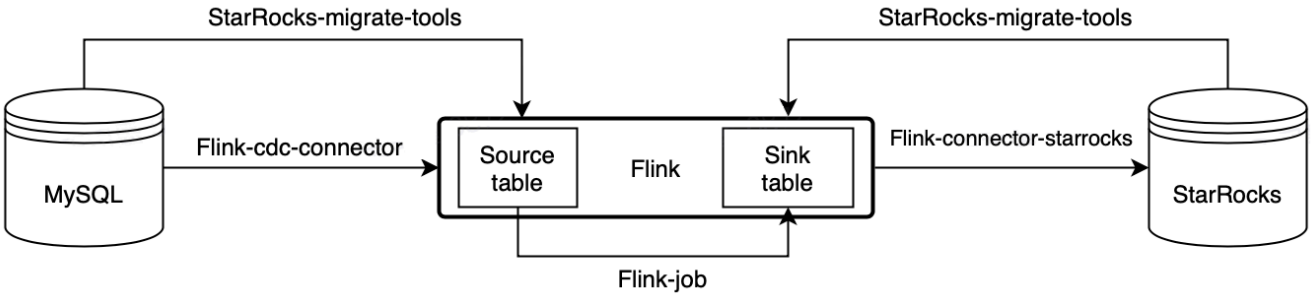
StarRocks supports multiple methods to synchronize data from MySQL to StarRocks in real time, delivering low latency real-time analytics of massive data.

This topic describes how to synchronize data from MySQL to StarRocks in real-time (within seconds) through Apache Flink®.

How it works

> **TIP**
>
> Flink CDC is used in the synchronization from MySQL to Flink. This topic uses Flink CDC whose version is less than 3.0, so SMT is used to synchronize table schemas. However, if Flink CDC 3.0 is used, it is not necessary to use SMT to synchronize table schemas to StarRocks. Flink CDC 3.0 can even synchronize the schemas of the entire MySQL database, the sharded databases and tables, and also supports schema changes synchronization. For detailed usage, see Streaming ELT from MySQL to StarRocks.

The following figure illustrates the entire synchronization process.



Real-time synchronization from MySQL through Flink to StarRocks is implemented in two stages: synchronizing database & table schema and synchronizing data. First, the SMT converts MySQL database & table schema into table creation statements for StarRocks.

Then, the Flink cluster runs Flink jobs to synchronize full and incremental MySQL data to StarRocks.

> **INFO**
>
> The synchronization process guarantees exactly-once semantics.

Synchronization process:

1. Synchronize database & table schema.

   The SMT reads the schema of the MySQL database & table to be synchronized and generates SQL files for creating a destination database & table in StarRocks. This operation is based on the MySQL and StarRocks information in SMT's configuration file.

2. Synchronize data.

   a. The Flink SQL client executes the data loading statement `INSERT INTO SELECT` to submit one or more Flink jobs to the Flink cluster.

   b. The Flink cluster runs the Flink jobs to obtain data. The Flink CDC connector first reads full historical data from the source database, then seamlessly switches to incremental reading, and sends the data to flink-connector-starrocks.

   c. flink-connector-starrocks accumulates data in mini-batches, and synchronizes each batch of data to StarRocks.

   > **INFO**
   >
   > Only data manipulation language (DML) operations in MySQL can be synchronized to StarRocks. Data definition language (DDL) operations cannot be synchronized.

Scenarios

Real-time synchronization from MySQL has a broad range of use cases where data is constantly changed. Take a real-world use case "real-time ranking of commodity sales" as an example.

Flink calculates the real-time ranking of commodity sales based on the original order table in MySQL and synchronizes the ranking to StarRocks' Primary Key table in real time. Users can connect a visualization tool to StarRocks to view the ranking in real time to gain on-demand operational insights.

Preparations

Download and install synchronization tools

To synchronize data from MySQL, you need to install the following tools: SMT, Flink, Flink CDC connector, and flink-connector-starrocks.

1. Download and install Flink, and start the Flink cluster. You can also perform this step by following the instructions in Flink official documentation.

   a. Install Java 8 or Java 11 in your operating system before you run Flink. You can run the following command to check the installed Java version.

   ```
   # View the Java version.
   java -version

   # Java 8 is installed if the following output is returned.
   java version "1.8.0_301"
   Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
   Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed
mode)
   ```

   b. Download the Flink installation package and decompress it. We recommend that you use Flink 1.14 or later. The minimum allowed version is Flink 1.11. This topic uses Flink 1.14.5.

   ```
   # Download Flink.
   wget https://archive.apache.org/dist/flink/flink-1.14.5/flink-
1.14.5-bin-scala_2.11.tgz
   # Decompress Flink.
   tar -xzf flink-1.14.5-bin-scala_2.11.tgz
   # Go to the Flink directory.
   cd flink-1.14.5
   ```

c. Start the Flink cluster.

```
   # Start the Flink cluster.
   ./bin/start-cluster.sh

   # The Flink cluster is started if the following output is
returned.
   Starting cluster.
   Starting standalonesession daemon on host.
   Starting taskexecutor daemon on host.
```

2. Download Flink CDC connector. This topic uses MySQL as the data source and therefore, `flink-sql-connector-mysql-cdc-x.x.x.jar` is downloaded. The connector version must match the Flink version. For detailed version mapping, see Supported Flink Versions. This topic uses Flink 1.14.5 and you can download `flink-sql-connector-mysql-cdc-2.2.0.jar`.

```
wget https://repo1.maven.org/maven2/com/ververica/flink-sql-
connector-mysql-cdc/2.1.1/flink-sql-connector-mysql-cdc-2.2.0.jar
```

3. Download flink-connector-starrocks. The version must match the Flink version.

> The flink-connector-starrocks package `x.x.x_flink-y.yy _ z.zz.jar` contains three version numbers:
>
> - `x.x.x` is the version number of flink-connector-starrocks.
> - `y.yy` is the supported Flink version.
> - `z.zz` is the Scala version supported by Flink. If the Flink version is 1.14.x or earlier, you must download a package that has the Scala version.
>
> This topic uses Flink 1.14.5 and Scala 2.11. Therefore, you can download the following package: `1.2.3_flink-14_2.11.jar`.

4. Move the JAR packages of Flink CDC connector (`flink-sql-connector-mysql-cdc-2.2.0.jar`) and flink-connector-starrocks (`1.2.3_flink-1.14_2.11.jar`) to the `lib` directory of Flink.

> **Note**

> If a Flink cluster is already running in your system, you must stop the Flink cluster and restart it to load and validate the JAR packages.
>
> ```
> $ ./bin/stop-cluster.sh
> $ ./bin/start-cluster.sh
> ```

5. Download and decompress the SMT package and place it in the `flink-1.14.5` directory. StarRocks provides SMT packages for Linux x86 and macos ARM64. You can choose one based on your operating system and CPU. You can also run the following command to obtain the SMT package.

```
# for Linux x86
wget https://releases.starrocks.io/resources/smt.tar.gz
# for macOS ARM64
wget
https://releases.starrocks.io/resources/smt_darwin_arm64.tar.gz
```

Enable MySQL binary log

To synchronize data from MySQL in real time, the system needs to read data from MySQL binary log (binlog), parse the data, and then synchronize the data to StarRocks. Make sure that MySQL binary log is enabled.

1. Edit the MySQL configuration file `my.cnf` (default path: `/etc/my.cnf`) to enable MySQL binary log.

```
# Enable MySQL Binlog.
log_bin = ON
# Configure the save path for the Binlog.
log_bin =/var/lib/mysql/mysql-bin
# Configure server_id.
# If server_id is not configured for MySQL 5.7.3 or later, the
MySQL service cannot be used.
server_id = 1
# Set the Binlog format to ROW.
binlog_format = ROW
# The base name of the Binlog file. An identifier is appended to
identify each Binlog file.
log_bin_basename =/var/lib/mysql/mysql-bin
# The index file of Binlog files, which manages the directory of
```

```
all Binlog files.
log_bin_index =/var/lib/mysql/mysql-bin.index
```

2. Run one of the following commands to restart MySQL for the modified configuration file to take effect.

```
# Use service to restart MySQL.
service mysqld restart
# Use mysqld script to restart MySQL.
/etc/init.d/mysqld restart
```

3. Connect to MySQL and check whether MySQL binary log is enabled.

```
-- Connect to MySQL.
mysql -h xxx.xx.xxx.xx -P 3306 -u root -pxxxxxx

-- Check whether MySQL binary log is enabled.
mysql> SHOW VARIABLES LIKE 'log_bin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

Synchronize database & table schema

1. Edit the SMT configuration file. Go to the SMT `conf` directory and edit the configuration file `config_prod.conf`, such as MySQL connection information, the matching rules of the database & table to be synchronized, and configuration information of flink-connector-starrocks.

```
[db]
type = mysql
host = xxx.xx.xxx.xx
port = 3306
user = user1
password = xxxxxx
```

```
[other]
# Number of BEs in StarRocks
be_num = 3
# `decimal_v3` is supported since StarRocks-1.18.1.
use_decimal_v3 = true
# File to save the converted DDL SQL
output_dir = ./result

[table-rule.1]
# Pattern to match databases for setting properties
database = ^demo.*$
# Pattern to match tables for setting properties
table = ^.*$

############################################
### Flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`. They
are auto-generated.
############################################
flink.starrocks.jdbc-url=jdbc:mysql://<fe_host>:<fe_query_port>
flink.starrocks.load-url= <fe_host>:<fe_http_port>
flink.starrocks.username=user2
flink.starrocks.password=xxxxxx
flink.starrocks.sink.properties.format=csv
flink.starrocks.sink.properties.column_separator=\x01
flink.starrocks.sink.properties.row_delimiter=\x02
flink.starrocks.sink.buffer-flush.interval-ms=15000
```

- `[db]`: information used to access the source database.
  - `type`: type of the source database. In this topic, the source database is `mysql`.
  - `host`: IP address of the MySQL server.
  - `port`: port number of the MySQL database, defaults to `3306`
  - `user`: username for accessing the MySQL database
  - `password`: password of the username
- `[table-rule]`: database & table matching rules and the corresponding flink-connector-starrocks configuration.
  - `Database`, `table`: the names of the database & table in MySQL. Regular expressions are supported.

- `flink.starrocks.*`: configuration information of flink-connector-starrocks. For more configurations and information, see flink-connector-starrocks.

  > If you need to use different flink-connector-starrocks configurations for different tables. For example, if some tables are frequently updated and you need to accelerate data loading, see Use different flink-connector-starrocks configurations for different tables. If you need to load multiple tables obtained from MySQL sharding into the same StarRocks table, see Synchronize multiple tables after MySQL sharding to one table in StarRocks.

- `[other]`: other information

  - `be_num`: The number of BEs in your StarRocks cluster (This parameter will be used for setting a reasonable number of tablets in subsequent StarRocks table creation).
  - `use_decimal_v3`: Whether to enable Decimal V3. After Decimal V3 is enabled, MySQL decimal data will be converted into Decimal V3 data when data is synchronized to StarRocks.
  - `output_dir`: The path to save the SQL files to be generated. The SQL files will be used to create a database & table in StarRocks and submit a Flink job to the Flink cluster. The default path is `./result` and we recommend that you retain the default settings.

2. Run the SMT to read the database & table schema in MySQL and generate SQL files in the `./result` directory based on the configuration file. The `starrocks-create.all.sql` file is used to create a database & table in StarRocks and the `flink-create.all.sql` file is used to submit a Flink job to the Flink cluster.

```
# Run the SMT.
./starrocks-migrate-tool

# Go to the result directory and check the files in this
directory.
cd result
ls result
flink-create.1.sql    smt.tar.gz              starrocks-
create.all.sql
flink-create.all.sql  starrocks-create.1.sql
```

3. Run the following command to connect to StarRocks and execute the `starrocks-create.all.sql` file to create a database and table in StarRocks. We recommend that

you use the default table creation statement in the SQL file to create a table of the Primary Key table.

> **Note**
>
> You can also modify the table creation statement based on your business needs and create a table that does not use the Primary Key table. However, the DELETE operation in the source MySQL database cannot be synchronized to the non-primary key table. Exercise caution when you create such a table.

```
mysql -h <fe_host> -P <fe_query_port> -u user2 -pxxxxxx <
starrocks-create.all.sql
```

If the data needs to be processed by Flink before it is written to the destination StarRocks table, the table schema will be different between the source and destination tables. In this case, you must modify the table creation statement. In this example, the destination table requires only the `product_id` and `product_name` columns and real-time ranking of commodity sales. You can use the following table creation statement.

```
CREATE DATABASE IF NOT EXISTS `demo`;

CREATE TABLE IF NOT EXISTS `demo`.`orders` (
`product_id` INT(11) NOT NULL COMMENT "",
`product_name` STRING NOT NULL COMMENT "",
`sales_cnt` BIGINT NOT NULL COMMENT ""
) ENGINE=olap
PRIMARY KEY(`product_id`)
DISTRIBUTED BY HASH(`product_id`) BUCKETS 1
PROPERTIES (
"replication_num" = "3"
);
```

Synchronize data

Run the Flink cluster and submit a Flink job to continuously synchronize full and incremental data from MySQL to StarRocks.

1. Go to the Flink directory and run the following command to run the `flink-create.all.sql` file on your Flink SQL client.

```
./bin/sql-client.sh -f flink-create.all.sql
```

This SQL file defines dynamic tables `source table` and `sink table`, query statement `INSERT INTO SELECT`, and specifies the connector, source database, and destination database. After this file is executed, a Flink job is submitted to the Flink cluster to start data synchronization.

> **Note**
>
> - Make sure that the Flink cluster has been started. You can start the Flink cluster by running `flink/bin/start-cluster.sh`.
> - If your Flink version is earlier than 1.13, you may not be able to directly run the SQL file `flink-create.all.sql`. You need to execute SQL statements one by one in this file in the command line interface (CLI) of the SQL client. You also need to escape the `\` character.
>
> ```
> 'sink.properties.column_separator' = '\\x01'
> 'sink.properties.row_delimiter' = '\\x02'
> ```

Process data during synchronization:

If you need to process data during synchronization, such as performing GROUP BY or JOIN on the data, you can modify the `flink-create.all.sql` file. The following example calculates real-time ranking of commodity sales by executing COUNT (*) and GROUP BY.

```
    $ ./bin/sql-client.sh -f flink-create.all.sql
    No default environment is specified.
    Searching for '/home/disk1/flink-1.13.6/conf/sql-client-
defaults.yaml'...not found.
    [INFO] Executing SQL from file.

    Flink SQL> CREATE DATABASE IF NOT EXISTS
`default_catalog`.`demo`;
    [INFO] Execute statement succeed.

    -- Create a dynamic table `source table` based on the order
table in MySQL.
    Flink SQL>
    CREATE TABLE IF NOT EXISTS
`default_catalog`.`demo`.`orders_src` (`order_id` BIGINT NOT
```

```
NULL,
    `product_id` INT NULL,
    `order_date` TIMESTAMP NOT NULL,
    `customer_name` STRING NOT NULL,
    `product_name` STRING NOT NULL,
    `price` DECIMAL(10, 5) NULL,
    PRIMARY KEY(`order_id`)
    NOT ENFORCED
    ) with ('connector' = 'mysql-cdc',
    'hostname' = 'xxx.xx.xxx.xxx',
    'port' = '3306',
    'username' = 'root',
    'password' = '',
    'database-name' = 'demo',
    'table-name' = 'orders'
    );
    [INFO] Execute statement succeed.

    -- Create a dynamic table `sink table`.
    Flink SQL>
    CREATE TABLE IF NOT EXISTS
`default_catalog`.`demo`.`orders_sink` (`product_id` INT NOT
NULL,
    `product_name` STRING NOT NULL,
    `sales_cnt` BIGINT NOT NULL,
    PRIMARY KEY(`product_id`)
    NOT ENFORCED
    ) with ('sink.max-retries' = '10',
    'jdbc-url' = 'jdbc:mysql://<fe_host>:<fe_query_port>',
    'password' = '',
    'sink.properties.strip_outer_array' = 'true',
    'sink.properties.format' = 'json',
    'load-url' = '<fe_host>:<fe_http_port>',
    'username' = 'root',
    'sink.buffer-flush.interval-ms' = '15000',
    'connector' = 'starrocks',
    'database-name' = 'demo',
    'table-name' = 'orders'
    );
    [INFO] Execute statement succeed.

    -- Implement real-time ranking of commodity sales, where
`sink table` is dynamically updated to reflect data changes in
`source table`.
    Flink SQL>
    INSERT INTO `default_catalog`.`demo`.`orders_sink` select
```

```
product_id,product_name, count(*) as cnt from
`default_catalog`.`demo`.`orders_src` group by
product_id,product_name;
    [INFO] Submitting SQL update statement to the cluster...
    [INFO] SQL update statement has been successfully submitted
to the cluster:
    Job ID: 5ae005c4b3425d8bb13fe660260a35da
```

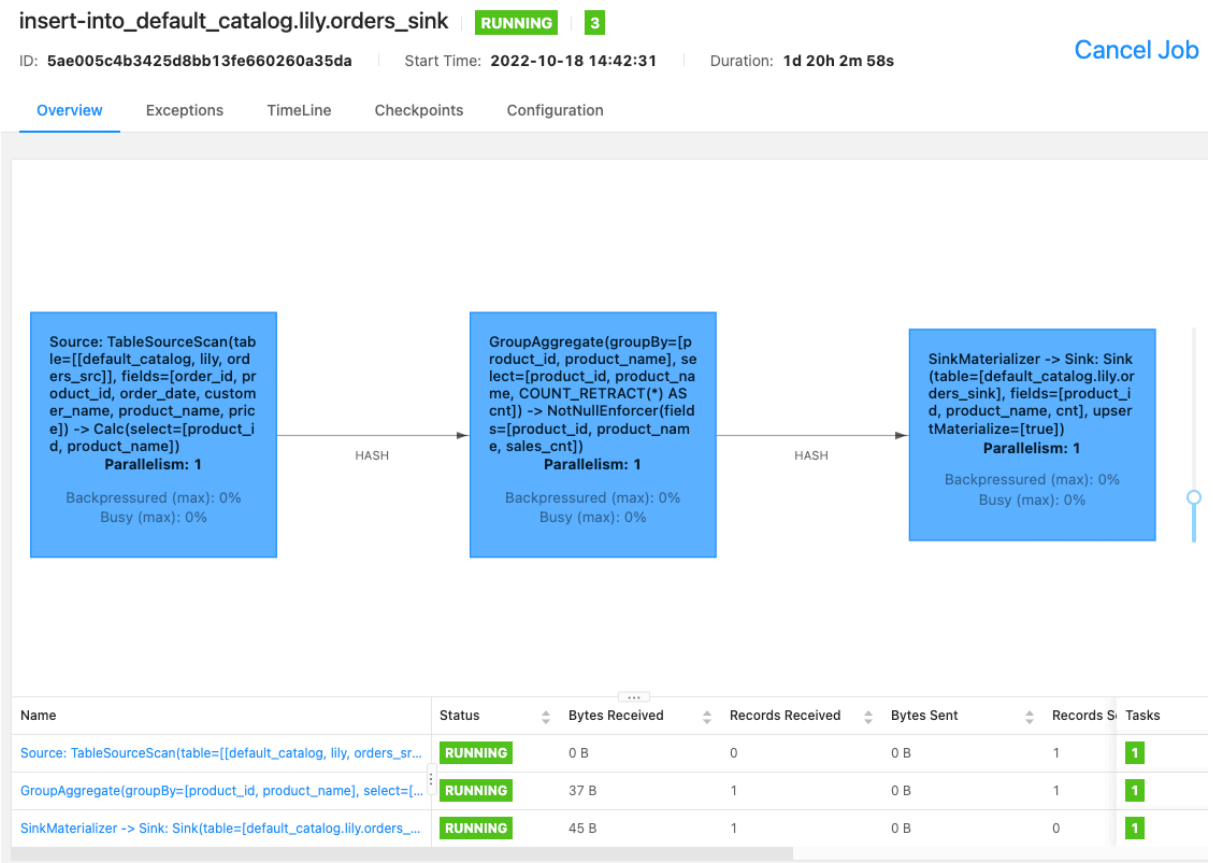
If you only need to synchronize only a portion of the data, such as data whose payment time is later than December 21, 2021, you can use the `WHERE` clause in `INSERT INTO SELECT` to set a filter condition, such as `WHERE pay_dt > '2021-12-21'`. Data that does not meet this condition will not be synchronized to StarRocks.

If the following result is returned, the Flink job has been submitted for full and incremental synchronization.

```
[INFO] Submitting SQL update statement to the cluster...
[INFO] SQL update statement has been successfully submitted to
the cluster:
Job ID: 5ae005c4b3425d8bb13fe660260a35da
```

2. You can use the Flink WebUI or run the `bin/flink list -running` command on your Flink SQL client to view Flink jobs that are running in the Flink cluster and the job IDs.

- Flink WebUI



- `bin/flink list -running`

```
    $ bin/flink list -running
    Waiting for response...
    ------------------ Running/Restarting Jobs -------------------
-
    13.10.2022 15:03:54 : 040a846f8b58e82eb99c8663424294d5 :
insert-into_default_catalog.lily.example_tbl1_sink (RUNNING)
    --------------------------------------------------------------
-
```

> **Note**
>
> If the job is abnormal, you can perform troubleshooting by using Flink WebUI or by viewing the log file in the `/log` directory of Flink 1.14.5.

FAQ

Use different flink-connector-starrocks configurations for different tables

If some tables in the data source are frequently updated and you want to accelerate the loading speed of flink-connector-starrocks, you must set a separate flink-connector-starrocks configuration for each table in the SMT configuration file `config_prod.conf`.

```
[table-rule.1]
# Pattern to match databases for setting properties
database = ^order.*$
# Pattern to match tables for setting properties
table = ^.*$

############################################
### Flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`. They are
auto-generated
############################################
flink.starrocks.jdbc-url=jdbc:mysql://<fe_host>:<fe_query_port>
flink.starrocks.load-url= <fe_host>:<fe_http_port>
flink.starrocks.username=user2
flink.starrocks.password=xxxxxx
flink.starrocks.sink.properties.format=csv
flink.starrocks.sink.properties.column_separator=\x01
flink.starrocks.sink.properties.row_delimiter=\x02
flink.starrocks.sink.buffer-flush.interval-ms=15000[table-rule.2]
# Pattern to match databases for setting properties
database = ^order2.*$
# Pattern to match tables for setting properties
table = ^.*$

############################################
### Flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`. They are
auto-generated
############################################
flink.starrocks.jdbc-url=jdbc:mysql://<fe_host>:<fe_query_port>
flink.starrocks.load-url= <fe_host>:<fe_http_port>
flink.starrocks.username=user2
flink.starrocks.password=xxxxxx
flink.starrocks.sink.properties.format=csv
flink.starrocks.sink.properties.column_separator=\x01
```

```
flink.starrocks.sink.properties.row_delimiter=\x02
flink.starrocks.sink.buffer-flush.interval-ms=10000
```

Synchronize multiple tables after MySQL sharding to one table in StarRocks

After sharding is performed, data in one MySQL table may be split into multiple tables or even distributed to multiple databases. All the tables have the same schema. In this case, you can set `[table-rule]` to synchronize these tables to one StarRocks table. For example, MySQL has two databases `edu_db_1` and `edu_db_2`, each of which has two tables `course_1 and course_2`, and the schema of all tables is the same. You can use the following `[table-rule]` configuration to synchronize all the tables to one StarRocks table.

> **Note**
>
> The name of the StarRocks table defaults to `course__auto_shard`. If you need to use a different name, you can modify it in the SQL files `starrocks-create.all.sql` and `flink-create.all.sql`

```
[table-rule.1]
# Pattern to match databases for setting properties
database = ^edu_db_[0-9]*$
# Pattern to match tables for setting properties
table = ^course_[0-9]*$

############################################
### Flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`. They are
auto-generated
############################################
flink.starrocks.jdbc-url = jdbc: mysql://xxx.xxx.x.x:xxxx
flink.starrocks.load-url = xxx.xxx.x.x:xxxx
flink.starrocks.username = user2
flink.starrocks.password = xxxxxx
flink.starrocks.sink.properties.format=csv
flink.starrocks.sink.properties.column_separator =\x01
flink.starrocks.sink.properties.row_delimiter =\x02
flink.starrocks.sink.buffer-flush.interval-ms = 5000
```

Import data in JSON format

Data in the preceding example is imported in CSV format. If you are unable to choose a suitable delimiter, you need to replace the following parameters of `flink.starrocks.*` in `[table-rule]`.

```
flink.starrocks.sink.properties.format=csv
flink.starrocks.sink.properties.column_separator =\x01
flink.starrocks.sink.properties.row_delimiter =\x02
```

Data is imported in JSON format after the following parameters are passed in.

```
flink.starrocks.sink.properties.format=json
flink.starrocks.sink.properties.strip_outer_array=true
```

> **Note**
>
> This method slightly slows down the loading speed.

Execute multiple INSERT INTO statements as one Flink job

You can use the STATEMENT SET syntax in the `flink-create.all.sql` file to execute multiple INSERT INTO statements as one Flink job, which prevents multiple statements from taking up too many Flink job resources and improves the efficiency of executing multiple queries.

> **Note**
>
> Flink supports the STATEMENT SET syntax from 1.13 onwards.

1. Open the `result/flink-create.all.sql` file.
2. Modify the SQL statements in the file. Move all the INSERT INTO statements to the end of the file. Place `EXECUTE STATEMENT SET BEGIN` before the first INSERT INTO statement and place `END;` after the last INSERT INTO statement.

> **Note**

> The positions of CREATE DATABASE and CREATE TABLE remain unchanged.

```
CREATE DATABASE IF NOT EXISTS db;
CREATE TABLE IF NOT EXISTS db.a1;
CREATE TABLE IF NOT EXISTS db.b1;
CREATE TABLE IF NOT EXISTS db.a2;
CREATE TABLE IF NOT EXISTS db.b2;
EXECUTE STATEMENT SET
BEGIN-- one or more INSERT INTO statements
INSERT INTO db.a1 SELECT * FROM db.b1;
INSERT INTO db.a2 SELECT * FROM db.b2;
END;
```

Continuously load data from Apache Flink®

StarRocks provides a self-developed connector named StarRocks Connector for Apache Flink® (Flink connector for short) to help you load data into a StarRocks table by using Flink. The basic principle is to accumulate the data and then load it all at a time into StarRocks through STREAM LOAD.

The Flink connector supports DataStream API, Table API & SQL, and Python API. It has a higher and more stable performance than flink-connector-jdbc provided by Apache Flink®.

> **NOTICE**
>
> Loading data into StarRocks tables with Flink connector needs SELECT and INSERT privileges. If you do not have these privileges, follow the instructions provided in GRANT to grant these privileges to the user that you use to connect to your StarRocks cluster.

Version requirements

Connector	Flink	StarRocks	Java	Scala
1.2.9	1.15,1.16,1.17,1.18	2.1 and later	8	2.11,2.12
1.2.8	1.13,1.14,1.15,1.16,1.17	2.1 and later	8	2.11,2.12
1.2.7	1.11,1.12,1.13,1.14,1.15	2.1 and later	8	2.11,2.12

Obtain Flink connector

You can obtain the Flink connector JAR file in the following ways:

- Directly download the compiled Flink connector JAR file.
- Add the Flink connector as a dependency in your Maven project and then download the JAR file.
- Compile the source code of the Flink connector into a JAR file by yourself.

The naming format of the Flink connector JAR file is as follows:

- Since Flink 1.15, it's `flink-connector-starrocks-${connector_version}_flink-${flink_version}.jar`. For example, if you install Flink 1.15 and you want to use Flink connector 1.2.7, you can use `flink-connector-starrocks-1.2.7_flink-1.15.jar`.
- Prior to Flink 1.15, it's `flink-connector-starrocks-${connector_version}_flink-${flink_version}_${scala_version}.jar`. For example, if you install Flink 1.14 and Scala 2.12 in your environment, and you want to use Flink connector 1.2.7, you can use `flink-connector-starrocks-1.2.7_flink-1.14_2.12.jar`.

> **NOTICE**
>
> In general, the latest version of the Flink connector only maintains compatibility with the three most recent versions of Flink.

Download the compiled Jar file

Directly download the corresponding version of the Flink connector Jar file from the Maven Central Repository.

Maven Dependency

In your Maven project's `pom.xml` file, add the Flink connector as a dependency according to the following format. Replace `flink_version`, `scala_version`, and `connector_version` with the respective versions.

- In Flink 1.15 and later

```
<dependency>
    <groupId>com.starrocks</groupId>
    <artifactId>flink-connector-starrocks</artifactId>
```

```
    <version>${connector_version}_flink-${flink_version}
</version>
</dependency>
```

- In versions earlier than Flink 1.15

```
<dependency>
    <groupId>com.starrocks</groupId>
    <artifactId>flink-connector-starrocks</artifactId>

<version>${connector_version}_flink-${flink_version}_${scala_versior
</version>
</dependency>
```

Compile by yourself

1. Download the Flink connector package.

2. Execute the following command to compile the source code of Flink connector into a JAR file. Note that `flink_version` is replaced with the corresponding Flink version.

   ```
   sh build.sh <flink_version>
   ```

   For example, if the Flink version in your environment is 1.15, you need to execute the following command:

   ```
   sh build.sh 1.15
   ```

3. Go to the `target/` directory to find the Flink connector JAR file, such as `flink-connector-starrocks-1.2.7_flink-1.15-SNAPSHOT.jar`, generated upon compilation.

> **NOTE**
>
> The name of Flink connector which is not formally released contains the `SNAPSHOT` suffix.

Options

Option	Required	Default value	Descriptio
connector	Yes	NONE	The connector that y to use. The value mu "starrocks".
jdbc-url	Yes	NONE	The address that is u connect to the MySQ of the FE. You can sp multiple addresss, w be separated by a co Format: `jdbc:mysql://<fe <fe_query_port1> <fe_host2>: <fe_query_port2> <fe_host3>: <fe_query_port3>`
load-url	Yes	NONE	The HTTP URL of th your StarRocks clust can specify multiple which must be sepa semicolon (;). Forma `<fe_host1>: <fe_http_port1>; <fe_host2>: <fe_http_port2>`.
database-name	Yes	NONE	The name of the Sta database into which to load data.

Option	Required	Default value	Descriptio
table-name	Yes	NONE	The name of the tab want to use to load d StarRocks.
username	Yes	NONE	The username of the that you want to use data into StarRocks. account needs SELE INSERT privileges.
password	Yes	NONE	The password of the preceding account.
sink.semantic	No	at-least-once	The semantic guarai sink. Valid values: **at once** and **exactly-o**
sink.version	No	AUTO	The interface used t data. This paramete supported from Flink connector version 1. onwards. • `V1`: Use Stream interface to load Connectors bef only support thi • `V2`: Use Stream transaction inte load data. It req StarRocks to be version 2.4. Rec `V2` because it c the memory usa provides a more exactly-once implementation

Option	Required	Default value	Descriptio
			• `AUTO`: If the ver StarRocks supp transaction Stre will choose `V2` automatically, o choose `V1`
sink.label-prefix	No	NONE	The label prefix usec Stream Load. Recom configure it if you ar exactly-once with cc 1.2.8 and later. See e once usage notes.
sink.buffer-flush.max-bytes	No	94371840(90M)	The maximum size c that can be accumul memory before bein StarRocks at a time. maximum value rang 64 MB to 10 GB. Set parameter to a large can improve loading performance but ma loading latency.
sink.buffer-flush.max-rows	No	500000	The maximum numb that can be accumul memory before bein StarRocks at a time. parameter is availab when you set `sink.` to `V1` and set `sink.semantic` to `least-once`. Valid v 64000 to 5000000.

Option	Required	Default value	Descriptio
sink.buffer-flush.interval-ms	No	300000	The interval at which flushed. This parame available only when `sink.semantic` to `least-once`. Valid v 1000 to 3600000. U
sink.max-retries	No	3	The number of times system retries to per Stream Load job. Th parameter is availab when you set `sink.` to `V1`. Valid values:
sink.connect.timeout-ms	No	30000	The timeout for esta HTTP connection. Va values: 100 to 6000 ms. Before Flink con v1.2.9, the default va `1000`.
sink.wait-for-continue.timeout-ms	No	10000	Supported since 1.2. timeout for waiting r of HTTP 100-continu the FE. Valid values: `60000`. Unit: ms
sink.ignore.update-before	No	true	Supported since ver Whether to ignore `UPDATE_BEFORE` rec Flink when loading d Primary Key tables. I parameter is set to f record is treated as operation to StarRoc

Option	Required	Default value	Descriptio
sink.properties.*	No	NONE	The parameters that to control Stream Lo behavior. For examp parameter `sink.properties.` specifies the format Stream Load, such a JSON. For a list of su parameters and thei descriptions, see ST LOAD.
sink.properties.format	No	csv	The format used for Load. The Flink conr transform each batc to the format before them to StarRocks. V values: `csv` and `js`
sink.properties.row_delimiter	No	\n	The row delimiter fo formatted data.
sink.properties.column_separator	No	\t	The column separat CSV-formatted data
sink.properties.max_filter_ratio	No	0	The maximum error of the Stream Load. maximum percentag records that can be out due to inadequa quality. Valid values: Default value: `0`. Se Load for details.
sink.parallelism	No	NONE	The parallelism of th connector. Only avai

Option	Required	Default value	Descriptio
			Flink SQL. If not set, planner will decide t parallelism. In the sc multi-parallelism, us to guarantee data is the correct order.
sink.properties.strict_mode	No	false	Specifies whether tc the strict mode for S Load. It affects the l behavior when there unqualified rows, su inconsistent column Valid values: `true` a `false`. Default valu See Stream Load for

Data type mapping between Flink and StarRocks

Flink data type	StarRocks data type
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
FLOAT	FLOAT

Flink data type	StarRocks data type
DOUBLE	DOUBLE
DECIMAL	DECIMAL
BINARY	INT
CHAR	STRING
VARCHAR	STRING
STRING	STRING
DATE	DATE
TIMESTAMP_WITHOUT_TIME_ZONE(N)	DATETIME
TIMESTAMP_WITH_LOCAL_TIME_ZONE(N)	DATETIME
ARRAY<T>	ARRAY<T>
MAP<KT,VT>	JSON STRING
ROW<arg T...>	JSON STRING

Usage notes

Exactly Once

- If you want sink to guarantee exactly-once semantics, we recommend you to upgrade StarRocks to 2.5 or later, and Flink connector to 1.2.4 or later
    - Since Flink connector 1.2.4, the exactly-once is redesigned based on Stream Load transaction interface provided by StarRocks since 2.4. Compared to the previous implementation based on non-transactional Stream Load non-transactional

interface, the new implementation reduces memory usage and checkpoint overhead, thereby enhancing real-time performance and stability of loading.

    - If the version of StarRocks is earlier than 2.4 or the version of Flink connector is earlier than 1.2.4, the sink will automatically choose the implementation based on Stream Load non-transactional interface.

- Configurations to guarantee exactly-once

    - The value of `sink.semantic` needs to be `exactly-once`.

    - If the version of Flink connector is 1.2.8 and later, it is recommended to specify the value of `sink.label-prefix`. Note that the label prefix must be unique among all types of loading in StarRocks, such as Flink jobs, Routine Load, and Broker Load.

        - If the label prefix is specified, the Flink connector will use the label prefix to clean up lingering transactions that may be generated in some Flink failure scenarios, such as the Flink job fails when a checkpoint is still in progress. These lingering transactions are generally in `PREPARED` status if you use `SHOW PROC '/transactions/<db_id>/running';` to view them in StarRocks. When the Flink job restores from checkpoint, the Flink connector will find these lingering transactions according to the label prefix and some information in checkpoint, and abort them. The Flink connector can not abort them when the Flink job exits because of the two-phase-commit mechanism to implement the exactly-once. When the Flink job exits, the Flink connector has not received the notification from Flink checkpoint coordinator whether the transactions should be included in a successful checkpoint, and it may lead to data loss if these transactions are aborted anyway. You can have an overview about how to achieve end-to-end exactly-once in Flink in this blogpost.

        - If the label prefix is not specified, lingering transactions will be cleaned up by StarRocks only after they time out. However the number of running transactions can reach the limitation of StarRocks `max_running_txn_num_per_db` if Flink jobs fail frequently before transactions time out. The timeout length is controlled by StarRocks FE configuration `prepared_transaction_default_timeout_second` whose default value is `86400` (1 day). You can set a smaller value to it to make transactions expired faster when the label prefix is not specified.

- If you are certain that the Flink job will eventually recover from checkpoint or savepoint after a long downtime because of stop or continuous failover, please adjust the following StarRocks configurations accordingly, to avoid data loss.

  - `prepared_transaction_default_timeout_second`: StarRocks FE configuration, default value is `86400`. The value of this configuration needs to be larger than the downtime of the Flink job. Otherwise, the lingering transactions that are included in a successful checkpoint may be aborted because of timeout before you restart the Flink job, which leads to data loss.

    Note that when you set a larger value to this configuration, it is better to specify the value of `sink.label-prefix` so that the lingering transactions can be cleaned according to the label prefix and some information in checkpoint, instead of due to timeout (which may cause data loss).

  - `label_keep_max_second` and `label_keep_max_num`: StarRocks FE configurations, default values are `259200` and `1000` respectively. For details, see FE configurations. The value of `label_keep_max_second` needs to be larger than the downtime of the Flink job. Otherwise, the Flink connector can not check the state of transactions in StarRocks by using the transaction labels saved in the Flink's savepoint or checkpoint and figure out whether these transactions are committed or not, which may eventually lead to data loss.

  These configurations are mutable and can be modified by using `ADMIN SET FRONTEND CONFIG`:

  ```
  ADMIN SET FRONTEND CONFIG
("prepared_transaction_default_timeout_second" = "3600");
  ADMIN SET FRONTEND CONFIG ("label_keep_max_second" = "259200");
  ADMIN SET FRONTEND CONFIG ("label_keep_max_num" = "1000");
  ```

Flush Policy

The Flink connector will buffer the data in memory, and flush them in batch to StarRocks via Stream Load. How the flush is triggered is different between at-least-once and exactly-once.

For at-least-once, the flush will be triggered when any of the following conditions are met:

- the bytes of buffered rows reaches the limit `sink.buffer-flush.max-bytes`
- the number of buffered rows reaches the limit `sink.buffer-flush.max-rows`. (Only valid for sink version V1)
- the elapsed time since the last flush reaches the limit `sink.buffer-flush.interval-ms`
- a checkpoint is triggered

For exactly-once, the flush only happens when a checkpoint is triggered.

Monitoring load metrics

The Flink connector provides the following metrics to monitor loading.

Metric	Type	Description
totalFlushBytes	counter	successfully flushed bytes.
totalFlushRows	counter	number of rows successfully flushed.
totalFlushSucceededTimes	counter	number of times that the data is successfully flushed.
totalFlushFailedTimes	counter	number of times that the data fails to be flushed.
totalFilteredRows	counter	number of rows filtered, which is also included in totalFlushRows.

Examples

The following examples show how to use the Flink connector to load data into a StarRocks table with Flink SQL or Flink DataStream.

Preparations

Create a StarRocks table

Create a database `test` and create a Primary Key table `score_board`.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`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`);
```

Set up Flink environment

- Download Flink binary Flink 1.15.2, and unzip it to directory `flink-1.15.2`.
- Download Flink connector 1.2.7, and put it into the directory `flink-1.15.2/lib`.
- Run the following commands to start a Flink cluster:

```
cd flink-1.15.2
./bin/start-cluster.sh
```

Run with Flink SQL

- Run the following command to start a Flink SQL client.

```
./bin/sql-client.sh
```

- Create a Flink table `score_board`, and insert values into the table via Flink SQL Client. Note you must define the primary key in the Flink DDL if you want to load data into a Primary Key table of StarRocks. It's optional for other types of StarRocks tables.

```
CREATE TABLE `score_board` (
    `id` INT,
    `name` STRING,
```

```
    `score` INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'starrocks',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',

    'table-name' = 'score_board',
    'username' = 'root',
    'password' = ''
);

INSERT INTO `score_board` VALUES (1, 'starrocks', 100), (2,
'flink', 100);
```

Run with Flink DataStream

There are several ways to implement a Flink DataStream job according to the type of the input records, such as a CSV Java `String`, a JSON Java `String` or a custom Java object.

- The input records are CSV-format `String`. See LoadCsvRecords for a complete example.

```
/**
 * Generate CSV-format records. Each record has three values
separated by "\t".
 * These values will be loaded to the columns `id`, `name`, and
`score` in the StarRocks table.
 */
String[] records = new String[]{
        "1\tstarrocks-csv\t100",
        "2\tflink-csv\t100"
};
DataStream<String> source = env.fromElements(records);

/**
 * Configure the connector with the required properties.
 * You also need to add properties "sink.properties.format" and
"sink.properties.column_separator"
 * to tell the connector the input records are CSV-format, and
the column separator is "\t".
 * You can also use other column separators in the CSV-format
```

```
records,
 * but remember to modify the "sink.properties.column_separator"
correspondingly.
 */
StarRocksSinkOptions options = StarRocksSinkOptions.builder()
        .withProperty("jdbc-url", jdbcUrl)
        .withProperty("load-url", loadUrl)
        .withProperty("database-name", "test")
        .withProperty("table-name", "score_board")
        .withProperty("username", "root")
        .withProperty("password", "")
        .withProperty("sink.properties.format", "csv")
        .withProperty("sink.properties.column_separator", "\t")
        .build();
// Create the sink with the options.
SinkFunction<String> starRockSink = StarRocksSink.sink(options);
source.addSink(starRockSink);
```

- The input records are JSON-format `String`. See LoadJsonRecords for a complete example.

```
/**
 * Generate JSON-format records.
 * Each record has three key-value pairs corresponding to the
columns `id`, `name`, and `score` in the StarRocks table.
 */
String[] records = new String[]{
        "{\"id\":1, \"name\":\"starrocks-json\", \"score\":100}",
        "{\"id\":2, \"name\":\"flink-json\", \"score\":100}",
};
DataStream<String> source = env.fromElements(records);

/**
 * Configure the connector with the required properties.
 * You also need to add properties "sink.properties.format" and
"sink.properties.strip_outer_array"
 * to tell the connector the input records are JSON-format and to
strip the outermost array structure.
 */
StarRocksSinkOptions options = StarRocksSinkOptions.builder()
        .withProperty("jdbc-url", jdbcUrl)
        .withProperty("load-url", loadUrl)
        .withProperty("database-name", "test")
        .withProperty("table-name", "score_board")
        .withProperty("username", "root")
```

```
        .withProperty("password", "")
        .withProperty("sink.properties.format", "json")
        .withProperty("sink.properties.strip_outer_array",
"true")
        .build();
// Create the sink with the options.
SinkFunction<String> starRockSink = StarRocksSink.sink(options);
source.addSink(starRockSink);
```

- The input records are custom Java objects. See LoadCustomJavaRecords for a complete example.
    - In this example, the input record is a simple POJO `RowData`.

```
public static class RowData {
        public int id;
        public String name;
        public int score;

        public RowData() {}

        public RowData(int id, String name, int score) {
            this.id = id;
            this.name = name;
            this.score = score;
        }
    }
```

    - The main program is as follows:

```
// Generate records which use RowData as the container.
RowData[] records = new RowData[]{
        new RowData(1, "starrocks-rowdata", 100),
        new RowData(2, "flink-rowdata", 100),
    };
DataStream<RowData> source = env.fromElements(records);

// Configure the connector with the required properties.
StarRocksSinkOptions options = StarRocksSinkOptions.builder()
        .withProperty("jdbc-url", jdbcUrl)
        .withProperty("load-url", loadUrl)
        .withProperty("database-name", "test")
        .withProperty("table-name", "score_board")
```

```
        .withProperty("username", "root")
        .withProperty("password", "")
        .build();

/**
 * The Flink connector will use a Java object array (Object[])
to represent a row to be loaded into the StarRocks table,
 * and each element is the value for a column.
 * You need to define the schema of the Object[] which matches
that of the StarRocks table.
 */
TableSchema schema = TableSchema.builder()
        .field("id", DataTypes.INT().notNull())
        .field("name", DataTypes.STRING())
        .field("score", DataTypes.INT())
        // When the StarRocks table is a Primary Key table,
you must specify notNull(), for example,
DataTypes.INT().notNull(), for the primary key `id`.
        .primaryKey("id")
        .build();
// Transform the RowData to the Object[] according to the
schema.
RowDataTransformer transformer = new RowDataTransformer();
// Create the sink with the schema, options, and transformer.
SinkFunction<RowData> starRockSink =
StarRocksSink.sink(schema, options, transformer);
source.addSink(starRockSink);
```

- The `RowDataTransformer` in the main program is defined as follows:

```
private static class RowDataTransformer implements
StarRocksSinkRowBuilder<RowData> {

    /**
     * Set each element of the object array according to the
input RowData.
     * The schema of the array matches that of the StarRocks
table.
     */
    @Override
    public void accept(Object[] internalRow, RowData rowData)
{
        internalRow[0] = rowData.id;
        internalRow[1] = rowData.name;
        internalRow[2] = rowData.score;
```

```
        // When the StarRocks table is a Primary Key table,
you need to set the last element to indicate whether the data
loading is an UPSERT or DELETE operation.
        internalRow[internalRow.length - 1] =
StarRocksSinkOP.UPSERT.ordinal();
    }
}
```

Synchronize data with Flink CDC 3.0 (with schema change supported)

Flink CDC 3.0 framework can be used to easily build a streaming ELT pipeline from CDC sources (such as MySQL and Kafka) to StarRocks. The pipeline can synchronize whole database, merged sharding tables, and schema changes from sources to StarRocks.

Since v1.2.9, the Flink connector for StarRocks is integrated into this framework as StarRocks Pipeline Connector. The StarRocks Pipeline Connector supports:

- Automatic creation of databases and tables
- Schema change synchronization
- Full and incremental data synchronization

For quick start, see Streaming ELT from MySQL to StarRocks using Flink CDC 3.0 with StarRocks Pipeline Connector.

Best practices

Load data to a Primary Key table

This section will show how to load data to a StarRocks Primary Key table to achieve partial updates and conditional updates. You can see Change data through loading for the introduction of those features. These examples use Flink SQL.

Preparations

Create a database `test` and create a Primary Key table `score_board` in StarRocks.

```
CREATE DATABASE `test`;
```

```
CREATE TABLE `test`.`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`);
```

Partial update

This example will show how to load data only to columns `id` and `name`.

1. Insert two data rows into the StarRocks table `score_board` in MySQL client.

```
mysql> INSERT INTO `score_board` VALUES (1, 'starrocks', 100),
(2, 'flink', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | starrocks |   100 |
|    2 | flink     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Create a Flink table `score_board` in Flink SQL client.

   - Define the DDL which only includes the columns `id` and `name`.
   - Set the option `sink.properties.partial_update` to `true` which tells the Flink connector to perform partial updates.
   - If the Flink connector version `<=` 1.2.7, you also need to set the option `sink.properties.columns` to `id,name,__op` to tells the Flink connector which columns need to be updated. Note that you need to append the field `__op` at the end. The field `__op` indicates that the data loading is an UPSERT or DELETE operation, and its values are set by the connector automatically.

```
CREATE TABLE `score_board` (
    `id` INT,
```

```
    `name` STRING,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'starrocks',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'score_board',
    'username' = 'root',
    'password' = '',
    'sink.properties.partial_update' = 'true',
    -- only for Flink connector version <= 1.2.7
    'sink.properties.columns' = 'id,name,__op'
);
```

3. Insert two data rows into the Flink table. The primary keys of the data rows are as same as these of rows in the StarRocks table. but the values in the column `name` are modified.

```
INSERT INTO `score_board` VALUES (1, 'starrocks-update'), (2,
'flink-update');
```

4. Query the StarRocks table in MySQL client.

```
mysql> select * from score_board;
+------+------------------+-------+
| id   | name             | score |
+------+------------------+-------+
|    1 | starrocks-update |   100 |
|    2 | flink-update     |   100 |
+------+------------------+-------+
2 rows in set (0.02 sec)
```

   You can see that only values for `name` change, and the values for `score` do not change.

Conditional update

This example will show how to do conditional update according to the value of column `score`. The update for an `id` takes effect only when the new value for `score` is has a greater or equal to the old value.

1. Insert two data rows into the StarRocks table in MySQL client.

```
mysql> INSERT INTO `score_board` VALUES (1, 'starrocks', 100),
(2, 'flink', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | starrocks |   100 |
|    2 | flink     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Create a Flink table `score_board` in the following ways:

   - Define the DDL including all of columns.
   - Set the option `sink.properties.merge_condition` to `score` to tell the connector to use the column `score` as the condition.
   - Set the option `sink.version` to `V1` which tells the connector to use Stream Load.

```
CREATE TABLE `score_board` (
    `id` INT,
    `name` STRING,
    `score` INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'starrocks',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'score_board',
    'username' = 'root',
    'password' = '',
    'sink.properties.merge_condition' = 'score',
    'sink.version' = 'V1'
    );
```

3. Insert two data rows into the Flink table. The primary keys of the data rows are as same as these of rows in the StarRocks table. The first data row has a smaller value in the column `score`, and the second data row has a larger value in the column `score`.

```
INSERT INTO `score_board` VALUES (1, 'starrocks-update', 99), (2,
'flink-update', 101);
```

4. Query the StarRocks table in MySQL client.

```
mysql> select * from score_board;
+------+--------------+-------+
| id   | name         | score |
+------+--------------+-------+
|    1 | starrocks    |   100 |
|    2 | flink-update |   101 |
+------+--------------+-------+
2 rows in set (0.03 sec)
```

You can see that only the values of the second data row change, and the values of the first data row do not change.

Load data into columns of BITMAP type

`BITMAP` is often used to accelerate count distinct, such as counting UV, see Use Bitmap for exact Count Distinct. Here we take the counting of UV as an example to show how to load data into columns of the `BITMAP` type.

1. Create a StarRocks Aggregate table in MySQL client.

   In the database `test`, create an Aggregate table `page_uv` where the column `visit_users` is defined as the `BITMAP` type and configured with the aggregate function `BITMAP_UNION`.

```
CREATE TABLE `test`.`page_uv` (
  `page_id` INT NOT NULL COMMENT 'page ID',
  `visit_date` datetime NOT NULL COMMENT 'access time',
  `visit_users` BITMAP BITMAP_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`);
```

2. Create a Flink table in Flink SQL client.

The column `visit_user_id` in the Flink table is of `BIGINT` type, and we want to load this column to the column `visit_users` of `BITMAP` type in the StarRocks table. So when defining the DDL of the Flink table, note that:

- Because Flink does not support `BITMAP`, you need to define a column `visit_user_id` as `BIGINT` type to represent the column `visit_users` of `BITMAP` type in the StarRocks table.
- You need to set the option `sink.properties.columns` to `page_id,visit_date,user_id,visit_users=to_bitmap(visit_user_id)`, which tells the connector the column mapping beween the Flink table and StarRocks table. Also you need to use `to_bitmap` function to tell the connector to convert the data of `BIGINT` type into `BITMAP` type.

```
CREATE TABLE `page_uv` (
    `page_id` INT,
    `visit_date` TIMESTAMP,
    `visit_user_id` BIGINT
) WITH (
    'connector' = 'starrocks',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'page_uv',
    'username' = 'root',
    'password' = '',
    'sink.properties.columns' =
'page_id,visit_date,visit_user_id,visit_users=to_bitmap(visit_user_i
);
```

3. Load data into Flink table in Flink SQL client.

```
INSERT INTO `page_uv` VALUES
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 13),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 33),
   (1, CAST('2020-06-23 02:30:30' AS TIMESTAMP), 13),
   (2, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23);
```

4. Calculate page UVs from the StarRocks table in MySQL client.

```
MySQL [test]> SELECT `page_id`, COUNT(DISTINCT `visit_users`)
FROM `page_uv` GROUP BY `page_id`;
+---------+----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+----------------------------+
|       2 |                          1 |
|       1 |                          3 |
+---------+----------------------------+
2 rows in set (0.05 sec)
```

Load data into columns of HLL type

`HLL` can be used for approximate count distinct, see Use HLL for approximate count distinct.

Here we take the counting of UV as an example to show how to load data into columns of the `HLL` type.

1. Create a StarRocks Aggregate table

   In the database `test`, create an Aggregate table `hll_uv` where the column `visit_users` is defined as the `HLL` type and configured with the aggregate function `HLL_UNION`.

   ```
   CREATE TABLE `hll_uv` (
     `page_id` INT NOT NULL COMMENT 'page ID',
     `visit_date` datetime NOT NULL COMMENT 'access time',
     `visit_users` HLL HLL_UNION NOT NULL COMMENT 'user ID'
   ) ENGINE=OLAP
   AGGREGATE KEY(`page_id`, `visit_date`)
   DISTRIBUTED BY HASH(`page_id`);
   ```

2. Create a Flink table in Flink SQL client.

   The column `visit_user_id` in the Flink table is of `BIGINT` type, and we want to load this column to the column `visit_users` of `HLL` type in the StarRocks table. So when defining the DDL of the Flink table, note that:

   - Because Flink does not support `BITMAP`, you need to define a column `visit_user_id` as `BIGINT` type to represent the column `visit_users` of `HLL`

type in the StarRocks table.

- You need to set the option `sink.properties.columns` to `page_id,visit_date,user_id,visit_users=hll_hash(visit_user_id)` which tells the connector the column mapping between Flink table and StarRocks table. Also you need to use `hll_hash` function to tell the connector to convert the data of `BIGINT` type into `HLL` type.

```
CREATE TABLE `hll_uv` (
    `page_id` INT,
    `visit_date` TIMESTAMP,
    `visit_user_id` BIGINT
) WITH (
    'connector' = 'starrocks',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'hll_uv',
    'username' = 'root',
    'password' = '',
    'sink.properties.columns' =
'page_id,visit_date,visit_user_id,visit_users=hll_hash(visit_user_ic
);
```

3. Load data into Flink table in Flink SQL client.

```
INSERT INTO `hll_uv` VALUES
   (3, CAST('2023-07-24 12:00:00' AS TIMESTAMP), 78),
   (4, CAST('2023-07-24 13:20:10' AS TIMESTAMP), 2),
   (3, CAST('2023-07-24 12:30:00' AS TIMESTAMP), 674);
```

4. Calculate page UVs from the StarRocks table in MySQL client.

```
mysql> SELECT `page_id`, COUNT(DISTINCT `visit_users`) FROM
`hll_uv` GROUP BY `page_id`;
**+---------+----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+----------------------------+
|       3 |                          2 |
|       4 |                          1 |
+---------+----------------------------+
2 rows in set (0.04 sec)
```

Change data through loading

Primary Key tables provided by StarRocks allows you to make data changes to StarRocks tables by running Stream Load, Broker Load, or Routine Load jobs. These data changes include inserts, updates, and deletions. However, Primary Key tables do not support changing data by using Spark Load or INSERT.

StarRocks also supports partial updates and conditional updates.

This topic uses CSV data as an example to describe how to make data changes to a StarRocks table through loading. The data file formats that are supported vary depending on the loading method of your choice.

> **NOTE**
>
> For CSV data, you can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.

Implementation

Primary Key tables provided by StarRocks support UPSERT and DELETE operations and does not distinguish INSERT operations from UPDATE operations.

When you create a load job, StarRocks supports adding a field named `__op` to the job creation statement or command. The `__op` field is used to specify the type of operation you want to perform.

> **NOTE**
>
> When you create a table, you do not need to add a column named `__op` to that table.

The method of defining the `__op` field varies depending on the loading method of your choice:

- If you choose Stream Load, define the `__op` field by using the `columns` parameter.

- If you choose Broker Load, define the `__op` field by using the SET clause.
- If you choose Routine Load, define the `__op` field by using the `COLUMNS` column.

You can decide whether to add the `__op` field based on the data changes you want to make. If you do not add the `__op` field, the operation type defaults to UPSERT. The major data change scenarios are as follows:

- If the data file you want to load involves only UPSERT operations, you do not need to add the `__op` field.
- If the data file you want to load involves only DELETE operations, you must add the `__op` field and specify the operation type as DELETE.
- If the data file you want to load involves both UPSERT and DELETE operations, you must add the `__op` field and make sure that the data file contains a column whose values are `0` or `1`. A value of `0` indicates an UPSERT operation, and a value of `1` indicates a DELETE operation.

Usage notes

- Make sure that each row in your data file has the same number of columns.
- The columns that involve data changes must include the primary key column.

Prerequisites

Broker Load

See the "Background information" section in Load data from HDFS or cloud storage.

Routine load

If you choose Routine Load, make sure that topics are created in your Apache Kafka® cluster. Assume that you have created four topics: `topic1`, `topic2`, `topic3`, and `topic4`.

Basic operations

This section provides examples of how to make data changes to a StarRocks table through loading. For detailed syntax and parameter descriptions, see STREAM LOAD, BROKER LOAD, and CREATE ROUTINE LOAD.

UPSERT

If the data file you want to load involves only UPSERT operations, you do not need to add the `__op` field.

> **NOTE**
>
> If you add the `__op` field:
>
> - You can specify the operation type as UPSERT.
> - You can leave the `__op` field empty, because the operation type defaults to UPSERT.

Data examples

1. Prepare a data file.

   a. Create a CSV file named `example1.csv` in your local file system. The file consists of three columns, which represent user ID, user name, and user score in sequence.

   ```
   101,Lily,100
   102,Rose,100
   ```

   b. Publish the data of `example1.csv` to `topic1` of your Kafka cluster.

2. Prepare a StarRocks table.

   a. Create a Primary Key table named `table1` in your StarRocks database `test_db`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

   ```
   CREATE TABLE `table1`
   (
   ```

```
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NOT NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

b. Insert a record into `table1`.

```
INSERT INTO table1 VALUES
    (101, 'Lily',80);
```

Load data

Run a load job to update the record whose `id` is `101` in `example1.csv` to `table1` and insert the record whose `id` is `102` in `example1.csv` into `table1`.

- Run a Stream Load job.
  - If you do not want to include the `__op` field, run the following command:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label1" \
    -H "column_separator:," \
    -T example1.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/test_db/table1/_stream_load
```

  - If you want to include the `__op` field, run the following command:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label2" \
    -H "column_separator:," \
    -H "columns:__op ='upsert'" \
    -T example1.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/test_db/table1/_stream_load
```

- Run a Broker Load job.
  - If you do not want to include the `__op` field, run the following command:

```
LOAD LABEL test_db.label1
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example1.csv")
    into table table1
    columns terminated by ","
    format as "csv"
)
WITH BROKER;
```

  - If you want to include the `__op` field, run the following command:

```
LOAD LABEL test_db.label2
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example1.csv")
    into table table1
    columns terminated by ","
    format as "csv"
    set (__op = 'upsert')
)
WITH BROKER;
```

- Run a Routine Load job.
  - If you do not want to include the `__op` field, run the following command:

```
CREATE ROUTINE LOAD test_db.table1 ON table1
COLUMNS TERMINATED BY ",",
COLUMNS (id, name, score)
```

```
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:
<kafka_broker_port>",
    "kafka_topic" = "test1",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

- If you want to include the `__op` field, run the following command:

```
CREATE ROUTINE LOAD test_db.table1 ON table1
COLUMNS TERMINATED BY ",",
COLUMNS (id, name, score, __op ='upsert')
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:
<kafka_broker_port>",
    "kafka_topic" = "test1",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

Query data

After the load is complete, query the data of `table1` to verify that the load is successful:

```
SELECT * FROM table1;
+------+------+-------+
| id   | name | score |
+------+------+-------+
```

```
|  101 | Lily |   100 |
|  102 | Rose |   100 |
+------+------+-------+
2 rows in set (0.02 sec)
```

As shown in the preceding query result, the record whose `id` is `101` in `example1.csv` has been updated to `table1`, and the record whose `id` is `102` in `example1.csv` has been inserted into `table1`.

DELETE

If the data file you want to load involves only DELETE operations, you must add the `__op` field and specify the operation type as DELETE.

Data examples

1. Prepare a data file.

   a. Create a CSV file named `example2.csv` in your local file system. The file consists of three columns, which represent user ID, user name, and user score in sequence.

   ```
   101,Jack,100
   ```

   b. Publish the data of `example2.csv` to `topic2` of your Kafka cluster.

2. Prepare a StarRocks table.

   a. Create a Primary Key table named `table2` in your StarRocks table `test_db`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

   ```
   CREATE TABLE `table2`
         (
       `id` int(11) NOT NULL COMMENT "user ID",
       `name` varchar(65533) NOT NULL COMMENT "user name",
       `score` int(11) NOT NULL COMMENT "user score"
   )
   ENGINE=OLAP
   PRIMARY KEY(`id`)
   DISTRIBUTED BY HASH(`id`);
   ```

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

b. Insert two records into `table2`.

```
INSERT INTO table2 VALUES
(101, 'Jack', 100),
(102, 'Bob', 90);
```

Load data

Run a load job to delete the record whose `id` is `101` in `example2.csv` from `table2`.

- Run a Stream Load job.

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label3" \
    -H "column_separator:," \
    -H "columns:__op='delete'" \
    -T example2.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/test_db/table2/_stream_load
```

- Run a Broker Load job.

```
LOAD LABEL test_db.label3
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example2.csv")
    into table table2
    columns terminated by ","
    format as "csv"
    set (__op = 'delete')
)
WITH BROKER;
```

- Run a Routine Load job.

```
CREATE ROUTINE LOAD test_db.table2 ON table2
COLUMNS(id, name, score, __op = 'delete')
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:
<kafka_broker_port>",
    "kafka_topic" = "test2",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

Query data

After the load is complete, query the data of `table2` to verify that the load is successful:

```
SELECT * FROM table2;
+------+------+-------+
| id   | name | score |
+------+------+-------+
|  102 | Bob  |    90 |
+------+------+-------+
1 row in set (0.00 sec)
```

As shown in the preceding query result, the record whose `id` is `101` in `example2.csv` has been deleted from `table2`.

UPSERT and DELETE

If the data file you want to load involves both UPSERT and DELETE operations, you must add the `__op` field and make sure that the data file contains a column whose values are `0` or `1`. A value of `0` indicates an UPSERT operation, and a value of `1` indicates a DELETE operation.

Data examples

1. Prepare a data file.

   a. Create a CSV file named `example3.csv` in your local file system. The file consists of four columns, which represent user ID, user name, user score, and operation type in sequence.

   ```
   101,Tom,100,1
   102,Sam,70,0
   103,Stan,80,0
   ```

   b. Publish the data of `example3.csv` to `topic3` of your Kafka cluster.

2. Prepare a StarRocks table.

   a. Create a Primary Key table named `table3` in your StarRocks database `test_db`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

   ```
   CREATE TABLE `table3`
   (
       `id` int(11) NOT NULL COMMENT "user ID",
       `name` varchar(65533) NOT NULL COMMENT "user name",
       `score` int(11) NOT NULL COMMENT "user score"
   )
   ENGINE=OLAP
   PRIMARY KEY(`id`)
   DISTRIBUTED BY HASH(`id`);
   ```

   > **NOTE**
   >
   > Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

   b. Insert two records into `table3`.

   ```
   INSERT INTO table3 VALUES
       (101, 'Tom', 100),
       (102, 'Sam', 90);
   ```

Load data

Run a load job to delete the record whose `id` is `101` in `example3.csv` from `table3`, update the record whose `id` is `102` in `example3.csv` to `table3`, and insert the record whose `id` is `103` in `example3.csv` into `table3`.

- Run a Stream Load job:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label4" \
    -H "column_separator:," \
    -H "columns: id, name, score, temp, __op = temp" \
    -T example3.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/test_db/table3/_stream_load
```

> **NOTE**
>
> In the preceding example, the fourth column that represents the operation type in `example3.csv` is temporarily named as `temp` and the `__op` field is mapped onto the `temp` column by using the `columns` parameter. As such, StarRocks can decide whether to perform an UPSERT or DELETE operation depending on the value in the fourth column of `example3.csv` is `0` or `1`.

- Run a Broker Load job:

```
LOAD LABEL test_db.label4
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example1.csv")
    into table table1
    columns terminated by ","
    format as "csv"
    (id, name, score, temp)
    set (__op=temp)
)
WITH BROKER;
```

- Run a Routine Load job:

```
CREATE ROUTINE LOAD test_db.table3 ON table3
COLUMNS(id, name, score, temp, __op = temp)
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:
<kafka_broker_port>",
    "kafka_topic" = "test3",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Query data

After the load is complete, query the data of `table3` to verify that the load is successful:

```
SELECT * FROM table3;
+------+------+-------+
| id   | name | score |
+------+------+-------+
|  102 | Sam  |    70 |
|  103 | Stan |    80 |
+------+------+-------+
2 rows in set (0.01 sec)
```

As shown in the preceding query result, the record whose `id` is `101` in `example3.csv` has been deleted from `table3`, the record whose `id` is `102` in `example3.csv` has been updated to `table3`, and the record whose `id` is `103` in `example3.csv` has been inserted into `table3`.

Partial updates

Since v2.2, StarRocks supports updating only the specified columns of a table that uses the Primary Key table. This section uses CSV as an example to describe how to perform partial updates.

> **NOTICE**
>
> When you perform a partial update, if the row to be updated does not exist, StarRocks inserts a new row, and fills default values in fields that are empty because no data updates are inserted into them.

Data examples

1. Prepare a data file.

   a. Create a CSV file named `example4.csv` in your local file system. The file consists of two columns, which represent user ID and user name in sequence.

   ```
   101,Lily
   102,Rose
   103,Alice
   ```

   b. Publish the data of `example4.csv` to `topic4` of your Kafka cluster.

2. Prepare a StarRocks table.

   a. Create a Primary Key table named `table4` in your StarRocks database `test_db`. The table consists of three columns: `id`, `name`, and `score`, of which `id` is the primary key.

   ```
   CREATE TABLE `table4`
   (
       `id` int(11) NOT NULL COMMENT "user ID",
       `name` varchar(65533) NOT NULL COMMENT "user name",
       `score` int(11) NOT NULL COMMENT "user score"
   )
   ENGINE=OLAP
   PRIMARY KEY(`id`)
   DISTRIBUTED BY HASH(`id`);
   ```

   > **NOTE**
   >
   > Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the

> number of buckets. For detailed information, see determine the number of buckets.

b. Insert a record into `table4`.

```
INSERT INTO table4 VALUES
    (101, 'Tom',80);
```

Load data

Run a load to update the data in the two columns of `example4.csv` to the `id` and `name` columns of `table4`.

- Run a Stream Load job:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label7" -H "column_separator:," \
    -H "partial_update:true" \
    -H "columns:id,name" \
    -T example4.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/test_db/table4/_stream_load
```

> **NOTE**
>
> If you choose Stream Load, you must set the `partial_update` parameter to `true` to enable the partial update feature. Additionally, you must use the `columns` parameter to specify the columns you want to update.

- Run a Broker Load job:

```
LOAD LABEL test_db.table4
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example4.csv")
    into table table4
    format as "csv"
    (id, name)
)
WITH BROKER
PROPERTIES
```

```
(
    "partial_update" = "true"
);
```

> **NOTE**
>
> If you choose Broker Load, you must set the `partial_update` parameter to `true` to enable the partial update feature. Additionally, you must use the `column_list` parameter to specify the columns you want to update.

- Run a Routine Load job:

```
CREATE ROUTINE LOAD test_db.table4 on table4
COLUMNS (id, name),
COLUMNS TERMINATED BY ','
PROPERTIES
(
    "partial_update" = "true"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:
<kafka_broker_port>",
    "kafka_topic" = "test4",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

> **NOTE**
>
> If you choose Broker Load, you must set the `partial_update` parameter to `true` to enable the partial update feature. Additionally, you must use the `COLUMNS` parameter to specify the columns you want to update.

Query data

After the load is complete, query the data of `table4` to verify that the load is successful:

```
SELECT * FROM table4;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
```

```
|  102 | Rose  |     0 |
|  101 | Lily  |    80 |
|  103 | Alice |     0 |
+------+-------+-------+
3 rows in set (0.01 sec)
```

As shown in the preceding query result, the record whose `id` is `101` in `example4.csv` has been updated to `table4`, and the records whose `id` are `102` and `103` in `example4.csv` have been Inserted into `table4`.

Conditional updates

From StarRocks v2.5 onwards, Primary Key tables support conditional updates. You can specify a non-primary key column as the condition to determine whether updates can take effect. As such, the update from a source record to a destination record takes effect only when the source data record has a greater or equal value than the destination data record in the specified column.

The conditional update feature is designed to resolve data disorder. If the source data is disordered, you can use this feature to ensure that new data will not be overwritten by old data.

> **NOTICE**
>
> - You cannot specify different columns as update conditions for the same batch of data.
> - DELETE operations do not support conditional updates.
> - Partial updates and conditional updates cannot be used simultaneously.
> - Only Stream Load and Routine Load support conditional updates.

Data examples

1. Prepare a data file.

   a. Create a CSV file named `example5.csv` in your local file system. The file consists of three columns, which represent user ID, version, and user score in sequence.

```
101,1,100
102,3,100
```

b. Publish the data of `example5.csv` to `topic5` of your Kafka cluster.

2. Prepare a StarRocks table.

a. Create a Primary Key table named `table5` in your StarRocks database `test_db`. The table consists of three columns: `id`, `version`, and `score`, of which `id` is the primary key.

```
CREATE TABLE `table5`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `version` int NOT NULL COMMENT "version",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`) DISTRIBUTED BY HASH(`id`) BUCKETS 10;
```

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

b. Insert a record into `table5`.

```
INSERT INTO table5 VALUES
    (101, 2, 80),
    (102, 2, 90);
```

Load data

Run a load to update the records whose `id` values are `101` and `102`, respectively, from `example5.csv` into `table5`, and specify that the updates take effect only when the `verion` value in each of the two records is greater or equal to their current `version` values.

- Run a Stream Load job:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label10" \
    -H "column_separator:," \
    -H "merge_condition:version" \
    -T example5.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/test_db/table5/_stream_load
```

- Run a Routine Load job:

```
CREATE ROUTINE LOAD test_db.table5 on table5
COLUMNS (id, version, score),
COLUMNS TERMINATED BY ','
PROPERTIES
(
    "merge_condition" = "version"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:
<kafka_broker_port>",
    "kafka_topic" = "topic5",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

Query data

After the load is complete, query the data of `table5` to verify that the load is successful:

```
SELECT * FROM table5;
+------+------+-------+
| id   | version | score |
+------+------+-------+
|  101 |       2 |   80 |
|  102 |       3 |  100 |
+------+------+-------+
2 rows in set (0.02 sec)
```

As shown in the preceding query result, the record whose `id` is `101` in `example5.csv` is not updated to `table5`, and the record whose `id` is `102` in `example5.csv` has been Inserted into `table5`.

Transform data at loading

StarRocks supports data transformation at loading.

This feature supports Stream Load, Broker Load, and Routine Load but does not support Spark Load.

This topic uses CSV data as an example to describe how to extract and transform data at loading. The data file formats that are supported vary depending on the loading method of your choice.

> **NOTE**
>
> For CSV data, you can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.

Scenarios

When you load a data file into a StarRocks table, the data of the data file may not be completely mapped onto the data of the StarRocks table. In this situation, you do not need to extract or transform the data before you load it into the StarRocks table. StarRocks can help you extract and transform the data during loading:

- Skip columns that do not need to be loaded.

  You can skip the columns that do not need to be loaded. Additionally, if the columns of the data file are in a different order than the columns of the StarRocks table, you can create a column mapping between the data file and the StarRocks table.

- Filter out rows you do not want to load.

  You can specify filter conditions based on which StarRocks filters out the rows that you do not want to load.

- Generate new columns from original columns.

Generated columns are special columns that are computed from the original columns of the data file. You can map the generated columns onto the columns of the StarRocks table.

- Extract partition field values from a file path.

  If the data file is generated from Apache Hive™, you can extract partition field values from the file path.

Prerequisites

Broker Load

See the "Background information" section in Load data from HDFS or cloud storage.

Routine load

If you choose Routine Load, make sure that topics are created in your Apache Kafka® cluster. Assume that you have created two topics: `topic1` and `topic2`.

Data examples

1. Create data files in your local file system.

   a. Create a data file named `file1.csv`. The file consists of four columns, which represent user ID, user gender, event date, and event type in sequence.

   ```
   354,female,2020-05-20,1
   465,male,2020-05-21,2
   576,female,2020-05-22,1
   687,male,2020-05-23,2
   ```

   b. Create a data file named `file2.csv`. The file consists of only one column, which represents date.

   ```
   2020-05-20
   2020-05-21
   ```

```
2020-05-22
2020-05-23
```

2. Create tables in your StarRocks database `test_db`.

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

a. Create a table named `table1`, which consists of three columns: `event_date`, `event_type`, and `user_id`.

```
MySQL [test_db]> CREATE TABLE table1
(
    `event_date` DATE COMMENT "event date",
     `event_type` TINYINT COMMENT "event type",
     `user_id` BIGINT COMMENT "user ID"
)
DISTRIBUTED BY HASH(user_id);
```

b. Create a table named `table2`, which consists of four columns: `date`, `year`, `month`, and `day`.

```
MySQL [test_db]> CREATE TABLE table2
(
     `date` DATE COMMENT "date",
     `year` INT COMMENT "year",
     `month` TINYINT COMMENT "month",
     `day` TINYINT COMMENT "day"
)
DISTRIBUTED BY HASH(date);
```

3. Upload `file1.csv` and `file2.csv` to the `/user/starrocks/data/input/` path of your HDFS cluster, publish the data of `file1.csv` to `topic1` of your Kafka cluster, and publish the data of `file2.csv` to `topic2` of your Kafka cluster.

Skip columns that do not need to be loaded

The data file that you want to load into a StarRocks table may contain some columns that cannot be mapped to any columns of the StarRocks table. In this situation, StarRocks supports loading only the columns that can be mapped from the data file onto the columns of the StarRocks table.

This feature supports loading data from the following data sources:

- Local file system
- HDFS and cloud storage

  > **NOTE**
  >
  > This section uses HDFS as an example.

- Kafka

In most cases, the columns of a CSV file are not named. For some CSV files, the first row is composed of column names, but StarRocks processes the content of the first row as common data rather than column names. Therefore, when you load a CSV file, you must temporarily name the columns of the CSV file **in sequence** in the job creation statement or command. These temporarily named columns are mapped **by name** onto the columns of the StarRocks table. Take note of the following points about the columns of the data file:

- The data of the columns that can be mapped onto and are temporarily named by using the names of the columns in the StarRocks table is directly loaded.
- The columns that cannot be mapped onto the columns of the StarRocks table are ignored, the data of these columns are not loaded.
- If some columns can be mapped onto the columns of the StarRocks table but are not temporarily named in the job creation statement or command, the load job reports errors.

This section uses `file1.csv` and `table1` as an example. The four columns of `file1.csv` are temporarily named as `user_id`, `user_gender`, `event_date`, and `event_type` in sequence. Among the temporarily named columns of `file1.csv`, `user_id`, `event_date`,

and `event_type` can be mapped onto specific columns of `table1`, whereas `user_gender` cannot be mapped onto any column of `table1`. Therefore, `user_id`, `event_date`, and `event_type` are loaded into `table1`, but `user_gender` is not.

Load data

Load data from a local file system

If `file1.csv` is stored in your local file system, run the following command to create a Stream Load job:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: user_id, user_gender, event_date, event_type" \
    -T file1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

> **NOTE**
>
> If you choose Stream Load, you must use the `columns` parameter to temporarily name the columns of the data file to create a column mapping between the data file and the StarRocks table.

For detailed syntax and parameter descriptions, see STREAM LOAD.

Load data from an HDFS cluster

If `file1.csv` is stored in your HDFS cluster, execute the following statement to create a Broker Load job:

```
LOAD LABEL test_db.label1
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/file1.csv")
    INTO TABLE `table1`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (user_id, user_gender, event_date, event_type)
)
WITH BROKER;
```

> **NOTE**
>
> If you choose Broker Load, you must use the `column_list` parameter to temporarily name the columns of the data file to create a column mapping between the data file and the StarRocks table.

For detailed syntax and parameter descriptions, see BROKER LOAD.

Load data from a Kafka cluster

If the data of `file1.csv` is published to `topic1` of your Kafka cluster, execute the following statement to create a Routine Load job:

```
CREATE ROUTINE LOAD test_db.table101 ON table1
    COLUMNS TERMINATED BY ",",
    COLUMNS(user_id, user_gender, event_date, event_type)
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic1",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

> **NOTE**
>
> If you choose Routine Load, you must use the `COLUMNS` parameter to temporarily name the columns of the data file to create a column mapping between the data file and the StarRocks table.

For detailed syntax and parameter descriptions, see CREATE ROUTINE LOAD.

Query data

After the load of data from your local file system, HDFS cluster, or Kafka cluster is complete, query the data of `table1` to verify that the load is successful:

```
MySQL [test_db]> SELECT * FROM table1;
+------------+------------+---------+
| event_date | event_type | user_id |
+------------+------------+---------+
| 2020-05-22 |          1 |     576 |
```

```
| 2020-05-20 |          1 |     354 |
| 2020-05-21 |          2 |     465 |
| 2020-05-23 |          2 |     687 |
+------------+------------+---------+
4 rows in set (0.01 sec)
```

Filter out rows that you do not want to load

When you load a data file into a StarRocks table, you may not want to load specific rows of the data file. In this situation, you can use the WHERE clause to specify the rows that you want to load. StarRocks filters out all rows that do not meet the filter conditions specified in the WHERE clause.

This feature supports loading data from the following data sources:

- Local file system
- HDFS and cloud storage

> **NOTE**
>
> This section uses HDFS as an example.

- Kafka

This section uses `file1.csv` and `table1` as an example. If you want to load only the rows whose event type is `1` from `file1.csv` into `table1`, you can use the WHERE clause to specify a filter condition `event_type = 1`.

Load data

Load data from a local file system

If `file1.csv` is stored in your local file system, run the following command to create a Stream Load job:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: user_id, user_gender, event_date, event_type" \
```

```
    -H "where: event_type=1" \
    -T file1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

For detailed syntax and parameter descriptions, see STREAM LOAD.

Load data from an HDFS cluster

If `file1.csv` is stored in your HDFS cluster, execute the following statement to create a Broker Load job:

```
LOAD LABEL test_db.label2
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/file1.csv")
    INTO TABLE `table1`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (user_id, user_gender, event_date, event_type)
    WHERE event_type = 1
)
WITH BROKER;
```

For detailed syntax and parameter descriptions, see BROKER LOAD.

Load data from a Kafka cluster

If the data of `file1.csv` is published to `topic1` of your Kafka cluster, execute the following statement to create a Routine Load job:

```
CREATE ROUTINE LOAD test_db.table102 ON table1
COLUMNS TERMINATED BY ",",
COLUMNS (user_id, user_gender, event_date, event_type)
WHERE event_type = 1
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic1",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

For detailed syntax and parameter descriptions, see CREATE ROUTINE LOAD.

Query data

After the load of data from your local file system, HDFS cluster, or Kafka cluster is complete, query the data of `table1` to verify that the load is successful:

```
MySQL [test_db]> SELECT * FROM table1;
+------------+------------+---------+
| event_date | event_type | user_id |
+------------+------------+---------+
| 2020-05-20 |          1 |     354 |
| 2020-05-22 |          1 |     576 |
+------------+------------+---------+
2 rows in set (0.01 sec)
```

Generate new columns from original columns

When you load a data file into a StarRocks table, some data of the data file may require conversions before the data can be loaded into the StarRocks table. In this situation, you can use functions or expressions in the job creation command or statement to implement data conversions.

This feature supports loading data from the following data sources:

- Local file system
- HDFS and cloud storage

  > **NOTE**
  >
  > This section uses HDFS as an example.

- Kafka

This section uses `file2.csv` and `table2` as an example. `file2.csv` consists of only one column that represents date. You can use the year, month, and day functions to extract the year, month, and day in each date from `file2.csv` and load the extracted data into the `year`, `month`, and `day` columns of `table2`.

Load data

Load data from a local file system

If `file2.csv` is stored in your local file system, run the following command to create a Stream Load job:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns:date,year=year(date),month=month(date),day=day(date)"
\
    -T file2.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table2/_stream_load
```

> **NOTE**
>
> - In the `columns` parameter, you must first temporarily name **all columns** of the data file, and then temporarily name the new columns that you want to generate from the original columns of the data file. As shown in the preceding example, the only column of `file2.csv` is temporarily named as `date`, and then the `year=year(date)`, `month=month(date)`, and `day=day(date)` functions are invoked to generate three new columns, which are temporarily named as `year`, `month`, and `day`.
> - Stream Load does not support `column_name = function(column_name)` but supports `column_name = function(column_name)`.

For detailed syntax and parameter descriptions, see STREAM LOAD.

Load data from an HDFS cluster

If `file2.csv` is stored in your HDFS cluster, execute the following statement to create a Broker Load job:

```
LOAD LABEL test_db.label3
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/file2.csv")
    INTO TABLE `table2`
    FORMAT AS "csv"
```

```
    COLUMNS TERMINATED BY ","
    (date)
    SET(year=year(date), month=month(date), day=day(date))
)
WITH BROKER;
```

> **NOTE**
>
> You must first use the `column_list` parameter to temporarily name **all columns** of the data file, and then use the SET clause to temporarily name the new columns that you want to generate from the original columns of the data file. As shown in the preceding example, the only column of `file2.csv` is temporarily named as `date` in the `column_list` parameter, and then the `year=year(date)`, `month=month(date)`, and `day=day(date)` functions are invoked in the SET clause to generate three new columns, which are temporarily named as `year`, `month`, and `day`.

For detailed syntax and parameter descriptions, see BROKER LOAD.

Load data from a Kafka cluster

If the data of `file2.csv` is published to `topic2` of your Kafka cluster, execute the following statement to create a Routine Load job:

```
CREATE ROUTINE LOAD test_db.table201 ON table2
    COLUMNS TERMINATED BY ",",
    COLUMNS(date,year=year(date),month=month(date),day=day(date))
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic2",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

> **NOTE**
>
> In the `COLUMNS` parameter, you must first temporarily name **all columns** of the data file, and then temporarily name the new columns that you want to generate from the original columns of the data file. As shown in the preceding example, the only column of `file2.csv` is temporarily named as `date`, and then the `year=year(date)`, `month=month(date)`, and `day=day(date)` functions are invoked to generate three new columns, which are temporarily named as `year`, `month`, and `day`.

For detailed syntax and parameter descriptions, see CREATE ROUTINE LOAD.

Query data

After the load of data from your local file system, HDFS cluster, or Kafka cluster is complete, query the data of `table2` to verify that the load is successful:

```
MySQL [test_db]> SELECT * FROM table2;
+------------+------+-------+------+
| date       | year | month | day  |
+------------+------+-------+------+
| 2020-05-20 | 2020 |  5    | 20   |
| 2020-05-21 | 2020 |  5    | 21   |
| 2020-05-22 | 2020 |  5    | 22   |
| 2020-05-23 | 2020 |  5    | 23   |
+------------+------+-------+------+
4 rows in set (0.01 sec)
```

Extract partition field values from a file path

If the file path that you specify contains partition fields, you can use the `COLUMNS FROM PATH AS` parameter to specify the partition fields you want to extract from the file paths. The partition fields in file paths are equivalent to the columns in data files. The `COLUMNS FROM PATH AS` parameter is supported only when you load data from an HDFS cluster.

For example, you want to load the following four data files generated from Hive:

```
/user/starrocks/data/input/date=2020-05-20/data
1,354
/user/starrocks/data/input/date=2020-05-21/data
2,465
/user/starrocks/data/input/date=2020-05-22/data
1,576
/user/starrocks/data/input/date=2020-05-23/data
2,687
```

The four data files are stored in the `/user/starrocks/data/input/` path of your HDFS cluster. Each of these data files is partitioned by partition field `date` and consists of two columns, which represent event type and user ID in sequence.

Load data from an HDFS cluster

Execute the following statement to create a Broker Load job, which enables you to extract the `date` partition field values from the `/user/starrocks/data/input/` file path and use a wildcard (*) to specify that you want to load all data files in the file path to `table1`:

```
LOAD LABEL test_db.label4
(
    DATA INFILE("hdfs://<fe_host>:
<fe_http_port>/user/starrocks/data/input/date=*/*")
    INTO TABLE `table1`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (event_type, user_id)
    COLUMNS FROM PATH AS (date)
    SET(event_date = date)
)
WITH BROKER;
```

> **NOTE**
>
> In the preceding example, the `date` partition field in the specified file path is equivalent to the `event_date` column of `table1`. Therefore, you need to use the SET clause to map the `date` partition field onto the `event_date` column. If the partition field in the specified file path has the same name as a column of the StarRocks table, you do not need to use the SET clause to create a mapping.

For detailed syntax and parameter descriptions, see BROKER LOAD.

Query data

After the load of data from your HDFS cluster is complete, query the data of `table1` to verify that the load is successful:

```
MySQL [test_db]> SELECT * FROM table1;
+------------+-----------+---------+
```

```
| event_date | event_type | user_id |
+------------+------------+---------+
| 2020-05-22 |          1 |     576 |
| 2020-05-20 |          1 |     354 |
| 2020-05-21 |          2 |     465 |
| 2020-05-23 |          2 |     687 |
+------------+------------+---------+
4 rows in set (0.01 sec)
```

Strict mode

Strict mode is an optional property that you can configure for data loads. It affects the loading behavior and the final loaded data.

This topic introduces what strict mode is and how to set strict mode.

Understand strict mode

During data loading, the data types of the source columns may not be completely consistent with the data types of the destination columns. In such cases, StarRocks performs conversions on the source column values that have inconsistent data types. Data conversions may fail due to various issues such as unmatched field data types and field length overflows. Source column values that fail to be properly converted are unqualified column values, and source rows that contain unqualified column values are referred to as "unqualified rows". Strict mode is used to control whether to filter out unqualified rows during data loading.

Strict mode works as follows:

- If strict mode is enabled, StarRocks loads only qualified rows. It filters out unqualified rows and returns details about the unqualified rows.
- If strict mode is disabled, StarRocks converts unqualified column values into `NULL` and loads unqualified rows that contain these `NULL` values together with qualified rows.

Note the following points:

- In actual business scenarios, both qualified and unqualified rows may contain `NULL` values. If the destination columns do not allow `NULL` values, StarRocks reports errors and filters out the rows that contain `NULL` values.

- The maximum percentage of unqualified rows that can be filtered out for a Stream Load, Broker Load, Routine Load, or Spark Load job is controlled by an optional job property `max_filter_ratio`. INSERT does not support setting the `max_filter_ratio` property.

For example, you want to load four rows that hold `\N` (`\N` denotes a `NULL` value), `abc`, `2000`, and `1` values respectively in a column from a CSV-formatted data file into a StarRocks table, and the data type of the destination StarRocks table column is TINYINT [-128, 127].

- The source column value `\N` is processed into `NULL` upon conversion to TINYINT.

  > **NOTE**
  >
  > `\N` is always processed into `NULL` upon conversion regardless of the destination data type.

- The source column value `abc` is processed into `NULL`, because its data type is not TINYINT and the conversion fails.
- The source column value `2000` is processed into `NULL`, because it is beyond the range supported by TINYINT and the conversion fails.
- The source column value `1` can be properly converted to a TINYINT-type value `1`.

If strict mode is disabled, StarRocks loads all the four rows.

If strict mode is enabled, StarRocks loads only the rows that hold `\N` or `1` and filters out the rows that hold `abc` or `2000`. The rows filtered out are counted against the maximum percentage of rows that can be filtered out due to inadequate data quality as specified by the `max_filter_ratio` parameter.

Final loaded data with strict mode disabled

Source column value	Column value upon conversion to TINYINT	Load result when destination column allows NULL values	Load result when destination column does not allow NULL values
\N	NULL	The value `NULL` is loaded.	An error is reported.
abc	NULL	The value `NULL` is loaded.	An error is reported.

Source column value	Column value upon conversion to TINYINT	Load result when destination column allows NULL values	Load result when destination column does not allow NULL values
2000	NULL	The value `NULL` is loaded.	An error is reported.
1	1	The value `1` is loaded.	The value `1` is loaded.

Final loaded data with strict mode enabled

Source column value	Column value upon conversion to TINYINT	Load result when destination column allows NULL values	Load result when destination column does not allow NULL values
\N	NULL	The value `NULL` is loaded.	An error is reported.
abc	NULL	The value `NULL` is not allowed and therefore is filtered out.	An error is reported.
2000	NULL	The value `NULL` is not allowed and therefore is filtered out.	An error is reported.
1	1	The value `1` is loaded.	The value `1` is loaded.

Set strict mode

If you run a Stream Load, Broker Load, Routine Load, or Spark Load job to load data, use the `strict_mode` parameter to set strict mode for the load job. Valid values are `true` and `false`. The default value is `false`. The value `true` enables strict mode, and the value `false` disables strict mode.

If you execute INSERT to load data, use the `enable_insert_strict` session variable to set strict mode. Valid values are `true` and `false`. The default value is `true`. The value `true` enables strict mode, and the value `false` disables strict mode.

Examples are as follows:

Stream Load

```
curl --location-trusted -u <username>:<password> \
    -H "strict_mode: {true | false}" \
    -T <file_name> -XPUT \
    http://<fe_host>:
<fe_http_port>/api/<database_name>/<table_name>/_stream_load
```

For detailed syntax and parameters about Stream Load, see STREAM LOAD.

Broker Load

```
LOAD LABEL [<database_name>.]<label_name>
(
    DATA INFILE ("<file_path>"[, "<file_path>" ...])
    INTO TABLE <table_name>
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "strict_mode" = "{true | false}"
)
```

The preceding code snippet uses HDFS as an example. For detailed syntax and parameters about Broker Load, see BROKER LOAD.

Routine Load

```
CREATE ROUTINE LOAD [<database_name>.]<job_name> ON <table_name>
PROPERTIES
(
    "strict_mode" = "{true | false}"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>[,
<kafka_broker2_ip>:<kafka_broker2_port>...]",
    "kafka_topic" = "<topic_name>"
)
```

The preceding code snippet uses Apache Kafka® as an example. For detailed syntax and parameters about Routine Load, see CREATE ROUTINE LOAD.

Spark Load

```
LOAD LABEL [<database_name>.]<label_name>
(
    DATA INFILE ("<file_path>"[, "<file_path>" ...])
    INTO TABLE <table_name>
)
WITH RESOURCE <resource_name>
(
    "spark.executor.memory" = "3g",
    "broker.username" = "<hdfs_username>",
    "broker.password" = "<hdfs_password>"
)
PROPERTIES
(
    "strict_mode" = "{true | false}"
)
```

The preceding code snippet uses HDFS as an example. For detailed syntax and parameters about Spark Load, see SPARK LOAD.

INSERT

```
SET enable_insert_strict = {true | false};
INSERT INTO <table_name> ...
```

For detailed syntax and parameters about INSERT, see INSERT.

Export data using EXPORT

This topic describes how to export data from specified tables or partitions in your StarRocks cluster as CSV data files to an external storage system, which can be a distributed file system HDFS or a cloud storage system such as AWS S3.

Background information

In v2.4 and earlier, StarRocks depends on brokers to set up connections between your StarRocks cluster and your external storage system when it uses the EXPORT statement to export data. Therefore, you need to input `WITH BROKER "<broker_name>"` to specify the broker you want to use in the EXPORT statement. This is called "broker-based unloading." A broker is an independent, stateless service that is integrated with a file-system interface, helping StarRocks export data to your external storage system.

From v2.5 onwards, StarRocks no longer depends on brokers to set up connections between your StarRocks cluster and your external storage system when it uses the EXPORT statement to export data. Therefore, you no longer need to specify a broker in the EXPORT statement, but you still need to retain the `WITH BROKER` keyword. This is called "broker-free unloading."

When your data is stored in HDFS, however, broker-free unloading may not work and you can resort to broker-based unloading:

- If you export data to multiple HDFS clusters, you need to deploy and configure an independent broker for each of these HDFS clusters.
- If you export data to a single HDFS cluster and you have configured multiple Kerberos users, you need to deploy one independent broker.

> **NOTE**
>
> You can use the SHOW BROKER statement to check for brokers that are deployed in your StarRocks cluster. If no brokers are deployed, you can deploy brokers by following the instructions provided in Deploy a broker.

Supported storage systems

- Distributed file system HDFS
- Cloud storage system such as AWS S3

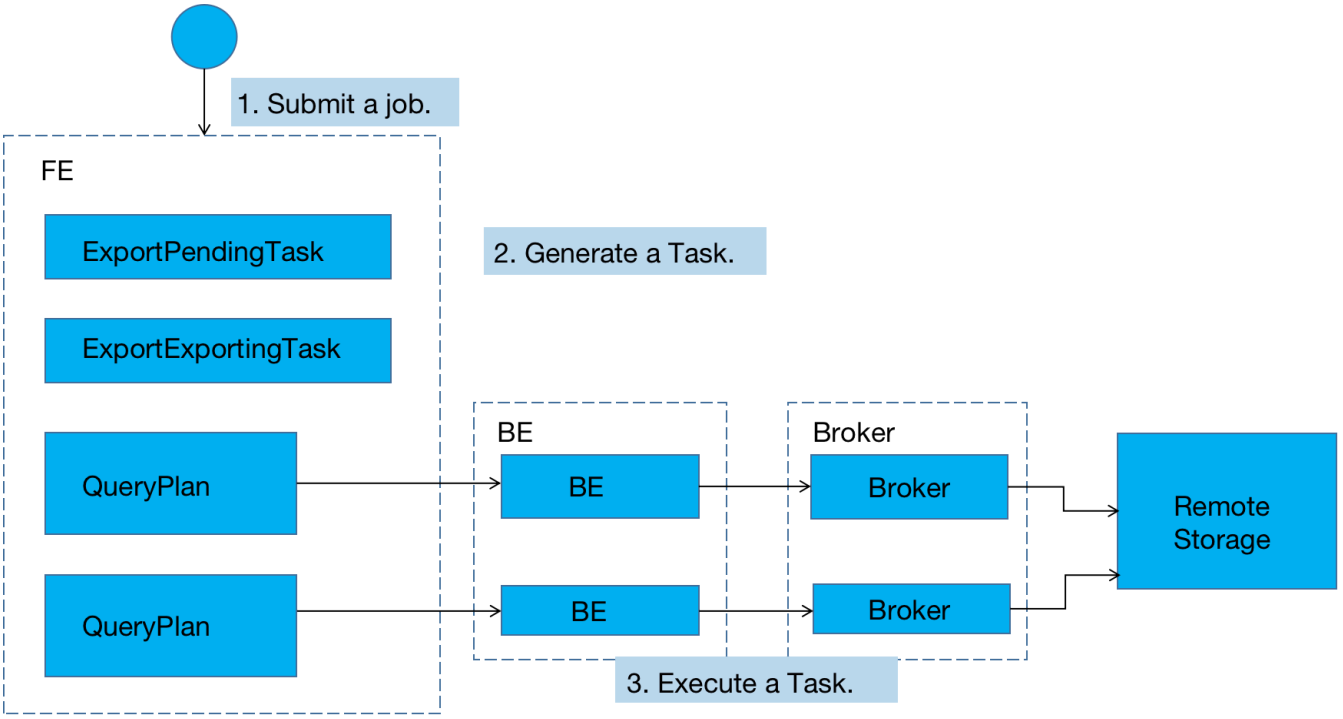
Precautions

- We recommend that the data you export at a time do not exceed a few dozen GB. If you export an exceedingly large amount of data at a time, the export may fail and the cost of retrying the export increases.
- If the source StarRocks table contains a large amount of data, we recommend that you export data from only a few partitions of the table each time until all data from the table is exported.
- If the FEs in your StarRocks cluster restart or a new leader FE is elected when an export job is running, the export job fails. In this situation, you must submit the export job again.
- If the FEs in your StarRocks cluster restart or a new leader FE is elected after an export job finishes, some of the job information returned by the SHOW EXPORT statement may be lost.
- StarRocks exports only the data of base tables. StarRocks does not export the data of materialized views created on base tables.
- Export jobs require data scans, which occupy I/O resources and consequently increase query latency.

Workflow

After you submit an export job, StarRocks identifies all tablets involved in the export job. Then, StarRocks divides the involved tablets into groups and generates query plans. The query plans are used to read data from the involved tablets and to write the data to a specified path of the destination storage system.

The following figure shows the general workflow.



The general workflow consists of the following three steps:

1. The user submits an export job to the leader FE.
2. The leader FE issues `snapshot` instructions to all BEs in the StarRocks cluster, so the BEs can take snapshots of the involved tablets to ensure the consistency of the data to be exported. The leader FE also generates multiple export tasks. Each export task is a query plan, and each query plan is used to process a portion of the involved tablets.
3. The leader FE distributes the export tasks to the BEs.

Principles

When StarRocks execute query plans, it first creates a temporary folder named `__starrocks_export_tmp_xxx` in the specified path of the destination storage system. In the name of the temporary folder, `xxx` represents the ID of the export job. An example name for the temporary folder is `__starrocks_export_tmp_921d8f80-7c9d-11eb-9342-acde48001122`. After StarRocks successfully executes a query plan, it generates a temporary file in the temporary folder and writes the exported data into the generated temporary file.

After all data is exported, StarRocks uses the RENAME statement to save the generated temporary files to a specified path.

Related parameters

This section describes some export-related parameters that you can configure in the FEs of your StarRocks cluster.

- `export_checker_interval_second`: the interval at which export jobs are scheduled. The default interval is 5 seconds. After you reconfigure this parameter for an FE, you need to restart the FE to make the new parameter setting take effect.
- `export_running_job_num_limit`: the maximum number of running export jobs that are allowed. If the number of running export jobs exceeds this limit, excessive export jobs enter a waiting state after `snapshot` is run. The default maximum number is 5. You can reconfigure this parameter when export jobs are running.
- `export_task_default_timeout_second`: the timeout period for export jobs. The default timeout period is 2 hours. You can reconfigure this parameter when export jobs are running.
- `export_max_bytes_per_be_per_task`: the maximum amount of data as compressed that can be exported per export task from each BE. This parameter provides a policy based on which StarRocks splits export jobs into export tasks that can be concurrently run. The default maximum amount is 256 MB.
- `export_task_pool_size`: the maximum number of export tasks that can be concurrently run by the thread pool. The default maximum number is 5.

Basic operations

Submit an export job

Suppose that your StarRocks database `db1` contains a table named `tbl1`. To export the data of columns `col1` and `col3` from partitions `p1` and `p2` of `tbl1` to the `export` path of your HDFS cluster, run the following command:

```
EXPORT TABLE db1.tbl1
PARTITION (p1,p2)
(col1, col3)
TO "hdfs://HDFS_IP:HDFS_Port/export/lineorder_"
```

```
PROPERTIES
(
    "column_separator"=",",
    "load_mem_limit"="2147483648",
    "timeout" = "3600"
)
WITH BROKER
(
    "username" = "user",
    "password" = "passwd"
);
```

For detailed syntax and parameter descriptions as well as the command examples of exporting data to AWS S3, see EXPORT.

Obtain the query ID of an export job

After you submit an export job, you can use the SELECT LAST_QUERY_ID() statement to query the query ID of the export job. With the query ID, you can view or cancel the export job.

For the detailed syntax and parameter descriptions, see last_query_id.

View the status of an export job

After you submit an export job, you can use the SHOW EXPORT statement to view the status of the export job. Example:

```
SHOW EXPORT WHERE queryid = "edee47f0-abe1-11ec-b9d1-00163e1e238f";
```

> **NOTE**
>
> In the preceding example, `queryid` is the query ID of the export job.

Information similar to the following output is returned:

```
JobId: 14008
State: FINISHED
Progress: 100%
TaskInfo: {"partitions":["*"],"mem limit":2147483648,"column
separator":",","line delimiter":"\n","tablet
```

```
num":1,"broker":"hdfs","coord
num":1,"db":"default_cluster:db1","tbl":"tbl3",columns:["col1",
"col3"]}
Path: oss://bj-test/export/
CreateTime: 2019-06-25 17:08:24
StartTime: 2019-06-25 17:08:28
FinishTime: 2019-06-25 17:08:34
Timeout: 3600
ErrorMsg: N/A
```

For detailed syntax and parameter descriptions, see SHOW EXPORT.

Cancel an export job

You can use the CANCEL EXPORT statement to cancel an export job that you have submitted. Example:

```
CANCEL EXPORT WHERE queryid = "921d8f80-7c9d-11eb-9342-acde48001122";
```

> **NOTE**
>
> In the preceding example, `queryid` is the query ID of the export job.

For detailed syntax and parameter descriptions, see CANCEL EXPORT.

Best practices

Query plan splitting

The number of query plans into which an export job is split varies depending on the number of tablets involved in the export job and on the maximum amount of data that can be processed per query plan. Export jobs are retried as query plans. If the amount of data processed by a query plan exceeds the maximum amount allowed, the query plan encounters errors such as jitters in remote storage. As a result, the cost of retrying the query plan increases. The maximum amount of data that can be processed per query plan by each BE is specified by the `export_max_bytes_per_be_per_task` parameter, which defaults to 256 MB. In a query plan, each BE is allocated at least one tablet and can export a

data amount that does not exceed the limit specified by the `export_max_bytes_per_be_per_task` parameter.

The multiple query plans of an export job are concurrently executed. You can use the FE parameter `export_task_pool_size` to specify the maximum number of export tasks that are allowed to concurrently run by the thread pool. This parameter defaults to `5`.

In normal cases, each query plan of an export job consists of only two parts: scanning and exporting. The logic for performing computations required by query plans does not consume much memory. Therefore, the default memory limit of 2 GB can meet most of your business requirements. However, in certain circumstances, such as when a query plan requires scanning many tablets on a BE or a tablet has many versions, the 2-GB memory capacity may be insufficient. In these circumstances, you need to use the `load_mem_limit` parameter to specify a higher memory capacity limit, such as 4 GB or 8 GB.

Read data from StarRocks using Spark connector

StarRocks provides a self-developed connector named StarRocks Connector for Apache Spark™ (Spark connector for short) to help you read data from a StarRocks table by using Spark. You can use Spark for complex processing and machine learning on the data you have read from StarRocks.

The Spark connector supports three reading methods: Spark SQL, Spark DataFrame, and Spark RDD.

You can use Spark SQL to create a temporary view on the StarRocks table, and then directly read data from the StarRocks table by using that temporary view.

You can also map the StarRocks table to a Spark DataFrame or a Spark RDD, and then read data from the Spark DataFrame or Spark RDD. We recommend the use of a Spark DataFrame.

> **NOTICE**
>
> Only users with the SELECT privilege on a StarRocks table can read data from this table. You can follow the instructions provided in GRANT to grant the privilege to a user.

Usage notes

- You can filter data on StarRocks before you read the data, thereby reducing the amount of data transferred.
- If the overhead of reading data is substantial, you can employ appropriate table design and filter conditions to prevent Spark from reading an excessive amount of data at a time. As such, you can reduce I/O pressure on your disk and network connection, thereby ensuring routine queries can be run properly.

Version requirements

Spark connector	Spark	StarRocks	Java	Scala
1.1.2	3.2, 3.3, 3.4, 3.5	2.5 and later	8	2.12
1.1.1	3.2, 3.3, 3.4	2.5 and later	8	2.12
1.1.0	3.2, 3.3, 3.4	2.5 and later	8	2.12
1.0.0	3.x	1.18 and later	8	2.12
1.0.0	2.x	1.18 and later	8	2.11

> **NOTICE**
>
> - Please see Upgrade Spark connector for behaviour changes among different connector versions.
> - The connector does not provide MySQL JDBC driver since version 1.1.1, and you need import the driver to the Spark classpath manually. You can find the driver on Maven Central.
> - In version 1.0.0, the Spark connector only supports reading data from StarRocks. From version 1.1.0 onwards, the Spark connector supports both reading data from and writing data to StarRocks.
> - Version 1.0.0 differs from version 1.1.0 in terms of parameters and data type mappings. See Upgrade Spark connector.
> - In general cases, no new features will be added to version 1.0.0. We recommend that you upgrade your Spark connector at your earliest opportunity.

Obtain Spark connector

Use one of the following methods to obtain the Spark connector **.jar** package that suits your business needs:

- Download a compiled package.
- Use Maven to add the dependencies required by the Spark connector. (This method is supported only for Spark connector 1.1.0 and later.)

- Manually compile a package.

Spark connector 1.1.0 and later

Spark connector **.jar** packages are named in the following format:

`starrocks-spark-connector-${spark_version}_${scala_version}-${connector_version}.jar`

For example, if you want to use Spark connector 1.1.0 with Spark 3.2 and Scala 2.12, you can choose `starrocks-spark-connector-3.2_2.12-1.1.0.jar`.

> **NOTICE**
>
> In normal cases, the latest Spark connector version can be used with the most recent three Spark versions.

Download a compiled package

You can obtain Spark connector **.jar** packages of various versions at Maven Central Repository.

Add Maven dependencies

Configure the dependencies required by the Spark connector as follows:

> **NOTICE**
>
> You must replace `spark_version`, `scala_version`, and `connector_version` with the Spark version, Scala version, and Spark connector version you use.

```
<dependency>
  <groupId>com.starrocks</groupId>
  <artifactId>starrocks-spark-connector-${spark_version}_${scala_version}</artifactId>
  <version>${connector_version}</version>
</dependency>
```

For example, if you want to use Spark connector 1.1.0 with Spark 3.2 and Scala 2.12, configure the dependencies as follows:

```
<dependency>
  <groupId>com.starrocks</groupId>
  <artifactId>starrocks-spark-connector-3.2_2.12</artifactId>
  <version>1.1.0</version>
</dependency>
```

Manually compile a package

1. Download the Spark connector code.
2. Use the following command to compile the Spark connector:

   > **NOTICE**
   >
   > You must replace `spark_version` with the Spark version you use.

   ```
   sh build.sh <spark_version>
   ```

   For example, if you want to use the Spark connector with Spark 3.2, compile the Spark connector as follows:

   ```
   sh build.sh 3.2
   ```

3. Go to the `target/` path, in which a Spark connector **.jar** package like `starrocks-spark-connector-3.2_2.12-1.1.0-SNAPSHOT.jar` is generated upon compilation.

   > **NOTICE**
   >
   > If you are using a Spark connector version that is not officially released, the name of the generated Spark connector **.jar** package contains `SNAPSHOT` as a suffix.

Spark connector 1.0.0

Download a compiled package

- Spark 2.x
- Spark 3.x

Manually compile a package

1. Download the Spark connector code.

   > **NOTICE**
   >
   > You must switch to `spark-1.0`.

2. Take one of the following actions to compile the Spark connector:

   - If you are using Spark 2.x, run the following command, which compiles the Spark connector to suit Spark 2.3.4 by default:

     ```
     sh build.sh 2
     ```

   - If you are using Spark 3.x, run the following command, which compiles the Spark connector to suit Spark 3.1.2 by default:

     ```
     sh build.sh 3
     ```

3. Go to the `output/` path, in which the `starrocks-spark2_2.11-1.0.0.jar` file is generated upon compilation. Then, copy the file to the classpath of Spark:

   - If your Spark cluster runs in `Local` mode, place the file into the `jars/` path.
   - If your Spark cluster runs in `Yarn` mode, place the file into the pre-deployment package.

You can use the Spark connector to read data from StarRocks only after you place the file into the specified location.

Parameters

This section describes the parameters you need to configure when you use the Spark connector to read data from StarRocks.

Common parameters

The following parameters apply to all three reading methods: Spark SQL, Spark DataFrame, and Spark RDD.

Parameter	Default value	Descripti
starrocks.fenodes	None	The HTTP URL of the FE i cluster. Format `<fe_hos` `<fe_http_port>`. You ca multiple URLs, which mus by a comma (,).
starrocks.table.identifier	None	The name of the StarRocl `<database_name>.<tab`
starrocks.request.retries	3	The maximum number of can retry to send a read r StarRocks.
starrocks.request.connect.timeout.ms	30000	The maximum amount of a read request sent to Sta out.
starrocks.request.read.timeout.ms	30000	The maximum amount of the reading for a request StarRocks times out.
starrocks.request.query.timeout.s	3600	The maximum amount of a query of data from Star The default timeout perio means that no timeout pe
starrocks.request.tablet.size	Integer.MAX_VALUE	The number of StarRocks into each Spark RDD part value of this parameter in larger number of Spark R be generated. A larger nu RDD partitions means hig on Spark but greater pres StarRocks.

Parameter	Default value	Descripti
starrocks.batch.size	4096	The maximum number of read from BEs at a time. I value of this parameter ca number of connections es between Spark and StarR mitigating extra time over network latency.
starrocks.exec.mem.limit	2147483648	The maximum amount of per query. Unit: bytes. Th memory limit is 2 GB.
starrocks.deserialize.arrow.async	false	Specifies whether to supp asynchronously convertir memory format to RowBa for the iteration of the Sp
starrocks.deserialize.queue.size	64	The size of the internal qu tasks for asynchronously Arrow memory format to This parameter is valid wh `starrocks.deserializ` is set to `true`.
starrocks.filter.query	None	The condition based on w filter data on StarRocks. Y multiple filter conditions, joined by `and`. StarRocks from the StarRocks table specified filter conditions is read by Spark.
starrocks.timezone	Default timezone of JVM	Supported since 1.1.1. The to convert StarRocks `DAT` `TimestampType`. The de timezone of JVM returned

Parameter	Default value	Descripti
		`ZoneId#systemDefault` could be a timezone nam `Asia/Shanghai`, or a zo `+08:00`.

Parameters for Spark SQL and Spark DataFrame

The following parameters apply only to the Spark SQL and Spark DataFrame reading methods.

Parameter	Default value	Description
starrocks.fe.http.url	None	The HTTP IP address of the FE. This parameter is supported from Spark connector 1.1.0 onwards. This parameter is equivalent to `starrocks.fenodes`. You only need to configure one of them. In Spark connector 1.1.0 and later, we recommend that you use `starrocks.fe.http.url` because `starrocks.fenodes` may be deprecated.
starrocks.fe.jdbc.url	None	The address that is used to connect to the MySQL server of the FE. Format: `jdbc:mysql://<fe_host>:<fe_query_port>`. **NOTICE** In Spark connector 1.1.0 and later, this parameter is mandatory.
user	None	The username of your StarRocks cluster account. The user needs the

Parameter	Default value	Description
		SELECT privilege on the StarRocks table.
starrocks.user	None	The username of your StarRocks cluster account. This parameter is supported from Spark connector 1.1.0 onwards. This parameter is equivalent to `user`. You only need to configure one of them. In Spark connector 1.1.0 and later, we recommend that you use `starrocks.user` because `user` may be deprecated.
password	None	The password of your StarRocks cluster account.
starrocks.password	None	The password of your StarRocks cluster account. This parameter is supported from Spark connector 1.1.0 onwards. This parameter is equivalent to `password`. You only need to configure one of them. In Spark connector 1.1.0 and later, we recommend that you use `starrocks.password` because `password` may be deprecated.
starrocks.filter.query.in.max.count	100	The maximum number of values supported by the IN expression during predicate pushdown. If the number of values specified in the IN expression exceeds this limit, the filter conditions specified in the IN expression are processed on Spark.

Parameters for Spark RDD

The following parameters apply only to the Spark RDD reading method.

Parameter	Default value	Description
starrocks.request.auth.user	None	The username of your StarRocks cluster account.
starrocks.request.auth.password	None	The password of your StarRocks cluster account.
starrocks.read.field	None	The StarRocks table column from which you want to read data. You can specify multiple columns, which must be separated by a comma (,).

Data type mapping between StarRocks and Spark

Spark connector 1.1.0 and later

StarRocks data type	Spark data type
BOOLEAN	DataTypes.BooleanType
TINYINT	DataTypes.ByteType
SMALLINT	DataTypes.ShortType
INT	DataTypes.IntegerType
BIGINT	DataTypes.LongType
LARGEINT	DataTypes.StringType

StarRocks data type	Spark data type
FLOAT	DataTypes.FloatType
DOUBLE	DataTypes.DoubleType
DECIMAL	DecimalType
CHAR	DataTypes.StringType
VARCHAR	DataTypes.StringType
STRING	DataTypes.StringType
DATE	DataTypes.DateType
DATETIME	DataTypes.TimestampType
JSON	DataTypes.StringType **NOTE:** This data type mapping is supported since Spark connector v1.1.2, and requires a StarRocks version of at least 2.5.13, 3.0.3, 3.1.0 or later.
ARRAY	Unsupported datatype
HLL	Unsupported datatype
BITMAP	Unsupported datatype

Spark connector 1.0.0

StarRocks data type	Spark data type
BOOLEAN	DataTypes.BooleanType
TINYINT	DataTypes.ByteType

StarRocks data type	Spark data type
SMALLINT	DataTypes.ShortType
INT	DataTypes.IntegerType
BIGINT	DataTypes.LongType
LARGEINT	DataTypes.StringType
FLOAT	DataTypes.FloatType
DOUBLE	DataTypes.DoubleType
DECIMAL	DecimalType
CHAR	DataTypes.StringType
VARCHAR	DataTypes.StringType
DATE	DataTypes.StringType
DATETIME	DataTypes.StringType
ARRAY	Unsupported datatype
HLL	Unsupported datatype
BITMAP	Unsupported datatype

The processing logic of the underlying storage engine used by StarRocks cannot cover an expected time range when DATE and DATETIME data types are directly used. Therefore, the Spark connector maps the DATE and DATETIME data types from StarRocks to the STRING data type from Spark, and generates readable string texts matching the date and time data read from StarRocks.

Upgrade Spark connector

Upgrade from version 1.0.0 to version 1.1.0

- Since 1.1.1, the Spark connector does not provide `mysql-connector-java` which is the official JDBC driver for MySQL, because of the limitations of the GPL license used by `mysql-connector-java`. However, the Spark connector still needs the `mysql-connector-java` to connect to StarRocks for the table metadata, so you need to add the driver to the Spark classpath manually. You can find the driver on MySQL site or Maven Central.

- In version 1.1.0, the Spark connector uses JDBC to access StarRocks to obtain more detailed table information. Therefore, you must configure `starrocks.fe.jdbc.url`.

- In version 1.1.0, some parameters are renamed. Both the old and new parameters are retained for now. For each pair of equivalent parameters, you only need to configure one of them, but we recommend that you use the new one because the old one may be deprecated.

    - `starrocks.fenodes` is renamed as `starrocks.fe.http.url`.
    - `user` is renamed as `starrocks.user`.
    - `password` is renamed as `starrocks.password`.

- In version 1.1.0, the mappings of some data types are adjusted based on Spark 3.x:

    - `DATE` in StarRocks is mapped to `DataTypes.DateType` (originally `DataTypes.StringType`) in Spark.
    - `DATETIME` in StarRocks is mapped to `DataTypes.TimestampType` (originally `DataTypes.StringType`) in Spark.

Examples

The following examples assume you have created a database named `test` in your StarRocks cluster and you have the permissions of user `root`. The parameter settings in the examples are based on Spark Connector 1.1.0.

Data example

Do as follows to prepare a sample table:

1. Go to the `test` database and create a table named `score_board`.

```
MySQL [test]> 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`) BUCKETS 1
PROPERTIES (
    "replication_num" = "3"
);
```

2. Insert data into the `score_board` table.

```
MySQL [test]> INSERT INTO score_board
VALUES
    (1, 'Bob', 21),
    (2, 'Stan', 21),
    (3, 'Sam', 22),
    (4, 'Tony', 22),
    (5, 'Alice', 22),
    (6, 'Lucy', 23),
    (7, 'Polly', 23),
    (8, 'Tom', 23),
    (9, 'Rose', 24),
    (10, 'Jerry', 24),
    (11, 'Jason', 24),
    (12, 'Lily', 25),
    (13, 'Stephen', 25),
    (14, 'David', 25),
    (15, 'Eddie', 26),
    (16, 'Kate', 27),
    (17, 'Cathy', 27),
    (18, 'Judy', 27),
    (19, 'Julia', 28),
    (20, 'Robert', 28),
    (21, 'Jack', 29);
```

3. Query the `score_board` table.

```
MySQL [test]> SELECT * FROM score_board;
+------+---------+-------+
| id   | name    | score |
+------+---------+-------+
|    1 | Bob     |    21 |
|    2 | Stan    |    21 |
|    3 | Sam     |    22 |
|    4 | Tony    |    22 |
|    5 | Alice   |    22 |
|    6 | Lucy    |    23 |
|    7 | Polly   |    23 |
|    8 | Tom     |    23 |
|    9 | Rose    |    24 |
|   10 | Jerry   |    24 |
|   11 | Jason   |    24 |
|   12 | Lily    |    25 |
|   13 | Stephen |    25 |
|   14 | David   |    25 |
|   15 | Eddie   |    26 |
|   16 | Kate    |    27 |
|   17 | Cathy   |    27 |
|   18 | Judy    |    27 |
|   19 | Julia   |    28 |
|   20 | Robert  |    28 |
|   21 | Jack    |    29 |
+------+---------+-------+
21 rows in set (0.01 sec)
```

Read data using Spark SQL

1. Run the following command in the Spark directory to start Spark SQL:

```
sh spark-sql
```

2. Run the following command to create a temporary view named `spark_starrocks` on the `score_board` table which belongs to the `test` database:

```
spark-sql> CREATE TEMPORARY VIEW spark_starrocks
           USING starrocks
           OPTIONS
           (
```

```
                "starrocks.table.identifier" = "test.score_board",
                "starrocks.fe.http.url" = "<fe_host>:
<fe_http_port>",
                "starrocks.fe.jdbc.url" = "jdbc:mysql://<fe_host>:
<fe_query_port>",
                "starrocks.user" = "root",
                "starrocks.password" = ""
            );
```

3. Run the following command to read data from the temporary view:

```
spark-sql> SELECT * FROM spark_starrocks;
```

Spark returns the following data:

```
1        Bob        21
2        Stan        21
3        Sam        22
4        Tony        22
5        Alice        22
6        Lucy        23
7        Polly        23
8        Tom        23
9        Rose        24
10        Jerry        24
11        Jason        24
12        Lily        25
13        Stephen        25
14        David        25
15        Eddie        26
16        Kate        27
17        Cathy        27
18        Judy        27
19        Julia        28
20        Robert        28
21        Jack        29
Time taken: 1.883 seconds, Fetched 21 row(s)
22/08/09 15:29:36 INFO thriftserver.SparkSQLCLIDriver: Time
taken: 1.883 seconds, Fetched 21 row(s)
```

Read data using Spark DataFrame

1. Run the following command in the Spark directory to start Spark Shell:

```
sh spark-shell
```

2. Run the following command to create a DataFrame named `starrocksSparkDF` on the `score_board` table which belongs to the `test` database:

```
scala> val starrocksSparkDF = spark.read.format("starrocks")
           .option("starrocks.table.identifier",
s"test.score_board")
           .option("starrocks.fe.http.url", s"<fe_host>:
<fe_http_port>")
           .option("starrocks.fe.jdbc.url",
s"jdbc:mysql://<fe_host>:<fe_query_port>")
           .option("starrocks.user", s"root")
           .option("starrocks.password", s"")
           .load()
```

3. Read data from the DataFrame. For example, if you want to read the first 10 rows, run the following command:

```
scala> starrocksSparkDF.show(10)
```

   Spark returns the following data:

```
+---+-----+-----+
| id| name|score|
+---+-----+-----+
|  1|  Bob|   21|
|  2| Stan|   21|
|  3|  Sam|   22|
|  4| Tony|   22|
|  5|Alice|   22|
|  6| Lucy|   23|
|  7|Polly|   23|
|  8|  Tom|   23|
|  9| Rose|   24|
| 10|Jerry|   24|
+---+-----+-----+
only showing top 10 rows
```

> **NOTE**
>
> By default, if you do not specify the number of rows you want to read, Spark returns the first 20 rows.

Read data using Spark RDD

1. Run the following command in the Spark directory to start Spark Shell:

```
sh spark-shell
```

2. Run the following command to create an RDD named `starrocksSparkRDD` on the `score_board` table which belongs to the `test` database.

```
scala> import com.starrocks.connector.spark._
scala> val starrocksSparkRDD = sc.starrocksRDD
           (
           tableIdentifier = Some("test.score_board"),
           cfg = Some(Map(
               "starrocks.fenodes" -> "<fe_host>:<fe_http_port>",
               "starrocks.request.auth.user" -> "root",
               "starrocks.request.auth.password" -> ""
           ))
           )
```

3. Read data from the RDD. For example, if you want to read the first 10 elements, run the following command:

```
scala> starrocksSparkRDD.take(10)
```

   Spark returns the following data:

```
res0: Array[AnyRef] = Array([1, Bob, 21], [2, Stan, 21], [3, Sam,
22], [4, Tony, 22], [5, Alice, 22], [6, Lucy, 23], [7, Polly,
23], [8, Tom, 23], [9, Rose, 24], [10, Jerry, 24])
```

   To read the entire RDD, run the following command:

```
scala> starrocksSparkRDD.collect()
```

Spark returns the following data:

```
res1: Array[AnyRef] = Array([1, Bob, 21], [2, Stan, 21], [3, Sam,
22], [4, Tony, 22], [5, Alice, 22], [6, Lucy, 23], [7, Polly,
23], [8, Tom, 23], [9, Rose, 24], [10, Jerry, 24], [11, Jason,
24], [12, Lily, 25], [13, Stephen, 25], [14, David, 25], [15,
Eddie, 26], [16, Kate, 27], [17, Cathy, 27], [18, Judy, 27], [19,
Julia, 28], [20, Robert, 28], [21, Jack, 29])
```

Best practices

When you read data from StarRocks using the Spark connector, you can use the `starrocks.filter.query` parameter to specify filter conditions based on which Spark prunes partitions, buckets, and prefix indexes to reduce the cost of data pulling. This section uses Spark DataFrame as an example to show how this is achieved.

Environment setup

Component	Version
Spark	Spark 2.4.4 and Scala 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_302)
StarRocks	2.2.0
Spark connector	starrocks-spark2_2.11-1.0.0.jar

Data example

Do as follows to prepare a sample table:

1. Go to the `test` database and create a table named `mytable`.

```
MySQL [test]> CREATE TABLE `mytable`
(
    `k` int(11) NULL COMMENT "bucket",
    `b` int(11) NULL COMMENT "",
    `dt` datetime NULL COMMENT "",
    `v` int(11) NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`k`,`b`, `dt`)
COMMENT "OLAP"
PARTITION BY RANGE(`dt`)
(
    PARTITION p202201 VALUES [('2022-01-01 00:00:00'), ('2022-02-
01 00:00:00')),
    PARTITION p202202 VALUES [('2022-02-01 00:00:00'), ('2022-03-
01 00:00:00')),
    PARTITION p202203 VALUES [('2022-03-01 00:00:00'), ('2022-04-
01 00:00:00'))
)
DISTRIBUTED BY HASH(`k`) BUCKETS 3
PROPERTIES (
    "replication_num" = "3"
);
```

2. Insert data into `mytable`.

```
MySQL [test]> INSERT INTO mytable
VALUES
     (1, 11, '2022-01-02 08:00:00', 111),
     (2, 22, '2022-02-02 08:00:00', 222),
     (3, 33, '2022-03-02 08:00:00', 333);
```

3. Query the `mytable` table.

```
MySQL [test]> select * from mytable;
+------+------+---------------------+------+
| k    | b    | dt                  | v    |
+------+------+---------------------+------+
|    1 |   11 | 2022-01-02 08:00:00 |  111 |
|    2 |   22 | 2022-02-02 08:00:00 |  222 |
|    3 |   33 | 2022-03-02 08:00:00 |  333 |
```

```
+------+------+--------------------+------+
3 rows in set (0.01 sec)
```

Full table scan

1. Run the following command in the Spark directory to create a DataFrame named `df` on `mytable` table which belongs to the `test` database:

```
scala>  val df = spark.read.format("starrocks")
        .option("starrocks.table.identifier", s"test.mytable")
        .option("starrocks.fenodes", s"<fe_host>:<fe_http_port>")
        .option("user", s"root")
        .option("password", s"")
        .load()
```

2. View the FE log file **fe.log** of your StarRocks cluster, and find the SQL statement executed to read data. Example:

```
2022-08-09 18:57:38,091 INFO (nioEventLoopGroup-3-10|196)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable`] from
external service [ user ['root'@'%']] for database [test] table
[mytable]
```

3. In the `test` database, use EXPLAIN to obtain the execution plan of the SELECT `k`,`b`,`dt`,`v` from `test`.`mytable` statement:

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from
`test`.`mytable`;
+-----------------------------------------------------------------
-------+
| Explain String
|
+-----------------------------------------------------------------
-------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v
|
|   PARTITION: UNPARTITIONED
```

```
|
|
|
|   RESULT SINK
|
|
|
|   1:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 01
|
|     UNPARTITIONED
|
|
|
|   0:OlapScanNode
|
|      TABLE: mytable
|
|      PREAGGREGATION: ON
|
|      partitions=3/3
|
|      rollup: mytable
|
|      tabletRatio=9/9
|
|
tabletList=41297,41299,41301,41303,41305,41307,41309,41311,41313
|
|      cardinality=3
|
|      avgRowSize=4.0
|
```

```
|      numNodes=0
|
+-----------------------------------------------------------------
-------+
26 rows in set (0.00 sec)
```

In this example, no pruning is performed. Therefore, Spark scans all of the three partitions (as suggested by `partitions=3/3`) that hold data, and scans all of the 9 tablets (as suggested by `tabletRatio=9/9`) in those three partitions.

Partition pruning

1. Run the following command, in which you use the `starrocks.filter.query` parameter to specify a filter condition `dt='2022-01-02 08:00:00` for partition pruning, in the Spark directory to create a DataFrame named `df` on the `mytable` table which belongs to the `test` database:

```
scala> val df = spark.read.format("starrocks")
       .option("starrocks.table.identifier", s"test.mytable")
       .option("starrocks.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"root")
       .option("password", s"")
       .option("starrocks.filter.query", "dt='2022-01-02
08:00:00'")
       .load()
```

2. View the FE log file **fe.log** of your StarRocks cluster, and find the SQL statement executed to read data. Example:

```
2022-08-09 19:02:31,253 INFO (nioEventLoopGroup-3-14|204)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable` where
dt='2022-01-02 08:00:00'] from external service [ user
['root'@'%']] for database [test] table [mytable]
```

3. In the `test` database, use EXPLAIN to obtain the execution plan of the SELECT `k`, `b`, `dt`, `v` from `test`.`mytable` where dt='2022-01-02 08:00:00' statement:

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from
`test`.`mytable` where dt='2022-01-02 08:00:00';
+-----------------------------------------------+
| Explain String                                |
+-----------------------------------------------+
| PLAN FRAGMENT 0                               |
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v      |
|   PARTITION: UNPARTITIONED                    |
|                                               |
|   RESULT SINK                                 |
|                                               |
|   1:EXCHANGE                                  |
|                                               |
| PLAN FRAGMENT 1                               |
|  OUTPUT EXPRS:                                |
|   PARTITION: RANDOM                           |
|                                               |
|   STREAM DATA SINK                            |
|     EXCHANGE ID: 01                           |
|     UNPARTITIONED                             |
|                                               |
|   0:OlapScanNode                              |
|      TABLE: mytable                           |
|      PREAGGREGATION: ON                       |
|      PREDICATES: 3: dt = '2022-01-02 08:00:00' |
|      partitions=1/3                           |
|      rollup: mytable                          |
|      tabletRatio=3/3                          |
|      tabletList=41297,41299,41301             |
|      cardinality=1                            |
|      avgRowSize=20.0                          |
|      numNodes=0                               |
+-----------------------------------------------+
27 rows in set (0.01 sec)
```

In this example, only partition pruning is performed, whereas bucket pruning is not. Therefore, Spark scans one of the three partitions (as suggested by `partitions=1/3`) and all of the tablets (as suggested by `tabletRatio=3/3`) in that partition.

Bucket pruning

1. Run the following command, in which you use the `starrocks.filter.query` parameter to specify a filter condition `k=1` for bucket pruning, in the Spark directory to

create a DataFrame named `df` on the `mytable` table which belongs to the `test` database:

```
scala> val df = spark.read.format("starrocks")
       .option("starrocks.table.identifier", s"test.mytable")
       .option("starrocks.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"root")
       .option("password", s"")
       .option("starrocks.filter.query", "k=1")
       .load()
```

2. View the FE log file **fe.log** of your StarRocks cluster, and find the SQL statement executed to read data. Example:

```
2022-08-09 19:04:44,479 INFO (nioEventLoopGroup-3-16|208)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable` where
k=1] from external service [ user ['root'@'%']] for database
[test] table [mytable]
```

3. In the `test` database, use EXPLAIN to obtain the execution plan of the SELECT `k`, `b`, `dt`, `v` from `test`.`mytable` where k=1 statement:

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from
`test`.`mytable` where k=1;
+-----------------------------------------+
| Explain String                          |
+-----------------------------------------+
| PLAN FRAGMENT 0                         |
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v |
|   PARTITION: UNPARTITIONED              |
|                                         |
|   RESULT SINK                           |
|                                         |
|   1:EXCHANGE                            |
|                                         |
| PLAN FRAGMENT 1                         |
|  OUTPUT EXPRS:                          |
|   PARTITION: RANDOM                     |
|                                         |
|   STREAM DATA SINK                      |
|     EXCHANGE ID: 01                     |
```

```
|     UNPARTITIONED                          |
|                                            |
|   0:OlapScanNode                           |
|      TABLE: mytable                        |
|      PREAGGREGATION: ON                    |
|      PREDICATES: 1: k = 1                  |
|      partitions=3/3                        |
|      rollup: mytable                       |
|      tabletRatio=3/9                       |
|      tabletList=41299,41305,41311          |
|      cardinality=1                         |
|      avgRowSize=20.0                       |
|      numNodes=0                            |
+--------------------------------------------+
27 rows in set (0.01 sec)
```

In this example, only bucket pruning is performed, whereas partition pruning is not. Therefore, Spark scans all of the three partitions (as suggested by `partitions=3/3`) that hold data, and scans all of the three tablets (as suggested by `tabletRatio=3/9`) to retrieve Hash values that meet the `k = 1` filter condition within those three partitions.

Partition pruning and bucket pruning

1. Run the following command, in which you use the `starrocks.filter.query` parameter to specify two filter conditions `k=7` and `dt='2022-01-02 08:00:00'` for bucket pruning and partition pruning, in the Spark directory to create a DataFrame named `df` on the `mytable` table on the `test` database:

```
scala> val df = spark.read.format("starrocks")
       .option("starrocks.table.identifier", s"test.mytable")
       .option("starrocks.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"")
       .option("password", s"")
       .option("starrocks.filter.query", "k=7 and dt='2022-01-02
08:00:00'")
       .load()
```

2. View the FE log file **fe.log** of your StarRocks cluster, and find the SQL statement executed to read data. Example:

```
2022-08-09 19:06:34,939 INFO (nioEventLoopGroup-3-18|212)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable` where
k=7 and dt='2022-01-02 08:00:00'] from external service [ user
['root'@'%']] for database [test] t
able [mytable]
```

3. In the `test` database, use EXPLAIN to obtain the execution plan of the SELECT `k`, `b`, `dt`, `v` from `test`.`mytable` where k=7 and dt='2022-01-02 08:00:00' statement:

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from
`test`.`mytable` where k=7 and dt='2022-01-02 08:00:00';
+-----------------------------------------------------------------+
| Explain String                                                  |
+-----------------------------------------------------------------+
| PLAN FRAGMENT 0                                                 |
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v                      |
|   PARTITION: RANDOM                                             |
|                                                                 |
|   RESULT SINK                                                   |
|                                                                 |
|   0:OlapScanNode                                                |
|      TABLE: mytable                                             |
|      PREAGGREGATION: ON                                         |
|      PREDICATES: 1: k = 7, 3: dt = '2022-01-02 08:00:00'        |
|      partitions=1/3                                             |
|      rollup: mytable                                            |
|      tabletRatio=1/3                                            |
|      tabletList=41301                                           |
|      cardinality=1                                              |
|      avgRowSize=20.0                                            |
|      numNodes=0                                                 |
+-----------------------------------------------------------------+
17 rows in set (0.00 sec)
```

In this example, both partition pruning and bucket pruning are performed. Therefore, Spark scans only one of the three partitions (as suggested by `partitions=1/3`) and only one tablet (as suggested by `tabletRatio=1/3`) in that partition.

Prefix index filtering

1. Insert more data records into a partition of the `mytable` table which belongs to the `test` database:

```
MySQL [test]> INSERT INTO mytable
VALUES
    (1, 11, "2022-01-02 08:00:00", 111),
    (3, 33, "2022-01-02 08:00:00", 333),
    (3, 33, "2022-01-02 08:00:00", 333),
    (3, 33, "2022-01-02 08:00:00", 333);
```

2. Query the `mytable` table:

```
MySQL [test]> SELECT * FROM mytable;
+------+------+---------------------+------+
| k    | b    | dt                  | v    |
+------+------+---------------------+------+
|    1 |   11 | 2022-01-02 08:00:00 |  111 |
|    1 |   11 | 2022-01-02 08:00:00 |  111 |
|    3 |   33 | 2022-01-02 08:00:00 |  333 |
|    3 |   33 | 2022-01-02 08:00:00 |  333 |
|    3 |   33 | 2022-01-02 08:00:00 |  333 |
|    2 |   22 | 2022-02-02 08:00:00 |  222 |
|    3 |   33 | 2022-03-02 08:00:00 |  333 |
+------+------+---------------------+------+
7 rows in set (0.01 sec)
```

3. Run the following command, in which you use the `starrocks.filter.query` parameter to specify a filter condition `k=1` for prefix index filtering, in the Spark directory to create a DataFrame named `df` on the `mytable` table which belongs to the `test` database:

```
scala> val df = spark.read.format("starrocks")
       .option("starrocks.table.identifier", s"test.mytable")
       .option("starrocks.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"root")
       .option("password", s"")
       .option("starrocks.filter.query", "k=1")
       .load()
```

4. In the `test` database, set `is_report_success` to `true` to enable profile reporting:

```
MySQL [test]> SET is_report_success = true;
Query OK, 0 rows affected (0.00 sec)
```

5. Use a browser to open the `http://<fe_host>:<http_http_port>/query` page, and view the profile of the SELECT * FROM mytable where k=1 statement. Example:

```
OLAP_SCAN (plan_node_id=0):
  CommonMetrics:
     - CloseTime: 1.255ms
     - OperatorTotalTime: 1.404ms
     - PeakMemoryUsage: 0.00
     - PullChunkNum: 8
     - PullRowNum: 2
       - __MAX_OF_PullRowNum: 2
       - __MIN_OF_PullRowNum: 0
     - PullTotalTime: 148.60us
     - PushChunkNum: 0
     - PushRowNum: 0
     - PushTotalTime: 0ns
     - SetFinishedTime: 136ns
     - SetFinishingTime: 129ns
  UniqueMetrics:
     - Predicates: 1: k = 1
     - Rollup: mytable
     - Table: mytable
     - BytesRead: 88.00 B
       - __MAX_OF_BytesRead: 88.00 B
       - __MIN_OF_BytesRead: 0.00
     - CachedPagesNum: 0
     - CompressedBytesRead: 844.00 B
       - __MAX_OF_CompressedBytesRead: 844.00 B
       - __MIN_OF_CompressedBytesRead: 0.00
     - CreateSegmentIter: 18.582us
     - IOTime: 4.425us
     - LateMaterialize: 17.385us
     - PushdownPredicates: 3
     - RawRowsRead: 2
       - __MAX_OF_RawRowsRead: 2
       - __MIN_OF_RawRowsRead: 0
     - ReadPagesNum: 12
       - __MAX_OF_ReadPagesNum: 12
       - __MIN_OF_ReadPagesNum: 0
     - RowsRead: 2
       - __MAX_OF_RowsRead: 2
```

```
      - __MIN_OF_RowsRead: 0
    - ScanTime: 154.367us
    - SegmentInit: 95.903us
      - BitmapIndexFilter: 0ns
      - BitmapIndexFilterRows: 0
      - BloomFilterFilterRows: 0
      - ShortKeyFilterRows: 3
        - __MAX_OF_ShortKeyFilterRows: 3
        - __MIN_OF_ShortKeyFilterRows: 0
      - ZoneMapIndexFilterRows: 0
    - SegmentRead: 2.559us
      - BlockFetch: 2.187us
      - BlockFetchCount: 2
        - __MAX_OF_BlockFetchCount: 2
        - __MIN_OF_BlockFetchCount: 0
      - BlockSeek: 7.789us
      - BlockSeekCount: 2
        - __MAX_OF_BlockSeekCount: 2
        - __MIN_OF_BlockSeekCount: 0
      - ChunkCopy: 25ns
      - DecompressT: 0ns
      - DelVecFilterRows: 0
      - IndexLoad: 0ns
      - PredFilter: 353ns
      - PredFilterRows: 0
      - RowsetsReadCount: 7
      - SegmentsReadCount: 3
        - __MAX_OF_SegmentsReadCount: 2
        - __MIN_OF_SegmentsReadCount: 0
      - TotalColumnsDataPageCount: 8
        - __MAX_OF_TotalColumnsDataPageCount: 8
        - __MIN_OF_TotalColumnsDataPageCount: 0
    - UncompressedBytesRead: 508.00 B
      - __MAX_OF_UncompressedBytesRead: 508.00 B
      - __MIN_OF_UncompressedBytesRead: 0.00
```

In this example, the filter condition `k = 1` can hit the prefix index. Therefore, Spark can filter out three rows (as suggested by `ShortKeyFilterRows: 3`).

Read data from StarRocks using Flink connector

StarRocks provides a self-developed connector named StarRocks Connector for Apache Flink® (Flink connector for short) to help you read data in bulk from a StarRocks cluster by using Flink.

The Flink connector supports two reading methods: Flink SQL and Flink DataStream. Flink SQL is recommended.

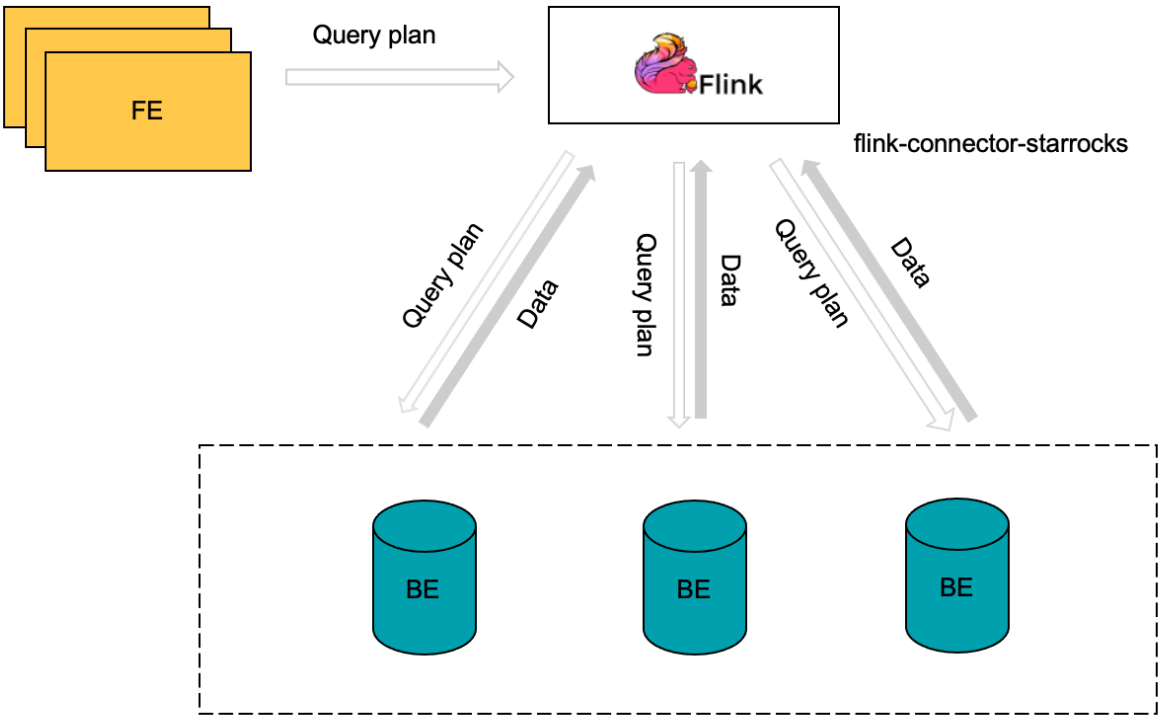
> **NOTE**
>
> The Flink connector also supports writing the data read by Flink to another StarRocks cluster or storage system. See Continuously load data from Apache Flink®.

Background information

Unlike the JDBC connector provided by Flink, the Flink connector of StarRocks supports reading data from multiple BEs of your StarRocks cluster in parallel, greatly accelerating read tasks. The following comparison shows the difference in implementation between the two connectors.

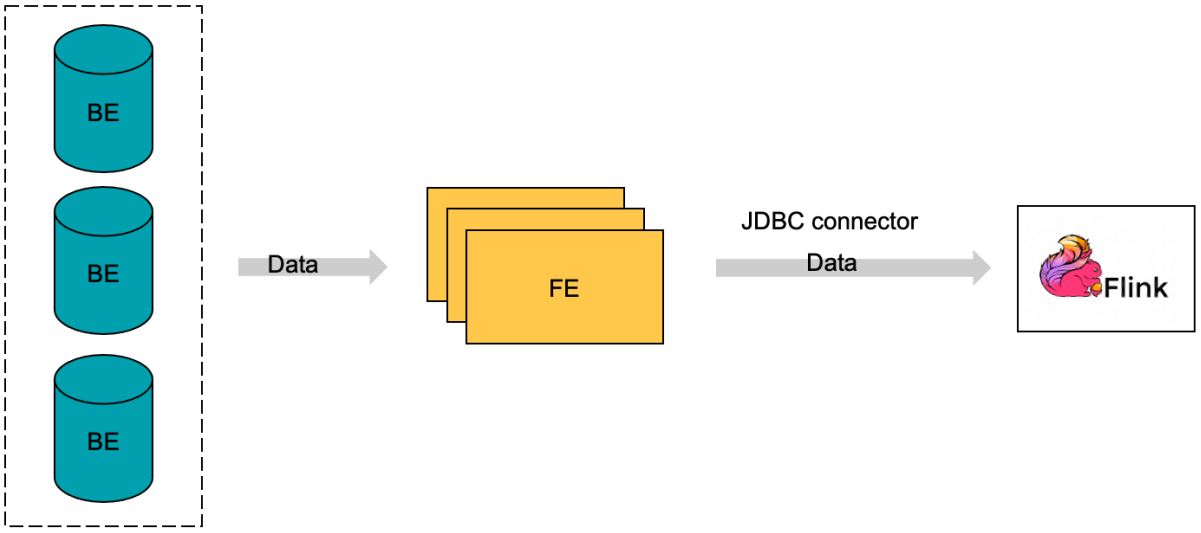
- Flink connector of StarRocks

  With the Flink connector of StarRocks, Flink can first obtain the query plan from the responsible FE, then distribute the obtained query plan as parameters to all the involved BEs, and finally obtain the data returned by the BEs.



- JDBC connector of Flink

  With the JDBC connector of Flink, Flink can only read data from individual FEs, one at a time. Data reads are slow.



Version requirements

Connector	Flink	StarRocks	Java	Scala
1.2.9	1.15,1.16,1.17,1.18	2.1 and later	8	2.11,2.12
1.2.8	1.13,1.14,1.15,1.16,1.17	2.1 and later	8	2.11,2.12
1.2.7	1.11,1.12,1.13,1.14,1.15	2.1 and later	8	2.11,2.12

Prerequisites

Flink has been deployed. If Flink has not been deployed, follow these steps to deploy it:

1. Install Java 8 or Java 11 in your operating system to ensure Flink can run properly. You can use the following command to check the version of your Java installation:

   ```
   java -version
   ```

   For example, if the following information is returned, Java 8 has been installed:

   ```
   openjdk version "1.8.0_322"
   OpenJDK Runtime Environment (Temurin)(build 1.8.0_322-b06)
   OpenJDK 64-Bit Server VM (Temurin)(build 25.322-b06, mixed mode)
   ```

2. Download and unzip the Flink package of your choice.

   > **NOTE**
   >
   > We recommend that you use Flink v1.14 or later. The minimum Flink version supported is v1.11.

   ```
   # Download the Flink package.
   wget https://dlcdn.apache.org/flink/flink-1.14.5/flink-1.14.5-bin-scala_2.11.tgz
   # Unzip the Flink package.
   tar -xzf flink-1.14.5-bin-scala_2.11.tgz
   # Go to the Flink directory.
   cd flink-1.14.5
   ```

3. Start your Flink cluster.

```
# Start your Flink cluster.
./bin/start-cluster.sh

# When the following information is displayed, your Flink cluster
has successfully started:
Starting cluster.
Starting standalonesession daemon on host.
Starting taskexecutor daemon on host.
```

You can also deploy Flink by following the instructions provided in Flink documentation.

Before you begin

Follow these steps to deploy the Flink connector:

1. Select and download the flink-connector-starrocks JAR package matching the Flink version that you are using.

   > **NOTICE**
   >
   > We recommend that you download the Flink connector package whose version is 1.2.x or later and whose matching Flink version has the same first two digits as the Flink version that you are using. For example, if you use Flink v1.14.x, you can download `flink-connector-starrocks-1.2.4_flink-1.14_x.yy.jar`.

2. If code debugging is needed, compile the Flink connector package to suit your business requirements.

3. Place the Flink connector package you downloaded or compiled into the `lib` directory of Flink.

4. Restart your Flink cluster.

Parameters

Common parameters

The following parameters apply to both the Flink SQL and Flink DataStream reading methods.

Parameter	Required	Data type	Description
connector	Yes	STRING	The type of connector that you want to u data. Set the value to `starrocks`.
scan-url	Yes	STRING	The address that is used to connect the web server. Format: `<fe_host>:<fe_h` The default port is `8030`. You can speci addresses, which must be separated wit (,). Example: `192.168.xxx.xxx:8030,192.168.xx`
jdbc-url	Yes	STRING	The address that is used to connect the client of the FE. Format: `jdbc:mysql://<fe_host>:<fe_query` The default port number is `9030`.
username	Yes	STRING	The username of your StarRocks cluster The account must have read permission StarRocks table you want to read. See U privileges.
password	Yes	STRING	The password of your StarRocks cluster
database-name	Yes	STRING	The name of the StarRocks database to StarRocks table you want to read belong
table-name	Yes	STRING	The name of the StarRocks table you wa
scan.connect.timeout-ms	No	STRING	The maximum amount of time after whic connection from the Flink connector to y StarRocks cluster times out. Unit: millise Default value: `1000`. If the amount of tin

Parameter	Required	Data type	Description
			establish the connection exceeds this lir task fails.
scan.params.keep-alive-min	No	STRING	The maximum amount of time during wh task keeps alive. The keep-alive time is regular basis by using a polling mechani minutes. Default value: `10`. We recomm set this parameter to a value that is grea equal to `5`.
scan.params.query-timeout-s	No	STRING	The maximum amount of time after whic task times out. The timeout duration is c during task execution. Unit: seconds. De `600`. If no read result is returned after th duration elapses, the read task stops.
scan.params.mem-limit-byte	No	STRING	The maximum amount of memory allowe on each BE. Unit: bytes. Default value: `1` equal to 1 GB.
scan.max-retries	No	STRING	The maximum number of times that the can be retried upon failures. Default valu number of times that the read task is ret this limit, the read task returns errors.

Parameters for Flink DataStream

The following parameters apply only to the Flink DataStream reading method.

Parameter	Required	Data type	Description
scan.columns	No	STRING	The column that you want to read. You can specify multiple columns, which must be separated by a comma (,).
scan.filter	No	STRING	The filter condition based on which you want to filter data.

Assume that in Flink you create a table that consists of three columns, which are `c1`, `c2`, `c3`. To read the rows whose values in the `c1` column of this Flink table are equal to `100`, you can specify two filter conditions `"scan.columns, "c1"` and `"scan.filter, "c1 = 100"`.

Data type mapping between StarRocks and Flink

The following data type mapping is valid only for Flink reading data from StarRocks. For the data type mapping used for Flink writing data into StarRocks, see Continuously load data from Apache Flink®.

StarRocks	Flink
NULL	NULL
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT

StarRocks	Flink
LARGEINT	STRING
FLOAT	FLOAT
DOUBLE	DOUBLE
DATE	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
DECIMALV2	DECIMAL
DECIMAL32	DECIMAL
DECIMAL64	DECIMAL
DECIMAL128	DECIMAL
CHAR	CHAR
VARCHAR	STRING

Examples

The following examples assume you have created a database named `test` in your StarRocks cluster and you have the permissions of user `root`.

> **NOTE**
>
> If a read task fails, you must re-create it.

Data example

1. Go to the `test` database and create a table named `score_board`.

```
MySQL [test]> 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"
);
```

2. Insert data into the `score_board` table.

```
MySQL [test]> INSERT INTO score_board
VALUES
    (1, 'Bob', 21),
    (2, 'Stan', 21),
    (3, 'Sam', 22),
    (4, 'Tony', 22),
    (5, 'Alice', 22),
    (6, 'Lucy', 23),
    (7, 'Polly', 23),
    (8, 'Tom', 23),
    (9, 'Rose', 24),
    (10, 'Jerry', 24),
    (11, 'Jason', 24),
    (12, 'Lily', 25),
    (13, 'Stephen', 25),
    (14, 'David', 25),
    (15, 'Eddie', 26),
    (16, 'Kate', 27),
    (17, 'Cathy', 27),
    (18, 'Judy', 27),
    (19, 'Julia', 28),
    (20, 'Robert', 28),
    (21, 'Jack', 29);
```

3. Query the `score_board` table.

```
MySQL [test]> SELECT * FROM score_board;
+------+---------+-------+
| id   | name    | score |
+------+---------+-------+
|    1 | Bob     |    21 |
|    2 | Stan    |    21 |
|    3 | Sam     |    22 |
|    4 | Tony    |    22 |
|    5 | Alice   |    22 |
|    6 | Lucy    |    23 |
|    7 | Polly   |    23 |
|    8 | Tom     |    23 |
|    9 | Rose    |    24 |
|   10 | Jerry   |    24 |
|   11 | Jason   |    24 |
|   12 | Lily    |    25 |
|   13 | Stephen |    25 |
|   14 | David   |    25 |
|   15 | Eddie   |    26 |
|   16 | Kate    |    27 |
|   17 | Cathy   |    27 |
|   18 | Judy    |    27 |
|   19 | Julia   |    28 |
|   20 | Robert  |    28 |
|   21 | Jack    |    29 |
+------+---------+-------+
21 rows in set (0.00 sec)
```

Read data using Flink SQL

1. In your Flink cluster, create a table named `flink_test` based on the schema of the source StarRocks table (which is `score_board` in this example). In the table creation command, you must configure the read task properties, including the information about the Flink connector, the source StarRock database, and the source StarRocks table.

```
CREATE TABLE flink_test
(
    `id` INT,
    `name` STRING,
    `score` INT
)
WITH
(
```

```
    'connector'='starrocks',
    'scan-url'='192.168.xxx.xxx:8030',
    'jdbc-url'='jdbc:mysql://192.168.xxx.xxx:9030',
    'username'='xxxxxx',
    'password'='xxxxxx',
    'database-name'='test',
    'table-name'='score_board'
);
```

2. Use SELECT to read data from StarRocks.

```
SELECT id, name FROM flink_test WHERE score > 20;
```

When you read data by using Flink SQL, take note of the following points:

- You can use only SQL statements like `SELECT ... FROM <table_name> WHERE ...` to read data from StarRocks. Of all aggregate functions, only `count` is supported.
- Predicate pushdown is supported. For example, if your query contains a filter condition `char_1 <> 'A' and int_1 = -126`, the filter condition will be pushed down to the Flink connector and transformed into a statement that can be executed by StarRocks before the query is run. You do not need to perform extra configurations.
- The LIMIT statement is not supported.
- StarRocks does not support the checkpointing mechanism. As a result, data consistency cannot be guaranteed if the read task fails.

Read data using Flink DataStream

1. Add the following dependencies to the `pom.xml` file:

```
<dependency>
    <groupId>com.starrocks</groupId>
    <artifactId>flink-connector-starrocks</artifactId>
    <!-- for Apache Flink® 1.15 -->
    <version>x.x.x_flink-1.15</version>
    <!-- for Apache Flink® 1.14 -->
    <version>x.x.x_flink-1.14_2.11</version>
    <version>x.x.x_flink-1.14_2.12</version>
    <!-- for Apache Flink® 1.13 -->
    <version>x.x.x_flink-1.13_2.11</version>
    <version>x.x.x_flink-1.13_2.12</version>
```

```
    <!-- for Apache Flink® 1.12 -->
    <version>x.x.x_flink-1.12_2.11</version>
    <version>x.x.x_flink-1.12_2.12</version>
    <!-- for Apache Flink® 1.11 -->
    <version>x.x.x_flink-1.11_2.11</version>
    <version>x.x.x_flink-1.11_2.12</version>
</dependency>
```

You must replace `x.x.x` in the preceding code example with the latest Flink connector version that you are using. See Version information.

2. Call the Flink connector to read data from StarRocks:

```
import com.starrocks.connector.flink.StarRocksSource;
import
com.starrocks.connector.flink.table.source.StarRocksSourceOptions;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironmer
import org.apache.flink.table.api.DataTypes;
import org.apache.flink.table.api.TableSchema;

public class StarRocksSourceApp {
        public static void main(String[] args) throws Exception {
            StarRocksSourceOptions options =
StarRocksSourceOptions.builder()
                    .withProperty("scan-url", "192.168.xxx.xxx:8030")
                    .withProperty("jdbc-url",
"jdbc:mysql://192.168.xxx.xxx:9030")
                    .withProperty("username", "root")
                    .withProperty("password", "")
                    .withProperty("table-name", "score_board")
                    .withProperty("database-name", "test")
                    .build();
            TableSchema tableSchema = TableSchema.builder()
                    .field("id", DataTypes.INT())
                    .field("name", DataTypes.STRING())
                    .field("score", DataTypes.INT())
                    .build();
            StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
            env.addSource(StarRocksSource.source(tableSchema,
options)).setParallelism(5).print();
            env.execute("StarRocks flink source");
        }
```

```
    }
```

What's next

After Flink successfully reads data from StarRocks, you can use the Flink WebUI to monitor the read task. For example, you can view the `totalScannedRows` metric on the **Metrics** page of the WebUI to obtain the number of rows that are successfully read. You can also use Flink SQL to perform calculations such as joins on the data you have read.

Data Lakes

Catalog

7 items

External table

StarRocks supports access to other data sources by using external tables. External tables are crea...

File external table

File external table is a special type of external table. It allows you to directly query Parquet and OR...

Data Cache

This topic describes the working principles of Data Cache and how to enable Data Cache to improv...

Catalog overview

Overview

This topic describes what a catalog is, and how to manage and query internal data and external dat...

Default catalog

This topic describes what the default catalog is, and how to query the internal data of StarRocks b...

Hive catalog

A Hive catalog is a kind of external catalog that enables you to query data from Apache Hive™ with...

Iceberg catalog

An Iceberg catalog is a kind of external catalog that enables you to query data from Apache Iceber...

Hudi catalog

A Hudi catalog is a kind of external catalog that enables you to query data from Apache Hudi witho...

Delta Lake catalog

A Delta Lake catalog is a kind of external catalog that enables you to query data from Delta Lake wi...

📄 Query external data

This topic guides you through querying data from external data sources by using external catalogs.

Overview

This topic describes what a catalog is, and how to manage and query internal data and external data by using a catalog.

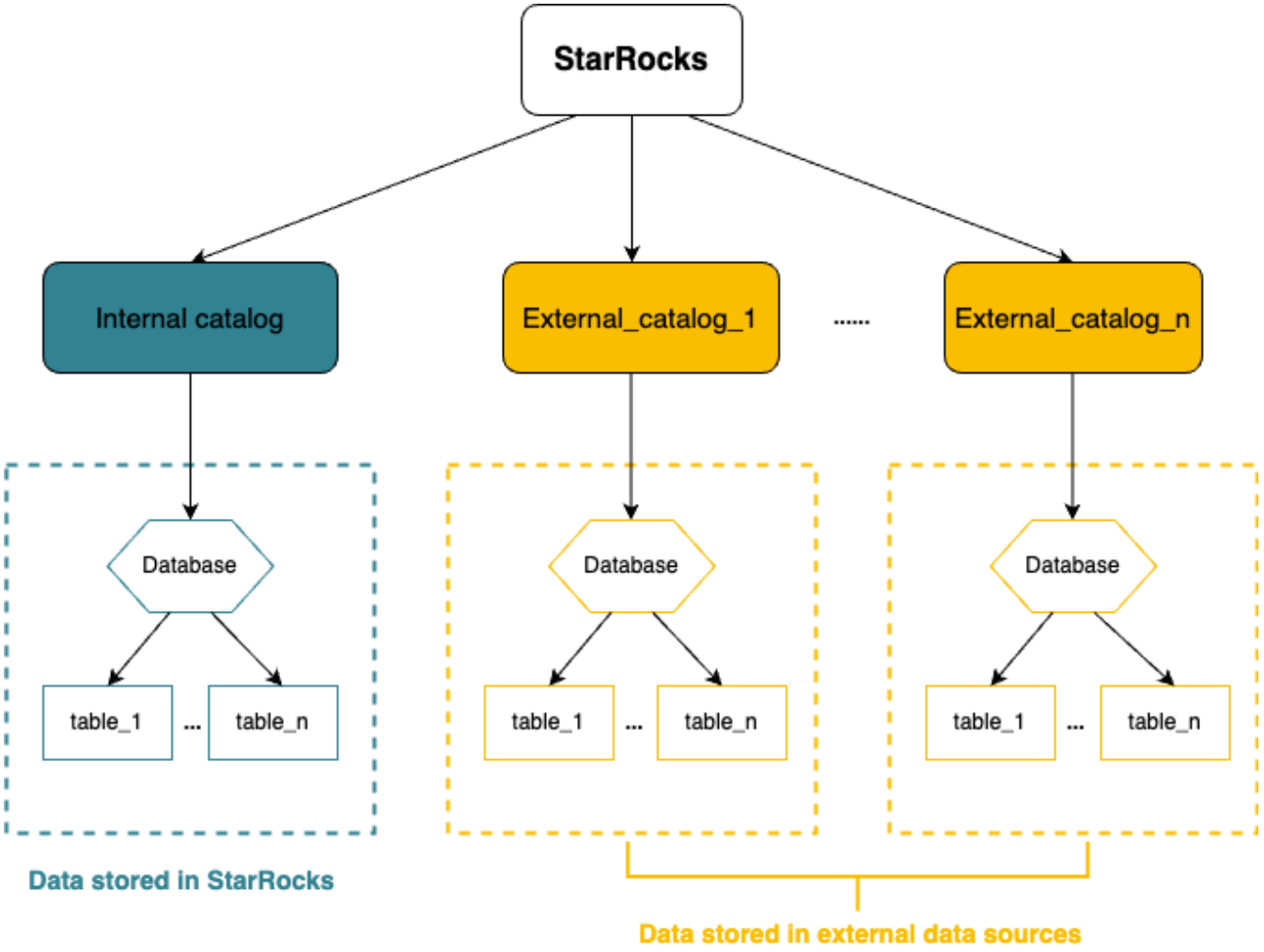
StarRocks supports the catalog feature from v2.3 onwards. Catalogs enable you to manage internal and external data in one system and offer a flexible way for you to easily query and analyze data that is stored in various external systems.

Basic concepts

- **Internal data**: refers to the data stored in StarRocks.
- **External data**: refers to the data stored in an external data source, such as Apache Hive™, Apache Iceberg, Apache Hudi, or Delta Lake.

Catalog

Currently, StarRocks provides two types of catalogs: internal catalog and external catalog.



- **Internal catalog** manages internal data of StarRocks. For example, if you execute the CREATE DATABASE or CREATE TABLE statements to create a database or a table, the database or table is stored in the internal catalog. Each StarRocks cluster has only one internal catalog named default catalog.

- **External catalog** acts like a link to externally managed metastores, which grants StarRocks direct access to external data sources. You can query external data directly with zero data loading or migration. Currently, StarRocks supports the following types of external catalogs:

    - Hive catalog: used to query data from Hive.
    - Iceberg catalog: used to query data from Iceberg.
    - Hudi catalog: used to query data from Hudi.
    - Delta Lake catalog: used to query data from Delta Lake.

    StarRocks interacts with the following two components of external data sources when you query external data:

- **Metastore service**: used by the FEs to access the metadata of external data sources. The FEs generate a query execution plan based on the metadata.
- **Data storage system**: used to store external data. Both distributed file systems and object storage systems can be used as data storage systems to store data files in various formats. After the FEs distribute the query execution plan to all BEs, all BEs scan the target external data in parallel, perform calculations, and then return the query result.

Query data

Query internal data

To query data in StarRocks, see Default catalog.

Query external data

To query data from external data sources, see Query external data.

Cross-catalog query

To perform a cross-catalog federated query from your current catalog, specify the data you want to query in the `catalog_name.database_name` or `catalog_name.database_name.table_name` format.

- Query `hive_table` in `hive_db` when the current session is `default_catalog.olap_db`.

```
SELECT * FROM hive_catalog.hive_db.hive_table;
```

- Query `olap_table` in `default_catalog` when the current session is `hive_catalog.hive_db`.

```
SELECT * FROM default_catalog.olap_db.olap_table;
```

- Perform a JOIN query on `hive_table` in `hive_catalog` and `olap_table` in `default_catalog` when the current session is `hive_catalog.hive_db`.

```
SELECT * FROM hive_table h JOIN
default_catalog.olap_db.olap_table o WHERE h.id = o.id;
```

- Perform a JOIN query on `hive_table` in `hive_catalog` and `olap_table` in `default_catalog` by using a JOIN clause when the current session is another catalog.

```
SELECT * FROM hive_catalog.hive_db.hive_table h JOIN
default_catalog.olap_db.olap_table o WHERE h.id = o.id;
```

Default catalog

This topic describes what the default catalog is, and how to query the internal data of StarRocks by using the default catalog.

StarRocks 2.3 and later provide an internal catalog to manage the internal data of StarRocks. Each StarRocks cluster has only one internal catalog named `default_catalog`. Currently, you cannot modify the name of the internal catalog or create a new internal catalog.

Query internal data

1. Connect your StarRocks cluster.
   - If you use the MySQL client to connect the StarRocks cluster, you go to `default_catalog` by default after connecting.
   - If you use JDBC to connect the StarRocks cluster, you can go directly to the destination database in the default catalog by specifying `default_catalog.db_name` when connecting.
2. (Optional) Execute the following statement to view all databases in StarRocks. See SHOW DATABASES to view the output of this statement.

   ```
   SHOW DATABASES;
   ```

   Or

   ```
   SHOW DATABASES FROM catalog_name;
   ```

3. (Optional) Execute the following statement to go to the destination database.

   ```
   USE db_name;
   ```

Or

```
USE default_catalog.db_name;
```

4. Query internal data. For more usages of the SELECT statement, see SELECT.

   ```
   SELECT * FROM table_name;
   ```

   If you do not specify the database in the preceding steps, you can directly specify it in a select query.

   ```
   SELECT * FROM db_name.table_name;
   ```

   Or

   ```
   SELECT * FROM default_catalog.db_name.table_name;
   ```

Examples

To query data in `olap_db.olap_table`, you can perform one of the following operations:

```
USE olap_db;
SELECT * FROM olap_table limit 1;
```

Or

```
SELECT * FROM olap_db.olap_table limit 1;
```

Or

```
SELECT * FROM default_catalog.olap_db.olap_table limit 1;
```

References

To query data from external data sources, see Query external data.

Hive catalog

A Hive catalog is a kind of external catalog that enables you to query data from Apache Hive™ without ingestion.

Also, you can directly transform and load data from Hive by using INSERT INTO based on Hive catalogs. StarRocks supports Hive catalogs from v2.4 onwards.

To ensure successful SQL workloads on your Hive cluster, your StarRocks cluster must be able to access the storage system and metastore of your Hive cluster. StarRocks supports the following storage systems and metastores:

- Object storage or distributed file system like AWS S3 or HDFS
- Metastore like Hive metastore or AWS Glue

Usage notes

- The file formats of Hive that StarRocks supports are Parquet, ORC, and Textfile:
    - Parquet files support the following compression formats: SNAPPY, LZ4, ZSTD, GZIP, and NO_COMPRESSION.
    - ORC files support the following compression formats: ZLIB, SNAPPY, LZO, LZ4, ZSTD, and NO_COMPRESSION.
- The data types of Hive that StarRocks does not support are INTERVAL, BINARY, and UNION. Additionally, StarRocks does not support the MAP and STRUCT data types for Textfile-formatted Hive tables.
- You can only use Hive catalogs to query data. You cannot use Hive catalogs to drop, delete, or insert data into your Hive cluster.

Integration preparations

Before you create a Hive catalog, make sure your StarRocks cluster can integrate with the storage system and metastore of your Hive cluster.

AWS IAM

If your Hive cluster uses AWS S3 as storage or AWS Glue as metastore, choose your suitable authentication method and make the required preparations to ensure that your StarRocks cluster can access the related AWS cloud resources.

The following authentication methods are recommended:

- Instance profile
- Assumed role
- IAM user

Of the above-mentioned three authentication methods, instance profile is the most widely used.

For more information, see Preparation for authentication in AWS IAM.

HDFS

If you choose HDFS as storage, configure your StarRocks cluster as follows:

- (Optional) Set the username that is used to access your HDFS cluster and Hive metastore. By default, StarRocks uses the username of the FE and BE processes to access your HDFS cluster and Hive metastore. You can also set the username by adding `export HADOOP_USER_NAME="<user_name>"` at the beginning of the **fe/conf/hadoop_env.sh** file of each FE and at the beginning of the **be/conf/hadoop_env.sh** file of each BE. After you set the username in these files, restart each FE and each BE to make the parameter settings take effect. You can set only one username for each StarRocks cluster.

- When you query Hive data, the FEs and BEs of your StarRocks cluster use the HDFS client to access your HDFS cluster. In most cases, you do not need to configure your StarRocks cluster to achieve that purpose, and StarRocks starts the HDFS client using the default configurations. You need to configure your StarRocks cluster only in the following situations:

    - High availability (HA) is enabled for your HDFS cluster: Add the **hdfs-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.

- View File System (ViewFs) is enabled for your HDFS cluster: Add the **core-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.

> **NOTE**
>
> If an error indicating an unknown host is returned when you send a query, you must add the mapping between the host names and IP addresses of your HDFS cluster nodes to the **/etc/hosts** path.

Kerberos authentication

If Kerberos authentication is enabled for your HDFS cluster or Hive metastore, configure your StarRocks cluster as follows:

- Run the `kinit -kt keytab_path principal` command on each FE and each BE to obtain Ticket Granting Ticket (TGT) from Key Distribution Center (KDC). To run this command, you must have the permissions to access your HDFS cluster and Hive metastore. Note that accessing KDC with this command is time-sensitive. Therefore, you need to use cron to run this command periodically.
- Add `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` to the **$FE_HOME/conf/fe.conf** file of each FE and to the **$BE_HOME/conf/be.conf** file of each BE. In this example, `/etc/krb5.conf` is the save path of the **krb5.conf** file. You can modify the path based on your needs.

Create a Hive catalog

Syntax

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "hive",
    GeneralParams,
    MetastoreParams,
    StorageCredentialParams,
```

```
    MetadataUpdateParams
)
```

Parameters

catalog_name

The name of the Hive catalog. The naming conventions are as follows:

- The name can contain letters, digits (0-9), and underscores (_). It must start with a letter.
- The name is case-sensitive and cannot exceed 1023 characters in length.

comment

The description of the Hive catalog. This parameter is optional.

type

The type of your data source. Set the value to `hive`.

GeneralParams

A set of general parameters.

The following table describes the parameters you can configure in `GeneralParams`.

Parameter	Required	Description
enable_recursive_listing	No	Specifies whether StarRocks reads data from a table and its partitions and from the subdirectories within the physical locations of the table and its partitions. Valid values: `true` and `false`. Default value: `true`. The value `true` specifies to recursively list subdirectories, and the value `false` specifies to ignore subdirectories.

MetastoreParams

A set of parameters about how StarRocks integrates with the metastore of your data source.

Hive metastore

If you choose Hive metastore as the metastore of your data source, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **NOTE**
>
> Before querying Hive data, you must add the mapping between the host names and IP addresses of your Hive metastore nodes to the `/etc/hosts` path. Otherwise, StarRocks may fail to access your Hive metastore when you start a query.

The following table describes the parameter you need to configure in `MetastoreParams`.

Parameter	Required	Description
hive.metastore.type	Yes	The type of metastore that you use for your Hive clust value to `hive`.
hive.metastore.uris	Yes	The URI of your Hive metastore. Format: `thrift://<metastore_IP_address>:<metastore_` If high availability (HA) is enabled for your Hive metast specify multiple metastore URIs and separate them wi (`,`), for example, `"thrift://<metastore_IP_addr <metastore_port_1>,thrift://<metastore_IP_a <metastore_port_2>,thrift://<metastore_IP_a <metastore_port_3>"`.

AWS Glue

If you choose AWS Glue as the metastore of your data source, take one of the following actions:

- To choose the instance profile-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the assumed role-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the IAM user-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

The following table describes the parameters you need to configure in `MetastoreParams`.

Parameter	Required	Description
hive.metastore.type	Yes	The type of metastore that you use for your Hive cluster. Set the value to `glue`.
aws.glue.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication. Valid values: `true` and `false`. Default value: `false`.
aws.glue.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS Glue Data Catalog. If you use the assumed role-based authentication method to access

Parameter	Required	Description
		AWS Glue, you must specify this parameter.
aws.glue.region	Yes	The region in which your AWS Glue Data Catalog resides. Example: `us-west-1`.
aws.glue.access_key	No	The access key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.secret_key	No	The secret key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS Glue and how to configure an access control policy in the AWS IAM Console, see Authentication parameters for accessing AWS Glue.

StorageCredentialParams

A set of parameters about how StarRocks integrates with your storage system. This parameter set is optional.

You need to configure `StorageCredentialParams` only when your Hive cluster uses AWS S3 as storage.

If your Hive cluster uses any other storage system, you can ignore `StorageCredentialParams`.

AWS S3

If you choose AWS S3 as storage for your Hive cluster, take one of the following actions:

- To choose the instance profile-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the assumed role-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the IAM user-based authentication method, configure `StorageCredentialParams` as follows:

```
"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>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you use the assumed role-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.region	Yes	The region in which your AWS S3 bucket resides. Example: `us-west-1`.

Parameter	Required	Description
aws.s3.access_key	No	The access key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.secret_key	No	The secret key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS S3 and how to configure an access control policy in AWS IAM Console, see Authentication parameters for accessing AWS S3.

S3-compatible storage system

Hive catalogs support S3-compatible storage systems from v2.5 onwards.

If you choose an S3-compatible storage system, such as MinIO, as storage for your Hive cluster, configure `StorageCredentialParams` as follows to ensure a successful integration:

```
"aws.s3.enable_ssl" = "{true | false}",
"aws.s3.enable_path_style_access" = "{true | false}",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.enable_ssl	Yes	Specifies whether to enable SSL connec Valid values: `true` and `false`. Default

Parameter	Required	Description
aws.s3.enable_path_style_access	Yes	Specifies whether to enable path-style a Valid values: `true` and `false`. Default you must set the value to `true`. Path-style URLs use the following forma `<region_code>.amazonaws.com/<buc` For example, if you create a bucket nam `BUCKET1` in the US West (Oregon) Regic the `alice.jpg` object in that bucket, yo path-style URL: `https://s3.us-west-` `EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Yes	The endpoint that is used to connect to storage system instead of AWS S3.
aws.s3.access_key	Yes	The access key of your IAM user.
aws.s3.secret_key	Yes	The secret key of your IAM user.

MetadataUpdateParams

A set of parameters about how StarRocks updates the cached metadata of Hive. This parameter set is optional.

StarRocks implements the automatic asynchronous update policy by default.

In most cases, you can ignore `MetadataUpdateParams` and do not need to tune the policy parameters in it, because the default values of these parameters already provide you with an out-of-the-box performance.

However, if the frequency of data updates in Hive is high, you can tune these parameters to further optimize the performance of automatic asynchronous updates.

> **NOTE**
>
> In most cases, if your Hive data is updated at a granularity of 1 hour or less, the data update frequency is considered high.

Parameter	Required	Description
enable_metastore_cache	No	Specifies whether StarRocks caches the metadata of Hive tables. Valid values: `true` and `false`. Default value: `true`. The value `true` enables the cache, and the value `false` disables the cache.
enable_remote_file_cache	No	Specifies whether StarRocks caches the metadata of the underlying data files of Hive tables or partitions. Valid values: `true` and `false`. Default value: `true`. The value `true` enables the cache, and the value `false` disables the cache.
metastore_cache_refresh_interval_sec	No	The time interval at which StarRocks asynchronously updates the metadata of Hive tables or partitions cached in itself. Unit: seconds. Default value: `7200`, which is 2 hours.
remote_file_cache_refresh_interval_sec	No	The time interval at which StarRocks asynchronously updates the metadata of the underlying data files of Hive tables or partitions cached in itself. Unit: seconds. Default value: `60`.
metastore_cache_ttl_sec	No	The time interval at which StarRocks automatically discards the metadata of Hive

Parameter	Required	Description
		tables or partitions cached in itself. Unit: seconds. Default value: `86400`, which is 24 hours.
remote_file_cache_ttl_sec	No	The time interval at which StarRocks automatically discards the metadata of the underlying data files of Hive tables or partitions cached in itself. Unit: seconds. Default value: `129600`, which is 36 hours.
enable_cache_list_names	No	Specifies whether StarRocks caches Hive partition names. Valid values: `true` and `false`. Default value: `false`. The value `true` enables the cache, and the value `false` disables the cache.

Examples

The following examples create a Hive catalog named `hive_catalog_hms` or `hive_catalog_glue`, depending on the type of metastore you use, to query data from your Hive cluster.

HDFS

If you use HDFS as storage, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
```

```
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

Instance profile-based authentication

- If you use Hive metastore in your Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Assumed role-based authentication

- If you use Hive metastore in your Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
```

```
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

IAM user-based authentication

- If you use Hive metastore in your Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "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" = "us-west-2"
);
```

S3-compatible storage system

Use MinIO as an example. Run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "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>"
);
```

View the schema of a Hive table

You can use one of the following syntaxes to view the schema of a Hive table:

- View schema

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- View schema and location from the CREATE statement

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Query a Hive table

1. Use the following syntax to view the databases in your Hive cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Use the following syntax to connect to your target Hive database:

```
USE <catalog_name>.<database_name>
```

3. Use the following syntax to query the Hive table:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Load data from Hive

Suppose you have an OLAP table named `olap_tbl`, you can transform and load data like below:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM hive_table
```

Manually or automatically update metadata cache

Manual update

By default, StarRocks caches the metadata of Hive and automatically updates the metadata in asynchronous mode to deliver better performance. Additionally, after some schema

changes or table updates are made on a Hive table, you can also use REFRESH EXTERNAL TABLE to manually update its metadata, thereby ensuring that StarRocks can obtain up-to-date metadata at its earliest opportunity and generate appropriate execution plans:

```
REFRESH EXTERNAL TABLE <table_name>
```

You need to manually update metadata in the following situations:

- A data file in an existing partition is changed, for example, by running the `INSERT OVERWRITE ... PARTITION ...` command.
- Schema changes are made on a Hive table.
- An existing Hive table is deleted by using the DROP statement, and a new Hive table with the same name as the deleted Hive table is created.
- You have specified `"enable_cache_list_names" = "true"` in `PROPERTIES` at the creation of your Hive catalog, and you want to query new partitions that you just created on your Hive cluster.

  > **NOTE**
  >
  > From v2.5.5 onwards, StarRocks provides the periodic Hive metadata cache refresh feature. For more information, see the below "Periodically refresh metadata cache" section of this topic. After you enable this feature, StarRocks refreshes your Hive metadata cache every 10 minutes by default. Therefore, manual updates are not needed in most cases. You need to perform a manual update only when you want to query new partitions immediately after the new partitions are created on your Hive cluster.

Note that the REFRESH EXTERNAL TABLE refreshes only the tables and partitions cached in your FEs.

Periodically refresh metadata cache

From v2.5.5 onwards, StarRocks can periodically refresh the cached metadata of the frequently accessed Hive catalogs to perceive data changes. You can configure the Hive metadata cache refresh through the following FE parameters:

Configuration item	Default	Descri
enable_background_refresh_connector_metadata	`true` in v3.0 `false` in v2.5	Whethe enable periodi Hive metada cache refresh After it enable StarRo polls th metast (Hive Metast or AWS Glue) o your Hi cluster, refresh the cac metada the frequer access Hive catalog perceiv data change `true` indicat enable Hive metada cache refresh

Configuration item	Default	Descri
		`false` indicat disable This ite an FE dynami parame You ca modify using t ADMIN FRONT CONFI comma
background_refresh_metadata_interval_millis	`600000` (10 minutes)	The int betwee two consec Hive metada cache refresh Unit: millisec This ite an FE dynami parame You ca modify using t ADMIN FRONT CONFI comma

Configuration item	Default	Descri
background_refresh_metadata_time_secs_since_last_access_secs	`86400` (24 hours)	The expirat time of Hive metada cache refresh task. Fo the Hiv catalog has bee access it has n been access for mor than th specifie time, StarRo stops refresh its cach metada For the catalog has not been access StarRo will not refresh cachec metada Unit: second

Configuration item	Default	Descri
		This ite an FE dynami parame You ca modify using t ADMIN FRONT CONFI comma
Using the periodic Hive metadata cache refresh feature and the metadata automatic asynchronous update policy together significantly accelerates data access, reduces the read load from external data sources, and improves query performance.		

Appendix: Understand automatic asynchronous update

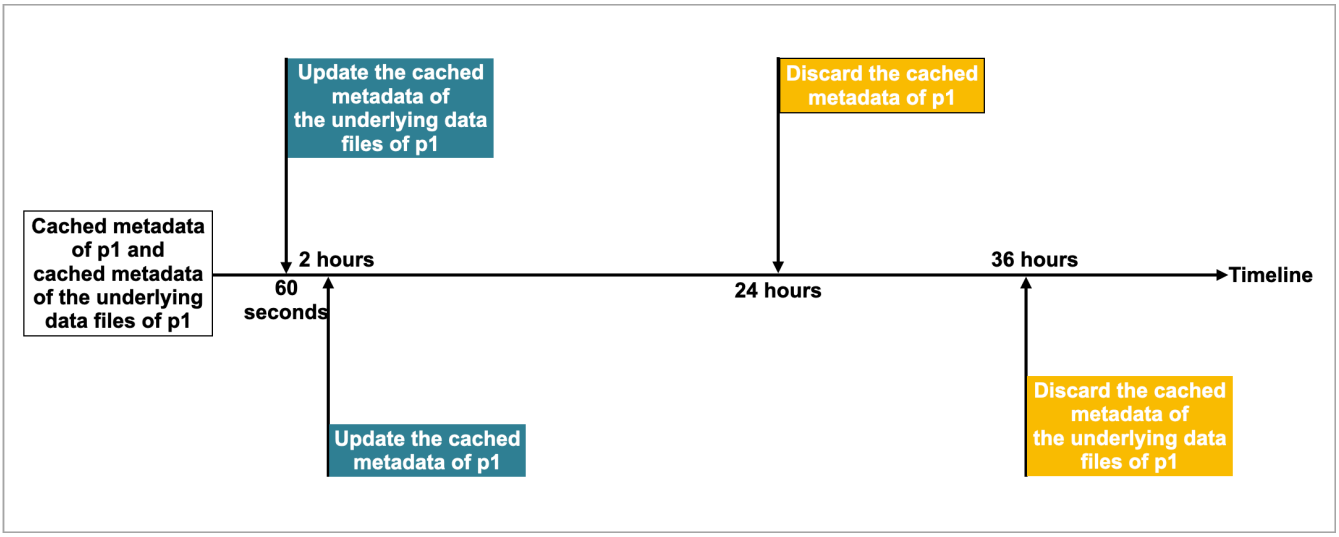
Automatic asynchronous update is the default policy that StarRocks uses to update the metadata in Hive catalogs.

By default (namely, when the `enable_metastore_cache` and `enable_remote_file_cache` parameters are both set to `true`), if a query hits a partition of a Hive table, StarRocks automatically caches the metadata of the partition and the metadata of the underlying data files of the partition. The cached metadata is updated by using the lazy update policy.

For example, there is a Hive table named `table2`, which has four partitions: `p1`, `p2`, `p3`, and `p4`. A query hits `p1`, and StarRocks caches the metadata of `p1` and the metadata of the underlying data files of `p1`. Assume that the default time intervals to update and discard the cached metadata are as follows:

- The time interval (specified by the `metastore_cache_refresh_interval_sec` parameter) to asynchronously update the cached metadata of `p1` is 2 hours.
- The time interval (specified by the `remote_file_cache_refresh_interval_sec` parameter) to asynchronously update the cached metadata of the underlying data files of `p1` is 60 seconds.
- The time interval (specified by the `metastore_cache_ttl_sec` parameter) to automatically discard the cached metadata of `p1` is 24 hours.
- The time interval (specified by the `remote_file_cache_ttl_sec` parameter) to automatically discard the cached metadata of the underlying data files of `p1` is 36 hours.

The following figure shows the time intervals on a timeline for easier understanding.



Then StarRocks updates or discards the metadata in compliance with the following rules:

- If another query hits `p1` again and the current time from the last update is less than 60 seconds, StarRocks does not update the cached metadata of `p1` or the cached metadata of the underlying data files of `p1`.
- If another query hits `p1` again and the current time from the last update is more than 60 seconds, StarRocks updates the cached metadata of the underlying data files of `p1`.
- If another query hits `p1` again and the current time from the last update is more than 2 hours, StarRocks updates the cached metadata of `p1`.
- If `p1` has not been accessed within 24 hours from the last update, StarRocks discards the cached metadata of `p1`. The metadata will be cached at the next query.

- If `p1` has not been accessed within 36 hours from the last update, StarRocks discards the cached metadata of the underlying data files of `p1`. The metadata will be cached at the next query.

Iceberg catalog

An Iceberg catalog is a kind of external catalog that enables you to query data from Apache Iceberg without ingestion.

Also, you can directly transform and load data from Iceberg by using INSERT INTO based on Iceberg catalogs. StarRocks supports Iceberg catalogs from v2.4 onwards.

To ensure successful SQL workloads on your Iceberg cluster, your StarRocks cluster must be able to access the storage system and metastore of your Iceberg cluster. StarRocks supports the following storage systems and metastores:

- Object storage or distributed file system like AWS S3 or HDFS
- Metastore like Hive metastore or AWS Glue

Usage notes

- The file formats of Iceberg that StarRocks supports are Parquet and ORC:
    - Parquet files support the following compression formats: SNAPPY, LZ4, ZSTD, GZIP, and NO_COMPRESSION.
    - ORC files support the following compression formats: ZLIB, SNAPPY, LZO, LZ4, ZSTD, and NO_COMPRESSION.
- Iceberg catalogs support v1 tables, but do not support v2 tables.

Integration preparations

Before you create an Iceberg catalog, make sure your StarRocks cluster can integrate with the storage system and metastore of your Iceberg cluster.

AWS IAM

If your Iceberg cluster uses AWS S3 as storage or AWS Glue as metastore, choose your suitable authentication method and make the required preparations to ensure that your StarRocks cluster can access the related AWS cloud resources.

The following authentication methods are recommended:

- Instance profile
- Assumed role
- IAM user

Of the above-mentioned three authentication methods, instance profile is the most widely used.

For more information, see Preparation for authentication in AWS IAM.

HDFS

If you choose HDFS as storage, configure your StarRocks cluster as follows:

- (Optional) Set the username that is used to access your HDFS cluster and Hive metastore. By default, StarRocks uses the username of the FE and BE processes to access your HDFS cluster and Hive metastore. You can also set the username by adding `export HADOOP_USER_NAME="<user_name>"` at the beginning of the **fe/conf/hadoop_env.sh** file of each FE and at the beginning of the **be/conf/hadoop_env.sh** file of each BE. After you set the username in these files, restart each FE and each BE to make the parameter settings take effect. You can set only one username for each StarRocks cluster.

- When you query Iceberg data, the FEs and BEs of your StarRocks cluster use the HDFS client to access your HDFS cluster. In most cases, you do not need to configure your StarRocks cluster to achieve that purpose, and StarRocks starts the HDFS client using the default configurations. You need to configure your StarRocks cluster only in the following situations:

    - High availability (HA) is enabled for your HDFS cluster: Add the **hdfs-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.
    - View File System (ViewFs) is enabled for your HDFS cluster: Add the **core-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.

> **NOTE**
>
> If an error indicating an unknown host is returned when you send a query, you must add the mapping between the host names and IP addresses of your HDFS cluster nodes to the **/etc/hosts** path.

Kerberos authentication

If Kerberos authentication is enabled for your HDFS cluster or Hive metastore, configure your StarRocks cluster as follows:

- Run the `kinit -kt keytab_path principal` command on each FE and each BE to obtain Ticket Granting Ticket (TGT) from Key Distribution Center (KDC). To run this command, you must have the permissions to access your HDFS cluster and Hive metastore. Note that accessing KDC with this command is time-sensitive. Therefore, you need to use cron to run this command periodically.
- Add `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` to the **$FE_HOME/conf/fe.conf** file of each FE and to the **$BE_HOME/conf/be.conf** file of each BE. In this example, `/etc/krb5.conf` is the save path of the **krb5.conf** file. You can modify the path based on your needs.

Create an Iceberg catalog

Syntax

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "iceberg",
    MetastoreParams,
    StorageCredentialParams
)
```

Parameters

catalog_name

The name of the Iceberg catalog. The naming conventions are as follows:

- The name can contain letters, digits (0-9), and underscores (_). It must start with a letter.
- The name is case-sensitive and cannot exceed 1023 characters in length.

comment

The description of the Iceberg catalog. This parameter is optional.

type

The type of your data source. Set the value to `iceberg`.

MetastoreParams

A set of parameters about how StarRocks integrates with the metastore of your data source.

Hive metastore

If you choose Hive metastore as the metastore of your data source, configure `MetastoreParams` as follows:

```
"iceberg.catalog.type" = "hive",
"iceberg.catalog.hive.metastore.uris" = "<hive_metastore_uri>"
```

> **NOTE**
>
> Before querying Iceberg data, you must add the mapping between the host names and IP addresses of your Hive metastore nodes to the `/etc/hosts` path. Otherwise, StarRocks may fail to access your Hive metastore when you start a query.

The following table describes the parameter you need to configure in `MetastoreParams`.

Parameter	Required	Descriptio
iceberg.catalog.type	Yes	The type of metastore that you use for the value to `hive`.
iceberg.catalog.hive.metastore.uris	Yes	The URI of your Hive metastore. Forma `thrift://<metastore_IP_address>`

Parameter	Required	Descriptio

		If high availability (HA) is enabled for y
		specify multiple metastore URIs and se
		(`,`), for example, `"thrift://<metas`
		`<metastore_port_1>,thrift://<me`
		`<metastore_port_2>,thrift://<me`
		`<metastore_port_3>"`.

AWS Glue

If you choose AWS Glue as the metastore of your data source, take one of the following actions:

- To choose the instance profile-based authentication method, configure `MetastoreParams` as follows:

```
"iceberg.catalog.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the assumed role-based authentication method, configure `MetastoreParams` as follows:

```
"iceberg.catalog.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the IAM user-based authentication method, configure `MetastoreParams` as follows:

```
"iceberg.catalog.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

The following table describes the parameters you need to configure in `MetastoreParams`.

Parameter	Required	Description
iceberg.catalog.type	Yes	The type of metastore that you use for your Iceberg cluster. Set the value to `glue`.
aws.glue.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.glue.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS Glue Data Catalog. If you use the assumed role-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.region	Yes	The region in which your AWS Glue Data Catalog resides. Example: `us-west-1`.
aws.glue.access_key	No	The access key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.secret_key	No	The secret key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS Glue and how to configure an access control policy in the AWS IAM Console, see Authentication parameters for accessing AWS Glue.

`StorageCredentialParams`

A set of parameters about how StarRocks integrates with your storage system. This parameter set is optional.

You need to configure `StorageCredentialParams` only when your Iceberg cluster uses AWS S3 as storage.

If your Iceberg cluster uses any other storage system, you can ignore `StorageCredentialParams`.

AWS S3

If you choose AWS S3 as storage for your Iceberg cluster, take one of the following actions:

- To choose the instance profile-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the assumed role-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the IAM user-based authentication method, configure `StorageCredentialParams` as follows:

```
"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>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you use the assumed role-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.region	Yes	The region in which your AWS S3 bucket resides. Example: `us-west-1`.
aws.s3.access_key	No	The access key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.secret_key	No	The secret key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS S3 and how to configure an access control policy in AWS IAM Console, see Authentication parameters for accessing AWS S3.

S3-compatible storage system

Iceberg catalogs support S3-compatible storage systems from v2.5 onwards.

If you choose an S3-compatible storage system, such as MinIO, as storage for your Iceberg cluster, configure `StorageCredentialParams` as follows to ensure a successful integration:

```
"aws.s3.enable_ssl" = "{true | false}",
"aws.s3.enable_path_style_access" = "{true | false}",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.enable_ssl	Yes	Specifies whether to enable SSL connec Valid values: `true` and `false`. Default
aws.s3.enable_path_style_access	Yes	Specifies whether to enable path-style a Valid values: `true` and `false`. Default you must set the value to `true`. Path-style URLs use the following forma `<region_code>.amazonaws.com/<buc` For example, if you create a bucket nam `BUCKET1` in the US West (Oregon) Regic the `alice.jpg` object in that bucket, yo path-style URL: `https://s3.us-west-` `EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Yes	The endpoint that is used to connect to storage system instead of AWS S3.
aws.s3.access_key	Yes	The access key of your IAM user.
aws.s3.secret_key	Yes	The secret key of your IAM user.

Examples

The following examples create an Iceberg catalog named `iceberg_catalog_hms` or `iceberg_catalog_glue`, depending on the type of metastore you use, to query data from your Iceberg cluster.

HDFS

If you use HDFS as storage, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "iceberg.catalog.hive.metastore.uris" = "thrift://xx.xx.xx:9083"
);
```

AWS S3

If you choose instance profile-based credential

- If you use Hive metastore in your Iceberg cluster, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "iceberg.catalog.hive.metastore.uris" =
"thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Iceberg cluster, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_glue
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

If you choose assumed role-based credential

- If you use Hive metastore in your HIceberg cluster, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "iceberg.catalog.hive.metastore.uris" =
"thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Iceberg cluster, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_glue
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

If you choose IAM user-based credential

- If you use Hive metastore in your Iceberg cluster, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
```

```
    "iceberg.catalog.hive.metastore.uris" =
"thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Iceberg cluster, run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_glue
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "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" = "us-west-2"
);
```

S3-compatible storage system

Use MinIO as an example. Run a command like below:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "iceberg.catalog.hive.metastore.uris" =
"thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "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>"
);
```

View the schema of an Iceberg table

You can use one of the following syntaxes to view the schema of an Iceberg table:

- View schema

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- View schema and location from the CREATE statement

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Query an Iceberg table

1. Use the following syntax to view the databases in your Iceberg cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Use the following syntax to connect to your target Iceberg database:

```
USE <catalog_name>.<database_name>
```

3. Use the following syntax to query the Iceberg table:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Load data from Iceberg

Suppose you have an OLAP table named `olap_tbl`, you can transform and load data like below:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM
iceberg_table
```

Hudi catalog

A Hudi catalog is a kind of external catalog that enables you to query data from Apache Hudi without ingestion.

Also, you can directly transform and load data from Hudi by using INSERT INTO based on Hudi catalogs. StarRocks supports Hudi catalogs from v2.4 onwards.

To ensure successful SQL workloads on your Hudi cluster, your StarRocks cluster must be able to access the storage system and metastore of your Hudi cluster. StarRocks supports the following storage systems and metastores:

- Object storage or distributed file system like AWS S3 or HDFS
- Metastore like Hive metastore or AWS Glue

Usage notes

- The file format of Hudi that StarRocks supports is Parquet. Parquet files support the following compression formats: SNAPPY, LZ4, ZSTD, GZIP, and NO_COMPRESSION.
- StarRocks provides complete support for Copy On Write (COW) tables and Merge On Read (MOR) tables from Hudi.

Integration preparations

Before you create a Hudi catalog, make sure your StarRocks cluster can integrate with the storage system and metastore of your Hudi cluster.

AWS IAM

If your Hudi cluster uses AWS S3 as storage or AWS Glue as metastore, choose your suitable authentication method and make the required preparations to ensure that your StarRocks cluster can access the related AWS cloud resources.

The following authentication methods are recommended:

- Instance profile
- Assumed role
- IAM user

Of the above-mentioned three authentication methods, instance profile is the most widely used.

For more information, see Preparation for authentication in AWS IAM.

HDFS

If you choose HDFS as storage, configure your StarRocks cluster as follows:

- (Optional) Set the username that is used to access your HDFS cluster and Hive metastore. By default, StarRocks uses the username of the FE and BE processes to access your HDFS cluster and Hive metastore. You can also set the username by adding `export HADOOP_USER_NAME="<user_name>"` at the beginning of the **fe/conf/hadoop_env.sh** file of each FE and at the beginning of the **be/conf/hadoop_env.sh** file of each BE. After you set the username in these files, restart each FE and each BE to make the parameter settings take effect. You can set only one username for each StarRocks cluster.

- When you query Hudi data, the FEs and BEs of your StarRocks cluster use the HDFS client to access your HDFS cluster. In most cases, you do not need to configure your StarRocks cluster to achieve that purpose, and StarRocks starts the HDFS client using the default configurations. You need to configure your StarRocks cluster only in the following situations:

    - High availability (HA) is enabled for your HDFS cluster: Add the **hdfs-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.
    - View File System (ViewFs) is enabled for your HDFS cluster: Add the **core-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.

> **NOTE**
>
> If an error indicating an unknown host is returned when you send a query, you must add the mapping between the host names and IP addresses of your HDFS cluster nodes to the **/etc/hosts** path.

Kerberos authentication

If Kerberos authentication is enabled for your HDFS cluster or Hive metastore, configure your StarRocks cluster as follows:

- Run the `kinit -kt keytab_path principal` command on each FE and each BE to obtain Ticket Granting Ticket (TGT) from Key Distribution Center (KDC). To run this command, you must have the permissions to access your HDFS cluster and Hive metastore. Note that accessing KDC with this command is time-sensitive. Therefore, you need to use cron to run this command periodically.
- Add `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` to the **$FE_HOME/conf/fe.conf** file of each FE and to the **$BE_HOME/conf/be.conf** file of each BE. In this example, `/etc/krb5.conf` is the save path of the **krb5.conf** file. You can modify the path based on your needs.

Create a Hudi catalog

Syntax

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "hudi",
    MetastoreParams,
    StorageCredentialParams,
    MetadataUpdateParams
)
```

Parameters

catalog_name

The name of the Hudi catalog. The naming conventions are as follows:

- The name can contain letters, digits (0-9), and underscores (_). It must start with a letter.
- The name is case-sensitive and cannot exceed 1023 characters in length.

comment

The description of the Hudi catalog. This parameter is optional.

type

The type of your data source. Set the value to `hudi`.

MetastoreParams

A set of parameters about how StarRocks integrates with the metastore of your data source.

Hive metastore

If you choose Hive metastore as the metastore of your data source, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **NOTE**
>
> Before querying Hudi data, you must add the mapping between the host names and IP addresses of your Hive metastore nodes to the `/etc/hosts` path. Otherwise, StarRocks may fail to access your Hive metastore when you start a query.

The following table describes the parameter you need to configure in `MetastoreParams`.

Parameter	Required	Description
hive.metastore.type	Yes	The type of metastore that you use for your Hudi clust value to `hive`.
hive.metastore.uris	Yes	The URI of your Hive metastore. Format: `thrift://<metastore_IP_address>:<metastore_` If high availability (HA) is enabled for your Hive metast specify multiple metastore URIs and separate them wi (`,`), for example, `"thrift://<metastore_IP_addre <metastore_port_1>,thrift://<metastore_IP_a`

Parameter	Required	Description
		`<metastore_port_2>,thrift://<metastore_IP_a` `<metastore_port_3>"`.

AWS Glue

If you choose AWS Glue as the metastore of your data source, take one of the following actions:

- To choose the instance profile-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the assumed role-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the IAM user-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

The following table describes the parameters you need to configure in `MetastoreParams`.

Parameter	Required	Description
hive.metastore.type	Yes	The type of metastore that you use for your Hudi cluster. Set the value to `glue`.
aws.glue.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.glue.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS Glue Data Catalog. If you use the assumed role-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.region	Yes	The region in which your AWS Glue Data Catalog resides. Example: `us-west-1`.
aws.glue.access_key	No	The access key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.secret_key	No	The secret key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS Glue and how to configure an access control policy in the AWS IAM Console, see Authentication parameters for accessing AWS Glue.

StorageCredentialParams

A set of parameters about how StarRocks integrates with your storage system. This parameter set is optional.

You need to configure `StorageCredentialParams` only when your Hudi cluster uses AWS S3 as storage.

If your Hudi cluster uses any other storage system, you can ignore `StorageCredentialParams`.

AWS S3

If you choose AWS S3 as storage for your Hudi cluster, take one of the following actions:

- To choose the instance profile-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the assumed role-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the IAM user-based authentication method, configure `StorageCredentialParams` as follows:

```
"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>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you use the assumed role-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.region	Yes	The region in which your AWS S3 bucket resides. Example: `us-west-1`.
aws.s3.access_key	No	The access key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.secret_key	No	The secret key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS S3 and how to configure an access control policy in AWS IAM Console, see Authentication parameters for accessing AWS S3.

S3-compatible storage system

Hudi catalogs support S3-compatible storage systems from v2.5 onwards.

If you choose an S3-compatible storage system, such as MinIO, as storage for your Hudi cluster, configure `StorageCredentialParams` as follows to ensure a successful integration:

```
"aws.s3.enable_ssl" = "{true | false}",
"aws.s3.enable_path_style_access" = "{true | false}",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.enable_ssl	Yes	Specifies whether to enable SSL connec Valid values: `true` and `false`. Default
aws.s3.enable_path_style_access	Yes	Specifies whether to enable path-style a Valid values: `true` and `false`. Default you must set the value to `true`. Path-style URLs use the following forma `<region_code>.amazonaws.com/<buc` For example, if you create a bucket nam `BUCKET1` in the US West (Oregon) Regic the `alice.jpg` object in that bucket, yo path-style URL: `https://s3.us-west-` `EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Yes	The endpoint that is used to connect to storage system instead of AWS S3.
aws.s3.access_key	Yes	The access key of your IAM user.
aws.s3.secret_key	Yes	The secret key of your IAM user.

MetadataUpdateParams

A set of parameters about how StarRocks updates the cached metadata of Hudi. This parameter set is optional.

StarRocks implements the automatic asynchronous update policy by default.

In most cases, you can ignore `MetadataUpdateParams` and do not need to tune the policy parameters in it, because the default values of these parameters already provide you with an out-of-the-box performance.

However, if the frequency of data updates in Hudi is high, you can tune these parameters to further optimize the performance of automatic asynchronous updates.

> **NOTE**
>
> In most cases, if your Hudi data is updated at a granularity of 1 hour or less, the data update frequency is considered high.

Parameter	Required	Description
enable_metastore_cache	No	Specifies whether StarRocks caches the metadata of Hudi tables. Valid values: `true` and `false`. Default value: `true`. The value `true` enables the cache, and the value `false` disables the cache.
enable_remote_file_cache	No	Specifies whether StarRocks caches the metadata of the underlying data files of Hudi tables or partitions. Valid values: `true` and `false`. Default value: `true`. The value `true` enables the cache, and the value `false` disables the cache.
metastore_cache_refresh_interval_sec	No	The time interval at which StarRocks asynchronously updates the metadata of Hudi tables or partitions cached in itself. Unit: seconds. Default value: `7200`, which is 2 hours.

Parameter	Required	Description
remote_file_cache_refresh_interval_sec	No	The time interval at which StarRocks asynchronously updates the metadata of the underlying data files of Hudi tables or partitions cached in itself. Unit: seconds. Default value: `60`.
metastore_cache_ttl_sec	No	The time interval at which StarRocks automatically discards the metadata of Hudi tables or partitions cached in itself. Unit: seconds. Default value: `86400`, which is 24 hours.
remote_file_cache_ttl_sec	No	The time interval at which StarRocks automatically discards the metadata of the underlying data files of Hudi tables or partitions cached in itself. Unit: seconds. Default value: `129600`, which is 36 hours.

Examples

The following examples create a Hudi catalog named `hudi_catalog_hms` or `hudi_catalog_glue`, depending on the type of metastore you use, to query data from your Hudi cluster.

HDFS

If you use HDFS as storage, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

If you choose instance profile-based credential

- If you use Hive metastore in your Hudi cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Hudi cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_glue
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

If you choose assumed role-based credential

- If you use Hive metastore in your Hudi cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Hudi cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_glue
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

If you choose IAM user-based credential

- If you use Hive metastore in your Hudi cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
```

```
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Hudi cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_glue
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "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" = "us-west-2"
);
```

S3-compatible storage system

Use MinIO as an example. Run a command like below:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "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>"
);
```

View the schema of a Hudi table

You can use one of the following syntaxes to view the schema of a Hudi table:

- View schema

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- View schema and location from the CREATE statement

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Query a Hudi table

1. Use the following syntax to view the databases in your Hudi cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Use the following syntax to connect to your target Hudi database:

```
USE <catalog_name>.<database_name>
```

3. Use the following syntax to query the Hudi table:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Load data from Hudi

Suppose you have an OLAP table named `olap_tbl`, you can transform and load data like below:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM hudi_table
```

Manually or automatically update metadata cache

Manual update

By default, StarRocks caches the metadata of Hudi and automatically updates the metadata in asynchronous mode to deliver better performance. Additionally, after some schema changes or table updates are made on a Hudi table, you can also use REFRESH EXTERNAL TABLE to manually update its metadata, thereby ensuring that StarRocks can obtain up-to-date metadata at its earliest opportunity and generate appropriate execution plans:

```
REFRESH EXTERNAL TABLE <table_name>
```

Appendix: Understand metadata automatic asynchronous update

Automatic asynchronous update is the default policy that StarRocks uses to update the metadata in Hudi catalogs.

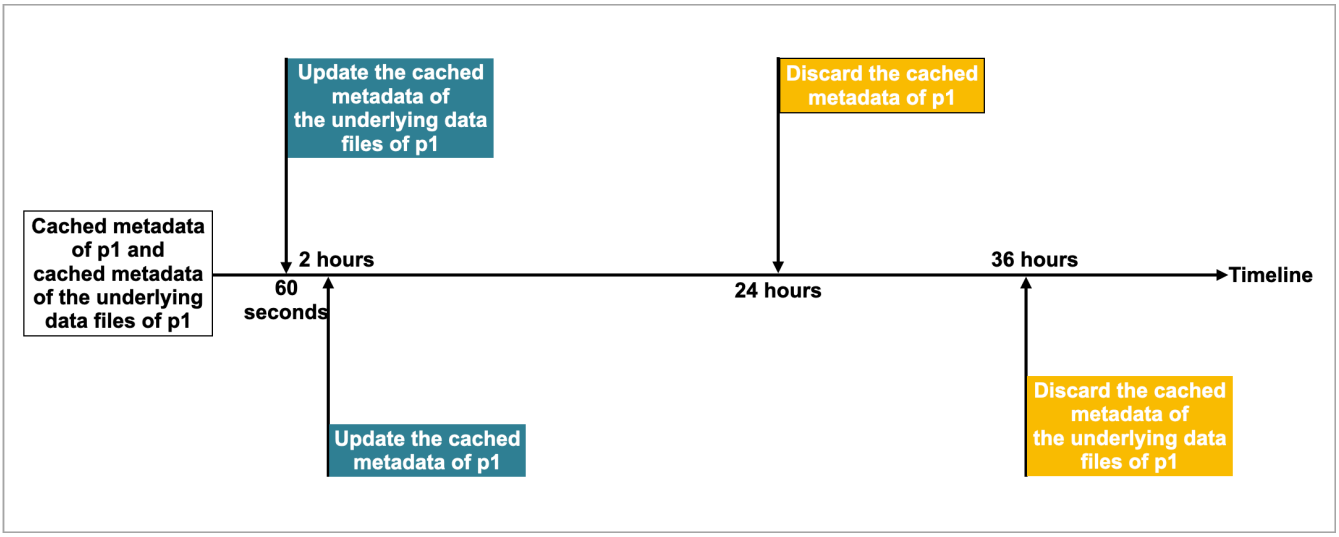
By default (namely, when the `enable_metastore_cache` and `enable_remote_file_cache` parameters are both set to `true`), if a query hits a partition of a Hudi table, StarRocks automatically caches the metadata of the partition and the metadata of the underlying data files of the partition. The cached metadata is updated by using the lazy update policy.

For example, there is a Hudi table named `table2`, which has four partitions: `p1`, `p2`, `p3`, and `p4`. A query hits `p1`, and StarRocks caches the metadata of `p1` and the metadata of the underlying data files of `p1`. Assume that the default time intervals to update and discard the cached metadata are as follows:

- The time interval (specified by the `metastore_cache_refresh_interval_sec` parameter) to asynchronously update the cached metadata of `p1` is 2 hours.
- The time interval (specified by the `remote_file_cache_refresh_interval_sec` parameter) to asynchronously update the cached metadata of the underlying data files of `p1` is 60 seconds.
- The time interval (specified by the `metastore_cache_ttl_sec` parameter) to automatically discard the cached metadata of `p1` is 24 hours.
- The time interval (specified by the `remote_file_cache_ttl_sec` parameter) to automatically discard the cached metadata of the underlying data files of `p1` is 36

hours.

The following figure shows the time intervals on a timeline for easier understanding.



Then StarRocks updates or discards the metadata in compliance with the following rules:

- If another query hits `p1` again and the current time from the last update is less than 60 seconds, StarRocks does not update the cached metadata of `p1` or the cached metadata of the underlying data files of `p1`.
- If another query hits `p1` again and the current time from the last update is more than 60 seconds, StarRocks updates the cached metadata of the underlying data files of `p1`.
- If another query hits `p1` again and the current time from the last update is more than 2 hours, StarRocks updates the cached metadata of `p1`.
- If `p1` has not been accessed within 24 hours from the last update, StarRocks discards the cached metadata of `p1`. The metadata will be cached at the next query.
- If `p1` has not been accessed within 36 hours from the last update, StarRocks discards the cached metadata of the underlying data files of `p1`. The metadata will be cached at the next query.

Delta Lake catalog

A Delta Lake catalog is a kind of external catalog that enables you to query data from Delta Lake without ingestion.

Also, you can directly transform and load data from Delta Lake by using INSERT INTO based on Delta Lake catalogs. StarRocks supports Delta Lake catalogs from v2.5 onwards.

To ensure successful SQL workloads on your Delta Lake cluster, your StarRocks cluster must be able to access the storage system and metastore of your Delta Lake cluster. StarRocks supports the following storage systems and metastores:

- Object storage or distributed file system like AWS S3 or HDFS
- Metastore like Hive metastore or AWS Glue

Usage notes

- The file format of Delta Lake that StarRocks supports is Parquet. Parquet files support the following compression formats: SNAPPY, LZ4, ZSTD, GZIP, and NO_COMPRESSION.
- The data types of Delta Lake that StarRocks does not support are MAP and STRUCT.

Integration preparations

Before you create a Delta Lake catalog, make sure your StarRocks cluster can integrate with the storage system and metastore of your Delta Lake cluster.

AWS IAM

If your Delta Lake cluster uses AWS S3 as storage or AWS Glue as metastore, choose your suitable authentication method and make the required preparations to ensure that your StarRocks cluster can access the related AWS cloud resources.

The following authentication methods are recommended:

- Instance profile
- Assumed role
- IAM user

Of the above-mentioned three authentication methods, instance profile is the most widely used.

For more information, see Preparation for authentication in AWS IAM.

HDFS

If you choose HDFS as storage, configure your StarRocks cluster as follows:

- (Optional) Set the username that is used to access your HDFS cluster and Hive metastore. By default, StarRocks uses the username of the FE and BE processes to access your HDFS cluster and Hive metastore. You can also set the username by adding `export HADOOP_USER_NAME="<user_name>"` at the beginning of the **fe/conf/hadoop_env.sh** file of each FE and at the beginning of the **be/conf/hadoop_env.sh** file of each BE. After you set the username in these files, restart each FE and each BE to make the parameter settings take effect. You can set only one username for each StarRocks cluster.

- When you query Delta Lake data, the FEs and BEs of your StarRocks cluster use the HDFS client to access your HDFS cluster. In most cases, you do not need to configure your StarRocks cluster to achieve that purpose, and StarRocks starts the HDFS client using the default configurations. You need to configure your StarRocks cluster only in the following situations:

    - High availability (HA) is enabled for your HDFS cluster: Add the **hdfs-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.
    - View File System (ViewFs) is enabled for your HDFS cluster: Add the **core-site.xml** file of your HDFS cluster to the **$FE_HOME/conf** path of each FE and to the **$BE_HOME/conf** path of each BE.

> **NOTE**
>
> If an error indicating an unknown host is returned when you send a query, you must add the mapping between the host names and IP addresses of your HDFS cluster nodes to the **/etc/hosts** path.

Kerberos authentication

If Kerberos authentication is enabled for your HDFS cluster or Hive metastore, configure your StarRocks cluster as follows:

- Run the `kinit -kt keytab_path principal` command on each FE and each BE to obtain Ticket Granting Ticket (TGT) from Key Distribution Center (KDC). To run this command, you must have the permissions to access your HDFS cluster and Hive metastore. Note that accessing KDC with this command is time-sensitive. Therefore, you need to use cron to run this command periodically.
- Add `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` to the **$FE_HOME/conf/fe.conf** file of each FE and to the **$BE_HOME/conf/be.conf** file of each BE. In this example, `/etc/krb5.conf` is the save path of the **krb5.conf** file. You can modify the path based on your needs.

Create a Delta Lake catalog

Syntax

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "deltalake",
    MetastoreParams,
    StorageCredentialParams,
    MetadataUpdateParams
)
```

Parameters

catalog_name

The name of the Delta Lake catalog. The naming conventions are as follows:

- The name can contain letters, digits (0-9), and underscores (_). It must start with a letter.
- The name is case-sensitive and cannot exceed 1023 characters in length.

comment

The description of the Delta Lake catalog. This parameter is optional.

type

The type of your data source. Set the value to `deltalake`.

MetastoreParams

A set of parameters about how StarRocks integrates with the metastore of your data source.

Hive metastore

If you choose Hive metastore as the metastore of your data source, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **NOTE**
>
> Before querying Delta Lake data, you must add the mapping between the host names and IP addresses of your Hive metastore nodes to the `/etc/hosts` path. Otherwise, StarRocks may fail to access your Hive metastore when you start a query.

The following table describes the parameter you need to configure in `MetastoreParams`.

Parameter	Required	Description
hive.metastore.type	Yes	The type of metastore that you use for your Delta Lake the value to `hive`.
hive.metastore.uris	Yes	The URI of your Hive metastore. Format: `thrift://<metastore_IP_address>:<metastore_` If high availability (HA) is enabled for your Hive metast specify multiple metastore URIs and separate them wi (`,`), for example, `"thrift://<metastore_IP_addre <metastore_port_1>,thrift://<metastore_IP_a`

Parameter	Required	Description
		`<metastore_port_2>,thrift://<metastore_IP_a` `<metastore_port_3>"`.

AWS Glue

If you choose AWS Glue as the metastore of your data source, take one of the following actions:

- To choose the instance profile-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the assumed role-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- To choose the IAM user-based authentication method, configure `MetastoreParams` as follows:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

The following table describes the parameters you need to configure in `MetastoreParams`.

Parameter	Required	Description
hive.metastore.type	Yes	The type of metastore that you use for your Delta Lake cluster. Set the value to `glue`.
aws.glue.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.glue.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS Glue Data Catalog. If you use the assumed role-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.region	Yes	The region in which your AWS Glue Data Catalog resides. Example: `us-west-1`.
aws.glue.access_key	No	The access key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.secret_key	No	The secret key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS Glue and how to configure an access control policy in the AWS IAM Console, see Authentication parameters for accessing AWS Glue.

StorageCredentialParams

A set of parameters about how StarRocks integrates with your storage system. This parameter set is optional.

You need to configure `StorageCredentialParams` only when your Delta Lake cluster uses AWS S3 as storage.

If your Delta Lake cluster uses any other storage system, you can ignore `StorageCredentialParams`.

AWS S3

If you choose AWS S3 as storage for your Delta Lake cluster, take one of the following actions:

- To choose the instance profile-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the assumed role-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the IAM user-based authentication method, configure `StorageCredentialParams` as follows:

```
"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>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you use the assumed role-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.region	Yes	The region in which your AWS S3 bucket resides. Example: `us-west-1`.
aws.s3.access_key	No	The access key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.secret_key	No	The secret key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS S3 and how to configure an access control policy in AWS IAM Console, see Authentication parameters for accessing AWS S3.

S3-compatible storage system

Delta Lake catalogs support S3-compatible storage systems from v2.5 onwards.

If you choose an S3-compatible storage system, such as MinIO, as storage for your Delta Lake cluster, configure `StorageCredentialParams` as follows to ensure a successful integration:

```
"aws.s3.enable_ssl" = "{true | false}",
"aws.s3.enable_path_style_access" = "{true | false}",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.enable_ssl	Yes	Specifies whether to enable SSL connec Valid values: `true` and `false`. Default
aws.s3.enable_path_style_access	Yes	Specifies whether to enable path-style a Valid values: `true` and `false`. Default you must set the value to `true`. Path-style URLs use the following forma `<region_code>.amazonaws.com/<buc` For example, if you create a bucket nam `BUCKET1` in the US West (Oregon) Regic the `alice.jpg` object in that bucket, yo path-style URL: `https://s3.us-west-` `EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Yes	The endpoint that is used to connect to storage system instead of AWS S3.
aws.s3.access_key	Yes	The access key of your IAM user.
aws.s3.secret_key	Yes	The secret key of your IAM user.

MetadataUpdateParams

A set of parameters about how StarRocks updates the cached metadata of Delta Lake. This parameter set is optional.

StarRocks implements the automatic asynchronous update policy by default.

In most cases, you can ignore `MetadataUpdateParams` and do not need to tune the policy parameters in it, because the default values of these parameters already provide you with an out-of-the-box performance.

However, if the frequency of data updates in Delta Lake is high, you can tune these parameters to further optimize the performance of automatic asynchronous updates.

> **NOTE**
>
> In most cases, if your Delta Lake data is updated at a granularity of 1 hour or less, the data update frequency is considered high.

Parameter	Required	Description
enable_metastore_cache	No	Specifies whether StarRocks caches the metadata of Delta Lake tables. Valid values: `true` and `false`. Default value: `true`. The value `true` enables the cache, and the value `false` disables the cache.
enable_remote_file_cache	No	Specifies whether StarRocks caches the metadata of the underlying data files of Delta Lake tables or partitions. Valid values: `true` and `false`. Default value: `true`. The value `true` enables the cache, and the value `false` disables the cache.

Parameter	Required	Description
metastore_cache_refresh_interval_sec	No	The time interval at which StarRocks asynchronously updates the metadata of Delta Lake tables or partitions cached in itself. Unit: seconds. Default value: `7200`, which is 2 hours.
remote_file_cache_refresh_interval_sec	No	The time interval at which StarRocks asynchronously updates the metadata of the underlying data files of Delta Lake tables or partitions cached in itself. Unit: seconds. Default value: `60`.
metastore_cache_ttl_sec	No	The time interval at which StarRocks automatically discards the metadata of Delta Lake tables or partitions cached in itself. Unit: seconds. Default value: `86400`, which is 24 hours.
remote_file_cache_ttl_sec	No	The time interval at which StarRocks automatically discards the metadata of the underlying data files of Delta Lake tables or partitions cached in itself. Unit: seconds. Default value: `129600`, which is 36 hours.

Examples

The following examples create a Delta Lake catalog named `deltalake_catalog_hms` or `deltalake_catalog_glue`, depending on the type of metastore you use, to query data from your Delta Lake cluster.

HDFS

If you use HDFS as storage, run a command like below:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

If you choose instance profile-based credential

- If you use Hive metastore in your Delta Lake cluster, run a command like below:

  ```
  CREATE EXTERNAL CATALOG deltalake_catalog_hms
  PROPERTIES
  (
      "type" = "deltalake",
      "hive.metastore.type" = "hive",
      "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
      "aws.s3.use_instance_profile" = "true",
      "aws.s3.region" = "us-west-2"
  );
  ```

- If you use AWS Glue in your Amazon EMR Delta Lake cluster, run a command like below:

  ```
  CREATE EXTERNAL CATALOG deltalake_catalog_glue
  PROPERTIES
  (
      "type" = "deltalake",
      "hive.metastore.type" = "glue",
      "aws.glue.use_instance_profile" = "true",
      "aws.glue.region" = "us-west-2",
      "aws.s3.use_instance_profile" = "true",
  ```

```
    "aws.s3.region" = "us-west-2"
);
```

If you choose assumed role-based credential

- If you use Hive metastore in your Delta Lake cluster, run a command like below:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Delta Lake cluster, run a command like below:

```
CREATE EXTERNAL CATALOG deltalake_catalog_glue
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

If you choose IAM user-based credential

- If you use Hive metastore in your Delta Lake cluster, run a command like below:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
```

```
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- If you use AWS Glue in your Amazon EMR Delta Lake cluster, run a command like below:

```
CREATE EXTERNAL CATALOG deltalake_catalog_glue
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "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" = "us-west-2"
);
```

S3-compatible storage system

Use MinIO as an example. Run a command like below:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "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>"
);
```

View the schema of a Delta Lake table

You can use one of the following syntaxes to view the schema of a Delta Lake table:

- View schema

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- View schema and location from the CREATE statement

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Query a Delta Lake table

1. Use the following syntax to view the databases in your Delta Lake cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Use the following syntax to connect to your target Delta Lake database:

```
USE <catalog_name>.<database_name>
```

3. Use the following syntax to query the Delta Lake table:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Load data from Delta Lake

Suppose you have an OLAP table named `olap_tbl`, you can transform and load data like below:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM
deltalake_table
```

Query external data

This topic guides you through querying data from external data sources by using external catalogs.

Prerequisites

External catalogs are created based on external data sources. For information about supported types of external catalogs, see Catalog.

Procedure

1. Connect your StarRocks cluster.
   - If you use the MySQL client to connect the StarRocks cluster, you go to `default_catalog` by default after connecting.
   - If you use JDBC to connect the StarRocks cluster, you can go directly to the destination database in the default catalog by specifying `default_catalog.db_name` when connecting.
2. (Optional) Execute the following statement to view all the catalogs and find the external catalog you have created. See SHOW CATALOGS to check the output of this statement.

```
SHOW CATALOGS;
```

3. (Optional) Execute the following statement to view all the databases in the external catalog. See SHOW DATABASES to check the output of this statement.

```
SHOW DATABASES FROM catalog_name;
```

4. (Optional) Execute the following statement to go to the destination database in the external catalog.

```
USE catalog_name.db_name;
```

5. Query external data. For more usages of the SELECT statement, see SELECT.

```
SELECT * FROM table_name;
```

If you do not specify the external catalog and database in the preceding steps, you can directly specify them in a select query.

```
SELECT * FROM catalog_name.db_name.table_name;
```

Examples

If you already created a Hive catalog named `hive1` and want to use `hive1` to query data from `hive_db.hive_table` of an Apache Hive™ cluster, you can perform one of the following operations:

```
USE hive1.hive_db;
SELECT * FROM hive_table limit 1;
```

Or

```
SELECT * FROM hive1.hive_db.hive_table limit 1;
```

References

To query data from your StarRocks cluster, see Default catalog.

External table

StarRocks supports access to other data sources by using external tables. External tables are created based on data tables that are stored in other data sources. StarRocks only stores the metadata of the data tables. You can use external tables to directly query data in other data sources. StarRocks supports the following data sources: MySQL, Elasticsearch, Hive, StarRocks, Apache Iceberg, and Apache Hudi. **Currently, you can only write data from another StarRocks cluster into the current StarRocks cluster. You cannot read data from it. For data sources other than StarRocks, you can only read data from these data sources.**

From 2.5 onwards, StarRocks provides the Data Cache feature, which accelerates hot data queriers on external data sources. For more information, see Data Cache.

MySQL external table

In the star schema, data is generally divided into dimension tables and fact tables. Dimension tables have less data but involve UPDATE operations. Currently, StarRocks does not support direct UPDATE operations (update can be implemented by using the Unique Key table). In some scenarios, you can store dimension tables in MySQL for direct data read.

To query MySQL data, you must create an external table in StarRocks and map it to the table in your MySQL database. You need to specify the MySQL connection information when creating the table.

```
CREATE EXTERNAL TABLE mysql_external_table
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=mysql
PROPERTIES
(
```

```
    "host" = "127.0.0.1",
    "port" = "3306",
    "user" = "mysql_user",
    "password" = "mysql_passwd",
    "database" = "mysql_db_test",
    "table" = "mysql_table_test"
);
```

Parameters:

- **host**: the connection address of the MySQL database
- **port**: the port number of the MySQL database
- **user**: the username to log in to MySQL
- **password**: the password to log in to MySQL
- **database**: the name of the MySQL database
- **table**: the name of the table in the MySQL database

StarRocks external table

From StarRocks 1.19 onwards, StarRocks allows you to use a StarRocks external table to write data from one StarRocks cluster to another. This achieves read-write separation and provides better resource isolation. You can first create a destination table in the destination StarRocks cluster. Then, in the source StarRocks cluster, you can create a StarRocks external table that has the same schema as the destination table and specify the information of the destination cluster and table in the `PROPERTIES` field.

Data can be written from a source cluster to a destination cluster by using INSERT INTO statement to write into a StarRocks external table. It can help realize the following goals:

- Data synchronization between StarRocks clusters.
- Read-write separation. Data is written to the source cluster, and data changes from the source cluster are synchronized to the destination cluster, which provides query services.

The following code shows how to create a destination table and an external table.

```
# Create a destination table in the destination StarRocks cluster.
CREATE TABLE t
(
```

```
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=olap
DISTRIBUTED BY HASH(k1) BUCKETS 10;

# Create an external table in the source StarRocks cluster.
CREATE EXTERNAL TABLE external_t
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=olap
DISTRIBUTED BY HASH(k1) BUCKETS 10
PROPERTIES
(
    "host" = "127.0.0.1",
    "port" = "9020",
    "user" = "user",
    "password" = "passwd",
    "database" = "db_test",
    "table" = "t"
);

# Write data from a source cluster to a destination cluster by
writing data into the StarRocks external table. The second statement
is recommended for the production environment.
insert into external_t values ('2020-10-11', 1, 1, 'hello', '2020-10-
11 10:00:00');
insert into external_t select * from other_table;
```

Parameters:

- **EXTERNAL:** This keyword indicates that the table to be created is an external table.
- **host:** This parameter specifies the IP address of the leader FE node of the destination StarRocks cluster.

- **port:** This parameter specifies the RPC port of the FE node of the destination StarRocks cluster.

  > **NOTE**
  >
  > To ensure that the source cluster to which the StarRocks external tables belong can access the destination StarRocks cluster, you must configure your network and firewall to allow access to the following ports:
  >
  > - The RPC port of the FE node. See `rpc_port` in the FE configuration file **fe/fe.conf**. The default RPC port is `9020`.
  > - The bRPC port of the BE node. See `brpc_port` in the BE configuration file **be/be.conf**. The default bRPC port is `8060`.

- **user:** This parameter specifies the username used to access the destination StarRocks cluster.

- **password:** This parameter specifies the password used to access the destination StarRocks cluster.

- **database:** This parameter specifies the database to which the destination table belongs.

- **table:** This parameter specifies the name of the destination table.

The following limits apply when you use a StarRocks external table:

- You can only run the INSERT INTO and SHOW CREATE TABLE commands on a StarRocks external table. Other data writing methods are not supported. In addition, you cannot query data from a StarRocks external table or perform DDL operations on the external table.
- The syntax of creating an external table is the same as creating a normal table, but the column names and other information in the external table must be the same as the destination table.
- The external table synchronizes table metadata from the destination table every 10 seconds. If a DDL operation is performed on the destination table, there may be a delay for data synchronization between the two tables.

Elasticsearch external table

StarRocks and Elasticsearch are two popular analytics systems. StarRocks is performant in large-scale distributed computing. Elasticsearch is ideal for full-text search. StarRocks combined with Elasticsearch can deliver a more complete OLAP solution.

Example of creating an Elasticsearch external table

Syntax

```
CREATE EXTERNAL TABLE elastic_search_external_table
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=ELASTICSEARCH
PROPERTIES (
    "hosts" = "http://192.168.0.1:9200,http://192.168.0.2:9200",
    "user" = "root",
    "password" = "root",
    "index" = "tindex",
    "type" = "_doc",
    "es.net.ssl" = "true"
);
```

The following table describes the parameters.

Parameter	Required	Default value	Description
hosts	Yes	None	The connection address of the Elasticsearch cluster. You can specify one or more addresses. StarRocks can parse the Elasticsearch version and index shard allocation from this address. StarRocks communicates with your Elasticsearch cluster based on the address returned by the `GET /_nodes/http` API operation.

Parameter	Required	Default value	Description
			Therefore, the value of the `host` parameter must be the same as the address returned by the `GET /_nodes/http` API operation. Otherwise, BEs may not be able to communicate with your Elasticsearch cluster.
index	Yes	None	The name of the Elasticsearch index that is created on the table in StarRocks. The name can be an alias. This parameter supports wildcards (*). For example, if you set `index` to `hello*`, StarRocks retrieves all indexes whose names start with `hello`.
user	No	Empty	The username that is used to log in to the Elasticsearch cluster with basic authentication enabled. Make sure you have access to `/*cluster/state/*nodes/http` and the index.
password	No	Empty	The password that is used to log in to the Elasticsearch cluster.
type	No	`_doc`	The type of the index. Default value: `_doc`. If you want to query data in Elasticsearch 8 and later versions, you do not need to configure this parameter because the mapping types have been removed in Elasticsearch 8 and later versions.

Parameter	Required	Default value	Description
es.nodes.wan.only	No	`false`	Specifies whether StarRocks only uses the addresses specified by `hosts` to access the Elasticsearch cluster and fetch data. • `true`: StarRocks only uses the addresses specified by `hosts` to access the Elasticsearch cluster and fetch data and does not sniff data nodes on which the shards of the Elasticsearch index reside. If StarRocks cannot access the addresses of the data nodes inside the Elasticsearch cluster, you need to set this parameter to `true`. • `false`: StarRocks uses the addresses specified by `host` to sniff data nodes on which the shards of the Elasticsearch cluster indexes reside. After StarRocks generates a query execution plan, the relevant BEs directly access the data nodes inside the Elasticsearch cluster to fetch data from the shards of indexes. If StarRocks can access the addresses of the data nodes inside the Elasticsearch cluster, we recommend that you retain the default value `false`.
es.net.ssl	No	`false`	Specifies whether the HTTPS protocol can be used to access your Elasticsearch cluster. Only StarRocks

Parameter	Required	Default value	Description
			2.4 and later versions support configuring this parameter. • `true`: Both the HTTPS and HTTP protocols can be used to access your Elasticsearch cluster. • `false`: Only the HTTP protocol can be used to access your Elasticsearch cluster.
enable_docvalue_scan	No	`true`	Specifies whether to obtain the values of the target fields from Elasticsearch columnar storage. In most cases, reading data from columnar storage outperforms reading data from row storage.
enable_keyword_sniff	No	`true`	Specifies whether to sniff TEXT-type fields in Elasticsearch based on KEYWORD-type fields. If this parameter is set to `false`, StarRocks performs matching after tokenization.

Columnar scan for faster queries

If you set `enable_docvalue_scan` to `true`, StarRocks follows these rules when it obtains data from Elasticsearch:

- **Try and see**: StarRocks automatically checks if columnar storage is enabled for the target fields. If so, StarRocks obtains all values in the target fields from columnar storage.
- **Auto-downgrading**: If any one of the target fields is unavailable in columnar storage, StarRocks parses and obtains all values in the target fields from row storage (`_source`).

> **NOTE**
>
> - Columnar storage is unavailable for TEXT-type fields in Elasticsearch. Therefore, if you query fields containing TEXT-type values, StarRocks obtains the values of the fields from `_source`.
> - If you query a large number (greater than or equal to 25) of fields, reading field values from `docvalue` does not show noticeable benefits compared with reading field values from `_source`.

Sniff KEYWORD-type fields

If you set `enable_keyword_sniff` to `true`, Elasticsearch allows direct data ingestion without an index because it will automatically create an index after ingestion. For STRING-type fields, Elasticsearch will create a field with both TEXT and KEYWORD types. This is how the Multi-Field feature of Elasticsearch works. The mapping is as follows:

```
"k4": {
   "type": "text",
   "fields": {
      "keyword": {
         "type": "keyword",
         "ignore_above": 256
      }
   }
}
```

For example, to conduct "=" filtering on `k4`, StarRocks on Elasticsearch will convert the filtering operation into an Elasticsearch TermQuery.

The original SQL filter is as follows:

```
k4 = "StarRocks On Elasticsearch"
```

The converted Elasticsearch query DSL is as follows:

```
"term" : {
    "k4": "StarRocks On Elasticsearch"

}
```

The first field of `k4` is TEXT, and it will be tokenized by the analyzer configured for `k4` (or by the standard analyzer if no analyzer has been configured for `k4`) after data ingestion. As a result, the first field will be tokenized into three terms: `StarRocks`, `On`, and `Elasticsearch`. The details are as follows:

```
POST /_analyze
{
  "analyzer": "standard",
  "text": "StarRocks On Elasticsearch"
}
```

The tokenization results are as follows:

```
{
   "tokens": [
      {
         "token": "starrocks",
         "start_offset": 0,
         "end_offset": 5,
         "type": "<ALPHANUM>",
         "position": 0
      },
      {
         "token": "on",
         "start_offset": 6,
         "end_offset": 8,
         "type": "<ALPHANUM>",
         "position": 1
      },
      {
         "token": "elasticsearch",
         "start_offset": 9,
         "end_offset": 11,
         "type": "<ALPHANUM>",
         "position": 2
      }
   ]
}
```

Suppose you conduct a query as follows:

```
"term" : {
    "k4": "StarRocks On Elasticsearch"
}
```

There is no term in the dictionary that matches the term `StarRocks On Elasticsearch`, and therefore no result will be returned.

However, if you have set `enable_keyword_sniff` to `true`, StarRocks will convert `k4 = "StarRocks On Elasticsearch"` to `k4.keyword = "StarRocks On Elasticsearch"` to match the SQL semantics. The converted `StarRocks On Elasticsearch` query DSL is as follows:

```
"term" : {
    "k4.keyword": "StarRocks On Elasticsearch"
}
```

`k4.keyword` is of the KEYWORD type. Therefore, the data is written into Elasticsearch as a complete term, allowing for successful matching.

Mapping of column data types

When you create an external table, you need to specify the data types of columns in the external table based on the data types of columns in the Elasticsearch table. The following table shows the mapping of column data types.

Elasticsearch	StarRocks
BOOLEAN	BOOLEAN
BYTE	TINYINT/SMALLINT/INT/BIGINT
SHORT	SMALLINT/INT/BIGINT
INTEGER	INT/BIGINT
LONG	BIGINT
FLOAT	FLOAT

Elasticsearch	StarRocks
DOUBLE	DOUBLE
KEYWORD	CHAR/VARCHAR
TEXT	CHAR/VARCHAR
DATE	DATE/DATETIME
NESTED	CHAR/VARCHAR
OBJECT	CHAR/VARCHAR
ARRAY	ARRAY

> **Note**
>
> - StarRocks reads the data of the NESTED type by using JSON-related functions.
> - Elasticsearch automatically flattens multi-dimensional arrays into one-dimensional arrays. StarRocks does the same. **The support for querying ARRAY data from Elasticsearch is added from v2.5.**

Predicate pushdown

StarRocks supports predicate pushdown. Filters can be pushed down to Elasticsearch for execution, which improves query performance. The following table lists the operators that support predicate pushdown.

SQL syntax	ES syntax
`=`	term query
`in`	terms query
`>=, <=, >, <`	range
`and`	bool.filter

SQL syntax	ES syntax
`or`	bool.should
`not`	bool.must_not
`not in`	bool.must_not + terms
`esquery`	ES Query DSL

Examples

The **esquery function** is used to push down queries **that cannot be expressed in SQL** (such as match and geoshape) to Elasticsearch for filtering. The first parameter in the esquery function is used to associate an index. The second parameter is a JSON expression of basic Query DSL, which is enclosed in brackets . **The JSON expression must have but only one root key**, such as match, geo_shape, or bool.

- match query

```
select * from es_table where esquery(k4, '{
    "match": {
       "k4": "StarRocks on elasticsearch"
    }
}');
```

- geo-related query

```
select * from es_table where esquery(k4, '{
  "geo_shape": {
     "location": {
        "shape": {
           "type": "envelope",
           "coordinates": [
              [
                 13,
                 53
              ],
              [
```

```
                14,
                52
            ]
        ]
      },
      "relation": "within"
    }
  }
}');
```

- bool query

```
select * from es_table where esquery(k4, ' {
    "bool": {
       "must": [
          {
             "terms": {
                "k1": [
                   11,
                   12
                ]
             }
          },
          {
             "terms": {
                "k2": [
                   100
                ]
             }
          }
       ]
    }
  }');
```

Usage notes

- Elasticsearch earlier than 5.x scans data in a different way than that later than 5.x. Currently, **only versions later than 5.x** are supported.
- Elasticsearch clusters with HTTP basic authentication enabled are supported.
- Querying data from StarRocks may not be as fast as directly querying data from Elasticsearch, such as count-related queries. The reason is that Elasticsearch directly

reads the metadata of target documents without the need to filter the real data, which accelerates the count query.

External table for a JDBC-compatible database

From v2.3.0, StarRocks provides external tables to query JDBC-compatible databases. This way, you can analyze the data of such databases in a blazing fast manner without the need to import the data into StarRocks. This section describes how to create an external table in StarRocks and query data in JDBC-compatible databases.

Prerequisites

Before you use a JDBC external table to query data, make sure that the FEs and BEs have access to the download URL of the JDBC driver. The download URL is specified by the `driver_url` parameter in the statement used for creating the JDBC resource.

Create and manage JDBC resources

Create a JDBC resource

Before you create an external table to query data from a database, you need to create a JDBC resource in StarRocks to manage the connection information of the database. The database must support the JDBC driver and is referred as the "target database". After creating the resource, you can use it to create external tables.

Execute the following statement to create a JDBC resource named `jdbc0`:

```
create external resource jdbc0
properties (
    "type"="jdbc",
    "user"="postgres",
    "password"="changeme",
    "jdbc_uri"="jdbc:postgresql://127.0.0.1:5432/jdbc_test",

"driver_url"="https://repo1.maven.org/maven2/org/postgresql/postgresql/
42.3.3.jar",
```

```
    "driver_class"="org.postgresql.Driver"
);
```

The required parameters in `properties` are as follows:

- `type`: the type of the resource. Set the value to `jdbc`.
- `user`: the username that is used to connect to the target database.
- `password`: the password that is used to connect to the target database.
- `jdbc_uri`: the URI that the JDBC driver uses to connect to the target database. The URI format must satisfy the database URI syntax. For the URI syntax of some common databases, visit the official websites of Oracle, PostgreSQL, SQL Server.

> Note: The URI must include the name of the target database. For example, in the preceding code example, `jdbc_test` is the name of the target database that you want to connect.

- `driver_url`: the download URL of the JDBC driver JAR package. An HTTP URL or file URL is supported, for example, `https://repo1.maven.org/maven2/org/postgresql/postgresql/42.3.3/postgresql-42.3.3.jar` or `file:///home/disk1/postgresql-42.3.3.jar`.
- `driver_class`: the class name of the JDBC driver. The JDBC driver class names of common databases are as follows:
  - MySQL: com.mysql.jdbc.Driver (MySQL 5.x and earlier), com.mysql.cj.jdbc.Driver (MySQL 6.x and later)
  - SQL Server: com.microsoft.sqlserver.jdbc.SQLServerDriver
  - Oracle: oracle.jdbc.driver.OracleDriver
  - PostgreSQL: org.postgresql.Driver

When the resource is being created, the FE downloads the JDBC driver JAR package by using the URL that is specified in the `driver_url` parameter, generates a checksum, and uses the checksum to verify the JDBC driver downloaded by BEs.

> Note: If the download of the JDBC driver JAR package fails, the creation of the resource also fails.

When BEs query the JDBC external table for the first time and find that the corresponding JDBC driver JAR package does not exist on their machines, BEs download the JDBC driver JAR package by using the URL that is specified in the `driver_url` parameter, and all JDBC driver JAR packages are saved in the `${STARROCKS_HOME}/lib/jdbc_drivers` directory.

View JDBC resources

Execute the following statement to view all JDBC resources in StarRocks:

```
SHOW RESOURCES;
```

> Note: The `ResourceType` column is `jdbc`.

Delete a JDBC resource

Execute the following statement to delete the JDBC resource named `jdbc0`:

```
DROP RESOURCE "jdbc0";
```

> Note: After a JDBC resource is deleted, all JDBC external tables that are created by using that JDBC resource are unavailable. However, the data in the target database is not lost. If you still need to use StarRocks to query data in the target database, you can create the JDBC resource and the JDBC external tables again.

Create a database

Execute the following statement to create and access a database named `jdbc_test` in StarRocks:

```
CREATE DATABASE jdbc_test;
USE jdbc_test;
```

> Note: The database name that you specify in the preceding statement does not need to be same as the name of the target database.

Create a JDBC external table

Execute the following statement to create a JDBC external table named `jdbc_tbl` in the database `jdbc_test`:

```
create external table jdbc_tbl (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=jdbc
properties (
    "resource" = "jdbc0",
    "table" = "dest_tbl"
);
```

The required parameters in `properties` are as follows:

- `resource`: the name of the JDBC resource used to create the external table.
- `table`: the target table name in the database.

For supported data types and data type mapping between StarRocks and target databases, see [Data type mapping](External_table.md#Data type mapping).

> Note:
>
> - Indexes are not supported.
> - You cannot use PARTITION BY or DISTRIBUTED BY to specify data distribution rules.

Query a JDBC external table

Before you query JDBC external tables, you must execute the following statement to enable the Pipeline engine:

```
set enable_pipeline_engine=true;
```

> Note: If the Pipeline engine is already enabled, you can skip this step.

Execute the following statement to query the data in the target database by using JDBC external tables.

```
select * from JDBC_tbl;
```

StarRocks supports predicate pushdown by pushing down filter conditions to the target table. Executing filter conditions as close as possible to the data source can improve query performance. Currently, StarRocks can push down operators, including the binary comparison operators (`>`, `>=`, `=`, `<`, and `<=`), `IN`, `IS NULL`, and `BETWEEN ... AND ...`. However, StarRocks can not push down functions.

Data type mapping

Currently, StarRocks can only query data of basic types in the target database, such as NUMBER, STRING, TIME, and DATE. If the ranges of data values in the target database are not supported by StarRocks, the query reports an error.

The mapping between the target database and StarRocks varies based on the type of the target database.

MySQL and StarRocks

MySQL	StarRocks
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
MEDIUMINTINT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL	DECIMAL
CHAR	CHAR

MySQL	StarRocks
VARCHAR	VARCHAR
DATE	DATE
DATETIME	DATETIME

Oracle and StarRocks

Oracle	StarRocks
CHAR	CHAR
VARCHARVARCHAR2	VARCHAR
DATE	DATE
SMALLINT	SMALLINT
INT	INT
BINARY_FLOAT	FLOAT
BINARY_DOUBLE	DOUBLE
DATE	DATE
DATETIME	DATETIME
NUMBER	DECIMAL

PostgreSQL and StarRocks

PostgreSQL	StarRocks
SMALLINTSMALLSERIAL	SMALLINT

PostgreSQL	StarRocks
INTEGERSERIAL	INT
BIGINTBIGSERIAL	BIGINT
BOOLEAN	BOOLEAN
REAL	FLOAT
DOUBLE PRECISION	DOUBLE
DECIMAL	DECIMAL
TIMESTAMP	DATETIME
DATE	DATE
CHAR	CHAR
VARCHAR	VARCHAR
TEXT	VARCHAR

SQL Server and StarRocks

SQL Server	StarRocks
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT

SQL Server	StarRocks
FLOAT	FLOAT
REAL	DOUBLE
DECIMALNUMERIC	DECIMAL
CHAR	CHAR
VARCHAR	VARCHAR
DATE	DATE
DATETIMEDATETIME2	DATETIME

Limits

- When you create JDBC external tables, you cannot create indexes on the tables or use PARTITION BY and DISTRIBUTED BY to specify data distribution rules for the tables.
- When you query JDBC external tables, StarRocks cannot push down functions to the tables.

Hive external table

Before using Hive external tables, make sure JDK 1.8 has been installed on your servers.

Create a Hive resource

A Hive resource corresponds to a Hive cluster. You must configure the Hive cluster used by StarRocks, such as the Hive metastore address. You must specify the Hive resource that is used by the Hive external table.

- Create a Hive resource named hive0.

```
CREATE EXTERNAL RESOURCE "hive0"
PROPERTIES (
  "type" = "hive",
  "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

- View the resources created in StarRocks.

```
SHOW RESOURCES;
```

- Delete the resource named `hive0`.

```
DROP RESOURCE "hive0";
```

You can modify `hive.metastore.uris` of a Hive resource in StarRocks 2.3 and later versions. For more information, see ALTER RESOURCE.

Create a database

```
CREATE DATABASE hive_test;
USE hive_test;
```

Create a Hive external table

Syntax

```
CREATE EXTERNAL TABLE table_name (
  col_name col_type [NULL | NOT NULL] [COMMENT "comment"]
) ENGINE=HIVE
PROPERTIES (
  "key" = "value"
);
```

Example: Create the external table `profile_parquet_p7` under the `rawdata` database in the Hive cluster corresponding to the `hive0` resource.

```
CREATE EXTERNAL TABLE `profile_wos_p7` (
  `id` bigint NULL,
  `first_id` varchar(200) NULL,
  `second_id` varchar(200) NULL,
  `p__device_id_list` varchar(200) NULL,
  `p__is_deleted` bigint NULL,
  `p_channel` varchar(200) NULL,
  `p_platform` varchar(200) NULL,
  `p_source` varchar(200) NULL,
  `p__city` varchar(200) NULL,
  `p__province` varchar(200) NULL,
  `p__update_time` bigint NULL,
  `p__first_visit_time` bigint NULL,
  `p__last_seen_time` bigint NULL
) ENGINE=HIVE
PROPERTIES (
  "resource" = "hive0",
  "database" = "rawdata",
  "table" = "profile_parquet_p7"
);
```

Description:

- Columns in the external table
  - The column names must be the same as column names in the Hive table.
  - The column order **does not need to be** the same as column order in the Hive table.
  - You can select only **some of the columns in the Hive table**, but you must select all the **partition key columns**.
  - Partition key columns of an external table do not need to be specified by using `partition by`. They must be defined in the same description list as other columns. You do not need to specify partition information. StarRocks will automatically synchronize this information from the Hive table.
  - Set `ENGINE` to HIVE.
- PROPERTIES:
  - **hive.resource**: the Hive resource that is used.
  - **database**: the Hive database.
  - **table**: the table in Hive. **view** is not supported.

- The following table describes the column data type mapping between Hive and StarRocks.

Column type of Hive	Column type of StarRocks	Description
INT/INTEGER	INT	
BIGINT	BIGINT	
TIMESTAMP	DATETIME	Precision and time zone information will be lost when you convert TIMESTAMP data into DATETIME data. You need to convert TIMESTAMP data into DATETIME data that does not have the time zone offset based on the time zone in sessionVariable.
STRING	VARCHAR	
VARCHAR	VARCHAR	
CHAR	CHAR	
DOUBLE	DOUBLE	
FLOAT	FLOAT	
DECIMAL	DECIMAL	
ARRAY	ARRAY	

> Note:
>
> - Currently, the supported Hive storage formats are Parquet, ORC, and CSV. If the storage format is CSV, quotation marks cannot be used as escape characters.
> - The SNAPPY and LZ4 compression formats are supported.

> - The maximum length of a Hive string column that can be queried is 1 MB. If a string column exceeds 1 MB, it will be processed as a null column.

Use a Hive external table

Query the total number of rows of `profile_wos_p7`.

```
select count(*) from profile_wos_p7;
```

Update cached Hive table metadata

- Hive partition information and the related file information are cached in StarRocks. The cache is refreshed at intervals specified by `hive_meta_cache_refresh_interval_s`. The default value is 7200. `hive_meta_cache_ttl_s` specifies the timeout duration of the cache and the default value is 86400.
  - The cached data can also be refreshed manually.
    a. If a partition is added or deleted from a table in Hive, you must run the `REFRESH EXTERNAL TABLE hive_t` command to refresh the table metadata cached in StarRocks. `hive_t` is the name of the Hive external table in StarRocks.
    b. If data in some Hive partitions is updated, you must refresh the cached data in StarRocks by running the `REFRESH EXTERNAL TABLE hive_t PARTITION ('k1=01/k2=02', 'k1=03/k2=04')` command. `hive_t` is the name of the Hive external table in StarRocks. `'k1=01/k2=02'` and `'k1=03/k2=04'` are the names of Hive partitions whose data is updated.
    c. When you run `REFRESH EXTERNAL TABLE hive_t`, StarRocks first checks if the column information of the Hive external table is the same as the column information of the Hive table returned by the Hive Metastore. If the schema of the Hive table changes, such as adding columns or removing columns, StarRocks synchronizes the changes to the Hive external table. After synchronization, the column order of the Hive external table remains the same as the column order of the Hive table, with the partition column being the last column.
- When Hive data is stored in the Parquet, ORC, and CSV format, you can synchronize schema changes (such as ADD COLUMN and REPLACE COLUMN) of a Hive table to a Hive external table in StarRocks 2.3 and later versions.

Access object storage

- The path of the FE configuration file is `fe/conf`, to which the configuration file can be added if you need to customize the Hadoop cluster. For example: If the HDFS cluster uses a highly available nameservice, you need to put `hdfs-site.xml` under `fe/conf`. If HDFS is configured with ViewFs, you need to put the `core-site.xml` under `fe/conf`.

- The path of the BE configuration file is `be/conf`, to which the configuration file can be added if you need to customize the Hadoop cluster. For example, if the HDFS cluster using a highly available nameservice, you need to put `hdfs-site.xml` under `be/conf`. If HDFS is configured with ViewFs, you need to put `core-site.xml` under `be/conf`.

- On the machine where BE is located, configure JAVA_HOME as a JDK environment rather than a JRE environment in the BE **startup script** `bin/start_be.sh`, for example, `export JAVA_HOME = <JDK path>`. You must add this configuration at the beginning of the script and restart the BE for the configuration to take effect.

- Configure Kerberos support:

    i. To log in with `kinit -kt keytab_path principal` to all FE/BE machines, you need to have access to Hive and HDFS. The kinit command login is only good for a period of time and needs to be put into crontab to be executed regularly.

    ii. Put `hive-site.xml/core-site.xml/hdfs-site.xml` under `fe/conf`, and put `core-site.xml/hdfs-site.xml` under `be/conf`.

    iii. Add `-Djava.security.krb5.conf=/etc/krb5.conf` to the value of the `JAVA_OPTS` option in the **$FE_HOME/conf/fe.conf** file. **/etc/krb5.conf** is the save path of the **krb5.conf** file. You can change the path based on your operating system.

    iv. Directly add `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` to the **$BE_HOME/conf/be.conf** file. **/etc/krb5.conf** is the save path of the **krb5.conf** file. You can change the path based on your operating system.

    v. When you add a Hive resource, you must pass in a domain name to `hive.metastore.uris`. In addition, you must add the mapping between Hive/HDFS domain names and IP addresses in the **/etc/hosts** file.

- Configure support for AWS S3: Add the following configuration to `fe/conf/core-site.xml` and `be/conf/core-site.xml`.

```
<configuration>
   <property>
      <name>fs.s3a.access.key</name>
      <value>******</value>
   </property>
   <property>
      <name>fs.s3a.secret.key</name>
      <value>******</value>
   </property>
   <property>
      <name>fs.s3a.endpoint</name>
      <value>s3.us-west-2.amazonaws.com</value>
   </property>
   <property>
   <name>fs.s3a.connection.maximum</name>
   <value>500</value>
   </property>
</configuration>
```

i. `fs.s3a.access.key`: the AWS access key ID.

ii. `fs.s3a.secret.key`: the AWS secret key.

iii. `fs.s3a.endpoint`: the AWS S3 endpoint to connect to.

iv. `fs.s3a.connection.maximum`: the maximum number of concurrent connections from StarRocks to S3. If an error `Timeout waiting for connection from poll` occurs during a query, you can set this parameter to a larger value.

(Deprecated) Iceberg external table

From v2.1.0, StarRocks allows you to query data from Apache Iceberg by using external tables. To query data in Iceberg, you need to create an Iceberg external table in StarRocks. When you create the table, you need to establish mapping between the external table and the Iceberg table you want to query.

Before you begin

Make sure that StarRocks has permissions to access the metadata service (such as Hive metastore), file system (such as HDFS), and object storage system (such as Amazon S3 and Alibaba Cloud Object Storage Service) used by Apache Iceberg.

Precautions

- The Iceberg external table can be used to query only the following types of data:
    - Versions 1 (Analytic Data Tables) tables. Versions 2 (Row-level Deletes) tables are not supported. For the differences between Versions 1 tables and Versions 2 tables, see Iceberg Table Spec.
    - Tables that are compressed in gzip (default format), Zstd, LZ4, or Snappy format.
    - Files that are stored in Parquet or ORC format.
- Iceberg external tables in StarRocks 2.3 and later versions support synchronizing schema changes of an Iceberg table while Iceberg external tables in versions earlier than StarRocks 2.3 do not. If the schema of an Iceberg table changes, you must delete the corresponding external table and create a new one.

Procedure

Step 1: Create an Iceberg resource

Before you create an Iceberg external table, you must create an Iceberg resource in StarRocks. The resource is used to manage the Iceberg access information. Additionally, you also need to specify this resource in the statement that is used to create the external table. You can create a resource based on your business requirements:

- If the metadata of an Iceberg table is obtained from a Hive metastore, you can create a resource and set the catalog type to `HIVE`.
- If the metadata of an Iceberg table is obtained from other services, you need to create a custom catalog. Then create a resource and set the catalog type to `CUSTOM`.

Create a resource whose catalog type is `HIVE`

For example, create a resource named `iceberg0` and set the catalog type to `HIVE`.

```
CREATE EXTERNAL RESOURCE "iceberg0"
PROPERTIES (
   "type" = "iceberg",
   "iceberg.catalog.type" = "HIVE",
   "iceberg.catalog.hive.metastore.uris" =
```

```
"thrift://xx.xx.xx.xx:9083"
);
```

The following table describes the related parameters.

Parameter	Description
type	The resource type. Set the value to `iceberg`.
iceberg.catalog.type	The catalog type of the resource. Both Hive catalog and custom catalog are supported. If you specify a Hive catalog, set the value to `HIVE`.If you specify a custom catalog, set the value to `CUSTOM`.
iceberg.catalog.hive.metastore.uris	The URI of the Hive metastore. The parameter value is in the following format: `thrift://< IP address of Iceberg metadata >:< port number >`. The port number defaults to 9083. Apache Iceberg uses a Hive catalog to access the Hive metastore and then queries the metadata of Iceberg tables.

Create a resource whose catalog type is `CUSTOM`

A custom catalog needs to inherit the abstract class BaseMetastoreCatalog, and you need to implement the IcebergCatalog interface. Additionally, the class name of a custom catalog cannot be duplicated with the name of the class that already exists in StarRock. After the catalog is created, package the catalog and its related files, and place them under the **fe/lib** path of each frontend (FE). Then restart each FE. After you complete the preceding operations, you can create a resource whose catalog is a custom catalog.

For example, create a resource named `iceberg1` and set the catalog type to `CUSTOM`.

```
CREATE EXTERNAL RESOURCE "iceberg1"
PROPERTIES (
   "type" = "iceberg",
   "iceberg.catalog.type" = "CUSTOM",
```

```
    "iceberg.catalog-impl" = "com.starrocks.IcebergCustomCatalog"
);
```

The following table describes the related parameters.

Parameter	Description
type	The resource type. Set the value to `iceberg`.
iceberg.catalog.type	The catalog type of the resource. Both Hive catalog and custom catalog are supported. If you specify a Hive catalog, set the value to `HIVE`. If you specify a custom catalog, set the value to `CUSTOM`.
iceberg.catalog-impl	The fully qualified class name of the custom catalog. FEs search for the catalog based on this name. If the catalog contains custom configuration items, you must add them to the `PROPERTIES` parameter as key-value pairs when you create an Iceberg external table.

You can modify `hive.metastore.uris` and `iceberg.catalog-impl` of a Iceberg resource in StarRocks 2.3 and later versions. For more information, see ALTER RESOURCE.

View Iceberg resources

```
SHOW RESOURCES;
```

Drop an Iceberg resource

For example, drop a resource named `iceberg0`.

```
DROP RESOURCE "iceberg0";
```

Dropping an Iceberg resource makes all external tables that reference this resource unavailable. However, the corresponding data in Apache Iceberg is not deleted. If you still need to query the data in Apache Iceberg, create a new resource and a new external table.

Step 2: (Optional) Create a database

For example, create a database named `iceberg_test` in StarRocks.

```
CREATE DATABASE iceberg_test;
USE iceberg_test;
```

> Note: The name of the database in StarRocks can be different from the name of the database in Apache Iceberg.

Step 3: Create an Iceberg external table

For example, create an Iceberg external table named `iceberg_tbl` in the database `iceberg_test`.

```
CREATE EXTERNAL TABLE `iceberg_tbl` (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=ICEBERG
PROPERTIES (
    "resource" = "iceberg0",
    "database" = "iceberg",
    "table" = "iceberg_table"
);
```

The following table describes the related parameters.

Parameter	Description
ENGINE	The engine name. Set the value to `ICEBERG`.
resource	The name of the Iceberg resource that the external table references.
database	The name of the database to which the Iceberg table belongs.
table	The name of the Iceberg table.

> Note:
>
> - The name of the external table can be different from the name of the Iceberg table.

> - The column names of the external table must be the same as those in the Iceberg table. The column order of the two tables can be different.

If you define configuration items in the custom catalog and want configuration items to take effect when you query data, you can add the configuration items to the `PROPERTIES` parameter as key-value pairs when you create an external table. For example, if you define a configuration item `custom-catalog.properties` in the custom catalog, you can run the following command to create an external table.

```
CREATE EXTERNAL TABLE `iceberg_tbl` (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=ICEBERG
PROPERTIES (
    "resource" = "iceberg0",
    "database" = "iceberg",
    "table" = "iceberg_table",
    "custom-catalog.properties" = "my_property"
);
```

When you create an external table, you need to specify the data types of columns in the external table based on the data types of columns in the Iceberg table. The following table shows the mapping of column data types.

Iceberg table	Iceberg external table
BOOLEAN	BOOLEAN
INT	TINYINT / SMALLINT / INT
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL(P, S)	DECIMAL
DATE	DATE / DATETIME

Iceberg table	Iceberg external table
TIME	BIGINT
TIMESTAMP	DATETIME
STRING	STRING / VARCHAR
UUID	STRING / VARCHAR
FIXED(L)	CHAR
BINARY	VARCHAR
LIST	ARRAY

StarRocks does not support querying Iceberg data whose data type is TIMESTAMPTZ, STRUCT, and MAP.

Step 4: Query the data in Apache Iceberg

After an external table is created, you can query the data in Apache Iceberg by using the external table.

```
select count(*) from iceberg_tbl;
```

Hudi external table

From v2.2.0, StarRocks allows you to query data from Hudi data lakes by using Hudi external tables, thus facilitating blazing-fast data lake analytics. This topic describes how to create a Hudi external table in your StarRocks cluster and use the Hudi external table to query data from a Hudi data lake.

Before you begin

Make sure that your StarRocks cluster is granted access to the Hive metastore, HDFS cluster, or bucket with which you can register Hudi tables.

Precautions

- Hudi external tables for Hudi are read-only and can be used only for queries.
- StarRocks supports querying Copy on Write and Merge On Read tables (MOR tables are supported from v2.5). For the differences between these two types of tables, see Table & Query Types.
- StarRocks supports the following two query types of Hudi: Snapshot Queries and Read Optimized Queries (Hudi only supports performing Read Optimized Queries on Merge On Read tables). Incremental Queries are not supported. For more information about the query types of Hudi, see Table & Query Types.
- StarRocks supports the following compression formats for Hudi files: gzip, zstd, LZ4, and Snappy. The default compression format for Hudi files is gzip.
- StarRocks cannot synchronize schema changes from Hudi managed tables. For more information, see Schema Evolution. If the schema of a Hudi managed table is changed, you must delete the associated Hudi external table from your StarRocks cluster and then re-create that external table.

Procedure

Step 1: Create and manage Hudi resources

You must create Hudi resources in your StarRocks cluster. The Hudi resources are used to manage the Hudi databases and external tables that you create in your StarRocks cluster.

Create a Hudi resource

Execute the following statement to create a Hudi resource named `hudi0`:

```
CREATE EXTERNAL RESOURCE "hudi0"
PROPERTIES (
    "type" = "hudi",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

The following table describes the parameters.

Parameter	Description
type	The type of the Hudi resource. Set the vaue to hudi.
hive.metastore.uris	The Thrift URI of the Hive metastore to which the Hudi resource connects. After connecting the Hudi resource to a Hive metastore, you can create and manage Hudi tables by using Hive. The Thrift URI is in the `<IP address of the Hive metastore>:<Port number of the Hive metastore>` format. The default port number is 9083.

From v2.3 onwards, StarRocks allows changing the `hive.metastore.uris` value of a Hudi resource. For more information, see ALTER RESOURCE.

View Hudi resources

Execute the following statement to view all Hudi resources that are created in your StarRocks cluster:

```
SHOW RESOURCES;
```

Delete a Hudi resource

Execute the following statement to delete the Hudi resource named `hudi0`:

```
DROP RESOURCE "hudi0";
```

> Note:
>
> Deleting a Hudi resource causes unavailability of all Hudi external tables that are created by using that Hudi resource. However, the deletion does not affect your data stored in Hudi. If you still want to query your data from Hudi by using StarRocks, you must re-create Hudi resources, Hudi databases, and Hudi external tables in your StarRocks cluster.

Step 2: Create Hudi databases

Execute the following statement to create and open a Hudi database named `hudi_test` in your StarRocks cluster:

```
CREATE DATABASE hudi_test;
USE hudi_test;
```

> Note:
>
> The name that you specify for the Hudi database in your StarRocks cluster does not need to be the same as the associated database in Hudi.

Step 3: Create Hudi external tables

Execute the following statement to create a Hudi external table named `hudi_tbl` in the `hudi_test` Hudi database:

```
CREATE EXTERNAL TABLE `hudi_tbl` (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=HUDI
PROPERTIES (
    "resource" = "hudi0",
    "database" = "hudi",
    "table" = "hudi_table"
);
```

The following table describes the parameters.

Parameter	Description
ENGINE	The query engine of the Hudi external table. Set the value to `HUDI`.
resource	The name of the Hudi resource in your StarRocks cluster.
database	The name of the Hudi database to which the Hudi external table belongs in your StarRocks cluster.
table	The Hudi managed table with which the Hudi external table is associated.

> Note:
>
> - The name that you specify for the Hudi external table does not need to be the same as the associated Hudi managed table.
> - The columns in the Hudi external table must have the same names but can be in a different sequence compared to their counterpart columns in the associated Hudi managed table.
> - You can select some or all columns from the associated Hudi managed table and create only the selected columns in the Hudi external table. The following table lists the mapping between the data types supported by Hudi and the data types supported by StarRocks.

Data types supported by Hudi	Data types supported by StarRocks
BOOLEAN	BOOLEAN
INT	TINYINT/SMALLINT/INT
DATE	DATE
TimeMillis/TimeMicros	TIME
TimestampMillis/TimestampMicros	DATETIME
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
STRING	CHAR/VARCHAR
ARRAY	ARRAY
DECIMAL	DECIMAL

> **Note**

> StarRocks does not support querying data of the STRUCT or MAP type, nor does it support querying data of the ARRAY type in Merge On Read tables.

Step 4: Query data from a Hudi external table

After you create a Hudi external table associated with a specific Hudi managed table, you do not need to load data into the Hudi external table. To query data from Hudi, execute the following statement:

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

File external table

File external table is a special type of external table. It allows you to directly query Parquet and ORC data files in external storage systems without loading data into StarRocks. In addition, file external tables do not rely on a metastore. In the current version, StarRocks supports the following external storage systems: HDFS, Amazon S3, and S3-compatible storage systems.

This feature is supported from StarRocks v2.5.

> **NOTE**
>
> The File external feature was designed to help with importing data into StarRocks NOT to perform queries against external systems as a normal operation. Although you can query and use functions like JOIN, it is not performant. A more performant solution would be to import the data into StarRocks.

Limits

- File external tables must be created in databases within the default_catalog. You can run SHOW CATALOGS to query catalogs created in the cluster.
- Only Parquet and ORC data files are supported.
- You can only use file external tables to query data in the target data file. Data write operations such as INSERT, DELETE, and DROP are not supported.

Prerequisites

Before you create a file external table, you must configure your StarRocks cluster so that StarRocks can access the external storage system where the target data file is stored. The configurations required for a file external table are the same as those required for a Hive catalog, except that you do not need to configure a metastore. See Hive catalog - Integration preparations for more information about configurations.

Create a database (Optional)

After connecting to your StarRocks cluster, you can create a file external table in an existing database or create a new database to manage file external tables. To query existing databases in the cluster, run SHOW DATABASES. Then you can run `USE <db_name>` to switch to the target database.

The syntax for creating a database is as follows.

```
CREATE DATABASE [IF NOT EXISTS] <db_name>
```

Create a file external table

After accessing the target database, you can create a file external table in this database.

Syntax

```
CREATE EXTERNAL TABLE <table_name>
(
    <col_name> <col_type> [NULL | NOT NULL] [COMMENT "<comment>"]
)
ENGINE=file
COMMENT ["comment"]
PROPERTIES
(
    FileLayoutParams,
    StorageCredentialParams
)
```

Parameters

Parameter	Required	Description
table_name	Yes	The name of the file external table. The naming conventions are as follows: • The name can contain letters, digits (0-9), and underscores (_). It must start with a letter.

Parameter	Required	Description
		• The name cannot exceed 64 characters in length.
col_name	Yes	The column name in the file external table. The column names in the file external table must be the same as those in the target data file but are not case-sensitive. The order of columns in the file external table can be different from that in the target data file.
col_type	Yes	The column type in the file external table. You need to specify this parameter based on the column type in the target data file. For more information, see Mapping of column types.
NULL | NOT NULL	No	Whether the column in the file external table is allowed to be NULL. • NULL: NULL is allowed. • NOT NULL: NULL is not allowed. You must specify this modifier based on the following rules: • If this parameter is not specified for the columns in the target data file, you can choose not to specify it for the columns in the file external table or specify NULL for the columns in the file external table. • If NULL is specified for the columns in the target data file, you can choose not to specify this parameter for the columns in the file external table or specify NULL for the columns in the file external table. • If NOT NULL is specified for the columns in the target data file, you must also specify NOT NULL for the columns in the file external table.
comment	No	The comment of column in the file external table.
ENGINE	Yes	The type of engine. Set the value to file.

Parameter	Required	Description
comment	No	The description of the file external table.
PROPERTIES	Yes	• `FileLayoutParams`: specifies the path and format of the target file. This property is required. • `StorageCredentialParams`: specifies the authentication information required for accessing object storage systems. This property is required only for AWS S3 and S3-compatible storage.

FileLayoutParams

A set of parameters for accessing the target data file.

```
"path" = "<file_path>",
"format" = "<file_format>"
"enable_recursive_listing" = "{ true | false }"
```

Parameter	Required	Description
path	Yes	The path of the data file. • If the data file is stored in HDFS, the path forr `address of HDFS>:<port>/<path>`. The d 8020. If you use the default port, you do not r • If the data file is stored in AWS S3 or other S3 system, the path format is `s3://<bucket na` Note the following rules when you enter the path: • If you want to access all files in a path, end th slash (`/`), such as `hdfs://x.x.x.x/user/hive/warehouse/a` When you run a query, StarRocks traverses al path. It does not traverse data files by using r • If you want to access a single file, enter a patl this file, such as

Parameter	Required	Description
		`hdfs://x.x.x.x/user/hive/warehouse/a` When you run a query, StarRocks only scans t
format	Yes	The format of the data file. Valid values: `parquet`
enable_recursive_listing	No	Specifies whether to recursively transverse all files path. Default value: `true`. The value `true` specif subdirectories, and the value `false` specifies to i

StorageCredentialParams (Optional)

A set of parameters about how StarRocks integrates with the target storage system. This parameter set is **optional**.

You need to configure `StorageCredentialParams` only when the target storage system is AWS S3 or S3-compatible storage.

For other storage systems, you can ignore `StorageCredentialParams`.

AWS S3

If you need to access a data file stored in AWS S3, configure the following authentication parameters in `StorageCredentialParams`.

- If you choose the instance profile-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- If you choose the assumed role-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<ARN of your assumed role>",
"aws.s3.region" = "<aws_s3_region>"
```

- If you choose the IAM user-based authentication method, configure `StorageCredentialParams` as follows:

```
"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>"
```

Parameter name	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method when you access AWS S3. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	Yes	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you use the assumed role-based authentication method to access AWS S3, you must specify this parameter. Then, StarRocks will assume this role when it accesses the target data file.
aws.s3.region	Yes	The region in which your AWS S3 bucket resides. Example: us-west-1.
aws.s3.access_key	No	The access key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.secret_key	No	The secret key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.

For information about how to choose an authentication method for accessing AWS S3 and how to configure an access control policy in the AWS IAM Console, see Authentication parameters for accessing AWS S3.

AWS S3-compatible storage

If you need to access an AWS S3-compatible storage system, such as MinIO, configure `StorageCredentialParams` as follows to ensure a successful integration:

```
"aws.s3.enable_ssl" = "{ true | false }",
"aws.s3.enable_path_style_access" = "{ true | false }",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.enable_ssl	Yes	Specifies whether to enable SSL connec Valid values: `true` and `false`. Default
aws.s3.enable_path_style_access	Yes	Specifies whether to enable path-style a Valid values: `true` and `false`. Default you must set the value to `true`. Path-style URLs use the following forma `<region_code>.amazonaws.com/<buc` For example, if you create a bucket nam `BUCKET1` in the US West (Oregon) Regic the `alice.jpg` object in that bucket, yo path-style URL: `https://s3.us-west-` `EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Yes	The endpoint used to connect to an S3- system instead of AWS S3.
aws.s3.access_key	Yes	The access key of your IAM user.

Parameter	Required	Description
aws.s3.secret_key	Yes	The secret key of your IAM user.

Mapping of column types

The following table provides the mapping of column types between the target data file and the file external table.

Data file	File external table
INT/INTEGER	INT
BIGINT	BIGINT
TIMESTAMP	DATETIME. Note that TIMESTAMP is converted to DATETIME without a time zone based on the time zone setting of the current session and loses some of its precision.
STRING	STRING
VARCHAR	VARCHAR
CHAR	CHAR
DOUBLE	DOUBLE
FLOAT	FLOAT
DECIMAL	DECIMAL
BOOLEAN	BOOLEAN
ARRAY	ARRAY
MAP	MAP

Data file	File external table
STRUCT	STRUCT

Examples

HDFS

Create a file external table named `t0` to query Parquet data files stored in an HDFS path.

```
USE db_example;

CREATE EXTERNAL TABLE t0
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path"="hdfs://x.x.x.x:8020/user/hive/warehouse/person_parq/",
    "format"="parquet"
);
```

AWS S3

Example 1: Create a file external table and use **instance profile** to access **a single Parquet file** in AWS S3.

```
USE db_example;

CREATE EXTERNAL TABLE table_1
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path" = "s3://bucket-test/folder1/raw_0.parquet",
    "format" = "parquet",
```

```
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Example 2: Create a file external table and use **assumed role** to access **all the ORC files** under the target file path in AWS S3.

```
USE db_example;

CREATE EXTERNAL TABLE table_1
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path" = "s3://bucket-test/folder1/",
    "format" = "orc",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::51234343412:role/role_name_in_aws_iam",
    "aws.s3.region" = "us-west-2"
);
```

Example 3: Create a file external table and use **IAM user** to access **all the ORC files** under the file path in AWS S3.

```
USE db_example;

CREATE EXTERNAL TABLE table_1
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path" = "s3://bucket-test/folder1/",
    "format" = "orc",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
```

```
    "aws.s3.region" = "us-west-2"
);
```

Query a file external table

Syntax:

```
SELECT <clause> FROM <file_external_table>
```

For example, to query data from the file external table `t0` created in Examples - HDFS, run the following command:

```
SELECT * FROM t0;

+--------+------+
| name   | id   |
+--------+------+
| jack   |    2 |
| lily   |    1 |
+--------+------+
2 rows in set (0.08 sec)
```

Manage file external tables

You can view the schema of the table using DESC or drop the table using DROP TABLE.

Data Cache

This topic describes the working principles of Data Cache and how to enable Data Cache to improve query performance on external data.

In data lake analytics, StarRocks works as an OLAP engine to scan data files stored in external storage systems, such as HDFS and Amazon S3. The I/O overhead increases as the number of files to scan increases. In addition, in some ad hoc scenarios, frequent access to the same data doubles I/O overhead.

To optimize the query performance in these scenarios, StarRocks 2.5 provides the Data Cache feature. This feature splits data in an external storage system into multiple blocks based on predefined policies and caches the data on StarRocks backends (BEs). This eliminates the need to pull data from external systems for each access request and accelerates queries and analysis on hot data. Data Cache only works when you query data from external storage systems by using external catalogs or external tables (excluding external tables for JDBC-compatible databases). It does not work when you query StarRocks native tables.

How it works

StarRocks splits data in an external storage system into multiple blocks of the same size (1 MB by default), and caches the data on BEs. Block is the smallest unit of data cache.

For example, if you query a Parquet file of 128 MB from Amazon S3, StarRocks splits the file into 128 blocks (1 MB for each block, recommended setting). The blocks are [0, 1 MB), [1 MB, 2 MB), [2 MB, 3 MB) ... [127 MB, 128 MB). StarRocks assigns a globally unique ID to each block, called a cache key. A cache key consists of the following three parts.

```
hash(filename) + filesize + blockId
```

The following table provides descriptions of each part.

Component item	Description
filename	The name of the data file.
filesize	The size of the data file, 1 MB by default.
blockId	The ID that StarRocks assigns to a block when splitting the data file. The ID is unique under the same data file but is not unique within your StarRocks cluster.

If the query hits the [1 MB, 2 MB) block, StarRocks performs the following operations:

1. Check whether the block exists in the cache.
2. If the block exists, StarRocks reads the block from the cache. If the block does not exist, StarRocks reads the block from Amazon S3 and caches it on a BE.

After Data Cache is enabled, StarRocks caches data blocks read from external storage systems. If you do not want to cache such data blocks, run the following command:

```
SET enable_populate_block_cache = false;
```

For more information about `enable_populate_block_cache`, see System variables.

Storage media of blocks

By default, StarRocks uses the memory of BE machines to cache blocks. It also supports using both the memory and disks as hybrid storage media of blocks. If BE machines are equipped with disks, such as NVMe drives or SSDs, you can use both the memory and disks to cache blocks. If you use cloud storage, such as Amazon EBS, for BE machines, we recommend that you use only the memory to cache blocks.

Cache replacement policies

StarRocks uses the least recently used (LRU) policy to cache and discard data.

- StarRocks reads data from memory, and then from disks if the data is not found in memory. Data read from disks is loaded into memory.
- Data deleted from memory is written into disks. Data removed from disks is discarded.

Enable Data Cache

Data Cache is disabled by default. To enable this feature, configure FEs and BEs in your StarRocks cluster.

Configurations for FEs

You can enable Data Cache for FEs by using one of the following methods:

- Enable Data Cache for a given session based on your requirements.

```
SET enable_scan_block_cache = true;
```

- Enable Data Cache for all active sessions.

```
SET GLOBAL enable_scan_block_cache = true;
```

Configurations for BEs

Add the following parameters to the **conf/be.conf** file of each BE. Then restart each BE to make the settings take effect.

Parameter	Description
block_cache_enable	Whether Data Cache is enabled. • `true`: Data Cache is enabled. • `false`: Data Cache is disabled. The value of this parameter defaults to `false`. To enable Data Cache, set the value of this parameter to `true`.

Parameter	Description
block_cache_disk_path	The paths of disks. We recommend that the number of paths you configured for this parameter is the same as the number of disks of your BE machine. Multiple paths need to be separated with semicolons (;). After you add this parameter, StarRocks automatically creates a file named **cachelib_data** to cache blocks.
block_cache_meta_path	The storage path of block metadata. You can customize the storage path. We recommend that you store the metadata under the **$STARROCKS_HOME** path.
block_cache_mem_size	The maximum amount of data that can be cached in the memory. Unit: bytes. The default value is `2147483648`, which is 2 GB. We recommend that you set the value of this parameter to at least 20 GB. If StarRocks reads a large amount of data from disks after Data Cache is enabled, consider increasing the value.
block_cache_disk_size	The maximum amount of data that can be cached in a single disk. For example, if you configure two disk paths for the `block_cache_disk_path` parameter and set the value of the `block_cache_disk_size` parameter as `21474836480` (20 GB), a maximum of 40 GB data can be cached in these two disks. The default value is `0`, which indicates that only the memory is used to cache data. Unit: bytes.

Examples of setting these parameters.

```
# Enable Data Cache.
block_cache_enable = true

# Configure the disk path. Assume the BE machine is equipped with two
disks.
block_cache_disk_path =
/home/disk1/sr/dla_cache_data/;/home/disk2/sr/dla_cache_data/
```

```
# Configure the metadata storage path.
block_cache_meta_path = /home/disk1/sr/dla_cache_meta/

# Set block_cache_mem_size to 2 GB.
block_cache_mem_size = 2147483648

# Set block_cache_disk_size to 1.2 TB.
block_cache_disk_size = 1288490188800
```

Check whether a query hits data cache

You can check whether a query hits data cache by analyzing the following metrics in the query profile:

- `BlockCacheReadBytes`: the amount of data that StarRocks reads directly from its memory and disks.
- `BlockCacheWriteBytes`: the amount of data loaded from an external storage system to StarRocks' memory and disks.
- `BytesRead`: the total amount of data that is read, including data that StarRocks reads from an external storage system, its memory, and disks.

Example 1: In this example, StarRocks reads a large amount of data (7.65 GB) from the external storage system and only a few data (518.73 MB) from the memory and disks. This means that few data caches were hit.

```
 - Table: lineorder
 - BlockCacheReadBytes: 518.73 MB
   - __MAX_OF_BlockCacheReadBytes: 4.73 MB
   - __MIN_OF_BlockCacheReadBytes: 16.00 KB
 - BlockCacheReadCounter: 684
   - __MAX_OF_BlockCacheReadCounter: 4
   - __MIN_OF_BlockCacheReadCounter: 0
 - BlockCacheReadTimer: 737.357us
 - BlockCacheWriteBytes: 7.65 GB
   - __MAX_OF_BlockCacheWriteBytes: 64.39 MB
   - __MIN_OF_BlockCacheWriteBytes: 0.00
 - BlockCacheWriteCounter: 7.887K (7887)
   - __MAX_OF_BlockCacheWriteCounter: 65
   - __MIN_OF_BlockCacheWriteCounter: 0
 - BlockCacheWriteTimer: 23.467ms
   - __MAX_OF_BlockCacheWriteTimer: 62.280ms
```

```
    - __MIN_OF_BlockCacheWriteTimer: 0ns
  - BufferUnplugCount: 15
    - __MAX_OF_BufferUnplugCount: 2
    - __MIN_OF_BufferUnplugCount: 0
  - BytesRead: 7.65 GB
    - __MAX_OF_BytesRead: 64.39 MB
    - __MIN_OF_BytesRead: 0.00
```

Example 2: In this example, StarRocks reads a large amount of data (46.08 GB) from data cache and no data from the external storage system, which means StarRocks reads data only from data cache.

```
Table: lineitem
- BlockCacheReadBytes: 46.08 GB
 - __MAX_OF_BlockCacheReadBytes: 194.99 MB
 - __MIN_OF_BlockCacheReadBytes: 81.25 MB
- BlockCacheReadCounter: 72.237K (72237)
 - __MAX_OF_BlockCacheReadCounter: 299
 - __MIN_OF_BlockCacheReadCounter: 118
- BlockCacheReadTimer: 856.481ms
 - __MAX_OF_BlockCacheReadTimer: 1s547ms
 - __MIN_OF_BlockCacheReadTimer: 261.824ms
- BlockCacheWriteBytes: 0.00
- BlockCacheWriteCounter: 0
- BlockCacheWriteTimer: 0ns
- BufferUnplugCount: 1.231K (1231)
 - __MAX_OF_BufferUnplugCount: 81
 - __MIN_OF_BufferUnplugCount: 35
- BytesRead: 46.08 GB
 - __MAX_OF_BytesRead: 194.99 MB
 - __MIN_OF_BytesRead: 81.25 MB
```

Query Acceleration

Gather statistics for CBO

This topic describes the basic concept of StarRocks CBO and how to collect statistics for the CBO....

Synchronous materialized view

This topic describes how to create, use, and manage a synchronous materialized view (Rollup).

Asynchronous materialized views

2 items

Colocate Join

For shuffle join and broadcast join, if the join condition is met, the data rows of the two joining tabl...

Use Lateral Join for column-to-row conversion

Column-to-row conversion is a common operation in ETL processing. Lateral is a special Join keyw...

Query Cache

The query cache is a powerful feature of StarRocks that can greatly enhance the performance of a...

Index

2 items

Computing the number of distinct values

2 items

Sorted streaming aggregate

Common aggregation methods in database systems include hash aggregate and sort aggregate.

Gather statistics for CBO

This topic describes the basic concept of StarRocks CBO and how to collect statistics for the CBO. StarRocks 2.4 introduces histograms to gather accurate data distribution statistics.

What is CBO

Cost-based optimizer (CBO) is critical to query optimization. After an SQL query arrives at StarRocks, it is parsed into a logical execution plan. The CBO rewrites and transforms the logical plan into multiple physical execution plans. The CBO then estimates the execution cost of each operator in the plan (such as CPU, memory, network, and I/O) and chooses a query path with the lowest cost as the final physical plan.

StarRocks CBO was launched in StarRocks 1.16.0 and is enabled by default from 1.19 onwards. Developed upon the Cascades framework, StarRocks CBO estimates cost based on various statistical information. It is capable of choosing an execution plan with the lowest cost among tens of thousands of execution plans, significantly improving the efficiency and performance of complex queries.

Statistics are important for the CBO. They determine whether cost estimation is accurate and useful. The following sections detail the types of statistical information, the collection policy, and how to collect statistics and view statistical information.

Types of statistical information

StarRocks collects a variety of statistics as the input for cost estimation.

Basic statistics

By default, StarRocks periodically collects the following basic statistics of tables and columns:

- row_count: total number of rows in a table

- data_size: data size of a column
- ndv: column cardinality, which is the number of distinct values in a column
- null_count: the amount of data with NULL values in a column
- min: minimum value in a column
- max: maximum value in a column

Basic statistics are stored in the `_statistics_.table_statistic_v1` table. You can view this table in the `_statistics_` database of your StarRocks cluster.

Histogram

StarRocks 2.4 introduces histograms to complement basic statistics. Histogram is considered an effective way of data representation. For tables with skewed data, histograms can accurately reflect data distribution.

StarRocks uses equi-height histograms, which are constructed on several buckets. Each bucket contains an equal amount of data. For data values that are frequently queried and have a major impact on selectivity, StarRocks allocates separate buckets for them. More buckets mean more accurate estimation, but may also cause a slight increase in memory usage. You can adjust the number of buckets and most common values (MCVs) for a histogram collection task.

Histograms are applicable to columns with highly skewed data and frequent queries.If your table data is uniformly distributed, you do not need to create histograms. Histograms can be created only on columns of numeric, DATE, DATETIME, or string types.

Currently, StarRocks supports only manual collection of histograms. Histograms are stored in the `histogram_statistics` table of the `_statistics_` database.

Collection types and methods

Data size and data distribution constantly change in a table. Statistics must be updated regularly to represent that data change. Before creating a statistics collection task, you must choose a collection type and method that best suit your business requirements.

StarRocks supports full and sampled collection, both can be performed automatically and manually. By default, StarRocks automatically collects full statistics of a table. It checks for any data updates every 5 minutes. If data change is detected, data collection will be automatically triggered. If you do not want to use automatic full collection, you can set the FE configuration item `enable_collect_full_statistic` to `false` and customize a collection task.

Collection type	Collection method	Description	Advan disad
Full collection	Automatic/manual	Scans the full table to gather statistics.Statistics are collected partition. If a partition has no data change, data will not be collected from this partition, reducing resource consumption.Full statistics are stored in the `_statistics_.column_statistics` table.	Advantage: are accurat helps the C accurate estimation.I It consume resources a From 2.5 or StarRocks a specify an a collection p reduces res consumptic
Sampled collection	Automatic/manual	Evenly extracts `N` rows of data from each partition of a table.Statistics are collected by table. Basic statistics of each column are stored as one record. Cardinality information (ndv) of a column is estimated based on the sampled data, which is not accurate. Sampled statistics are stored in the `_statistics_.table_statistic_v1` table.	Advantage: less system and is fast.I The statisti complete, v affect the a cost estima

Collect statistics

StarRocks offers flexible statistics collection methods. You can choose automatic, manual, or custom collection, whichever suits your business scenarios.

Automatic collection

For basic statistics, StarRocks automatically collects full statistics of a table by default, without requiring manual operations. For tables on which no statistics have been collected, StarRocks automatically collects statistics within the scheduling period. For tables on which statistics have been collected, StarRocks updates the total number of rows and modified rows in the tables, and persists this information regularly for judging whether to trigger automatic collection.

From 2.5 onwards, StarRocks allows you to specify a collection period for automatic full collection, which prevents cluster performance jitter caused by automatic full collection. This period is specified by FE parameters `statistic_auto_analyze_start_time` and `statistic_auto_analyze_end_time`.

Conditions that will trigger automatic collection:

- Table data has changed since previous statistics collection.
- The collection time falls within the range of the configured collection period. (The default collection period is all day.)
- The update time of the previous collecting job is earlier than the latest update time of partitions.
- The health of table statistics is below the specified threshold (`statistic_auto_collect_ratio`).

> Formula for calculating statistics health:
>
> If the number of partitions with data updated is less than 10, the formula is `1 - (Number of updated rows since previous collection/Total number of rows)`. If the number of partitions with data updated is greater than or equal to 10, the formula is `1 - MIN(Number of updated rows since previous collection/Total number of rows, Number of updated partitions since previous collection/Total number of partitions)`.

In addition, StarRocks allows you to configure collection policies based on table size and table update frequency:

- For tables with small data volume, **statistics are collected in real time with no restrictions, even though the table data is frequently updated. The `statistic_auto_collect_small_table_size` parameter can be used to determine whether a table is a small table or a large table. You can also use `statistic_auto_collect_small_table_interval` to configure collection intervals for small tables.

- For tables with large data volume, the following restrictions apply:

    - The default collection interval is not less than 12 hours, which can be configured using `statistic_auto_collect_large_table_interval`.

    - When the collection interval is met and the statistics health is lower than the threshold for automatic sampled collection (`statistic_auto_collect_sample_threshold`), sampled collection is triggered.

    - When the collection interval is met and the statistics health is higher than the threshold for automatic sampled collection (`statistic_auto_collect_sample_threshold`) and lower than the automatic collection threshold (`statistic_auto_collect_ratio`), full collection is triggered.

    - When the size of the largest partition to collect data (`statistic_max_full_collect_data_size`) is greater than 100 GB, sampled collection is triggered.

    - Only statistics of partitions whose update time is later than the time of the previous collection task are collected. Statistics of partitions with no data change are not collected.

> **TIP**
>
> After the data of a table is changed, manually triggering a sampled collection task for this table will make the update time of the sampled collection task later than the data update time, which will not trigger automatic full collection for this table in this scheduling period.

Automatic full collection is enabled by default and run by the system using the default settings.

The following table describes the default settings. If you need to modify them, run the **ADMIN SET CONFIG** command.

FE configuration item	Type	Default value	Descripti
enable_statistic_collect	BOOLEAN	TRUE	Whether to collect statistics. T switch is turned on b default.
enable_collect_full_statistic	BOOLEAN	TRUE	Whether to enable automatic collection. This switch turned on b default.
statistic_collect_interval_sec	LONG	300	The interva for checkir data updat during automatic collection. Unit: secor
statistic_auto_collect_ratio	FLOAT	0.8	The thresh for determinin whether th statistics fo automatic collection a healthy. If

FE configuration item	Type	Default value	Descripti
			statistics health is b this thresh automatic collection i triggered.
statistic_auto_analyze_start_time	STRING	00:00:00	The start ti of automat collection. Value rang `00:00:00` `23:59:59`
statistic_auto_analyze_end_time	STRING	23:59:59	The end tir of automat collection. Value rang `00:00:00` `23:59:59`
statistic_auto_collect_small_table_size	LONG	5368709120	The thresh for determinin whether a table is a s table for automatic collection. table whos size is grea than this v is consider a large tab whereas a table whos

FE configuration item	Type	Default value	Descripti
			size is less than or equ to this valu considered small table Unit: Byte. Default val 53687091 (5 GB).
statistic_auto_collect_small_table_interval	LONG	0	The interva for automatica collecting t statistics o small table Unit: secor
statistic_auto_collect_large_table_interval	LONG	43200	The interva for automatica collecting t statistics o large table Unit: secor Default val 43200 (12 hours).
statistic_auto_collect_sample_threshold	DOUBLE	0.3	The statist health threshold f triggering automatic sampled collection. the health

FE configuration item	Type	Default value	Descripti
			value of statistics is lower than threshold, automatic sampled collection i triggered.
statistic_max_full_collect_data_size	LONG	107374182400	The size of largest partition fo automatic collection t collect dat Unit: Byte. Default val 107374182 (100 GB). I partition exceeds th value, full collection i discarded sampled collection i performed instead.
statistic_full_collect_buffer	LONG	20971520	The maxim buffer size taken by automatic collection tasks. Unit Byte. Defa value:

FE configuration item	Type	Default value	Descripti
			20971520 MB).
statistic_collect_max_row_count_per_query	INT	5000000000	The maxim number of rows to qu for a single analyze tas An analyze task will be split into multiple queries if t value is exceeded.
statistic_collect_too_many_version_sleep	LONG	600000	The sleep t of automat collection tasks if the table on wl the collect task runs h too many c versions. U ms. Defaul value: 600 (10 minute

You can rely on automatic jobs for a majority of statistics collection, but if you have specific requirements, you can manually create a task by executing the ANALYZE TABLE statement or customize an automatic task by executing the CREATE ANALYZE statement.

Manual collection

You can use ANALYZE TABLE to create a manual collection task. By default, manual collection is a synchronous operation. You can also set it to an asynchronous operation. In asynchronous mode, after you run ANALYZE TABLE, the system immediately returns whether this statement is successful. However, the collection task will be running in the background and you do not have to wait for the result. You can check the status of the task by running SHOW ANALYZE STATUS. Asynchronous collection is suitable for tables with large data volume, whereas synchronous collection is suitable for tables with small data volume. **Manual collection tasks are run only once after creation. You do not need to delete manual collection tasks.**

Manually collect basic statistics

```
ANALYZE [FULL|SAMPLE] TABLE tbl_name (col_name [,col_name])
[WITH SYNC | ASYNC MODE]
PROPERTIES (property [,property]);
```

Parameter description:

- Collection type
  - FULL: indicates full collection.
  - SAMPLE: indicates sampled collection.
  - If no collection type is specified, full collection is used by default.
- `col_name`: columns from which to collect statistics. Separate multiple columns with commas (`,`). If this parameter is not specified, the entire table is collected.
- [WITH SYNC | ASYNC MODE]: whether to run the manual collection task in synchronous or asynchronous mode. Synchronous collection is used by default if you do not specify this parameter.
- `PROPERTIES`: custom parameters. If `PROPERTIES` is not specified, the default settings in the `fe.conf` file are used. The properties that are actually used can be viewed via the `Properties` column in the output of SHOW ANALYZE STATUS.

PROPERTIES	Type	Default value	Description
statistic_sample_collect_rows	INT	200000	The minimum number of rows to collect for sampled collection.If the parameter value exceeds the actual number of rows in your table, full collection is performed.

Examples

Manual full collection

```
-- Manually collect full stats of a table using default settings.
ANALYZE TABLE tbl_name;

-- Manually collect full stats of a table using default settings.
ANALYZE FULL TABLE tbl_name;

-- Manually collect stats of specified columns in a table using
default settings.
ANALYZE TABLE tbl_name(c1, c2, c3);
```

Manual sampled collection

```
-- Manually collect partial stats of a table using default settings.
ANALYZE SAMPLE TABLE tbl_name;

-- Manually collect stats of specified columns in a table, with the
number of rows to collect specified.
ANALYZE SAMPLE TABLE tbl_name (v1, v2, v3) PROPERTIES(
    "statistic_sample_collect_rows" = "1000000"
);
```

Manually collect histograms

```
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON col_name [, col_name]
[WITH SYNC | ASYNC MODE]
```

```
[WITH N BUCKETS]
PROPERTIES (property [,property]);
```

Parameter description:

- `col_name`: columns from which to collect statistics. Separate multiple columns with commas (`,`). If this parameter is not specified, the entire table is collected. This parameter is required for histograms.
- [WITH SYNC | ASYNC MODE]: whether to run the manual collection task in synchronous or asynchronous mode. Synchronous collection is used by default if you not specify this parameter.
- `WITH N BUCKETS`: `N` is the number of buckets for histogram collection. If not specified, the default value in `fe.conf` is used.
- PROPERTIES: custom parameters. If `PROPERTIES` is not specified, the default settings in `fe.conf` are used.

PROPERTIES	Type	Default value	Description
statistic_sample_collect_rows	INT	200000	The minimum number of rows to collect. If the parameter value exceeds the actual number of rows in your table, full collection is performed.
histogram_buckets_size	LONG	64	The default bucket number for a histogram.
histogram_mcv_size	INT	100	The number of most common values (MCV) for a histogram.
histogram_sample_ratio	FLOAT	0.1	The sampling ratio for a histogram.

PROPERTIES	Type	Default value	Description
histogram_max_sample_row_count	LONG	10000000	The maximum number of rows to collect for a histogram.

The number of rows to collect for a histogram is controlled by multiple parameters. It is the larger value between `statistic_sample_collect_rows` and table row count * `histogram_sample_ratio`. The number cannot exceed the value specified by `histogram_max_sample_row_count`. If the value is exceeded, `histogram_max_sample_row_count` takes precedence.

The properties that are actually used can be viewed via the `Properties` column in the output of SHOW ANALYZE STATUS.

Examples

```
-- Manually collect histograms on v1 using the default settings.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1;

-- Manually collect histograms on v1 and v2, with 32 buckets, 32
MCVs, and 50% sampling ratio.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1,v2 WITH 32 BUCKETS
PROPERTIES(
   "histogram_mcv_size" = "32",
   "histogram_sample_ratio" = "0.5"
);
```

Custom collection

Customize an automatic collection task

You can use the CREATE ANALYZE statement to customize an automatic collection task.

Before creating a custom automatic collection task, you must disable automatic full collection (`enable_collect_full_statistic = false`). Otherwise, custom tasks cannot take effect.

```
-- Automatically collect stats of all databases.
CREATE ANALYZE [FULL|SAMPLE] ALL PROPERTIES (property [,property]);

-- Automatically collect stats of all tables in a database.
CREATE ANALYZE [FULL|SAMPLE] DATABASE db_name
PROPERTIES (property [,property]);

-- Automatically collect stats of specified columns in a table.
CREATE ANALYZE [FULL|SAMPLE] TABLE tbl_name (col_name [,col_name])
PROPERTIES (property [,property]);
```

Parameter description:

- Collection type
  - FULL: indicates full collection.
  - SAMPLE: indicates sampled collection.
  - If no collection type is specified, full collection is used by default.
- `col_name`: columns from which to collect statistics. Separate multiple columns with commas (`,`). If this parameter is not specified, the entire table is collected.
- `PROPERTIES`: custom parameters. If `PROPERTIES` is not specified, the default settings in `fe.conf` are used.

PROPERTIES	Type	Default value	Description
statistic_auto_collect_ratio	FLOAT	0.8	The threshold for determining whether the statistics for automatic collection are healthy. If the statistics health is below this threshold, automatic collection is triggered.
statistics_max_full_collect_data_size	INT	100	The size of the largest partition for automatic collection to collect data.

PROPERTIES	Type	Default value	Description
			Unit: GB.If a partition exceeds this value, full collection is discarded and sampled collection is performed instead.
statistic_sample_collect_rows	INT	200000	The minimum number of rows to collect.If the parameter value exceeds the actual number of rows in your table, full collection is performed.
statistic_exclude_pattern	String	null	The name of the database or table that needs to be excluded in the job. You can specify the database and table that do not collect statistics in the job. Note that this is a regular expression pattern, and the match content is `database.table`.

Examples

Automatic full collection

```
-- Automatically collect full stats of all databases.
CREATE ANALYZE ALL;

-- Automatically collect full stats of a database.
CREATE ANALYZE DATABASE db_name;

-- Automatically collect full stats of all tables in a database.
CREATE ANALYZE FULL DATABASE db_name;
```

```
-- Automatically collect full stats of specified columns in a table.
CREATE ANALYZE TABLE tbl_name(c1, c2, c3);

-- Automatically collect stats of all databases, excluding specified
database 'db_name'.
CREATE ANALYZE ALL PROPERTIES (
   "statistic_exclude_pattern" = "db_name\."
);
```

Automatic sampled collection

```
-- Automatically collect stats of all tables in a database with
default settings.
CREATE ANALYZE SAMPLE DATABASE db_name;

-- Automatically collect stats of all tables in a database, excluding
specified table 'db_name.tbl_name'.
CREATE ANALYZE SAMPLE DATABASE db_name PROPERTIES (
   "statistic_exclude_pattern" = "db_name.tbl_name"
);

-- Automatically collect stats of specified columns in a table, with
statistics health and the number of rows to collect specified.
CREATE ANALYZE SAMPLE TABLE tbl_name(c1, c2, c3) PROPERTIES (
   "statistic_auto_collect_ratio" = "0.5",
   "statistic_sample_collect_rows" = "1000000"
);
```

View custom collection tasks

```
SHOW ANALYZE JOB [WHERE predicate]
```

You can filter results by using the WHERE clause. The statement returns the following columns.

Column	Description
Id	The ID of the collection task.

Column	Description
Database	The database name.
Table	The table name.
Columns	The column names.
Type	The type of statistics, including `FULL` and `SAMPLE`.
Schedule	The type of scheduling. The type is `SCHEDULE` for an automatic task.
Properties	Custom parameters.
Status	The task status, including PENDING, RUNNING, SUCCESS, and FAILED.
LastWorkTime	The time of the last collection.
Reason	The reason why the task failed. NULL is returned if task execution was successful.

Examples

```
-- View all the custom collection tasks.
SHOW ANALYZE JOB

-- View custom collection tasks of database `test`.
SHOW ANALYZE JOB where `database` = 'test';
```

Delete a custom collection task

```
DROP ANALYZE <ID>;
```

The task ID can be obtained by using the SHOW ANALYZE JOB statement.

Examples

```
DROP ANALYZE 266030;
```

View status of collection tasks

You can view the status of all the current tasks by running the SHOW ANALYZE STATUS statement. This statement cannot be used to view the status of custom collection tasks. To view the status of custom collection tasks, use SHOW ANALYZE JOB.

```
SHOW ANALYZE STATUS [WHERE predicate];
```

You can use `LILE or WHERE` to filter the information to return.

This statement returns the following columns.

List name	Description
Id	The ID of the collection task.
Database	The database name.
Table	The table name.
Columns	The column names.
Type	The type of statistics, including FULL, SAMPLE, and HISTOGRAM.
Schedule	The type of scheduling. `ONCE` means manual and `SCHEDULE` means automatic.
Status	The status of the task.
StartTime	The time when the task starts executing.
EndTime	The time when the task execution ends.

List name	Description
Properties	Custom parameters.
Reason	The reason why the task failed. NULL is returned if the execution was successful.

View statistics

View metadata of basic statistics

```
SHOW STATS META [WHERE];
```

This statement returns the following columns.

Column	Description
Database	The database name.
Table	The table name.
Columns	The column names.
Type	The type of statistics. `FULL` means full collection and `SAMPLE` means sampled collection.
UpdateTime	The latest statistics update time for the current table.
Properties	Custom parameters.
Healthy	The health of statistical information.

View metadata of histograms

```
SHOW HISTOGRAM META [WHERE];
```

This statement returns the following columns.

Column	Description
Database	The database name.
Table	The table name.
Column	The columns.
Type	Type of statistics. The value is `HISTOGRAM` for histograms.
UpdateTime	The latest statistics update time for the current table.
Properties	Custom parameters.

Delete statistics

You can delete statistical information you do not need. When you delete statistics, both the data and metadata of the statistics are deleted, as well as the statistics in expired cache. Note that if an automatic collection task is ongoing, previously deleted statistics may be collected again. You can use `SHOW ANALYZE STATUS` to view the history of collection tasks.

Delete basic statistics

```
DROP STATS tbl_name
```

Delete histograms

```
ANALYZE TABLE tbl_name DROP HISTOGRAM ON col_name [, col_name];
```

Cancel a collection task

You can use the KILL ANALYZE statement to cancel a **running** collection task, including manual and custom tasks.

```
KILL ANALYZE <ID>;
```

The task ID for a manual collection task can be obtained from SHOW ANALYZE STATUS. The task ID for a custom collection task can be obtained from SHOW ANALYZE SHOW ANALYZE JOB.

FE configuration items

FE configuration item	Type	Default value	Descript
enable_statistic_collect	BOOLEAN	TRUE	Whether to collect statistics. T parameter turned on l default.
enable_collect_full_statistic	BOOLEAN	TRUE	Whether to enable automatic statistics collection. parameter turned on l default.
statistic_auto_collect_ratio	FLOAT	0.8	The thresh for determ whether th statistics fo automatic

FE configuration item	Type	Default value	Descript
			collection a healthy. If statistics h is below th threshold, automatic collection i triggered.
statistic_max_full_collect_data_size	LONG	107374182400	The size of largest par for automa collection t collect dat Unit: Byte. partition exceeds th value, full collection i discarded sampled collection i performed instead.
statistic_collect_max_row_count_per_query	INT	5000000000	The maxim number of to query fo single anal task. An analyze tas will be split multiple qu if this value exceeded.

FE configuration item	Type	Default value	Descript
statistic_collect_interval_sec	LONG	300	The interva checking d updates du automatic collection. seconds.
statistic_auto_analyze_start_time	STRING	00:00:00	The start ti of automat collection. Value rang `00:00:00` `23:59:59`
statistic_auto_analyze_end_time	STRING	23:59:59	The end tir automatic collection. Value rang `00:00:00` `23:59:59`
statistic_sample_collect_rows	LONG	200000	The minim number of to collect f sampled collection. parameter value exce the actual number of in your tab full collecti performed
statistic_collect_concurrency	INT	3	The maxim number of

FE configuration item	Type	Default value	Descript
			manual collection t that can ru parallel. Th value defau to 3, which means you run a maxi of three ma collections tasks in pa If the value exceeded, incoming t will be in th PENDING s waiting to l scheduled.
histogram_buckets_size	LONG	64	The defaul bucket nur for a histog
histogram_mcv_size	LONG	100	The numbe most comr values (MC for a histog
histogram_sample_ratio	FLOAT	0.1	The sampli ratio for a histogram.
histogram_max_sample_row_count	LONG	10000000	The maxim number of to collect f histogram.

FE configuration item	Type	Default value	Descript
statistic_manager_sleep_time_sec	LONG	60	The interva which meta is schedule Unit: secor The systen performs t following operations based on t interval:Cre tables for storing statistics.D statistics tl have been deleted.De expired statistics.
statistic_analyze_status_keep_second	LONG	259200	The duratic retain the history of collection tasks. The default valu 3 days. Un seconds.

References

- To query FE configuration items, run ADMIN SHOW CONFIG.
- To modify FE configuration items, run ADMIN SET CONFIG.

Synchronous materialized view

This topic describes how to create, use, and manage a **synchronous materialized view (Rollup)**.

For a synchronous materialized view, all changes in the base table are simultaneously updated to the corresponding synchronous materialized views. The refresh of a synchronous materialized view is triggered automatically. Synchronous materialized views are significantly inexpensive to maintain and update, making them suitable for transparent acceleration of real-time, single-table aggregate queries.

Synchronous materialized views in StarRocks can be created only on a single base table from the default catalog. They are essentially a special index for query acceleration. You cannot query synchronous materialized views directly.

From v2.4 onwards, StarRocks provides asynchronous materialized views, which supports creation on multiple tables and more aggregation operators. For the usage of **asynchronous materialized views**, see Asynchronous materialized view.

The following table compares the asynchronous materialized views (ASYNC MVs) in StarRocks v2.5, v2.4, and the synchronous materialized view (SYNC MV) in the perspective of features that they support:

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
ASYNC MV	Yes	Yes	Yes	• Asynchronous refresh • Manual refresh	Multiple tables from: • Default catalog • External catalogs

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
					(v2.5) • Existing materialized views (v2.5) • Existing views (v3.1)
SYNC MV (Rollup)	Limited choices of aggregate functions	No	Yes	Synchronous refresh during data loading	Single table in the default catalog

Basic concepts

- **Base table**

  Base tables are the driving tables of a materialized view.

  For StarRocks' synchronous materialized views, base tables must be a single native table from the default catalog. StarRocks supports creating synchronous materialized views on Duplicate Key tables and Aggregate tables.

- **Refresh**

  A synchronous materialized view updates itself every time the data in the base table changes. You do not need to trigger the refresh manually.

- **Query rewrite**

  Query rewrite means that when executing a query on base tables with materialized views built on, the system automatically judges whether the pre-computed results in the materialized view can be reused for the query. If they can be reused, the system will load the data directly from the relevant materialized view to avoid the time- and resource-consuming computations or joins.

Synchronous materialized views support query rewrite based on some of the aggregate operators. For more information, see Correspondence of aggregate operators.

Preparation

Before creating a synchronous materialized view, check if your data warehouse is eligible for query acceleration through synchronous materialized views. For example, check if the queries reuse certain sub-query statements.

The following example is based on the table `sales_records`, which contains the transaction ID `record_id`, salesperson ID `seller_id`, store ID `store_id`, date `sale_date`, and sales amount `sale_amt` for each transaction. Follow these steps to create the table and insert data into it:

```
CREATE TABLE sales_records(
    record_id INT,
    seller_id INT,
    store_id INT,
    sale_date DATE,
    sale_amt BIGINT
) DISTRIBUTED BY HASH(record_id);

INSERT INTO sales_records
VALUES
    (001,01,1,"2022-03-13",8573),
    (002,02,2,"2022-03-14",6948),
    (003,01,1,"2022-03-14",4319),
    (004,03,3,"2022-03-15",8734),
    (005,03,3,"2022-03-16",4212),
    (006,02,2,"2022-03-17",9515);
```

The business scenario of this example demands frequent analyses on the sales amounts of different stores. As a result, the `sum()` function is used on each query, consuming a massive amount of compute resources. You can run the query to record its time, and view its query profile by using EXPLAIN command.

```
MySQL > SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
+----------+-----------------+
```

```
| store_id | sum(`sale_amt`) |
+----------+-----------------+
|        2 |           16463 |
|        3 |           12946 |
|        1 |           12892 |
+----------+-----------------+
3 rows in set (0.02 sec)

MySQL > EXPLAIN SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
+-----------------------------------------------------------------------
--------+
| Explain String
|
+-----------------------------------------------------------------------
--------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:3: store_id | 6: sum
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   4:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: HASH_PARTITIONED: 3: store_id
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 04
|
|     UNPARTITIONED
```

```
|
|
|
|   3:AGGREGATE (merge finalize)
|
|   |  output: sum(6: sum)
|
|   |  group by: 3: store_id
|
|   |
|
|   2:EXCHANGE
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 02
|
|     HASH_PARTITIONED: 3: store_id
|
|
|
|   1:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  output: sum(5: sale_amt)
|
|   |  group by: 3: store_id
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: sales_records
|
|      PREAGGREGATION: ON
```

```
|
|      partitions=1/1
|
|      rollup: sales_records
|
|      tabletRatio=10/10
|
|
tabletList=12049,12053,12057,12061,12065,12069,12073,12077,12081,12085
|
|      cardinality=1
|
|      avgRowSize=2.0
|
|      numNodes=0
|
+-----------------------------------------------------------------------
--------+
45 rows in set (0.00 sec)
```

It can be observed that the query takes about 0.02 seconds, and no synchronous materialized view is used to accelerate the query because the value of `rollup` field in the query profile is `sales_records`, which is the base table.

Create a synchronous materialized view

You can create a synchronous materialized view based on a specific query statement using CREATE MATERIALIZED VIEW.

Based on the table `sales_records` and the query statement mentioned above, the following example creates the synchronous materialized view `store_amt` to analyze the sum of sales amount in each store.

```
CREATE MATERIALIZED VIEW store_amt AS
SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
```

> **CAUTION**

- When using aggregate functions in synchronous materialized views, you must use the GROUP BY clause and specify at least one GROUP BY column in your SELECT list.
- Synchronous materialized views do not support using one aggregate function on multiple columns. Query statements in the form of `sum(a+b)` are not supported.
- Synchronous materialized views do not support using multiple aggregate functions on one column. Query statements in the form of `select sum(a), min(a) from table` are not supported.
- JOIN and WHERE clauses are not supported when creating a synchronous materialized view.
- When using ALTER TABLE DROP COLUMN to drop a specific column in a base table, you need to ensure that all synchronous materialized views of the base table do not contain the dropped column, otherwise the drop operation cannot be performed. To drop a column that used in synchronous materialized views, you need to first drop all synchronous materialized views that contain the column, and then drop the column.
- Creating too many synchronous materialized views for a table will affect the data loading efficiency. When data is being loaded to the base table, the data in synchronous materialized views and base table are updated synchronously. If the base table contains `n` synchronous materialized views, the efficiency of loading data into the base table is about the same as that of loading data into `n` tables.
- Currently, StarRocks does not support creating multiple synchronous materialized views at the same time. A new synchronous materialized view can only be created when the previous one is completed.

Check the building status of a synchronous materialized view

Creating a synchronous materialized view is an asynchronous operation. Executing CREATE MATERIALIZED VIEW successfully indicates that the task of creating the materialized view is submitted successfully. You can view the building status of the synchronous materialized view in a database via SHOW ALTER MATERIALIZED VIEW.

```
MySQL > SHOW ALTER MATERIALIZED VIEW\G
*************************** 1. row ***************************
          JobId: 12090
```

```
      TableName: sales_records
     CreateTime: 2022-08-25 19:41:10
   FinishedTime: 2022-08-25 19:41:39
  BaseIndexName: sales_records
RollupIndexName: store_amt
       RollupId: 12091
  TransactionId: 10
          State: FINISHED
            Msg:
       Progress: NULL
        Timeout: 86400
1 row in set (0.00 sec)
```

The `RollupIndexName` section indicates the name of the synchronous materialized view, and `State` section indicates if the building is completed.

Query with the synchronous materialized view

The synchronous materialized view you created contains the complete set of pre-computed results in accordance with the query statement. Subsequent queries use the data within it. You can run the same query to test the query time as you did in the preparation.

```
MySQL > SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
+----------+-----------------+
| store_id | sum(`sale_amt`) |
+----------+-----------------+
|        2 |           16463 |
|        3 |           12946 |
|        1 |           12892 |
+----------+-----------------+
3 rows in set (0.01 sec)
```

It can be observed that the query time is reduced to 0.01 seconds.

Check if a query hits the synchronous materialized view

Execute EXPLAIN command again to check if the query hits the synchronous materialized view.

```
MySQL > EXPLAIN SELECT store_id, SUM(sale_amt) FROM sales_records
GROUP BY store_id;
+-----------------------------------------------------------------------------
--------+
| Explain String
|
+-----------------------------------------------------------------------------
--------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:3: store_id | 6: sum
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   4:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: HASH_PARTITIONED: 3: store_id
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 04
|
|     UNPARTITIONED
|
```

```
|
|
|   3:AGGREGATE (merge finalize)
|
|   |  output: sum(6: sum)
|
|   |  group by: 3: store_id
|
|   |
|
|   2:EXCHANGE
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 02
|
|     HASH_PARTITIONED: 3: store_id
|
|
|
|   1:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  output: sum(5: sale_amt)
|
|   |  group by: 3: store_id
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: sales_records
|
|      PREAGGREGATION: ON
|
```

```
|      partitions=1/1
|
|      rollup: store_amt
|
|      tabletRatio=10/10
|
|
tabletList=12092,12096,12100,12104,12108,12112,12116,12120,12124,12128
|
|      cardinality=6
|
|      avgRowSize=2.0
|
|      numNodes=0
|
+-----------------------------------------------------------------------
--------+
45 rows in set (0.00 sec)
```

It can be observed that the value of `rollup` section in the query profile is now `store_amt`, which is the synchronous materialized view you have built. That means this query has hit the synchronous materialized view.

Show synchronous materialized views

You can execute DESC <tbl_name> ALL to check the schema of a table and its subordinate synchronous materialized views.

```
MySQL > DESC 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   |
+--------------+--------------+----------+-------+-----+-------
+---------+-------+
8 rows in set (0.00 sec)
```

Drop a synchronous materialized view

Under the following circumstances, you need to drop the synchronous materialized view:

- You have created a wrong materialized view and you need to drop it before the building completed.
- You have created too many materialized views, which results in a huge drop in load performance, and some of the materialized views are duplicate.
- The frequency of the involved queries is low, and you can tolerate a relatively high query latency.

Drop an unfinished synchronous materialized view

You can drop a synchronous materialized view that is being created by canceling the in-progress creation task. First, you need to get the job ID `JobID` of the materialized view creation task by checking the building status of the materialized view. After getting the job ID, you need to cancel the creation task with the CANCEL ALTER command.

```
CANCEL ALTER TABLE ROLLUP FROM sales_records (12090);
```

Drop an existing synchronous materialized view

You can drop an existing synchronous materialized view with the DROP MATERIALIZED VIEW command.

```
DROP MATERIALIZED VIEW store_amt;
```

Best practices

Exact count distinct

The following example is based on an advertisement business analysis table `advertiser_view_record`, which records the date that the ad is viewed `click_time`, the name of the ad `advertiser`, the channel of the ad `channel`, and the ID of the user who viewed the ID `user_id`.

```
CREATE TABLE advertiser_view_record(
    click_time DATE,
    advertiser VARCHAR(10),
    channel VARCHAR(10),
    user_id INT
) distributed BY hash(click_time);
```

Analysis is mainly focused on the UV of the ads.

```
SELECT advertiser, channel, count(distinct user_id)
FROM advertiser_view_record
GROUP BY advertiser, channel;
```

To accelerate exact count distinct, you can create a synchronous materialized view based on this table and use the bitmap_union function to pre-aggregate the data.

```
CREATE MATERIALIZED VIEW advertiser_uv AS
SELECT advertiser, channel, bitmap_union(to_bitmap(user_id))
FROM advertiser_view_record
GROUP BY advertiser, channel;
```

After the synchronous materialized view is created, the sub-query `count(distinct user_id)` in the subsequent queries will be automatically rewritten as `bitmap_union_count (to_bitmap(user_id))` so that they can hit the synchronous materialized view.

Approximate count distinct

Use the table `advertiser_view_record` above as an example again. To accelerate approximate count distinct, you can create a synchronous materialized view based on this table and use the hll_union() function to pre-aggregate the data.

```
CREATE MATERIALIZED VIEW advertiser_uv2 AS
SELECT advertiser, channel, hll_union(hll_hash(user_id))
FROM advertiser_view_record
GROUP BY advertiser, channel;
```

Set extra sort keys

Suppose that the base table `tableA` contains columns `k1`, `k2` and `k3`, where only `k1` and `k2` are sort keys. If the query including the sub-query `where k3=x` must be accelerated, you can create a synchronous materialized view with `k3` as the first column.

```
CREATE MATERIALIZED VIEW k3_as_key AS
SELECT k3, k2, k1
FROM tableA
```

Correspondence of aggregate functions

When a query is executed with a synchronous materialized view, the original query statement will be automatically rewritten and used to query the intermediate results stored in the synchronous materialized view. The following table shows the correspondence between the aggregate function in the original query and the aggregate function used to construct the synchronous materialized view. You can select the corresponding aggregate function to build a synchronous materialized view according to your business scenario.

aggregate function in the original query	aggregate function of the materialized view
sum	sum
min	min

aggregate function in the original query	aggregate function of the materialized view
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

Asynchronous materialized views

This topic describes how to understand, create, use, and manage an asynchronous materialized view. Asynchronous materialized views are supported from StarRocks v2.4 onwards.

Compared with synchronous materialized views, asynchronous materialized views support multi-table join and more aggregate functions. The refresh of asynchronous materialized views can be triggered manually or by scheduled tasks. You can also refresh some of the partitions instead of the whole materialized view, greatly reducing the cost of refresh. In addition, asynchronous materialized views support a variety of query rewrite scenarios, allowing automatic, transparent query acceleration.

For the scenario and usage of the synchronous materialized views (Rollup), see Synchronous materialized view (Rollup).

Overview

Applications in databases often perform complex queries on large tables. Such queries involve multi-table joins and aggregations on tables that contain billions of rows. Processing these queries can be expensive, in terms of system resources and the time it takes to compute the results.

Asynchronous materialized views in StarRocks are designed to tackle these issues. An asynchronous materialized view is a special physical table that holds pre-computed query results from one or more base tables. When you perform complex queries on the base table, StarRocks returns the pre-computed results from the relevant materialized views to process these queries. This way, query performance can be improved because repetitive complex calculations are avoided. This performance difference can be significant when a query is run frequently or is sufficiently complex.

Additionally, asynchronous materialized views are especially useful for building mathematical models upon your data warehouse. By doing so, you can provide a unified

data specification to upper-layer applications, shield the underlying implementation, or protect the raw data security of the base tables.

Understand materialized views in StarRocks

StarRocks v2.3 and earlier versions provided a synchronous materialized view that can be built only on a single table. Synchronous materialized views, or the Rollup, retain higher data freshness and lower refreshing costs. However, compared to asynchronous materialized views supported from v2.4 onwards, synchronous materialized views have many limitations. You have limited choices of aggregation operators when you want to build a synchronous materialized view to accelerate or rewrite your queries.

The following table compares the asynchronous materialized views (ASYNC MV) and the synchronous materialized view (SYNC MV) in StarRocks in the perspective of features that they support:

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
ASYNC MV	Yes	Yes	Yes	• Asynchronous refresh • Manual refresh	Multiple tables from: • Default catalog • External catalogs (v2.5) • Existing materialized views (v2.5)
SYNC MV (Rollup)	Limited choices of aggregate functions	No	Yes	Synchronous refresh during data loading	Single table in the default catalog

Basic concepts

- **Base table**

  Base tables are the driving tables of a materialized view.

  For StarRocks' asynchronous materialized views, base tables can be StarRocks native tables in the default catalog, tables in external catalogs (supported from v2.5), or even existing asynchronous materialized views (supported from v2.5). StarRocks supports creating asynchronous materialized views on all types of StarRocks tables.

- **Refresh**

  When you create an asynchronous materialized view, its data reflects only the state of the base tables at that time. When the data in the base tables change, you need to refresh the materialized view to keep the changes synchronized.

  Currently, StarRocks supports two generic refreshing strategies:

  - ASYNC: Asynchronous refresh mode. Each time the base table data changes, the materialized view is automatically refreshed according to the pre-defined refresh interval.
  - MANUAL: Manual refresh mode. The materialized view will not be automatically refreshed. The refresh tasks can only be triggered manually by users.

- **Query rewrite**

  Query rewrite means that when executing a query on base tables with materialized views built on, the system automatically judges whether the pre-computed results in the materialized view can be reused for the query. If they can be reused, the system will load the data directly from the relevant materialized view to avoid the time- and resource-consuming computations or joins.

  From v2.5, StarRocks supports automatic, transparent query rewrite based on the SPJG-type asynchronous materialized views. The SPJG-type materialized views refer to materialized views whose plan only includes Scan, Filter, Project, and Aggregate types of operators.

Decide when to create a materialized view

You can create an asynchronous materialized view if you have the following demands in your data warehouse environment:

- **Accelerating queries with repetitive aggregate functions**

  Suppose that most queries in your data warehouse include the same sub-query with an aggregate function, and these queries have consumed a huge proportion of your computing resources. Based on this sub-query, you can create an asynchronous materialized view, which will compute and store all results of the sub-query. After the materialized view is built, StarRocks rewrites all queries that contain the sub-query, loads the intermediate results stored in the materialized view, and thus accelerates these queries.

- **Regular JOIN of multiple tables**

  Suppose that you need to regularly join multiple tables in your data warehouse to make a new wide table. You can build an asynchronous materialized view for these tables, and set the ASYNC refreshing strategy that triggers refreshing tasks at a fixed time interval. After the materialized view is built, query results are returned directly from the materialized view, and thus the latency caused by JOIN operations is avoided.

- **Data warehouse layering**

  Suppose that your data warehouse contains a mass of raw data, and queries in it require a complex set of ETL operations. You can build multiple layers of asynchronous materialized views to stratify the data in your data warehouse, and thus decompose the query into a series of simple sub-queries. It can significantly reduce repetitive computation, and, more importantly, help your DBA identify the problem with ease and efficiency. Beyond that, data warehouse layering helps decouple raw data and statistical data, protecting the security of sensitive raw data.

- **Accelerating queries in data lakes**

  Querying a data lake can be slow due to network latency and object storage throughput. You can enhance the query performance by building an asynchronous materialized view on top of the data lake. Moreover, StarRocks can intelligently rewrite queries to use the existing materialized views, saving you the trouble of modifying your queries manually.

For specific use cases of asynchronous materialized views, refer to the following content:

- Data Modeling
- Query Rewrite

- Data Lake Query Acceleration

Create an asynchronous materialized view

StarRocks' asynchronous materialized views can be created on the following base tables:

- StarRocks' native tables (all StarRocks table types are supported)
- Tables in external catalogs, including
  - Hive catalog (Since v2.5)
  - Hudi catalog (Since v2.5)
  - Iceberg catalog (Since v2.5)
- Existing asynchronous materialized views (Since v2.5)

Before you begin

The following examples involve two base tables in the default catalog:

- The table `goods` records the item ID `item_id1`, the item name `item_name`, and the item price `price`.
- The table `order_list` records the order ID `order_id`, client ID `client_id`, item ID `item_id2`, and order date `order_date`.

The column `goods.item_id1` is equivalent to the column `order_list.item_id2`.

Execute the following statements to create the tables and insert data into them:

```
CREATE TABLE goods(
    item_id1          INT,
    item_name         STRING,
    price             FLOAT
) DISTRIBUTED BY HASH(item_id1);

INSERT INTO goods
VALUES
    (1001,"apple",6.5),
    (1002,"pear",8.0),
    (1003,"potato",2.2);
```

```
CREATE TABLE order_list(
    order_id          INT,
    client_id         INT,
    item_id2          INT,
    order_date        DATE
) DISTRIBUTED BY HASH(order_id);

INSERT INTO order_list
VALUES
    (10001,101,1001,"2022-03-13"),
    (10001,101,1002,"2022-03-13"),
    (10002,103,1002,"2022-03-13"),
    (10002,103,1003,"2022-03-14"),
    (10003,102,1003,"2022-03-14"),
    (10003,102,1001,"2022-03-14");
```

The scenario in the following example demands frequent calculations of the total of each order. It requires frequent joins of the two base tables and intensive usage of the aggregate function `sum()`. Besides, the business scenario demands the data refresh at an interval of one day.

The query statement is as follows:

```
SELECT
    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;
```

Create the materialized view

You can create a materialized view based on a specific query statement using CREATE MATERIALIZED VIEW.

Based on the table `goods`, `order_list`, and the query statement mentioned above, the following example creates the materialized view `order_mv` to analyze the total of each order. The materialized view is set to refresh itself at an interval of one day.

```
CREATE MATERIALIZED VIEW order_mv
DISTRIBUTED BY HASH(`order_id`)
REFRESH ASYNC START('2022-09-01 10:00:00') EVERY (interval 1 day)
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;
```

> **NOTE**
>
> - While creating an asynchronous materialized view, you must specify either the data distribution strategy or the refresh strategy of the materialized view, or both.
> - You can set different partitioning and bucketing strategies for an asynchronous materialized view from those of its base tables, but you must include the partition keys and bucket keys of the materialized views in the query statement used to create the materialized view.
> - Asynchronous materialized views support a dynamic partitioning strategy in a longer span. For example, if the base table is partitioned at an interval of one day, you can set the materialized view to be partitioned at an interval of one month.
> - Currently, StarRocks does not support creating asynchronous materialized views with the list partitioning strategy or based on tables that are created with the list partitioning strategy.
> - The query statement used to create a materialized view does not support random functions, including rand(), random(), uuid(), and sleep().
> - Asynchronous materialized views support a variety of data types. For more information, see CREATE MATERIALIZED VIEW - Supported data types.
> - By default, executing a CREATE MATERIALIZED VIEW statement immediately triggers the refresh task, which can consume a certain proportion of the system resources. If you want to defer the refresh task, you can add the REFRESH DEFERRED parameter to your CREATE MATERIALIZED VIEW statement.

- **About refresh mechanisms of asynchronous materialized views**

  Currently, StarRocks supports two ON DEMAND refresh strategies: MANUAL refresh and ASYNC refresh.

In StarRocks v2.5, asynchronous materialized views further support a variety of asynchronous refreshing mechanisms to control the cost of refresh and increase the success rate:

- If an MV has many large partitions, each refresh can consume a large amount of resources. In v2.5, StarRocks supports splitting refresh tasks. You can specify the maximum number of partitions to be refreshed, and StarRocks performs refresh in batches, with a batch size smaller or equal to the specified maximum number of partitions. This feature ensures large asynchronous materialized views are stably refreshed, enhancing the stability and robustness of data modeling.
- You can specify the time to live (TTL) for partitions of an asynchronous materialized view, reducing the storage size taken by the materialized view.
- You can specify the refresh range to refresh only the latest few partitions, reducing the refresh overhead.
- You can specify the base tables where data changes will not automatically trigger a refresh of the corresponding materialized view.
- You can assign a resource group to the refresh task.

For more information, see the **PROPERTIES** section in CREATE MATERIALIZED VIEW - Parameters. You can also modify the mechanisms of an existing asynchronous materialized view using ALTER MATERIALIZED VIEW.

> **CAUTION**
>
> To prevent full refresh operations from exhausting system resources and causing task failures, it is recommended to create partitioned materialized views based on partitioned base tables. This ensures that when data updates occur within a base table partition, only the corresponding partition of the materialized view are refreshed, rather than refreshing the entire materialized view. For more information, please refer to Data Modeling with Materialized Views - Partitioned Modeling.

- **About nested materialized views**

StarRocks v2.5 supports creating nested asynchronous materialized views. You can build asynchronous materialized views based on existing asynchronous materialized views. The refreshing strategy for each materialized view does not affect the materialized views on the upper or lower layers. Currently, StarRocks does not limit the number of nesting levels. In a production environment, we recommend that the number of nesting layers does not exceed THREE.

- **About external catalog materialized views**

  StarRocks supports building asynchronous materialized views based on Hive Catalog (since v2.5), Hudi Catalog (since v2.5), and Iceberg Catalog (since v2.5). Creating a materialized view on external catalogs is similar to creating an asynchronous materialized view on the default catalog, but with some usage restrictions. For more information, please refer to Data lake query acceleration with materialized views.

Manually refresh an asynchronous materialized view

You can refresh an asynchronous materialized view regardless of its refreshing strategy via REFRESH MATERIALIZED VIEW. StarRocks v2.5 supports refreshing specific partitions of an asynchronous materialized view by specifying partition names.

```
REFRESH MATERIALIZED VIEW order_mv;
```

You can cancel a refresh task submitted via an asynchronous call using CANCEL REFRESH MATERIALIZED VIEW.

Query the asynchronous materialized view directly

The asynchronous materialized view you created is essentially a physical table that contains the complete set of pre-computed results in accordance with the query statement. Therefore, you can directly query the materialized view after the materialized view is refreshed for the first time.

```
MySQL > SELECT * FROM order_mv;
+----------+-------------------+
| order_id | total             |
+----------+-------------------+
|    10001 |              14.5 |
|    10002 | 10.200000047683716 |
|    10003 |  8.700000047683716 |
```

```
+----------+--------------------+
3 rows in set (0.01 sec)
```

> **NOTE**
>
> You can directly query an asynchronous materialized view, but the results may be inconsistent with what you get from the query on its base tables.

Rewrite and accelerate queries with the asynchronous materialized view

StarRocks v2.5 supports automatic and transparent query rewrite based on the SPJG-type asynchronous materialized views. The SPJG-type materialized views query rewrite includes single table query rewrite, Join query rewrite, aggregation query rewrite, Union query rewrite and query rewrite based on nested materialized views. For more information, please refer to Query Rewrite with Materialized Views.

Currently, StarRocks supports rewriting queries on asynchronous materialized views that are created on the default catalog or an external catalog such as a Hive catalog, Hudi catalog, or Iceberg catalog. When querying data in the default catalog, StarRocks ensures strong consistency of results between the rewritten query and the original query by excluding materialized views whose data is inconsistent with the base table. When the data in a materialized view expires, the materialized view will not be used as a candidate materialized view. When querying data in external catalogs, StarRocks does not ensure a strong consistency of the results because StarRocks cannot perceive the data changes in external catalogs. For more about asynchronous materialized views that are created based on an external catalog, please refer to Data lake query acceleration with materialized views.

Manage an asynchronous materialized view

Alter an asynchronous materialized view

You can alter the property of an asynchronous materialized view using ALTER MATERIALIZED VIEW.

- Enable an inactive materialized view.

```
ALTER MATERIALIZED VIEW order_mv ACTIVE;
```

- Rename an asynchronous materialized view.

```
ALTER MATERIALIZED VIEW order_mv RENAME order_total;
```

- Alter the refreshing interval of an asynchronous materialized view to 2 days.

```
ALTER MATERIALIZED VIEW order_mv REFRESH ASYNC EVERY(INTERVAL 2
DAY);
```

Show asynchronous materialized views

You can view the asynchronous materialized views in your database by using SHOW MATERIALIZED VIEW or querying the system metadata table in Information Schema.

- Check all asynchronous materialized views in your database.

```
SHOW MATERIALIZED VIEW;
```

- Check a specific asynchronous materialized view.

```
SHOW MATERIALIZED VIEW WHERE NAME = "order_mv";
```

- Check specific asynchronous materialized views by matching the name.

```
SHOW MATERIALIZED VIEW WHERE NAME LIKE "order%";
```

- Check all asynchronous materialized views by querying the metadata table `materialized_views` in Information Schema. For more information, please refer to information_schema.materialized_views.

```

```

```
SELECT * FROM information_schema.materialized_views;
```

Check the definition of asynchronous materialized view

You can check the query used to create an asynchronous materialized view via SHOW CREATE MATERIALIZED VIEW.

```
SHOW CREATE MATERIALIZED VIEW order_mv;
```

Check the execution status of asynchronous materialized view

You can check the execution (building or refreshing) status of an asynchronous materialized view by querying the `tasks` and `task_runs` metadata tables in StarRocks' Information Schema.

The following example checks the execution status of the materialized view that was created most recently:

1. Check the `TASK_NAME` of the most recent task in the table `tasks`.

```
mysql> select * from information_schema.tasks  order by CREATE_TIME
desc limit 1\G;
*************************** 1. row ***************************
  TASK_NAME: mv-59299
CREATE_TIME: 2022-12-12 17:33:51
  SCHEDULE: MANUAL
  DATABASE: ssb_1
DEFINITION: insert overwrite hive_mv_lineorder_flat_1 SELECT
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_linenumber`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_custkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_partkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderpriorit
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_ordtotalpric
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_revenue`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`p_mfgr`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`s_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_city`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_nation`,
```

```
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate`
FROM `hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`
WHERE
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate` =
'1997-01-01'
EXPIRE_TIME: NULL
1 row in set (0.02 sec)
```

2. Check the execution status in the table `task_runs` using the `TASK_NAME` you have found.

```
mysql> select * from information_schema.task_runs where task_name='m
59299' order by CREATE_TIME \G;
*************************** 1. row ***************************
    QUERY_ID: d9cef11f-7a00-11ed-bd90-00163e14767f
    TASK_NAME: mv-59299
  CREATE_TIME: 2022-12-12 17:39:19
  FINISH_TIME: 2022-12-12 17:39:22
        STATE: SUCCESS
    DATABASE: ssb_1
  DEFINITION: insert overwrite hive_mv_lineorder_flat_1 SELECT
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_linenumber`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_custkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_partkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderpriorit
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_ordtotalpric
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_revenue`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`p_mfgr`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`s_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_city`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate`
FROM `hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`
WHERE
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate` =
'1997-01-01'
  EXPIRE_TIME: 2022-12-15 17:39:19
  ERROR_CODE: 0
ERROR_MESSAGE: NULL
    PROGRESS: 100%
2 rows in set (0.02 sec)
```

Drop an asynchronous materialized view

You can drop an asynchronous materialized view via DROP MATERIALIZED VIEW.

```
DROP MATERIALIZED VIEW order_mv;
```

Relevant session variables

The following variables control the behaviour of an asynchronous materialized view:

- `analyze_mv`: Whether and how to analyze the materialized view after refresh. Valid values are an empty string (Do not analyze), `sample` (Sampled statistics collection), and `full` (Full statistics collection). Default is `sample`.
- `enable_materialized_view_rewrite`: Whether to enable the automatic rewrite for materialized view. Valid values are `true` (Default since v2.5) and `false`.

Data modeling with materialized views

This topic describes how to perform data modeling with the help of StarRocks' asynchronous mate...

Query rewrite with materialized views

This topic describes how to leverage StarRocks' asynchronous materialized views to rewrite and ac...

Data lake query acceleration with materialized views

This topic describes how to optimize query performance in your data lake using StarRocks' asynch...

Data modeling with materialized views

This topic describes how to perform data modeling with the help of StarRocks' asynchronous materialized views. By doing so, you can greatly simplify the ETL pipeline of your data warehouses, and significantly improve data quality and query performance.

Overview

Data modeling is the process of cleaning, layering, aggregating, and associating data with rational methodologies. It can create a comprehensible representation of the raw data that is too rough, too complicated, or too costly to analyze directly, and provide actionable insights into the data.

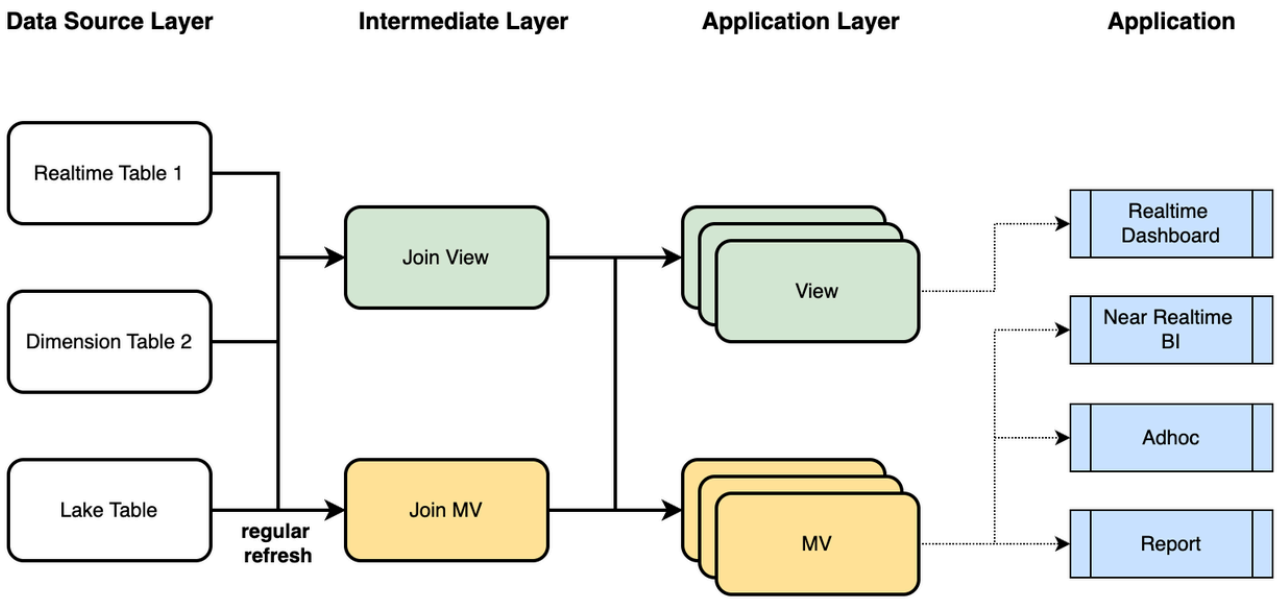
However, the common challenge in real-world data modeling is that the modeling process struggles to keep up with the pace of business development, and it is difficult to measure the return on investment for data modeling efforts. Despite the fact that the modeling methodologies are simple, business experts are required to have a solid background in data organization and governance, which is a complex process. In the early stages of business, decision-makers rarely spare sufficient resources for data modeling, and it is challenging to see the value that data modeling can bring. Moreover, business models can change rapidly, and the modeling methodologies themselves need iteration and evolution. Therefore, many data analysts tend to avoid modeling and use raw data directly, thus inevitably leading to issues of data quality and query performance. When the need for modeling arises, it becomes difficult to restructure data analytics patterns that have already been established to match those of the data models.

Using materialized views for data modeling can effectively address these problems. StarRocks' asynchronous materialized views can:

- **Simplify the data warehouse architecture**: Because StarRocks can provide a one-stop data governance experience, you do not need to maintain other data processing

systems, saving the human and system resources spent on them.

- **Ease the data modeling experience**: Any data analyst with only basic SQL knowledge is capable of data modeling with StarRocks. Data modeling is no longer the exclusive province of experienced data engineers.
- **Reduce maintenance complexity**: StarRocks' asynchronous materialized views can automatically manage the lineage relationships and dependencies across data layers, eliminating the need for an entire data platform to handle this task.



In real-world situations, you can perform data modeling by combining the usage of StarRocks' views (logical views) and asynchronous materialized views as follows:

1. Use views to associate real-time data with dimension data, and use materialized views to associate historical data from the data lake with dimension data. Perform necessary data cleaning and semantic mapping to obtain the detail data for the Intermediate Layer, which reflects the semantics required in your business scenarios.
2. In the Application Layer, perform data Join, Aggregation, Union, and Window calculations tailored to different business scenarios. This will yield views for real-time pipelines and materialized views for near-real-time pipelines.
3. On the Application side, choose an appropriate Analytical Data Store (ADS) for query analysis based on your timeliness and performance requirements. These ADS can serve real-time dashboards, near-real-time BI, ad-hoc queries, and scheduled reports.

During this process, you will leverage several built-in capabilities of StarRocks, which will be elaborated in the following section.

Capabilities of asynchronous materialized view

StarRocks' asynchronous materialized view possesses the following atomic functions that can aid in data modeling:

- **Automatic refresh**: After data is loaded into base tables, materialized views can be automatically refreshed. You do not need to maintain a scheduling task externally.
- **Partitioned refresh**: Near-real-time calculations can be achieved through the partitioned refresh of the materialized views built on tables featuring time series.
- **Synergy with views**: You can achieve multi-layer modeling by using materialized views and logical views, thus enabling the reuse of the Intermediate Layer and simplification of data models.
- **Schema change**: You can alter the calculation results through simple SQL statements, without the need to modify the complex data pipelines.

With these features, you can design comprehensive and adaptable data models to meet various business needs and scenarios.

Automatic refresh

When creating an asynchronous materialized view, you can specify the refresh strategy using the REFRESH clause. Currently, StarRocks supports the following refresh strategies of asynchronous materialized views:

- **Automatic Refresh** (`REFRESH ASYNC`): Refresh tasks are triggered each time data in the base tables changes. Data dependencies are automatically managed by the materialized view.
- **Scheduled Refresh** (`REFRESH ASYNC EVERY (INTERVAL <refresh_interval>)`): Refresh tasks are triggered at regular intervals, for example, every minute, day, or month. If there are no data changes in the base tables, the refresh task will not be triggered.
- **Manual Refresh** (`REFRESH MANUAL`): Refresh tasks are triggered only by executing REFRESH MATERIALIZED VIEW manually. This refresh strategy can be used when you maintain an external scheduling framework to trigger refresh tasks.

Syntax:

```
CREATE MATERIALIZED VIEW <name>
REFRESH
    [ ASYNC |
      ASYNC [START <time>] EVERY(<interval>) |
      MANUAL
    ]
AS <query>
```

Partitioned refresh

When creating an asynchronous materialized view, you can specify the PARTITION BY clause to associate the partitions of the base table with those of the materialized view, thus achieving partition-level refresh.

- `PARTITION BY <column>`: You can reference the same partitioning column for the base table and the materialized view. As a result, the base table and the materialized view are partitioned on the same granularity.
- `PARTITION BY date_trunc(<column>)`: You can use the date_trunc function to assign a different partition strategy (on the level of granularity) for the materialized view by truncating the time unit.
- `PARTITION BY { time_slice | date_slice }(<column>)`: In comparison to date_trunc, time_slice and date_slice offer more flexible time granularity adjustments, allowing for finer control over partitioning based on time.

Syntax:

```
CREATE MATERIALIZED VIEW <name>
REFRESH ASYNC
PARTITION BY
    [
        <base_table_column> |
        date_trunc(<granularity>, <base_table_column>) |
        time_slice(<base_table_column>, <granularity>) |
        date_slice(<base_table_column>, <granularity>)
    ]
AS <query>
```

Synergy with views

- Materialized views can be created based on views. In this case, when the base tables referenced by the view undergo data changes, the materialized view can be automatically refreshed.
- You can also create materialized views based on other materialized views, enabling multi-level cascading refresh mechanisms.
- Views can be created based on materialized views, which are equivalent to regular tables.

Schema change

- You can perform an atomic exchange between two asynchronous materialized views using the ALTER MATERIALIZED VIEW SWAP statement. This allows you to create a new materialized view with added columns or changed column types and then replace the old one with it.
- The definition of views can be directly modified using the ALTER VIEW statement.
- Regular tables in StarRocks can be modified using either SWAP or ALTER operations.
- Furthermore, when there are changes in the base tables (which can be materialized views, views, or regular tables), it triggers cascading changes in the corresponding materialized views.

Layered modeling

In many real-world business scenarios, there are various forms of data sources, including real-time detail data, dimension data, and historical data from data lakes. On the other hand, business requirements call for diverse analytical methods, such as real-time dashboards, near-real-time BI queries, ad-hoc queries, and scheduled reports. Different scenarios have different demands - some require flexibility, some prioritize performance, while others emphasize cost-effectiveness.

Clearly, a single solution cannot adequately address such versatile demands. StarRocks can efficiently address these needs by combining the usage of views and materialized views. Because views maintain no physical data, each time a view is queried, the query is parsed and executed according to the definition of the view. In comparison, materialized views, which hold the pre-computed results, can prevent the overhead of repetitive execution. Views are suitable for expressing business semantics and simplifying SQL complexity, but they cannot reduce the costs of query execution. Materialized Views, on the other hand,

optimize the query performance through pre-computation and are suitable for streamlining ETL pipelines.

Below is a summary of the differences between views and materialized views:

	View	Materialized view
Use cases	Business modeling, data governance	Data modeling, transparent acceleration, data lake integration
Storage cost	No storage cost	Storage cost incurred by storing pre-computed results
Update cost	No update cost	Refresh cost incurred when base table data updates
Performance benefits	No performance benefit	Query acceleration introduced by reusing pre-computes results
Data real-time attribute	The latest data is returned because queries against views are computed in real-time.	Data may not be up-to-date because results are pre-computed.
Dependency	Views become invalid if base table names are changed because they reference base tables by name.	Changes of base table names do not affect the availability of the materialized views, which reference base tables by ID.
Creation Syntax	CREATE VIEW	CREATE MATERIALIZED VIEW
Modification Syntax	ALTER VIEW	ALTER MATERIALIZED VIEW

You can use the following statements to modify your views, materialized views, and base tables:

```
-- Modify a table.
ALTER TABLE <table_name> ADD COLUMN <column_desc>;

-- Swap two tables.
ALTER TABLE <table1> SWAP WITH <table2>;

-- Modify the definition of a view.
ALTER VIEW <view_name> AS <query>;

-- Swap two materialized views
-- (by swapping the name of the two materialized views without
affecting the data within).
ALTER MATERIALIZED VIWE <mv1> SWAP WITH <mv2>;

-- Re-activate a materialized view.
ALTER MATERIALIZED VIEW <mv_name> ACTIVE;
```

Schema changes follow these principles:

- Rename and Swap operations on a table will set the dependent materialized views inactive. For Schema Change operations, the dependent materialized views are set inactive only when Schema Change operations are performed on the base table columns referenced by the materialized views.
- If you change the definition of a view, the dependent materialized views are set to inactive.
- If a materialized view is swapped, any nested materialized views built on it are set to inactive.
- The inactive status cascades upwards until there are no materialized view dependencies.
- Inactive materialized views cannot be refreshed or used for automatic query rewrite.
- Inactive materialized views can still be queried directly, but data consistency is not guaranteed until they are active again.

Whereas the data consistency of inactive materialized views cannot be guaranteed, you can restore the functionality of them using the following methods:

- **Manual repair**: You can manually repair an inactive materialized view by executing ALTER MATERIALIZED VIEW `<mv_name>` ACTIVE. This statement will recreate the materialized view based on its original SQL definition. Please note that the SQL

definition must still be valid after the underlying schema changes, otherwise, the operation will fail.

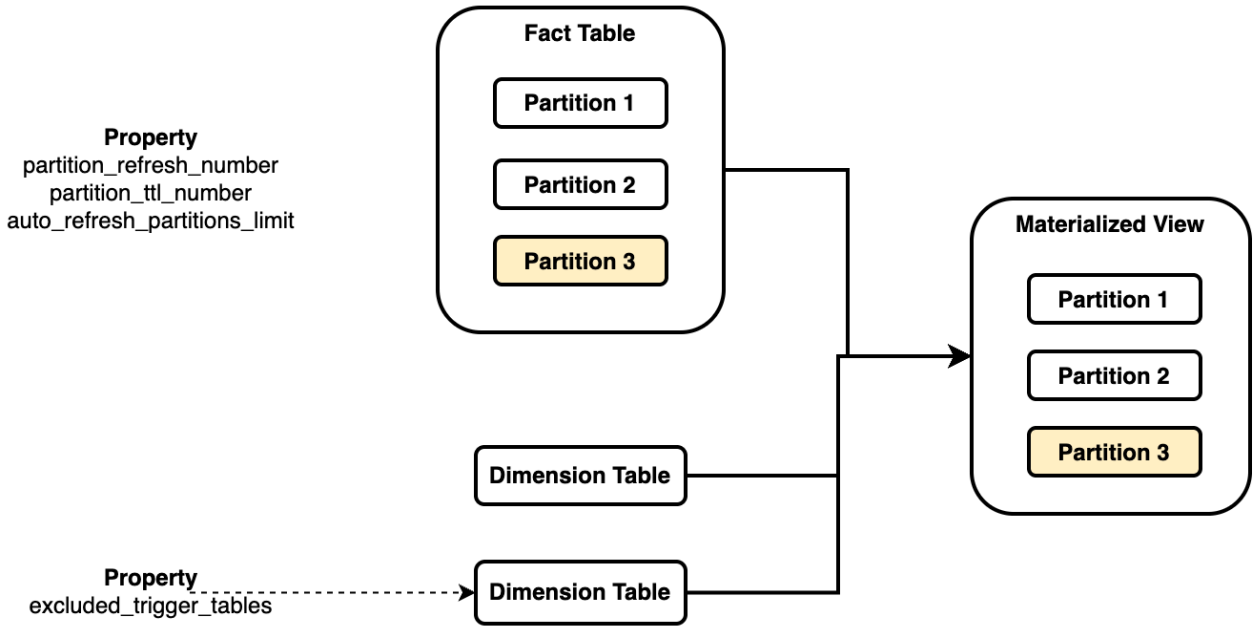
- **Automatic repair**: StarRocks will attempt to automatically activate inactive materialized views. However, the timeliness of this process cannot be guaranteed.

Partitioned modeling

In addition to layered modeling, partitioned modeling is also an essential aspect of data modeling. Data modeling often involves associating data based on business semantics and setting data's Time-To-Live (TTL) as per timeliness requirements. Partitioned modeling plays a significant role in this process.

Partitioned modeling is an essential aspect of data modeling, complementing layered modeling. It involves associating data based on business semantics and setting Time-To-Live (TTL) for data as per timeliness requirements. Data partitioning plays a significant role in this process.

Different ways of associating data give rise to various modeling approaches, such as star schemas and snowflake schemas. These models have something in common - they all use fact tables and dimension tables. Some business scenarios require multiple large fact tables, while others deal with complex dimension tables and the relationships among them. StarRocks' materialized views support partition association for fact tables, meaning that the fact table is partitioned, and the materialized view's join results are partitioned in the same way.



As the above figure shows, a materialized view associates a fact table with multiple dimension tables:

- You need to reference the partition key of a specific base table (usually the fact table) as the materialized view's partitioning key (`PARTITION BY fact_tbl.col`) to associate their partitioning strategies. Each materialized view can only be associated with one base table.
- When the data in a partition of the referenced table changes, the corresponding partition in the materialized view is refreshed without affecting other partitions.
- When the data in the non-referenced tables changes, the entire materialized view is refreshed by default. However, you can choose to ignore the data changes in certain non-referenced base tables so that the materialized view will not be refreshed when data in these tables changes.

Such partition association supports various business scenarios:

- **Fact table updates**: You can partition the fact table at a fine-grained level, such as daily or hourly. After the fact table is updated, the corresponding partitions in the materialized view are automatically refreshed.
- **Dimension table updates**: Usually, data updates in dimension tables will lead to the refresh of all associated results, which can be costly. You can choose to ignore the data updates in some dimension tables to avoid refreshing the entire materialized view, or you can specify a time range so that only the partitions within the time range can be refreshed.
- **External table automatic refresh**: In external data sources like Apache Hive or Apache Iceberg, data changes at the partition level. StarRocks' materialized views can subscribe to changes in external catalogs at the partition level, and only refresh the corresponding partition of the materialized view.
- **TTL**: When setting the partitioning strategy for a materialized view, you can set the number of recent partitions to retain, thus keeping only the most recent data. This is useful in business scenarios when analysts only query up-to-date data from a certain window of time, and they do not need to retain all the historical data.

Several parameters can be used to control refresh behavior:

- `partition_refresh_number`: the number of partitions to refresh in each refresh operation.
- `partition_ttl_number`: the number of recent partitions to retain.

- `excluded_trigger_tables`: the table whose data changes can be ignored to avoid triggering the automatic refresh.
- `auto_refresh_partitions_limit`: the number of partitions to refresh in each automatic refresh operation.

For more information, see CREATE MATERIALIZED VIEW.

Currently, partitioned materialized views have the following limitations:

- You can only build a partitioned materialized view based on a partitioned table.
- You can only use the DATE or DATETIME type columns as the partitioning key. The STRING data type is not supported.
- You can only perform partition roll-up using the date_trunc, time_slice, and date_slice functions.
- You can only specify a single column as the partitioning key. Multiple partitioning columns are not supported.

Summary

Utilizing StarRocks' asynchronous materialized views for data modeling offers the advantage of simplifying pipeline management and enhancing the efficiency and flexibility of data modeling through a declarative modeling language.

Apart from the data modeling, StarRocks' asynchronous materialized views find application in various scenarios involving transparent acceleration and data lake integration. This facilitates further exploration of data value and improves data efficiency.

Query rewrite with materialized views

This topic describes how to leverage StarRocks' asynchronous materialized views to rewrite and accelerate queries.

Overview

StarRocks' asynchronous materialized view uses a widely adopted transparent query rewrite algorithm based on the SPJG (select-project-join-group-by) form. Without the need to modify the query statement, StarRocks can automatically rewrite queries against the base tables into queries against the corresponding materialized view that contains the pre-computed results. As a result, materialized views can help you significantly reduce computational costs, and substantially accelerate query execution.

The query rewrite feature based on asynchronous materialized views is particularly useful in the following scenarios:

- **Pre-aggregation of metrics**

  You can use materialized views to create a pre-aggregated metric layer if you are dealing with a high dimensionality of data.

- **Joins of wide tables**

  Materialized views allow you to transparently accelerate queries with joins of multiple large wide tables in complex scenarios.

- **Query acceleration in the data lake**

  Building an external catalog-based materialized view can easily accelerate queries against data in your data lake.

  > **NOTE**

> Asynchronous materialized views created on base tables in a JDBC catalog do not support query rewrite.

Features

StarRocks' asynchronous materialized view-based automatic query rewrite features the following attributes:

- **Strong data consistency**: If the base tables are native tables, StarRocks ensures that the results obtained through the materialized view-based query rewrite are consistent with the results returned from the direct query against the base tables.
- **Staleness rewrite**: StarRocks supports staleness rewrite, allowing you to tolerate a certain level of data expiration to cope with scenarios with frequent data changes.
- **Multi-table joins**: StarRocks' asynchronous materialized view supports various types of joins, including some complex join scenarios like View Delta Joins and Derivable Joins, allowing you to accelerate queries in scenarios involving large wide tables.
- **Aggregation rewrite**: StarRocks can rewrite queries with aggregations to improve report performance.
- **Nested materialized view**: StarRocks supports rewriting complex queries based on nested materialized views, expanding the scope of queries that can be rewritten.
- **Union rewrite**: You can combine the Union rewrite feature with the TTL (Time-to-Live) of the materialized view's partitions to achieve the separation of the hot and cold data, which allows you to query hot data from materialized views and historical data from the base table.
- **Materialized views on views**: You can accelerate the queries in scenarios with data modeling based on views.
- **Materialized views on external catalogs**: You can accelerate queries in data lakes.
- **Complex expression rewrite**: It can handle complex expressions, including function calls and arithmetic operations, catering to advanced analytical and calculation requirements.

These features will be elaborated in the following sections.

Join rewrite

StarRocks supports rewriting queries with various types of joins, including Inner Join, Cross Join, Left Outer Join, Full Outer Join, Right Outer Join, Semi Join, and Anti Join.

The following is an example of rewriting queries with joins. Create two base tables as follows:

```
CREATE TABLE customer (
  c_custkey     INT(11)     NOT NULL,
  c_name        VARCHAR(26) NOT NULL,
  c_address     VARCHAR(41) NOT NULL,
  c_city        VARCHAR(11) NOT NULL,
  c_nation      VARCHAR(16) NOT NULL,
  c_region      VARCHAR(13) NOT NULL,
  c_phone       VARCHAR(16) NOT NULL,
  c_mktsegment  VARCHAR(11) NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(c_custkey)
DISTRIBUTED BY HASH(c_custkey) BUCKETS 12;

CREATE TABLE lineorder (
  lo_orderkey         INT(11) NOT NULL,
  lo_linenumber       INT(11) NOT NULL,
  lo_custkey          INT(11) NOT NULL,
  lo_partkey          INT(11) NOT NULL,
  lo_suppkey          INT(11) NOT NULL,
  lo_orderdate        INT(11) NOT NULL,
  lo_orderpriority    VARCHAR(16) NOT NULL,
  lo_shippriority     INT(11) NOT NULL,
  lo_quantity         INT(11) NOT NULL,
  lo_extendedprice    INT(11) NOT NULL,
  lo_ordtotalprice    INT(11) NOT NULL,
  lo_discount         INT(11) NOT NULL,
  lo_revenue          INT(11) NOT NULL,
  lo_supplycost       INT(11) NOT NULL,
  lo_tax              INT(11) NOT NULL,
  lo_commitdate       INT(11) NOT NULL,
  lo_shipmode         VARCHAR(11) NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(lo_orderkey)
DISTRIBUTED BY HASH(lo_orderkey) BUCKETS 48;
```

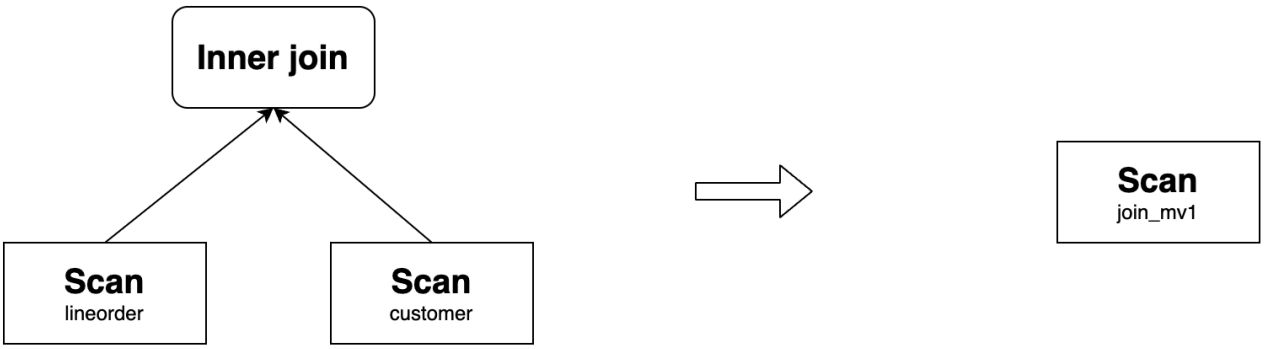
With the above base tables, you can create a materialized view as follows:

```
CREATE MATERIALIZED VIEW join_mv1
DISTRIBUTED BY HASH(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, lo_partkey, c_name,
```

```
c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

Such a materialized view can rewrite the following query:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```



StarRocks supports rewriting join queries with complex expressions, such as arithmetic operations, string functions, date functions, CASE WHEN expressions, and OR predicates. For example, the above materialized view can rewrite the following query:

```
SELECT
    lo_orderkey,
    lo_linenumber,
    (2 * lo_revenue + 1) * lo_linenumber,
    upper(c_name),
    substr(c_address, 3)
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

In addition to the conventional scenario, StarRocks further supports rewriting join queries in more complicated scenarios.

Query Delta Join rewrite

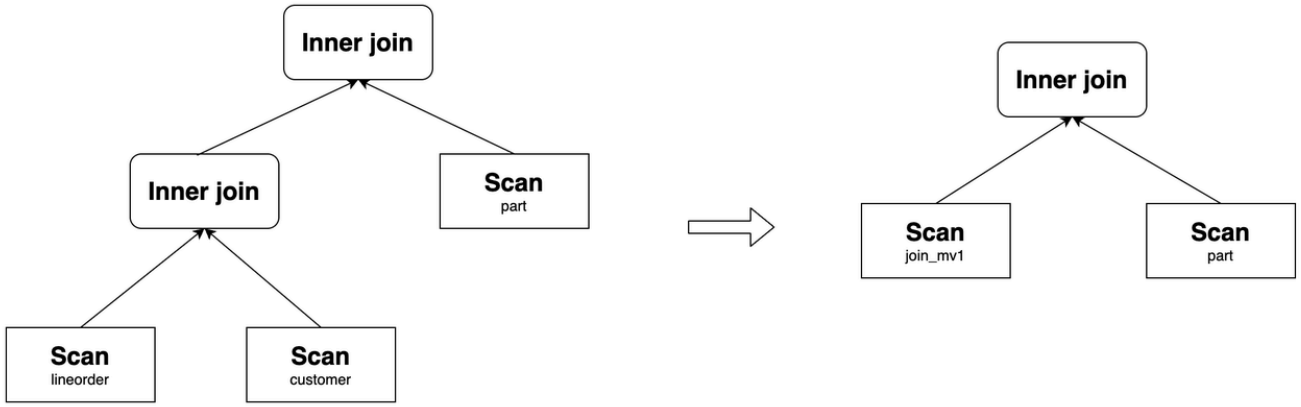
Query Delta Join refers to a scenario in which the tables joined in a query are a superset of the tables joined in a materialized view. For instance, consider the following query that

involves joins of three tables: `lineorder`, `customer`, and `part`. If the materialized view `join_mv1` contains only the join of `lineorder` and `customer`, StarRocks can rewrite the query using `join_mv1`.

Example:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address,
p_name
FROM
    lineorder INNER JOIN customer ON lo_custkey = c_custkey
    INNER JOIN part ON lo_partkey = p_partkey;
```

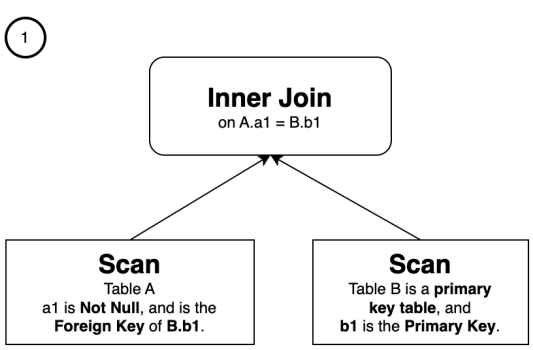
Its original query plan and the one after the rewrite are as follows:

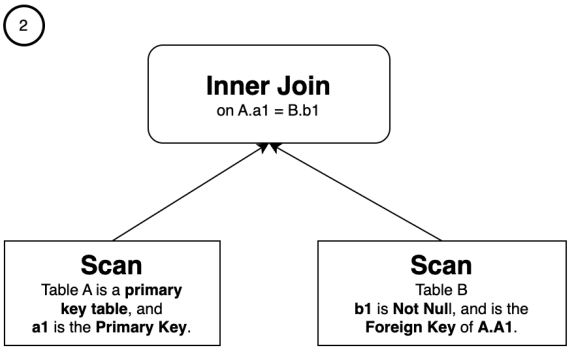


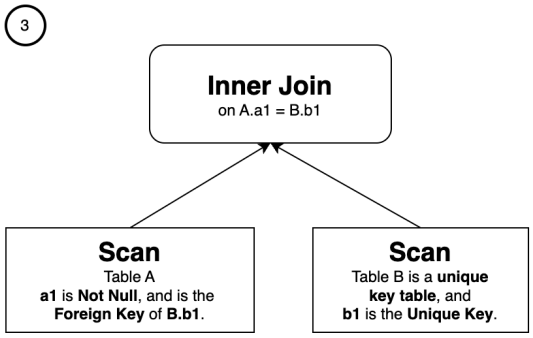
View Delta Join rewrite

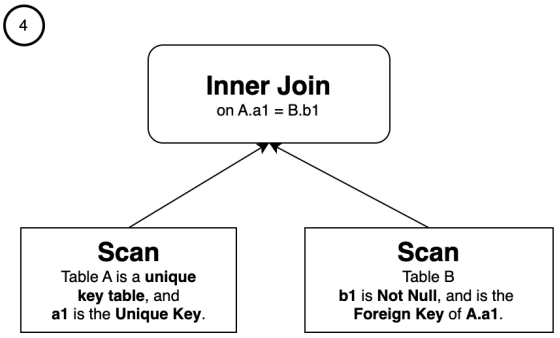
View Delta Join refers to a scenario in which the tables joined in a query are a subset of the tables joined in a materialized view. This feature is typically used in scenarios involving large wide tables. For example, in the context of the Star Schema Benchmark (SSB), you can create a materialized view that joins all tables to improve query performance. Through testing, it has been found that query performance for multi-table joins can achieve the same level of performance as querying the corresponding large wide table after transparently rewriting the queries through the materialized view.

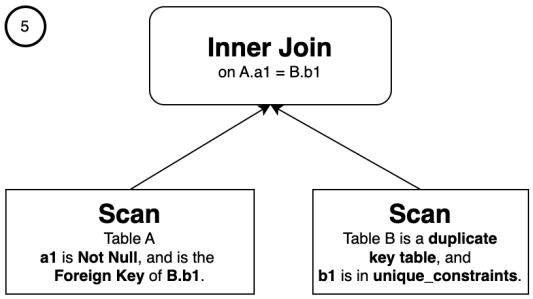
To perform a View Delta Join rewrite, the materialized view must contain the 1:1 cardinality preservation join that does not exist in the query. Here are the nine types of joins that are considered cardinality preservation joins, and satisfying any one of them enables View Delta Join rewriting:

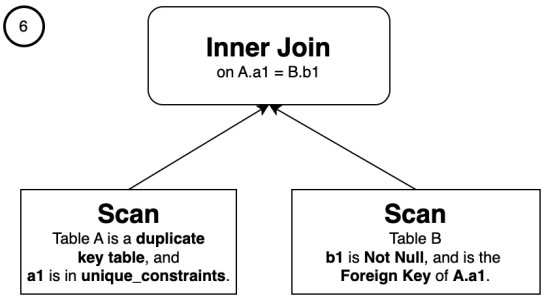


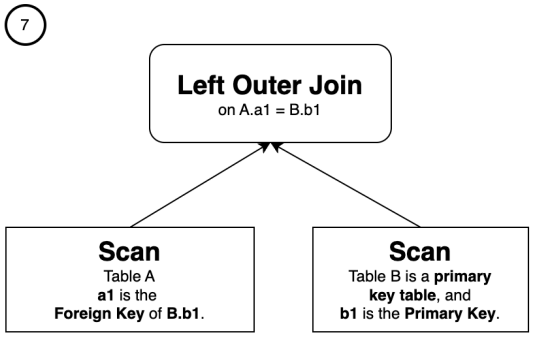


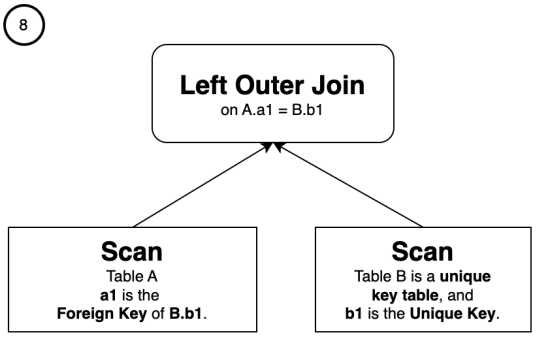


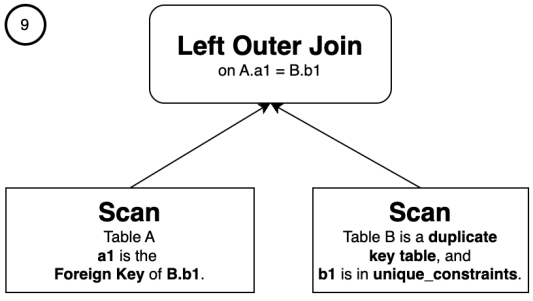












Take SSB tests as an example, create the following base tables:

```
CREATE TABLE customer (
  c_custkey         INT(11)       NOT NULL,
  c_name            VARCHAR(26)   NOT NULL,
  c_address         VARCHAR(41)   NOT NULL,
  c_city            VARCHAR(11)   NOT NULL,
  c_nation          VARCHAR(16)   NOT NULL,
  c_region          VARCHAR(13)   NOT NULL,
  c_phone           VARCHAR(16)   NOT NULL,
  c_mktsegment      VARCHAR(11)   NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(c_custkey)
DISTRIBUTED BY HASH(c_custkey) BUCKETS 12
PROPERTIES (
"unique_constraints" = "c_custkey"   -- Specify the unique
constraints.
);

CREATE TABLE dates (
  d_datekey          DATE          NOT NULL,
  d_date             VARCHAR(20)   NOT NULL,
  d_dayofweek        VARCHAR(10)   NOT NULL,
  d_month            VARCHAR(11)   NOT NULL,
  d_year             INT(11)       NOT NULL,
  d_yearmonthnum     INT(11)       NOT NULL,
  d_yearmonth        VARCHAR(9)    NOT NULL,
  d_daynuminweek     INT(11)       NOT NULL,
  d_daynuminmonth    INT(11)       NOT NULL,
  d_daynuminyear     INT(11)       NOT NULL,
  d_monthnuminyear   INT(11)       NOT NULL,
  d_weeknuminyear    INT(11)       NOT NULL,
  d_sellingseason    VARCHAR(14)   NOT NULL,
  d_lastdayinweekfl  INT(11)       NOT NULL,
  d_lastdayinmonthfl INT(11)       NOT NULL,
  d_holidayfl        INT(11)       NOT NULL,
  d_weekdayfl        INT(11)       NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(d_datekey)
DISTRIBUTED BY HASH(d_datekey) BUCKETS 1
PROPERTIES (
"unique_constraints" = "d_datekey"   -- Specify the unique
constraints.
);

CREATE TABLE supplier (
  s_suppkey          INT(11)       NOT NULL,
```

```
  s_name              VARCHAR(26)   NOT NULL,
  s_address           VARCHAR(26)   NOT NULL,
  s_city              VARCHAR(11)   NOT NULL,
  s_nation            VARCHAR(16)   NOT NULL,
  s_region            VARCHAR(13)   NOT NULL,
  s_phone             VARCHAR(16)   NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(s_suppkey)
DISTRIBUTED BY HASH(s_suppkey) BUCKETS 12
PROPERTIES (
"unique_constraints" = "s_suppkey"   -- Specify the unique
constraints.
);

CREATE TABLE part (
  p_partkey           INT(11)       NOT NULL,
  p_name              VARCHAR(23)   NOT NULL,
  p_mfgr              VARCHAR(7)    NOT NULL,
  p_category          VARCHAR(8)    NOT NULL,
  p_brand             VARCHAR(10)   NOT NULL,
  p_color             VARCHAR(12)   NOT NULL,
  p_type              VARCHAR(26)   NOT NULL,
  p_size              TINYINT(11)   NOT NULL,
  p_container         VARCHAR(11)   NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(p_partkey)
DISTRIBUTED BY HASH(p_partkey) BUCKETS 12
PROPERTIES (
"unique_constraints" = "p_partkey"   -- Specify the unique
constraints.
);

CREATE TABLE lineorder (
  lo_orderdate        DATE          NOT NULL, -- Specify it as NOT
NULL.
  lo_orderkey         INT(11)       NOT NULL,
  lo_linenumber       TINYINT       NOT NULL,
  lo_custkey          INT(11)       NOT NULL, -- Specify it as NOT
NULL.
  lo_partkey          INT(11)       NOT NULL, -- Specify it as NOT
NULL.
  lo_suppkey          INT(11)       NOT NULL, -- Specify it as NOT
NULL.
  lo_orderpriority    VARCHAR(100)  NOT NULL,
  lo_shippriority     TINYINT       NOT NULL,
  lo_quantity         TINYINT       NOT NULL,
```

```
  lo_extendedprice    INT(11)       NOT NULL,
  lo_ordtotalprice    INT(11)       NOT NULL,
  lo_discount         TINYINT       NOT NULL,
  lo_revenue          INT(11)       NOT NULL,
  lo_supplycost       INT(11)       NOT NULL,
  lo_tax              TINYINT       NOT NULL,
  lo_commitdate       DATE          NOT NULL,
  lo_shipmode         VARCHAR(100)  NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(lo_orderdate,lo_orderkey)
PARTITION BY RANGE(lo_orderdate)
(PARTITION p1 VALUES [("0000-01-01"), ("1993-01-01")),
PARTITION p2 VALUES [("1993-01-01"), ("1994-01-01")),
PARTITION p3 VALUES [("1994-01-01"), ("1995-01-01")),
PARTITION p4 VALUES [("1995-01-01"), ("1996-01-01")),
PARTITION p5 VALUES [("1996-01-01"), ("1997-01-01")),
PARTITION p6 VALUES [("1997-01-01"), ("1998-01-01")),
PARTITION p7 VALUES [("1998-01-01"), ("1999-01-01")))
DISTRIBUTED BY HASH(lo_orderkey) BUCKETS 48
PROPERTIES (
"foreign_key_constraints" = "
    (lo_custkey) REFERENCES customer(c_custkey);
    (lo_partkey) REFERENCES part(p_partkey);
    (lo_suppkey) REFERENCES supplier(s_suppkey)" -- Specify the
Foreign Keys.
);
```

Create the materialized view `lineorder_flat_mv` that joins `lineorder`, `customer`, `supplier`, `part`, and `dates`:

```
CREATE MATERIALIZED VIEW lineorder_flat_mv
DISTRIBUTED BY HASH(LO_ORDERDATE, LO_ORDERKEY) BUCKETS 48
PARTITION BY LO_ORDERDATE
REFRESH MANUAL
PROPERTIES (
    "partition_refresh_number"="1"
)
AS SELECT /*+ SET_VAR(query_timeout = 7200) */     -- Set timeout for
the refresh operation.
       l.LO_ORDERDATE        AS LO_ORDERDATE,
       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_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,
d.D_DATE              AS D_DATE,
d.D_DAYOFWEEK         AS D_DAYOFWEEK,
d.D_MONTH             AS D_MONTH,
d.D_YEAR              AS D_YEAR,
d.D_YEARMONTHNUM      AS D_YEARMONTHNUM,
d.D_YEARMONTH         AS D_YEARMONTH,
d.D_DAYNUMINWEEK      AS D_DAYNUMINWEEK,
d.D_DAYNUMINMONTH     AS D_DAYNUMINMONTH,
d.D_DAYNUMINYEAR      AS D_DAYNUMINYEAR,
d.D_MONTHNUMINYEAR    AS D_MONTHNUMINYEAR,
d.D_WEEKNUMINYEAR     AS D_WEEKNUMINYEAR,
d.D_SELLINGSEASON     AS D_SELLINGSEASON,
d.D_LASTDAYINWEEKFL   AS D_LASTDAYINWEEKFL,
```

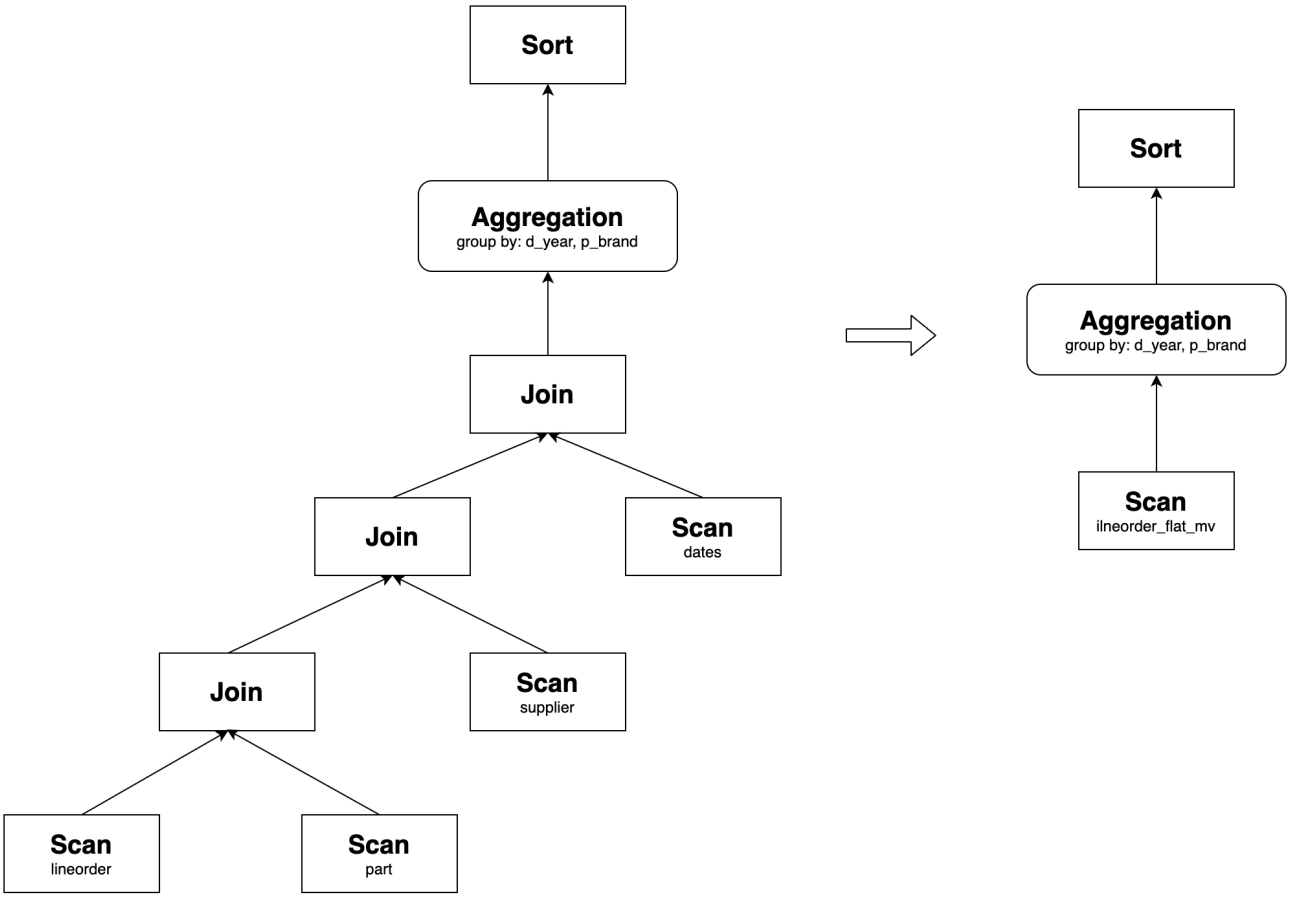
```
        d.D_LASTDAYINMONTHFL  AS D_LASTDAYINMONTHFL,
        d.D_HOLIDAYFL         AS D_HOLIDAYFL,
        d.D_WEEKDAYFL         AS D_WEEKDAYFL
    FROM lineorder            AS l
        INNER JOIN customer   AS c ON c.C_CUSTKEY = l.LO_CUSTKEY
        INNER JOIN supplier   AS s ON s.S_SUPPKEY = l.LO_SUPPKEY
        INNER JOIN part       AS p ON p.P_PARTKEY = l.LO_PARTKEY
        INNER JOIN dates      AS d ON l.LO_ORDERDATE = d.D_DATEKEY;
```

SSB Q2.1 involves joining four tables, but it lacks the `customer` table compared to the materialized view `lineorder_flat_mv`. In `lineorder_flat_mv`, `lineorder INNER JOIN customer` is essentially a cardinality preservation join. Therefore, logically, this join can be eliminated without affecting the query results. As a result, Q2.1 can be rewritten using `lineorder_flat_mv`.

SSB Q2.1:

```
SELECT sum(lo_revenue) AS lo_revenue, d_year, p_brand
FROM lineorder
JOIN dates ON lo_orderdate = d_datekey
JOIN part ON lo_partkey = p_partkey
JOIN supplier ON lo_suppkey = s_suppkey
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY d_year, p_brand
ORDER BY d_year, p_brand;
```

Its original query plan and the one after the rewrite are as follows:



Similarly, other queries in the SSB can also be transparently rewritten using `lineorder_flat_mv`, thus optimizing query performance.

Join Derivability rewrite

Join Derivability refers to a scenario in which the join types in the materialized view and the query are not consistent, but the materialized view's join results contain the results of the query's join. Currently, it supports two scenarios - joining three or more tables, and joining two tables.

- **Scenario one: Joining three or more tables**

  Suppose the materialized view contains a Left Outer Join between tables `t1` and `t2` and an Inner Join between tables `t2` and `t3`. In both joins, the join condition includes columns from `t2`.

  The query, on the other hand, contains an Inner Join between t1 and t2, and an Inner Join between t2 and t3. In both joins, the join condition includes columns from t2.

  In this case, the query can be rewritten using the materialized view. This is because in the materialized view, the Left Outer Join is executed first, followed by the Inner Join.

The right table generated by the Left Outer Join has no results for the matching (that is, columns in the right table are NULL). These results are subsequently filtered out during the Inner Join. Therefore, the logic of the materialized view and the query is equivalent, and the query can be rewritten.

Example:

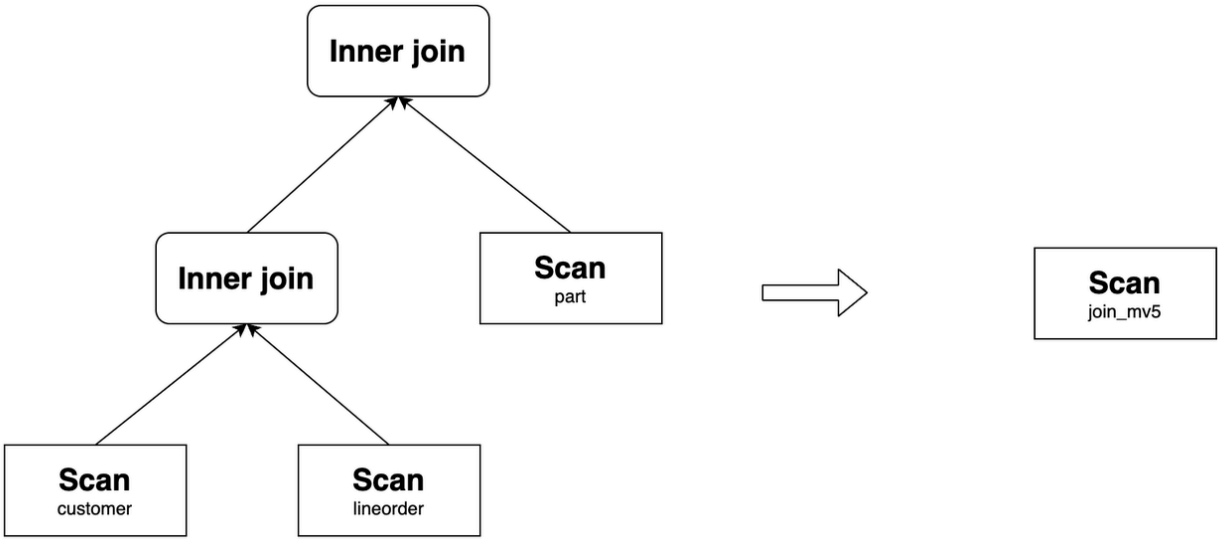
Create the materialized view `join_mv5`:

```
CREATE MATERIALIZED VIEW join_mv5
PARTITION BY lo_orderdate
DISTRIBUTED BY hash(lo_orderkey)
PROPERTIES (
  "partition_refresh_number" = "1"
)
AS
SELECT lo_orderkey, lo_orderdate, lo_linenumber, lo_revenue,
c_custkey, c_address, p_name
FROM customer LEFT OUTER JOIN lineorder
ON c_custkey = lo_custkey
INNER JOIN part
ON p_partkey = lo_partkey;
```

`join_mv5` can rewrite the following query:

```
SELECT lo_orderkey, lo_orderdate, lo_linenumber, lo_revenue,
c_custkey, c_address, p_name
FROM customer INNER JOIN lineorder
ON c_custkey = lo_custkey
INNER JOIN part
ON p_partkey = lo_partkey;
```

Its original query plan and the one after the rewrite are as follows:

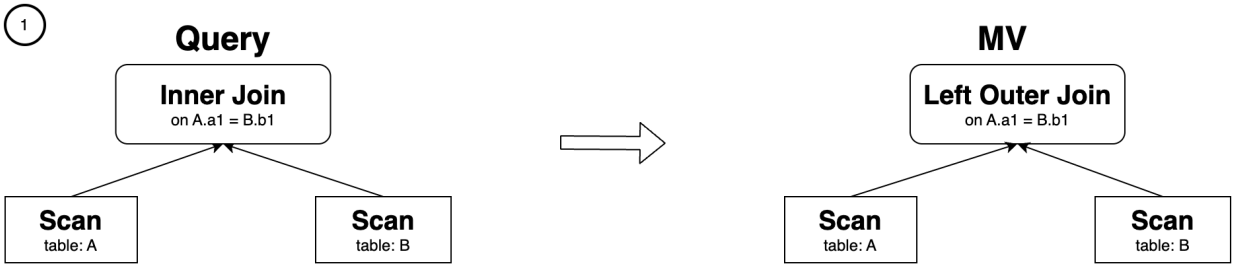


Similarly, if the materialized view is defined as `t1 INNER JOIN t2 INNER JOIN t3`, and the query is `LEFT OUTER JOIN t2 INNER JOIN t3`, the query can also be rewritten. Furthermore, this rewriting capability extends to scenarios involving more than three tables.

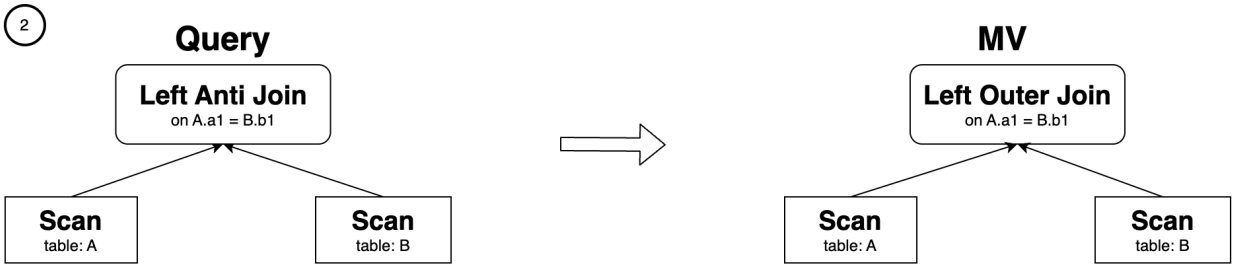
- **Scenario two: Joining two tables**

  The Join Derivability Rewrite feature involving two tables supports the following specific cases:

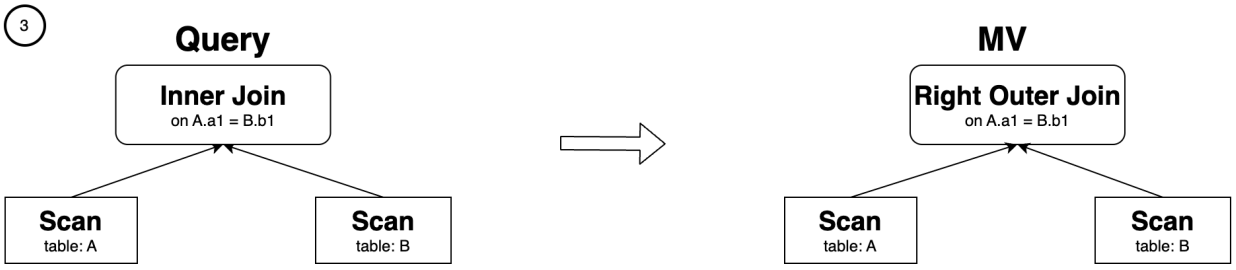
1


Query
Inner Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B
MV
Left Outer Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B

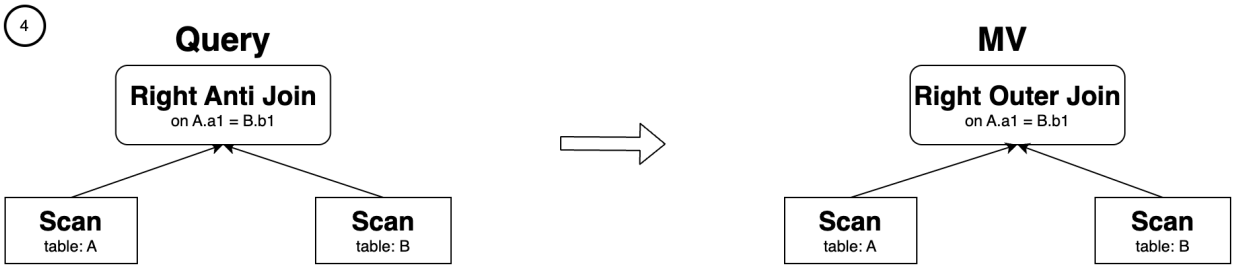
2


Query
Left Anti Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B
MV
Left Outer Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B

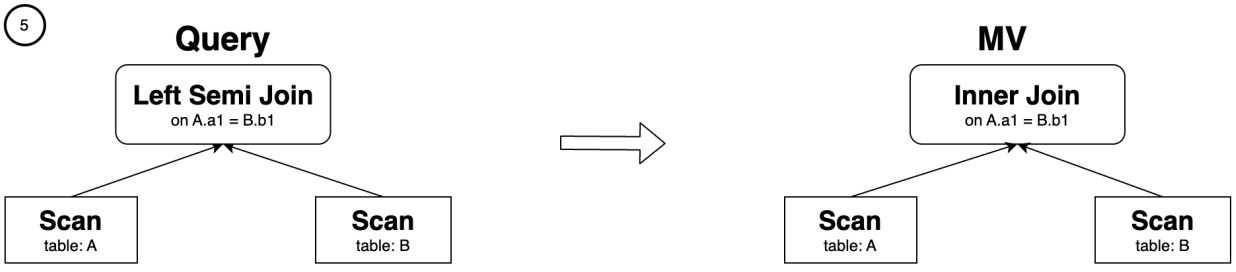
3


Query
Inner Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B
MV
Right Outer Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B

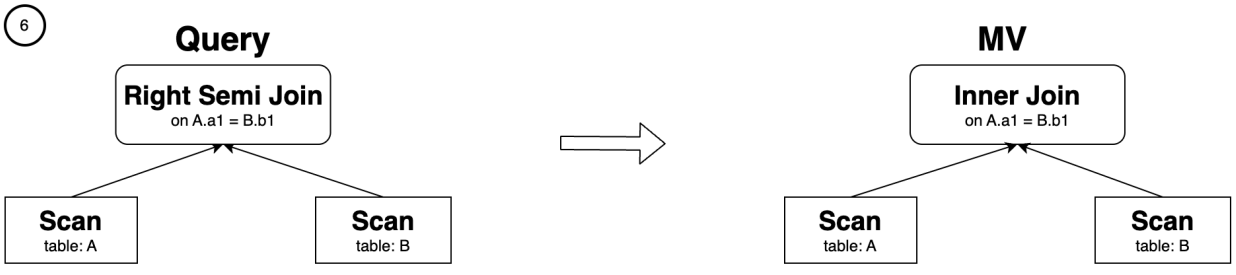
4


Query
Right Anti Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B
MV
Right Outer Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B

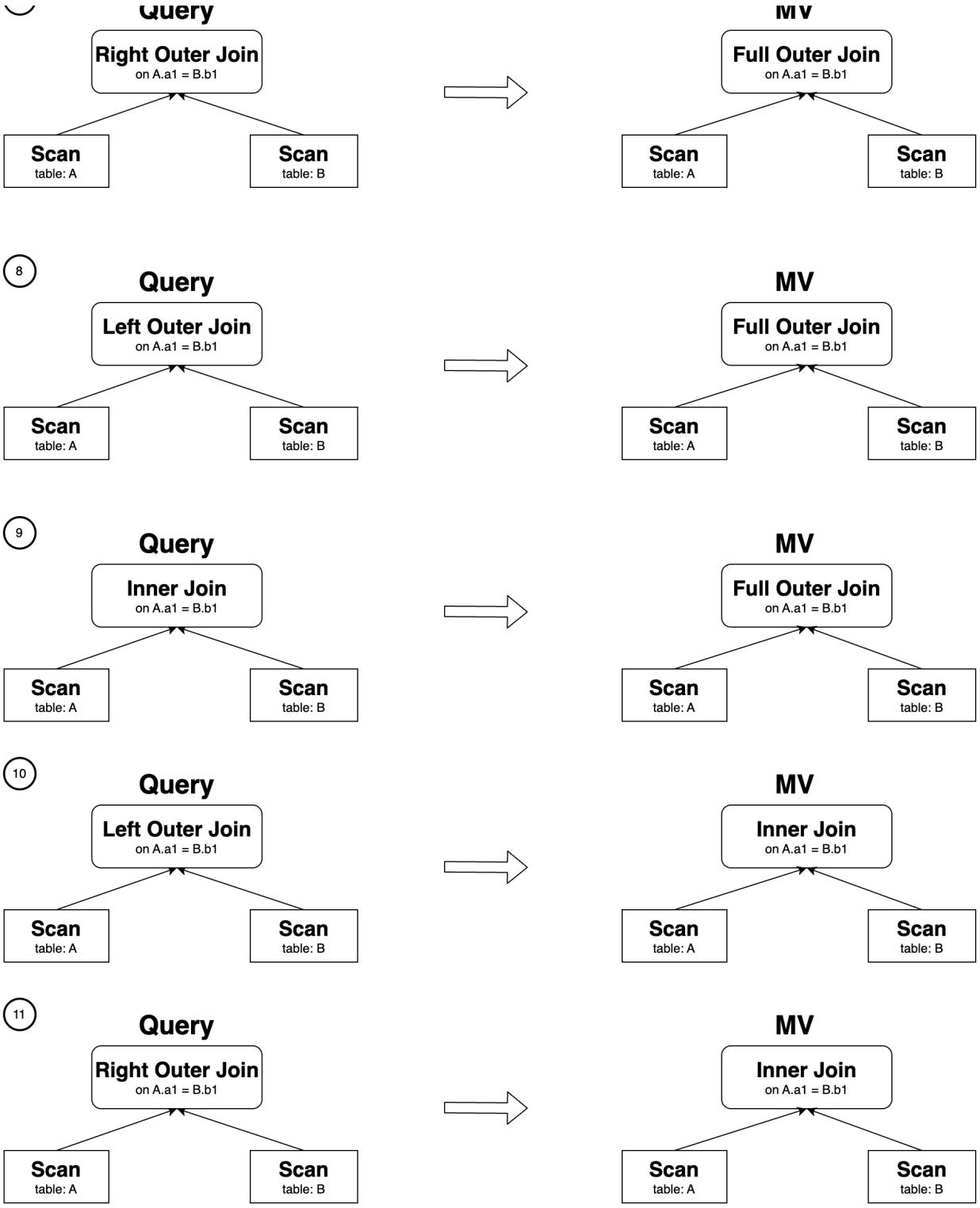
5


Query
Left Semi Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B
MV
Inner Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B

6


Query
Right Semi Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B
MV
Inner Join
on A.a1 = B.b1
Scan
table: A
Scan
table: B

7



In cases 1 to 9, filtering predicates must be added to the rewritten result to ensure semantic equivalence. For example, create a materialized view as follows:

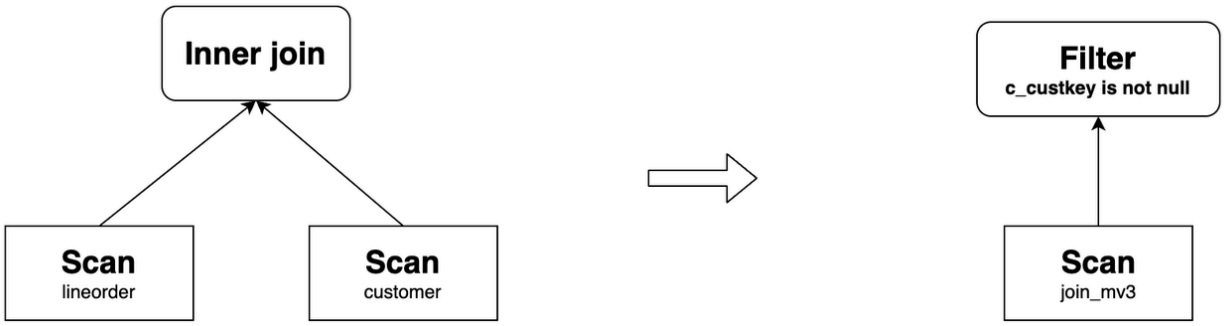
```
CREATE MATERIALIZED VIEW join_mv3
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey,
c_address
```

```
FROM lineorder LEFT OUTER JOIN customer
ON lo_custkey = c_custkey;
```

The following query can be rewritten using `join_mv3`, and the predicate `c_custkey IS NOT NULL` is added to the rewritten result:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey,
c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

Its original query plan and the one after the rewrite are as follows:



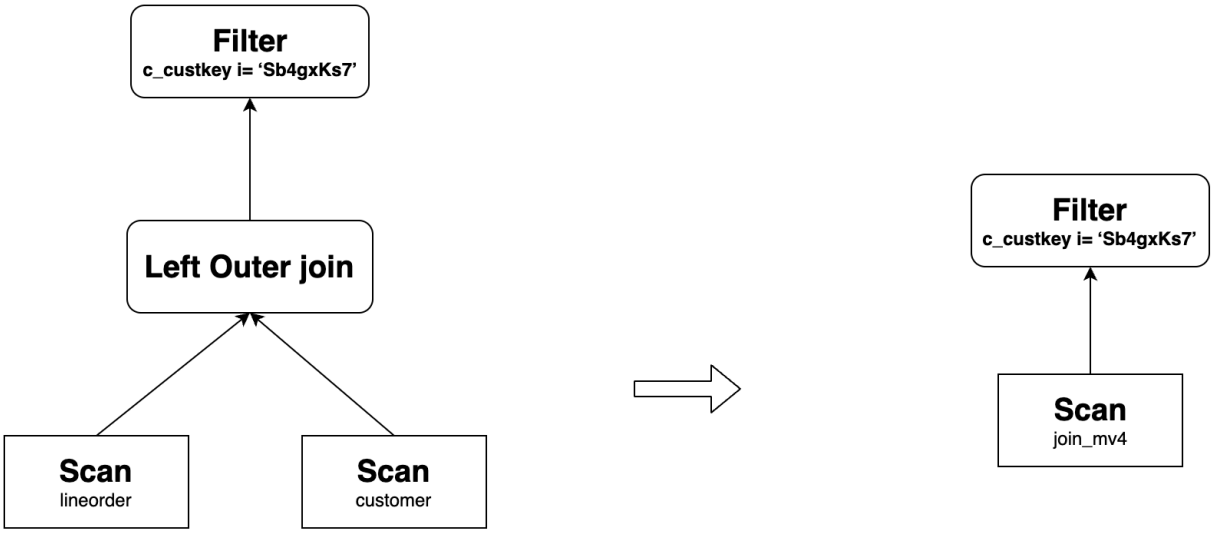
In case 10, the Left Outer Join query must include the filtering predicate `IS NOT NULL` in the right table, for example, `=`, `<>`, `>`, `<`, `<=`, `>=`, `LIKE`, `IN`, `NOT LIKE`, or `NOT IN`. For example, create a materialized view as follows:

```
CREATE MATERIALIZED VIEW join_mv4
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey,
c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

`join_mv4` can rewrite the following query, where `customer.c_address = "Sb4gxKs7"` is the filtering predicate `IS NOT NULL`:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey,
c_address
FROM lineorder LEFT OUTER JOIN customer
ON lo_custkey = c_custkey
WHERE customer.c_address = "Sb4gxKs7";
```

Its original query plan and the one after the rewrite are as follows:



Aggregation rewrite

StarRocks' asynchronous materialized view supports rewriting multi-table aggregate queries with all available aggregate functions, including bitmap_union, hll_union, and percentile_union. For example, create a materialized view as follows:

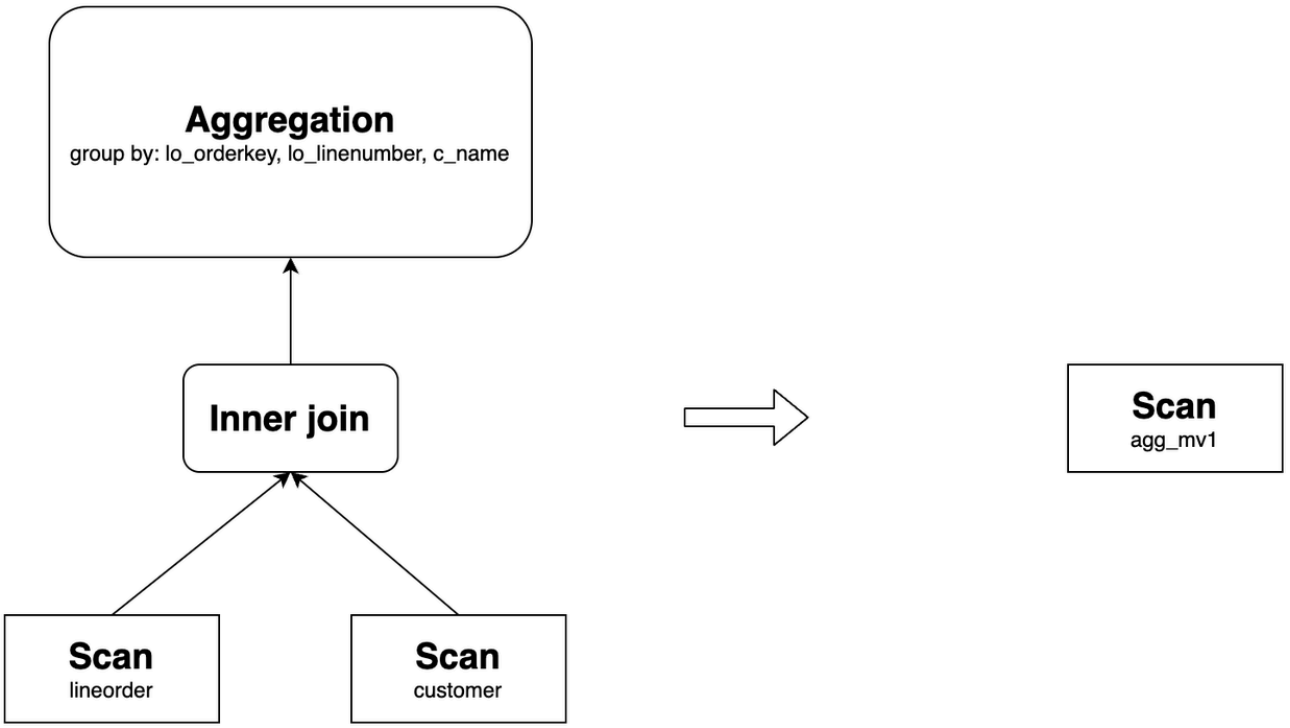
```
CREATE MATERIALIZED VIEW agg_mv1
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  lo_linenumber,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey, lo_linenumber, c_name;
```

It can rewrite the following query:

```
SELECT
  lo_orderkey,
  lo_linenumber,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
```

```
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey, lo_linenumber, c_name;
```

Its original query plan and the one after the rewrite are as follows:



The following sections expound on the scenarios where the Aggregation Rewrite feature can be useful.

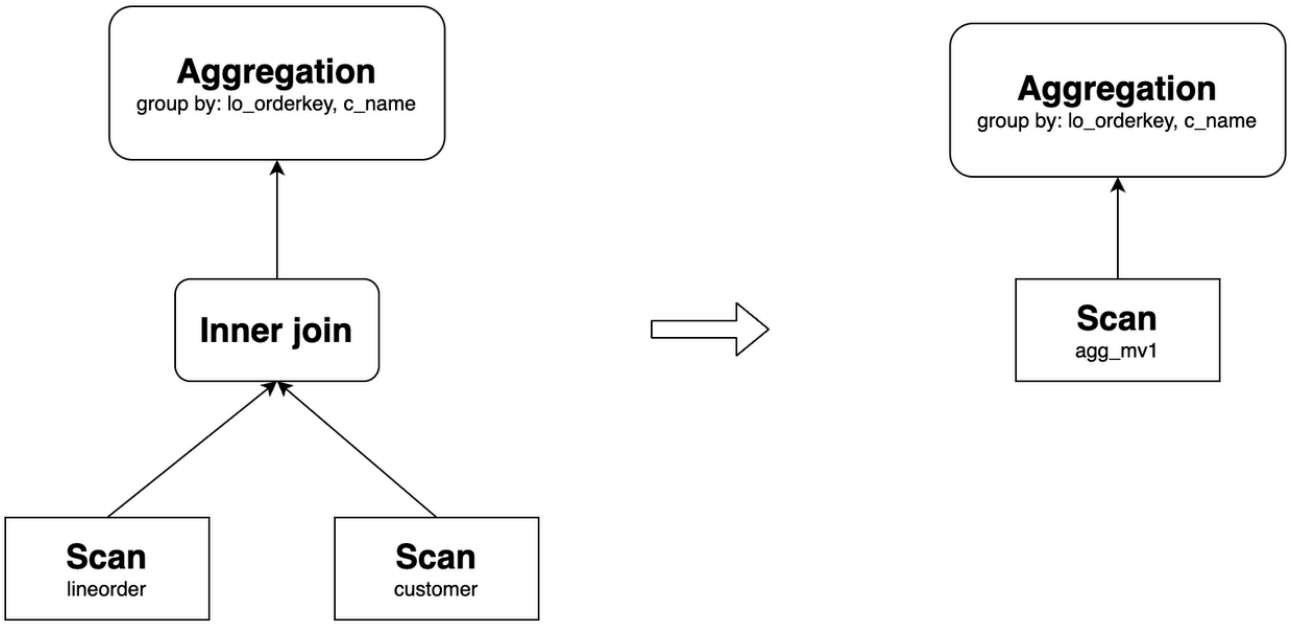
Aggregation Rollup rewrite

StarRocks supports rewriting queries with Aggregation Rollup, that is, StarRocks can rewrite aggregate queries with a `GROUP BY a` clause using an asynchronous materialized view created with a `GROUP BY a,b` clause. For example, the following query can be rewritten using `agg_mv1`:

```
SELECT
  lo_orderkey,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
```

```
ON lo_custkey = c_custkey
GROUP BY lo_orderkey, c_name;
```

Its original query plan and the one after the rewrite are as follows:



> **NOTE**
>
> Currently, rewriting the grouping set, grouping set with rollup, or grouping set with cube is not supported.

Only certain aggregate functions support query rewrite with Aggregate Rollup. In the preceding example, if the materialized view `order_agg_mv` uses `count(distinct client_id)` instead of `bitmap_union(to_bitmap(client_id))`, StarRocks cannot rewrite the queries with Aggregate Rollup.

The following table shows the correspondence between the aggregate functions in the original query and the aggregate function used to build the materialized view. You can select the corresponding aggregate functions to build a materialized view according to your business scenario.

Aggregate function suppprted in original queries	Function supported Aggregate Rollup in materialized view
sum	sum

Aggregate function supppprted in original queries	Function supported Aggregate Rollup in materialized view
count	count
min	min
max	max
avg	sum / 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

DISTINCT aggregates without the corresponding GROUP BY column cannot be rewritten with Aggregate Rollup. However, from StarRocks v3.1 onwards, if a query with an Aggregate Rollup DISTINCT aggregate function does not have a GROUP BY column but an equal predicate, it can also be rewritten by the relevant materialized view because StarRocks can convert the equal predicates into a GROUP BY constant expression.

In the following example, StarRocks can rewrite the query with the materialized view `order_agg_mv1`.

```
CREATE MATERIALIZED VIEW order_agg_mv1
DISTRIBUTED BY HASH(`order_id`) BUCKETS 12
REFRESH ASYNC START('2022-09-01 10:00:00') EVERY (interval 1 day)
AS
SELECT
    order_date,
    count(distinct client_id)
FROM order_list
GROUP BY order_date;


-- Query
```

```
SELECT
    order_date,
    count(distinct client_id)
FROM order_list WHERE order_date='2023-07-03';
```

COUNT DISTINCT rewrite

StarRocks supports rewriting COUNT DISTINCT calculations into bitmap-based calculations, enabling high-performance, precise deduplication using materialized views. For example, create a materialized view as follows:

```
CREATE MATERIALIZED VIEW distinct_mv
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, bitmap_union(to_bitmap(lo_custkey)) AS
distinct_customer
FROM lineorder
GROUP BY lo_orderkey;
```

It can rewrite the following query:

```
SELECT lo_orderkey, count(distinct lo_custkey)
FROM lineorder
GROUP BY lo_orderkey;
```

Nested materialized view rewrite

StarRocks supports rewriting queries using nested materialized view. For example, create the materialized views `join_mv2`, `agg_mv2`, and `agg_mv3` as follows:

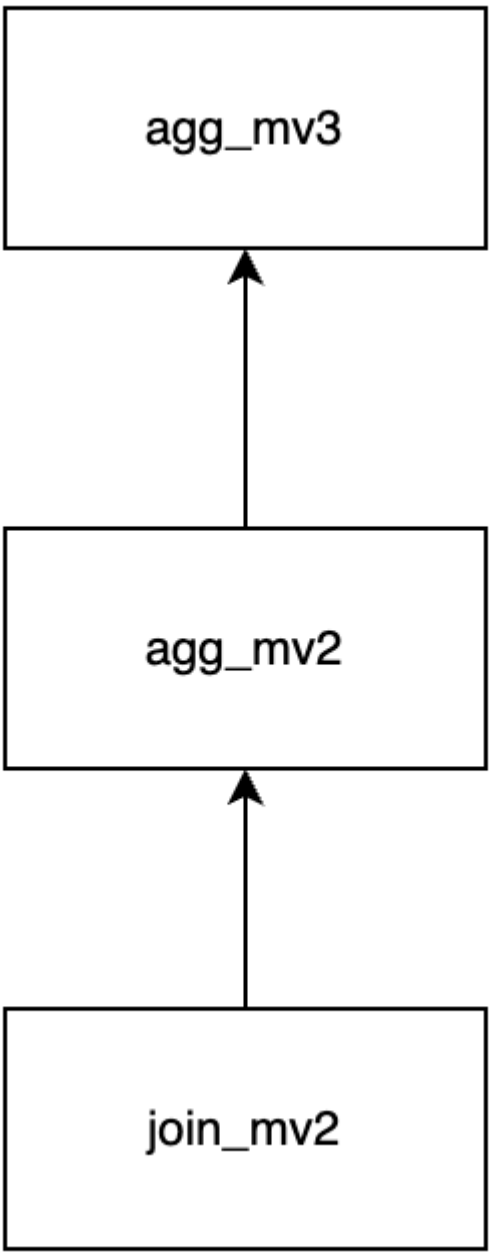
```
CREATE MATERIALIZED VIEW join_mv2
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;


CREATE MATERIALIZED VIEW agg_mv2
```

```
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  lo_linenumber,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM join_mv2
GROUP BY lo_orderkey, lo_linenumber, c_name;

CREATE MATERIALIZED VIEW agg_mv3
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  sum(total_revenue) AS total_revenue,
  max(max_discount) AS max_discount
FROM agg_mv2
GROUP BY lo_orderkey;
```

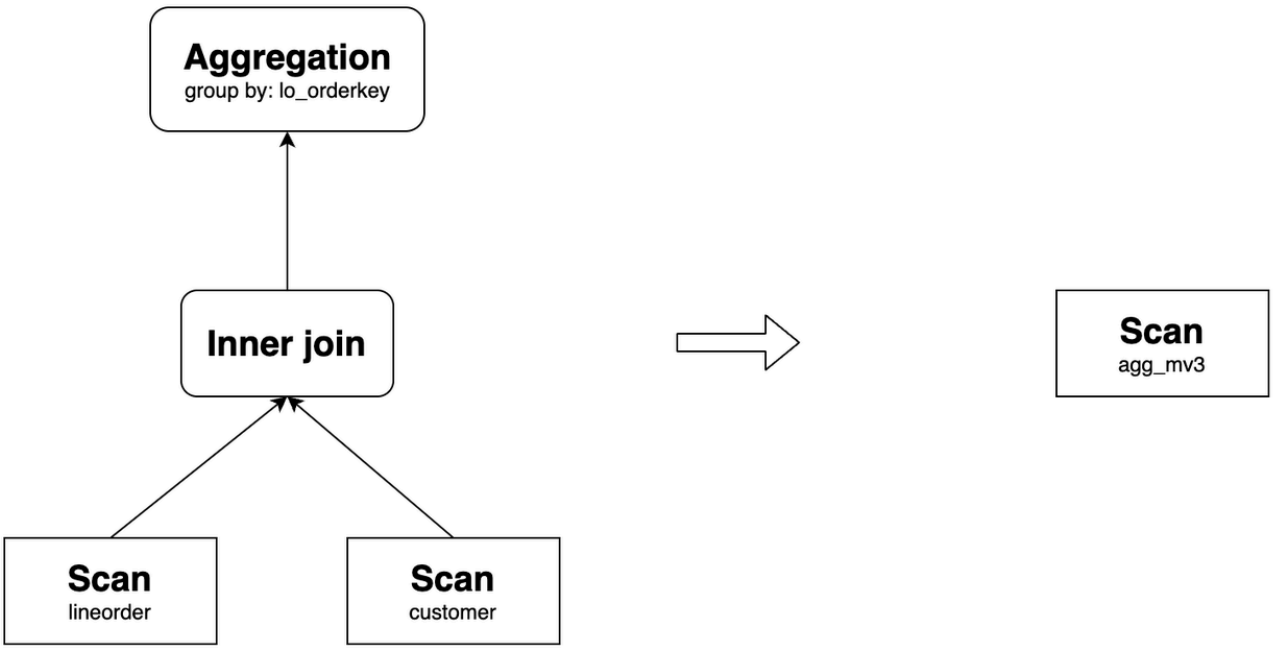
Their relationship is as follows:



`agg_mv3` can rewrite the following query:

```
SELECT
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey;
```

Its original query plan and the one after the rewrite are as follows:



Union rewrite

Predicate Union rewrite

When the predicate scope of a materialized view is a subset of the predicate scope of a query, the query can be rewritten using a UNION operation.

For example, create a materialized view as follows:

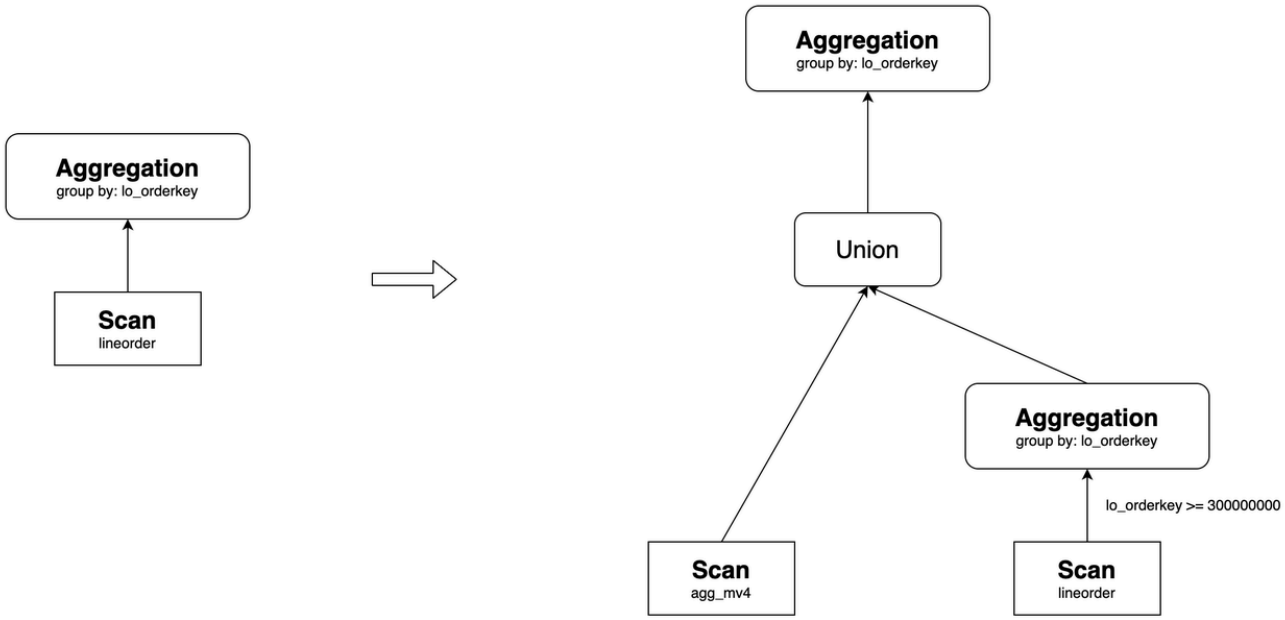
```
CREATE MATERIALIZED VIEW agg_mv4
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
WHERE lo_orderkey < 300000000
GROUP BY lo_orderkey;
```

It can rewrite the following query:

```
select
  lo_orderkey,
```

```
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
GROUP BY lo_orderkey;
```

Its original query plan and the one after the rewrite are as follows:



In this context, `agg_mv5` contains data where `lo_orderkey < 300000000`. Data where `lo_orderkey >= 300000000` is directly obtained from the base table `lineorder`. Finally, these two sets of data are combined using a UNION operation and then aggregated to obtain the final result.

Partition Union rewrite

Suppose you created a partitioned materialized view based on a partitioned table. When the partition range that a rewritable query scanned is a superset of the most recent partition range of the materialized view, the query will be rewritten using a UNION operation.

For example, consider the following materialized view `agg_mv4`. Its base table `lineorder` currently contains partitions from `p1` to `p7`, and the materialized view also contains partitions from `p1` to `p7`.

```
CREATE MATERIALIZED VIEW agg_mv5
DISTRIBUTED BY hash(lo_orderkey)
PARTITION BY RANGE(lo_orderdate)
REFRESH MANUAL
```

```
AS
SELECT
  lo_orderdate,
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
GROUP BY lo_orderkey;
```

If a new partition `p8`, with a partition range of `[("19990101"), ("20000101"))`, is added to `lineorder`, the following query can be rewritten using a UNION operation:

```
SELECT
  lo_orderdate,
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
GROUP BY lo_orderkey;
```

Its original query plan and the one after the rewrite are as follows:

As shown above, `agg_mv5` contains the data from partitions `p1` to `p7`, and the data from partition `p8` is directly queried from `lineorder`. Finally, these two sets of data are combined using a UNION operation.

View-based materialized view rewrite

StarRocks supports creating materialized views based on views. Subsequent queries against the views can be transparently rewritten.

For example, create the following views:

```
CREATE VIEW customer_view1
AS
SELECT c_custkey, c_name, c_address
FROM customer;
```

```
CREATE VIEW lineorder_view1
AS
SELECT lo_orderkey, lo_linenumber, lo_custkey, lo_revenue
FROM lineorder;
```

Then, create the following materialized view based on the views:

```
CREATE MATERIALIZED VIEW join_mv1
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name
FROM lineorder_view1 INNER JOIN customer_view1
ON lo_custkey = c_custkey;
```

During query rewrite, queries against `customer_view1` and `lineorder_view1` are automatically expanded to the base tables and then transparently matched and rewritten.

External catalog-based materialized view rewrite

StarRocks supports building asynchronous materialized views on Hive catalogs, Hudi catalogs, and Iceberg catalogs, and transparently rewriting queries with them. External catalog-based materialized views support most of the query rewrite capabilities, but there are some limitations:

- Hudi, Iceberg, or JDBC catalog-based materialized views do not support Union rewrite.
- Hudi, Iceberg, or JDBC catalog-based materialized views do not support View Delta Join rewrite.
- Hudi, Iceberg, or JDBC catalog-based materialized views do not support the incremental refresh of partitions.

Configure query rewrite

You can configure the asynchronous materialized view query rewrite through the following session variables:

Variable	Default	Description
enable_materialized_view_union_rewrite	true	Boolean value to control if to enable materialized view Union query rewrite.
enable_rule_based_materialized_view_rewrite	true	Boolean value to control if to enable rule-based materialized view query rewrite. This variable is mainly used in single-table query rewrite.
nested_mv_rewrite_max_level	3	The maximum levels of nested materialized views that can be used for query rewrite. Type: INT. Range: [1, +∞). The value of `1` indicates that materialized views created on other materialized views will not be used for query rewrite.

Check if a query is rewritten

You can check if your query is rewritten by viewing its query plan using the EXPLAIN statement. If the field `TABLE` under the section `OlapScanNode` shows the name of the corresponding materialized view, it means that the query has been rewritten based on the materialized view.

```
mysql> EXPLAIN SELECT
    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;
+-----------------------------------+
| Explain String                    |
+-----------------------------------+
```

```
| PLAN FRAGMENT 0                       |
|  OUTPUT EXPRS:1: order_id | 8: sum |
|   PARTITION: RANDOM                   |
|                                       |
|   RESULT SINK                         |
|                                       |
|   1:Project                           |
|   |  <slot 1> : 9: order_id           |
|   |  <slot 8> : 10: total             |
|   |                                   |
|   0:OlapScanNode                      |
|      TABLE: order_mv                  |
|      PREAGGREGATION: ON               |
|      partitions=1/1                   |
|      rollup: order_mv                 |
|      tabletRatio=0/12                 |
|      tabletList=                      |
|      cardinality=3                    |
|      avgRowSize=4.0                   |
|      numNodes=0                       |
+---------------------------------------+
20 rows in set (0.01 sec)
```

Disable query rewrite

By default, StarRocks enables query rewrite for asynchronous materialized views created based on the default catalog. You can disable this feature by setting the session variable `enable_materialized_view_rewrite` to `false`.

For asynchronous materialized views created based on an external catalog, you can disable this feature by setting the materialized view property `force_external_table_query_rewrite` to `false` using ALTER MATERIALIZED VIEW.

Limitations

In terms of materialized view-based query rewrite, StarRocks currently has the following limitations:

- StarRocks does not support rewriting queries with non-deterministic functions, including rand, random, uuid, and sleep.

- StarRocks does not support rewriting queries with window functions.
- Materialized views defined with statements containing LIMIT, ORDER BY, UNION, EXCEPT, INTERSECT, MINUS, GROUPING SETS, WITH CUBE, or WITH ROLLUP cannot be used for query rewrite.
- Strong consistency of query results is not guaranteed between base tables and materialized views built on external catalogs.
- Asynchronous materialized views created on base tables in a JDBC catalog do not support query rewrite.

Data lake query acceleration with materialized views

This topic describes how to optimize query performance in your data lake using StarRocks' asynchronous materialized views.

StarRocks offers out-of-the-box data lake query capabilities, which are highly effective for exploratory queries and analysis of data in the lake. In most scenarios, Data Cache can provide block-level file caching, avoiding performance degradation caused by remote storage jitter and a large number of I/O operations.

However, when it comes to building complex and efficient reports using data from the lake or further accelerating these queries, you may still encounter performance challenges. With asynchronous materialized views, you can achieve higher concurrency and better query performance for reports and data applications on the lake.

Overview

StarRocks supports building asynchronous materialized views based on external catalogs such as Hive catalog, Iceberg catalog, and Hudi catalog. External catalog-based materialized views are particularly useful in the following scenarios:

- **Transparent Acceleration of data lake reports**

  To ensure the query performance of data lake reports, data engineers typically need to work closely with data analysts to probe into the construction logic of the acceleration layer for reports. If the acceleration layer requires further updates, they must update the construction logic, processing schedules, and query statements accordingly.

  Through the query rewrite capability of materialized views, report acceleration can be made transparent and imperceptible to users. When slow queries are identified, data engineers can analyze the pattern of slow queries and create materialized views on demand. Application-side queries are then intelligently rewritten and transparently

accelerated by the materialized view, allowing for rapid improvement in query performance without modifying the logic of the business application or the query statement.

- **Incremental calculation of real-time data associated with historical data**

  Suppose your business application requires the association of real-time data in StarRocks native tables and historical data in the data lake for incremental calculations. In this situation, materialized views can provide a straightforward solution. For example, if the real-time fact table is a native table in StarRocks and the dimension table is stored in the data lake, you can easily perform incremental calculations by constructing materialized views that associate the native table with the table in the external data sources.

- **Rapid construction of metric layers**

  Calculating and processing metrics may encounter challenges when dealing with a high dimensionality of data. You can use materialized views, which allows you to perform data pre-aggregation and rollup, to create a relatively lightweight metric layer. Moreover, materialized views can be refreshed automatically, further reducing the complexity of metric calculations.

Materialized views, Data Cache, and native tables in StarRocks are all effective methods to achieve significant boosts in query performance. The following table compares their major differences:

	Data Cache	Materialized view	Native table
Data loading and updates	Queries automatically trigger data caching.	Refresh tasks are triggered automatically.	Supports various import methods but requires manual maintenance of import tasks
Data caching granularity	• Supports block-level data caching	Stores the precomputed query results	Stores data based on table schema

	Data Cache	Materialized view	Native table
	• Follows LRU cache eviction mechanism • No computation results are cached		
Query performance	Data Cache ≤ Materialized view = Native table		
Query statement	• No need to modify query statements against the data lake • Once queries hit the cache, computation occurs.	• No need to modify query statements against the data lake • Leverages query rewrite to reuse precomputed results	Requires modification of query statements to query the native table

Compared to directly querying lake data or loading data into native tables, materialized views offer several unique advantages:

- **Local storage acceleration**: Materialized views can leverage StarRocks' acceleration advantages with local storage, such as indexes, partitioning, bucketing, and collocate groups, resulting in better query performance compared to querying data from the data lake directly.
- **Zero maintenance for loading tasks**: Materialized views update data transparently via automatic refresh tasks. There's no need to maintain loading tasks to perform scheduled data updates. Additionally, Hive catalog-based materialized views can detect data changes and perform incremental refreshes at the partition level.
- **Intelligent query rewrite**: Queries can be transparently rewritten to use materialized views. You can benefit from acceleration instantly without the need to modify the query statements your application uses.

Therefore, we recommend using materialized views in the following scenarios:

- Even when Data Cache is enabled, query performance does not meet your requirements for query latency and concurrency.
- Queries involve reusable components, such as fixed aggregation functions or join patterns.
- Data is organized in partitions, while queries involve aggregation on a relatively high level (e.g., aggregating by day).

In the following scenarios, we recommend prioritizing acceleration through Data Cache:

- Queries do not have many reusable components and may scan any data from the data lake.
- Remote storage has significant fluctuations or instability, which could potentially impact access.

Create external catalog-based materialized views

Creating a materialized view on tables in external catalogs is similar to creating a materialized view on StarRiocks' native tables. You only need to set a suitable refresh strategy in accordance with the data source you are using, and manually enable query rewrite for external catalog-based materialized views.

Choose a suitable refresh strategy

Currently, StarRocks cannot detect partition-level data changes in Hudi catalogs, Iceberg catalogs, and JDBC catalogs. Therefore, a full-size refresh is performed once the task is triggered.

For Hive catalogs, you can enable the Hive metadata cache refresh feature to allow StarRocks to detect data changes at the partition level. Please note that the partitioning keys of the materialized view must be included in that of the base table. When this feature is enabled, StarRocks periodically accesses the Hive Metastore Service (HMS) or AWS Glue to check the metadata information of recently queried hot data. As a result, StarRocks can:

- Refresh only the partitions with data changes to avoid full-size refresh, reducing resource consumption caused by refresh.
- Ensure data consistency to some extent during query rewrite. If there are data changes in the base table in the data lake, the query will not be rewritten to use the materialized view.

> **NOTE**
>
> You can still choose to tolerate a certain level of data inconsistency by setting the property `mv_rewrite_staleness_second` when creating the materialized view. For more information, see CREATE MATERIALIZED VIEW.

To enable the Hive metadata cache refresh feature, you can set the following FE dynamic configuration item using ADMIN SET FRONTEND CONFIG:

Configuration item	Default	Descri
enable_background_refresh_connector_metadata	true in v3.0 false in v2.5	Whethe enable periodi Hive metada cache refresh After it enable StarRo polls th metast (Hive Metast or AWS Glue) o your Hi cluster, refresh the cac metada

Configuration item	Default	Descri
		the frequer access Hive catalog perceiv data change true indicat enable Hive metada cache refresh false indicat disable
background_refresh_metadata_interval_millis	600000 (10 minutes)	The int betwee two consec Hive metada cache refresh Unit: millisec
background_refresh_metadata_time_secs_since_last_access_secs	86400 (24 hours)	The expirat time of Hive metada cache refresh

Configuration item	Default	Descri
		task. F the Hiv catalog has be access it has n been access for mor than th specifi time, StarRo stops refresh its cach metada For the catalog has not been access StarRo will not refresh cachec metada Unit: seconc

Enable query rewrite for external catalog-based materialized views

By default, StarRocks does not support query rewrite for materialized views built on Hudi, Iceberg, and JDBC catalogs because query rewrite in this scenario cannot ensure a strong consistency of results. You can enable this feature by setting the property

`force_external_table_query_rewrite` to `true` when creating the materialized view. For materialized views built on tables in Hive catalogs, the query rewrite is enabled by default.

Example:

```
CREATE MATERIALIZED VIEW ex_mv_par_tbl
PARTITION BY emp_date
DISTRIBUTED BY hash(empid)
PROPERTIES (
"force_external_table_query_rewrite" = "true"
)
AS
select empid, deptno, emp_date
from `hive_catalog`.`emp_db`.`emps_par_tbl`
where empid < 5;
```

In scenarios involving query rewriting, if you use a very complex query statement to build a materialized view, we recommend that you split the query statement and construct multiple simple materialized views in a nested manner. Nested materialized views are more versatile and can accommodate a broader range of query patterns.

Best practices

In real-world business scenarios, you can identify queries with high execution latency and resource consumption by analyzing audit logs. You can further use query profiles to pinpoint the specific stages where the query is slow. The following sections provide instructions and examples on how to boost data lake query performance with materialized views.

Case One: Accelerate join calculation in data lake

You can use materialized views to accelerate join queries in the data lake.

Suppose the following queries on your Hive catalog are particularly slow:

```
--Q1
SELECT SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates
```

```
WHERE
    lo_orderdate = d_datekey
    AND d_year = 1993
    AND lo_discount BETWEEN 1 AND 3
    AND lo_quantity < 25;

--Q2
SELECT SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates
WHERE
    lo_orderdate = d_datekey
    AND d_yearmonth = 'Jan1994'
    AND lo_discount BETWEEN 4 AND 6
    AND lo_quantity BETWEEN 26 AND 35;

--Q3
SELECT SUM(lo_revenue), d_year, p_brand
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates,
hive.ssb_1g_csv.part, hive.ssb_1g_csv.supplier
WHERE
    lo_orderdate = d_datekey
    AND lo_partkey = p_partkey
    AND lo_suppkey = s_suppkey
    AND p_brand BETWEEN 'MFGR#2221' AND 'MFGR#2228'
    AND s_region = 'ASIA'
GROUP BY d_year, p_brand
ORDER BY d_year, p_brand;
```

By analyzing their query profiles, you may notice that the query execution time is mostly spent on the hash join between the table `lineorder` and the other dimension tables on the column `lo_orderdate`.

Here, Q1 and Q2 perform aggregation after joining `lineorder` and `dates`, while Q3 performs aggregation after joining `lineorder`, `dates`, `part`, and `supplier`.

Therefore, you can utilize the View Delta Join rewrite capability of StarRocks to build a materialized view that joins `lineorder`, `dates`, `part`, and `supplier`.

```
CREATE MATERIALIZED VIEW lineorder_flat_mv
DISTRIBUTED BY HASH(LO_ORDERDATE, LO_ORDERKEY) BUCKETS 48
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
```

```
    -- Specify the unique constraints.
    "unique_constraints" = "
    hive.ssb_1g_csv.supplier.s_suppkey;
    hive.ssb_1g_csv.part.p_partkey;
    hive.ssb_1g_csv.dates.d_datekey",
    -- Specify the Foreign Keys.
    "foreign_key_constraints" = "
    hive.ssb_1g_csv.lineorder(lo_partkey) REFERENCES
hive.ssb_1g_csv.part(p_partkey);
    hive.ssb_1g_csv.lineorder(lo_suppkey) REFERENCES
hive.ssb_1g_csv.supplier(s_suppkey);
    hive.ssb_1g_csv.lineorder(lo_orderdate) REFERENCES
hive.ssb_1g_csv.dates(d_datekey)",
    -- Enable query rewrite for the external catalog-based
materialized view.
    "force_external_table_query_rewrite" = "TRUE"
)
AS SELECT
       l.LO_ORDERDATE AS LO_ORDERDATE,
       l.LO_ORDERKEY AS LO_ORDERKEY,
       l.LO_PARTKEY AS LO_PARTKEY,
       l.LO_SUPPKEY AS LO_SUPPKEY,
       l.LO_QUANTITY AS LO_QUANTITY,
       l.LO_EXTENDEDPRICE AS LO_EXTENDEDPRICE,
       l.LO_DISCOUNT AS LO_DISCOUNT,
       l.LO_REVENUE AS LO_REVENUE,
       s.S_REGION AS S_REGION,
       p.P_BRAND AS P_BRAND,
       d.D_YEAR AS D_YEAR,
       d.D_YEARMONTH AS D_YEARMONTH
   FROM hive.ssb_1g_csv.lineorder AS l
            INNER JOIN hive.ssb_1g_csv.supplier AS s ON s.S_SUPPKEY =
l.LO_SUPPKEY
            INNER JOIN hive.ssb_1g_csv.part AS p ON p.P_PARTKEY =
l.LO_PARTKEY
            INNER JOIN hive.ssb_1g_csv.dates AS d ON l.LO_ORDERDATE =
d.D_DATEKEY;
```

Case Two: Accelerate aggregations and aggregations on joins in data lake

Materialized views can be used to accelerate aggregation queries, whether they are on a single table or involve multiple tables.

- Single-table aggregation query

  For typical queries on a single table, their query profile will reveal that the AGGREGATE node consumes a significant amount of time. You can use common aggregation operators to construct materialized views.

  Suppose the following is a slow query:

```
--Q4
SELECT
lo_orderdate, count(distinct lo_orderkey)
FROM hive.ssb_1g_csv.lineorder
GROUP BY lo_orderdate
ORDER BY lo_orderdate limit 100;
```

  Q4 calculates the daily count of unique orders. Because the count distinct calculation can be computationally expensive, you can create the following two types of materialized views to accelerate:

```
CREATE MATERIALIZED VIEW mv_2_1
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT
lo_orderdate, count(distinct lo_orderkey)
FROM hive.ssb_1g_csv.lineorder
GROUP BY lo_orderdate;

CREATE MATERIALIZED VIEW mv_2_2
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT
-- lo_orderkey must be the BIGINT type so that it can be used for
query rewrite.
lo_orderdate, bitmap_union(to_bitmap(lo_orderkey))
FROM hive.ssb_1g_csv.lineorder
GROUP BY lo_orderdate;
```

Please note that, in this context, do not create materialized views with LIMIT and ORDER BY clauses to avoid rewrite failures. For more information on the query rewrite limitations, see Query rewrite with materialized views - Limitations.

- Multi-table aggregation query

  In scenarios involving aggregations of join results, you can create nested materialized views on existing materialized views that join the tables to further aggregate the join results. For example, based on the example in Case One, you can create the following materialized view to accelerate Q1 and Q2 because their aggregation patterns are similar:

```
CREATE MATERIALIZED VIEW mv_2_3
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT
lo_orderdate, lo_discount, lo_quantity, d_year, d_yearmonth,
SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM lineorder_flat_mv
GROUP BY lo_orderdate, lo_discount, lo_quantity, d_year,
d_yearmonth;
```

  Of course, it is possible to perform both join and aggregation calculations within a single materialized view. While these types of materialized views may have fewer opportunities for query rewrite (due to their specific calculations), they typically occupy less storage space after aggregation. Your choice can be based on your specific use case.

```
CREATE MATERIALIZED VIEW mv_2_4
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    "force_external_table_query_rewrite" = "TRUE"
)
AS
SELECT lo_orderdate, lo_discount, lo_quantity, d_year,
d_yearmonth, SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates
WHERE lo_orderdate = d_datekey
```

```
GROUP BY lo_orderdate, lo_discount, lo_quantity, d_year,
d_yearmonth;
```

Case Three: Accelerate joins on aggregations in data lake

In some scenarios, it may be necessary to perform aggregation calculations on one table first and then perform join queries with other tables. To fully use StarRocks' query rewrite capabilities, we recommend constructing nested materialized views. For example:

```
--Q5
SELECT * FROM  (
    SELECT
      l.lo_orderkey, l.lo_orderdate, c.c_custkey, c_region,
sum(l.lo_revenue)
    FROM
      hive.ssb_1g_csv.lineorder l
      INNER JOIN (
        SELECT distinct c_custkey, c_region
        from
          hive.ssb_1g_csv.customer
        WHERE
          c_region IN ('ASIA', 'AMERICA')
      ) c ON l.lo_custkey = c.c_custkey
      GROUP BY  l.lo_orderkey, l.lo_orderdate, c.c_custkey, c_region
  ) c1
WHERE
  lo_orderdate = '19970503';
```

Q5 first performs an aggregation query on the `customer` table and then performs a join and aggregation with the `lineorder` table. Similar queries might involve different filters on `c_region` and `lo_orderdate`. To leverage query rewrite capabilities, you can create two materialized views, one for aggregation and another for the join.

```
--mv_3_1
CREATE MATERIALIZED VIEW mv_3_1
DISTRIBUTED BY HASH(c_custkey)
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    "force_external_table_query_rewrite" = "TRUE"
)
AS
```

```
SELECT distinct c_custkey, c_region from hive.ssb_1g_csv.customer;

--mv_3_2
CREATE MATERIALIZED VIEW mv_3_2
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    "force_external_table_query_rewrite" = "TRUE"
)
AS
SELECT l.lo_orderdate, l.lo_orderkey, mv.c_custkey, mv.c_region,
sum(l.lo_revenue)
FROM hive.ssb_1g_csv.lineorder l
INNER JOIN mv_3_1 mv
ON l.lo_custkey = mv.c_custkey
GROUP BY l.lo_orderkey, l.lo_orderdate, mv.c_custkey, mv.c_region;
```

Case Four: Separate hot and cold data for real-time data and historical data in data lake

Consider the following scenario: data updated within the past three days is directly written into StarRocks, while less recent data is checked and batch-written into Hive. However, there are still queries that may involve data from the past seven days. In this case, you can create a simple model with materialized views to expire the data automatically.

```
CREATE MATERIALIZED VIEW mv_4_1
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT lo_orderkey, lo_orderdate, lo_revenue
FROM hive.ssb_1g_csv.lineorder
WHERE lo_orderdate<=current_date()
AND lo_orderdate>=date_add(current_date(), INTERVAL -4 DAY);
```

You can further build views or materialized views upon it based on the logic of the upper-layer applications.

Usage

- CREATE
- ALTER
- DROP
- SHOW

CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW

Description

Creates a materialized view. For usage information about materialized views, see Synchronous materialized view and Asynchronous materialized view.

> **CAUTION**
>
> Only users with the CREATE MATERIALIZED VIEW privilege in the database where the base table resides can create a materialized view.

Creating a materialized view is an asynchronous operation. Running this command successfully indicates that the task of creating the materialized view is submitted successfully. You can view the building status of a synchronous materialized view in a database via SHOW ALTER MATERIALIZED VIEW command, and view that of an asynchronous materialized view by querying the metadata tables `tasks` and `task_runs` in Information Schema.

StarRocks supports asynchronous materialized views from v2.4. The major differences between asynchronous materialized views and synchronous materialized views in previous versions are as follows:

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
ASYNC MV	Yes	Yes	Yes	• Asynchronous refresh • Manual refresh	Multiple tables from: • Default catalog

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
					• External catalogs (v2.5) • Existing materialized views (v2.5)
SYNC MV (Rollup)	Limited choices of aggregate functions	No	Yes	Synchronous refresh during data loading	Single table in the default catalog

Synchronous materialized view

Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [database.]<mv_name>
[COMMENT ""]
[PROPERTIES ("key"="value", ...)]
AS
<query_statement>
```

Parameters in brackets [] are optional.

Parameters

mv_name (required)

The name of the materialized view. The naming requirements are as follows:

- The name must consist of letters (a-z or A-Z), digits (0-9), or underscores (_), and it can only start with a letter.

- The length of the name cannot exceed 64 characters.
- The name is case-sensitive.

COMMENT (optional)

Comment on the materialized view. Note that `COMMENT` must be placed after `mv_name`. Otherwise, the materialized view cannot be created.

query_statement (required)

The query statement to create the materialized view. Its result is the data in the materialized view. The syntax is as follows:

```
SELECT select_expr[, select_expr ...]
[GROUP BY column_name[, column_name ...]]
[ORDER BY column_name[, column_name ...]]
```

- select_expr (required)

  All columns in the query statement, that is, all columns in the materialized view schema. This parameter supports the following values:

  - Simple columns or aggregate columns such as `SELECT a, abs(b), min(c) FROM table_a`, where `a`, `b`, and `c` are the names of columns in the base table. If you do not specify column names for the materialized view, StarRocks automatically assigns names to the columns.
  - Expressions such as `SELECT a+1 AS x, b+2 AS y, c*c AS z FROM table_a`, where `a+1`, `b+2` and `c*c` are the expressions that reference the columns in the base tables, and `x`, `y` and `z` are the aliases assigned to the columns in the materialized view.

  > **NOTE**
  >
  > - You must specify at least one column in `select_expr`.
  > - Synchronous materialized views only support aggregate functions on a single column. Query statements in the form of `sum(a+b)` are not supported.
  > - When creating a synchronous materialized view with an aggregate function, you must specify the GROUP BY clause, and specify at least one GROUP BY column in `select_expr`.

> - Synchronous materialized views do not support clauses such as JOIN, WHERE, and the HAVING clause of GROUP BY.
> - When using ALTER TABLE DROP COLUMN to drop a specific column in a base table, you must ensure that all synchronous materialized views of the base table do not contain the dropped column, otherwise the drop operation will fail. Before you drop the column, you must first drop all synchronous materialized views that contain the column.
> - Creating too many synchronous materialized views for a table will affect the data load efficiency. When data is being loaded to the base table, the data in synchronous materialized view and base table will be updated synchronously. If a base table contains `n` synchronous materialized views, the efficiency of loading data into the base table is about the same as the efficiency of loading data into `n` tables.

- GROUP BY (optional)

  The GROUP BY column of the query. If this parameter is not specified, the data will not be grouped by default.

- ORDER BY (optional)

  The ORDER BY column of the query.

  - Columns in the ORDER BY clause must be declared in the same order as the columns in `select_expr`.
  - If the query statement contains a GROUP BY clause, the ORDER BY columns must be identical to the GROUP BY columns.
  - If this parameter is not specified, the system will automatically supplement the ORDER BY column according to the following rules:
    - If the materialized view is the AGGREGATE type, all GROUP BY columns are automatically used as sort keys.
    - If the materialized view is not the AGGREGATE type, StarRocks automatically selects sort keys based on the prefix columns.

Automatic query rewrite with synchronous materialized view

When a query that follows the pattern of a synchronous materialized view is executed, the original query statement is automatically rewritten and the intermediate results stored in the

materialized view are used.

The following table shows the correspondence between the aggregate function in the original query and the aggregate function used to construct the materialized view. You can select the corresponding aggregate function to build a materialized view according to your business scenario.

aggregate function in the original query	aggregate function of the materialized view
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

Asynchronous materialized view

Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [database.]<mv_name>
[COMMENT ""]
-- You must specify either `distribution_desc` or `refresh_scheme`,
or both.
-- distribution_desc
DISTRIBUTED BY HASH(<bucket_key>[,<bucket_key2> ...]) [BUCKETS
<bucket_number>]
-- refresh_scheme
```

```
[REFRESH
    [ASYNC | ASYNC [START (<start_time>)] EVERY (INTERVAL
<refresh_interval>) | MANUAL]
]
-- partition_expression
[PARTITION BY
    {<date_column> | date_trunc(fmt, <date_column>)}
]
-- order_by_expression
[ORDER BY (<sort_key>)]
[PROPERTIES ("key"="value", ...)]
AS
<query_statement>
```

Parameters in brackets [] are optional.

Parameters

mv_name (required)

The name of the materialized view. The naming requirements are as follows:

- The name must consist of letters (a-z or A-Z), digits (0-9), or underscores (_), and it can only start with a letter.
- The length of the name cannot exceed 64 characters.
- The name is case-sensitive.

> **CAUTION**
>
> Multiple materialized views can be created on the same base table, but the names of the materialized views in the same database cannot be duplicated.

COMMENT (optional)

Comment on the materialized view. Note that `COMMENT` must be placed after `mv_name`. Otherwise, the materialized view cannot be created.

distribution_desc (reuqired)

The bucketing strategy of the asynchronous materialized view.

Syntax

```
DISTRIBUTED BY HASH (<bucket_key1>[,<bucket_key2> ...]) [BUCKETS
<bucket_number>]
```

For more information, see Data distribution.

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

refresh_scheme (optional)

> **NOTE**
>
> While creating an asynchronous materialized view, you must specify either `distribution_desc` or `refresh_scheme`, or both.

The refresh strategy of the asynchronous materialized view. Valid values:

- `ASYNC`: Automatic refresh mode. Each time the base table data changes, the materialized view is automatically refreshed.
- `ASYNC [START (<start_time>)] EVERY(INTERVAL <interval>)`: Regular refresh mode. The materialized view is refreshed regularly at the interval defined. You can specify the interval as `EVERY (interval n day/hour/minute/second)` using the following units: `DAY`, `HOUR`, `MINUTE`, and `SECOND`. The default value is `10 MINUTE`. You can further specify the refresh start time as `START('yyyy-MM-dd hh:mm:ss')`. If the start time is not specified, the current time is used. Example: `ASYNC START ('2023-09-12 16:30:25') EVERY (INTERVAL 5 MINUTE)`.
- `MANUAL`: Manual refresh mode. The materialized view will not be refreshed unless you trigger a refresh task manually.

If this parameter is not specified, the default value `MANUAL` is used.

partition_expression (optional)

The partitioning strategy of the asynchronous materialized view. As for the current version of StarRocks, only one partition expression is supported when creating an asynchronous materialized view.

> **CAUTION**
>
> Currently, asynchronous materialized views do not support the list partitioning strategy.

Valid values:

- `column_name`: The name of the column used for partitioning. The expression `PARTITION BY dt` means to partition the materialized view according to the `dt` column.
- date_trunc function: The function used to truncate the time unit. `PARTITION BY date_trunc("MONTH", dt)` means that the `dt` column is truncated to month as the unit for partitioning. The date_trunc function supports truncating time to units including `YEAR`, `MONTH`, `DAY`, `HOUR`, and `MINUTE`.

If this parameter is not specified, no partitioning strategy is adopted by default.

PROPERTIES (optional)

Properties of the asynchronous materialized view. You can modify the properties of an existing materialized view using ALTER MATERIALIZED VIEW.

- `replication_num`: The number of materialized view replicas to create.
- `storage_medium`: Storage medium type. Valid values: `HDD` and `SSD`.
- `storage_cooldown_time`: the storage cooldown time for a partition. If both HDD and SSD storage mediums are used, data in the SSD storage is moved to the HDD storage after the time specified by this property. Format: "yyyy-MM-dd HH:mm:ss". The specified time must be later than the current time. If this property is not explicitly specified, the storage cooldown is not performed by default.
- `partition_ttl_number`: The number of most recent materialized view partitions to keep. For the partitions with a start time earlier than the current time, after the number of these partitions exceeds this value, less recent partitions will be deleted. StarRocks will periodically check materialized view partitions according to the time interval specified in the FE configuration item `dynamic_partition_check_interval_seconds`, and automatically delete expired partitions. If you enabled the dynamic partitioning strategy, the partitions created in advance are not counted in. When the value is `-1`, all partitions of the materialized view will be preserved. Default: `-1`.
- `partition_refresh_number`: In a single refresh, the maximum number of partitions to refresh. If the number of partitions to be refreshed exceeds this value, StarRocks will

split the refresh task and complete it in batches. Only when the previous batch of partitions is refreshed successfully, StarRocks will continue to refresh the next batch of partitions until all partitions are refreshed. If any of the partitions fail to be refreshed, no subsequent refresh tasks will be generated. When the value is `-1`, the refresh task will not be split. Default: `-1`.

- `excluded_trigger_tables`: If a base table of the materialized view is listed here, the automatic refresh task will not be triggered when the data in the base table is changed. This parameter only applies to load-triggered refresh strategy, and is usually used together with the property `auto_refresh_partitions_limit`. Format: `[db_name.]table_name`. When the value is an empty string, any data change in all base tables triggers the refresh of the corresponding materialized view. The default value is an empty string.
- `auto_refresh_partitions_limit`: The number of most recent materialized view partitions that need to be refreshed when a materialized view refresh is triggered. You can use this property to limit the refresh range and reduce the refresh cost. However, because not all the partitions are refreshed, the data in the materialized view may not be consistent with the base table. Default: `-1`. When the value is `-1`, all partitions will be refreshed. When the value is a positive integer N, StarRocks sorts the existing partitions in chronological order, and refreshes the current partition and N-1 most recent partitions. If the number of partitions is less than N, StarRocks refreshes all existing partitions. If there are dynamic partitions created in advance in your materialized view, StarRocks refreshes all pre-created partitions.

> **CAUTION**
>
> The Unique Key constraints and Foreign Key constraints are only used for query rewrite. The Foreign Key constraint checks are not guaranteed when data is loaded into the table. You must ensure the data loaded into the table meets the constraints.

query_statement (required)

The query statement to create the asynchronous materialized view.

> **CAUTION**
>
> Currently, StarRocks does not support creating asynchronous materialized views with base tables created with the list partitioning strategy.

Query an asynchronous materialized view

An asynchronous materialized view is a physical table. You can operate it as any regular table **except that you cannot directly load data into an asynchronous materialized view**.

Automatic query rewrite with asynchronous materialized view

StarRocks v2.5 supports automatic and transparent query rewrite based on the SPJG-type asynchronous materialized views. The SPJG-type materialized views refer to materialized views whose plan only includes Scan, Filter, Project, and Aggregate types of operators. The SPJG-type materialized views query rewrite includes single table query rewrite, Join query rewrite, aggregation query rewrite, Union query rewrite and query rewrite based on nested materialized views.

See Asynchronous materialized view - Rewrite queries with the asynchronous materialized view for further information.

Supported data types

- Asynchronous materialized views created based on the StarRocks default catalog support the following data types:
    - **Date**: DATE, DATETIME
    - **String**: CHAR, VARCHAR
    - **Numeric**: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, PERCENTILE
    - **Semi-structured**: ARRAY, JSON
    - **Other**: BITMAP, HLL

> **NOTE**
>
> BITMAP, HLL, and PERCENTILE have been supported since v2.4.5.

- Asynchronous materialized views created based on the StarRocks external catalogs support the following data types:
    - 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

Usage notes

- The current version of StarRocks does not support creating multiple materialized views at the same time. A new materialized view can only be created when the one before is completed.
- About synchronous materialized views:
    - Synchronous materialized views only support aggregate functions on a single column. Query statements in the form of `sum(a+b)` are not supported.
    - Synchronous materialized views support only one aggregate function for each column of the base table. Query statements such as `select sum(a), min(a) from table` are not supported.
    - When creating a synchronous materialized view with an aggregate function, you must specify the GROUP BY clause, and specify at least one GROUP BY column in SELECT.
    - Synchronous materialized views do not support clauses such as JOIN, WHERE, and the HAVING clause of GROUP BY.

    - When using ALTER TABLE DROP COLUMN to drop a specific column in a base table, you must ensure that all synchronous materialized views of the base table do not contain the dropped column, otherwise, the drop operation will fail. Before you drop the column, you must first drop all synchronous materialized views that contain the column.
    - Creating too many synchronous materialized views for a table will affect the data load efficiency. When data is being loaded to the base table, the data in the synchronous materialized view and the base table will be updated synchronously. If a base table contains `n` synchronous materialized views, the efficiency of loading data into the base table is about the same as the efficiency of loading data into `n` tables.

- About nested asynchronous materialized views:

    - The refresh strategy for each materialized view only applies to the corresponding materialized view.
    - Currently, StarRocks does not limit the number of nesting levels. In a production environment, we recommend that the number of nesting layers not exceed THREE.

- About external catalog asynchronous materialized views:

    - External catalog materialized view only support async fixed-interval refresh and manual refresh.
    - Strict consistency is not guaranteed between the materialized view and the base tables in the external catalog.
    - Currently, building materialized views based on external resources is not supported.
    - Currently, StarRocks cannot perceive if the base table data in the external catalog has changed, so all partitions will be refreshed by default every time the base table is refreshed. You can manually refresh only some of partitions using REFRESH MATERIALIZED VIEW.

Examples

Examples of synchronous materialized views

The schema of the base table is as follows:

```
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     |       |
+-------+--------+------+------+---------+-------+
```

Example 1: Create a synchronous materialized view that only contains the columns of the original table (k1, k2).

```
create materialized view k1_k2 as
select k1, k2 from duplicate_table;
```

The materialized view contains only two columns k1 and k2 without any aggregation.

```
+----------------+-------+--------+------+------+---------+-------+
| IndexName      | Field | Type   | Null | Key  | Default | Extra |
+----------------+-------+--------+------+------+---------+-------+
| k1_k2          | k1    | INT    | Yes  | true | N/A     |       |
|                | k2    | INT    | Yes  | true | N/A     |       |
+----------------+-------+--------+------+------+---------+-------+
```

Example 2: Create a synchronous materialized view sorted by k2.

```
create materialized view k2_order as
select k2, k1 from duplicate_table order by k2;
```

The materialized view's schema is shown below. The materialized view contains only two columns k2 and k1, where column k2 is a sort column without any aggregation.

```
+----------------+-------+--------+------+-------+---------+-------+
| IndexName      | Field | Type   | Null | Key   | Default | Extra |
+----------------+-------+--------+------+-------+---------+-------+
| k2_order       | k2    | INT    | Yes  | true  | N/A     |       |
```

```
|                 | k1    | INT    | Yes   | false | N/A     | NONE  |
+-----------------+-------+--------+-------+-------+---------+-------+
```

Example 3: Create a synchronous materialized view grouped by k1 and k2, and a SUM aggregation on k3.

```
create materialized view k1_k2_sumk3 as
select k1, k2, sum(k3) from duplicate_table group by k1, k2;
```

The materialized view's schema is shown below. The materialized view contains three columns k1, k2 and sum (k3), where k1, k2 are grouped columns, and sum (k3) is the sum of the k3 columns grouped according to k1 and 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   |
+-----------------+-------+--------+------+-------+---------+-------+
```

Because the materialized view does not declare a sort column, and it adopts an aggregation function, StarRocks supplements the grouped columns k1 and k2 by default.

Example 4: Create a synchronous materialized view to remove duplicate rows.

```
create materialized view deduplicate as
select k1, k2, k3, k4 from duplicate_table group by k1, k2, k3, k4;
```

The materialized view's schema is shown below. The materialized view contains k1, k2, k3, and k4 columns, and there are no duplicate rows.

```
+-----------------+-------+--------+------+-------+---------+-------+
| 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     |       |
+-----------------+-------+--------+------+-------+---------+-------+
```

Example 5: Create a non-aggregated synchronous materialized view that does not declare a sort column.

The schema of the base table is shown below:

```
+-------+--------------+------+-------+---------+-------+
| 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  |
+-------+--------------+------+-------+---------+-------+
```

The materialized view contains k3, k4, k5, k6, and k7 columns, and no sort column is declared. Create the materialized view with the following statement:

```
create materialized view mv_1 as
select k3, k4, k5, k6, k7 from all_type_table;
```

StarRocks automatically uses k3, k4, and k5 as the sort columns by default. The sum of the bytes occupied by these three column types is 4 (INT) + 8 (BIGINT) + 16 (DECIMAL) = 28 < 36. So these three columns are added as sort columns.

The materialized view's schema is as follows.

```
+---------------+-------+-------------+------+-------+---------+---
----+
| 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  |
+----------------+-------+--------------+------+-------+---------+---
----+
```

It can be observed that the `key` field of the k3, k4, and k5 columns is `true`, which indicates that they are the sort keys. The key field of the k6, and k7 columns is `false`, which indicates that they are not the sort keys.

Examples of asynchronous materialized views

The following examples are based on the base tables below:

```
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"
);
```

Example 1: Create a non-partitioned materialized view.

```
CREATE MATERIALIZED VIEW lo_mv1
DISTRIBUTED BY HASH(`lo_orderkey`)
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
order by lo_orderkey;
```

Example 2: Create a partitioned materialized view.

```
CREATE MATERIALIZED VIEW lo_mv2
PARTITION BY `lo_orderdate`
DISTRIBUTED BY HASH(`lo_orderkey`)
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
order by lo_orderkey;

-- Use the date_trunc() function to partition the materialized view
by month.
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;
```

Example 3: Create an asynchronous materialized view.

```
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;
```

ALTER MATERIALIZED VIEW

ALTER MATERIALIZED VIEW

Description

This SQL statement can：

- Alter the name of an asynchronous materialized view.
- Alter the refresh strategy of an asynchronous materialized view.
- Alter the status of an asynchronous materialized view to active or inactive.
- Alter the properties of an asynchronous materialized view.

  You can use this SQL statement to alter the following properties:

  - `partition_ttl_number`
  - `partition_refresh_number`
  - `resource_group`
  - `auto_refresh_partitions_limit`
  - `excluded_trigger_tables`
  - `mv_rewrite_staleness_second`
  - `unique_constraints`
  - `foreign_key_constraints`
  - `colocate_with`
  - All session variable-related properties. For information on session variables, see System variables。

Syntax

```
ALTER MATERIALIZED VIEW [db_name.]<mv_name>
    { RENAME [db_name.]<new_mv_name>
    | REFRESH <new_refresh_scheme_desc>
    | ACTIVE | INACTIVE
    | SET ( "<key>" = "<value>"[,...]) }
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
mv_name	yes	The name of the materialized view to alter.
new_refresh_scheme_desc	no	New async refresh strategy, see SQL Reference - CREATE MATERIALIZED VIEW - Parameters for details.
new_mv_name	no	New name for the materialized view.
ACTIVE	no	Set the status of the materialized view to active. StarRocks automatically sets a materialized view to inactive if any of its base tables is changed, for example, dropped and re-created, to prevent the situation that original metadata mismatches the changed base table. Inactive materialized views cannot be used for query acceleration or query rewrite. You can use this SQL to activate the materialized view after changing the base tables.
INACTIVE	no	Set the status of the materialized view to inactive. An inactive asynchronous materialized view cannot be refreshed. But you can still query it as a table.

Parameter	Required	Description
key	no	The name of the property to alter, see SQL Reference - CREATE MATERIALIZED VIEW - Parameters for details. **NOTE** If you want to alter a session variable-related property of the materialized view, you must add a `session.` prefix to the property, for example, `session.query_timeout`. You do not need to specify the prefix for non-session properties, for example, `mv_rewrite_staleness_second`.
value	no	The value of the property to alter.

Example

Example 1: Alter the name of the materialized view

```
ALTER MATERIALIZED VIEW lo_mv1 RENAME lo_mv1_new_name;
```

Example 2: Alter the refresh interval of the materialized view

```
ALTER MATERIALIZED VIEW lo_mv2 REFRESH ASYNC EVERY(INTERVAL 1 DAY);
```

Example 3: Alter the materialized view's properties.

```
-- Change mv1's query_timeout to 40000 seconds.
ALTER MATERIALIZED VIEW mv1 SET ("session.query_timeout" = "40000");
-- Change mv1's mv_rewrite_staleness_second to 600 seconds.
ALTER MATERIALIZED VIEW mv1 SET ("mv_rewrite_staleness_second" =
"600");
```

Example 4: Alter the materialized view's status to active.

```
ALTER MATERIALIZED VIEW order_mv ACTIVE;
```

DROP MATERIALIZED VIEW

DROP MATERIALIZED VIEW

Description

Drops a materialized view.

You cannot drop a synchronous materialized view that is being created in process with this command. To drop a synchronous materialized view that is being created in process, see Synchronous materialized View - Drop an unfinished materialized view for further instructions.

> **CAUTION**
>
> Only users with the `DROP_PRIV` privilege in the database where the base table resides can drop a materialized view.

Syntax

```
DROP MATERIALIZED VIEW [IF EXISTS] [database.]mv_name
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
IF EXISTS	no	If this parameter is specified, StarRocks will not throw an exception when deleting a materialized view that does not exist. If this parameter is not specified, the system will throw

Parameter	Required	Description
		an exception when deleting a materialized view that does not exist.
mv_name	yes	The name of the materialized view to delete.

Examples

Example 1: Drop an existing materialized view

1. View all existing materialized views in the database.

```
MySQL > SHOW MATERIALIZED VIEW\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. Drop the materialized view `order_mv1`.

```
DROP MATERIALIZED VIEW order_mv1;
```

3. Check if the dropped materialized view exists.

```
MySQL > SHOW MATERIALIZED VIEW;
Empty set (0.01 sec)
```

Example 2: Drop a non-existing materialized view

- If the parameter `IF EXISTS` is specified, StarRocks will not throw an exception when deleting a materialized view that does not exist.

```
MySQL > DROP MATERIALIZED VIEW IF EXISTS k1_k2;
Query OK, 0 rows affected (0.00 sec)
```

- If the parameter `IF EXISTS` is not specified, the system will throw an exception when deleting a materialized view that does not exist.

```
MySQL > DROP MATERIALIZED VIEW k1_k2;
ERROR 1064 (HY000): Materialized view k1_k2 is not find
```

SHOW MATERIALIZED VIEW

SHOW MATERIALIZED VIEW

Description

Shows all or one specific asynchronous materialized view.

Syntax

```
SHOW MATERIALIZED VIEW
[FROM db_name]
[
WHERE
[NAME { = "mv_name" | LIKE "mv_name_matcher"}
]
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
db_name	no	The name of the database to which the materialized view resides. If this parameter is not specified, the current database is used by default.
mv_name	no	The name of the materialized view to show.
mv_name_matcher	no	The matcher used to filter materialized views.

Returns

Return	Description
id	The ID of the materialized view.
database_name	The name of the database in which the materialized view resides.
name	The name of the materialized view.
rows	The number of data rows in the materialized view.
text	The statement used to create the materialized view.

Examples

```
MySQL > SHOW MATERIALIZED VIEW\G
*************************** 1. row ***************************
           id: 10099
         name: order_mv
database_name: test
         text: CREATE MATERIALIZED VIEW `order_mv` (`order_id`,
`total`)
DISTRIBUTED BY HASH(`order_id`)
REFRESH ASYNC START("2022-09-01 10:00:00") EVERY(INTERVAL 1 DAY)
PROPERTIES (
"replication_num" = "3",
"storage_medium" = "HDD"
)
AS SELECT `order_list`.`order_id`, sum(`goods`.`price`) AS `total`
FROM `wlc`.`order_list` INNER JOIN `wlc`.`goods` ON
`goods`.`item_id1` = `order_list`.`item_id2`
GROUP BY `order_list`.`order_id`;
         rows: 3
1 row in set (0.02 sec)
```

Colocate Join

For shuffle join and broadcast join, if the join condition is met, the data rows of the two joining tables are merged into a single node to complete the join. Neither of these two join methods can avoid latency or overhead caused by data network transmission between nodes.

The core idea is to keep bucketing key, number of copies, and copy placement consistent for tables in the same Colocation Group. If the join column is a bucketing key, the computing node only needs to do local join without getting data from other nodes. Colocate Join supports equi joins.

This document introduces the principle, implementation, usage, and considerations of Colocate Join.

Terminology

- **Colocation Group (CG)**: A CG will contain one or more Tables. The Tables within a CG have the same bucketing and replica placement, and are described using the Colocation Group Schema.
- **Colocation Group Schema (CGS)**: A CGS contains the bucketing key, number of buckets, and number of replicas of a CG.

Principle

Colocate Join is to form a CG with a set of Tables having the same CGS, and ensure that the corresponding bucket copies of these Tables will fall on the same set of BE nodes. When the tables in the CG perform Join operations on the bucketed columns, the local data can be joined directly, saving time from transferring data between nodes.

Bucket Seq is obtained by `hash(key) mod buckets`. Suppose a Table has 8 buckets, then there are [0, 1, 2, 3, 4, 5, 6, 7] 8 buckets, and each Bucket has one or more sub-tables, the number of sub-tables depends on the number of partitions. If it is a multi-partitioned table, there will be multiple tablets.

In order to have the same data distribution, tables within the same CG must comply with the following.

1. Tables within the same CG must have the identical bucketing key (type, number, order) and the same number of buckets so that the data slices of multiple tables can be distributed and controlled one by one. The bucketing key is the columns specified in the table creation statement `DISTRIBUTED BY HASH(col1, col2, ...)`. The bucketing key determines which columns of data are Hashed into different Bucket Seqs. The name of the bucketing key can vary for tables within the same CG.The bucketing columns can be different in the creation statement, but the order of the corresponding data types in `DISTRIBUTED BY HASH(col1, col2, ...)` should be exactly the same .
2. Tables within the same CG must have the same number of partition copies. If not, it may happen that a tablet copy has no corresponding copy in the partition of the same BE.
3. Tables within the same CG may have different numbers of partitions and different partition keys.

When creating a table, the CG is specified by the attribute `"colocate_with" = "group_name"` in the table PROPERTIES. If the CG does not exist, it means the table is the first table of the CG and called Parent Table. The data distribution of the Parent Table (type, number and order of split bucket keys, number of copies and number of split buckets) determines the CGS. If the CG exists, check whether the data distribution of the table is consistent with the CGS.

The copy placement of tables within the same CG satisfies:

1. The mapping between the Bucket Seq and BE nodes of all the Tables is the same as that of the Parent Table.
2. The mapping between the Bucket Seq and BE nodes of all the Partitions in the Parent Table is the same as that of the first Partition.
3. The mapping between the Bucket Seq and BE nodes of the first Partition of the Parent Table is determined using the native Round Robin algorithm.

The consistent data distribution and mapping guarantee that the data rows with the same value taken by bucketing key fall on the same BE. Therefore, when using the bucketing key to join columns, only local joins are required.

Usage

Table creation

When creating a table, you can specify the attribute `"colocate_with" = "group_name"` in PROPERTIES to indicate that the table is a Colocate Join table and belongs to a specified Colocation Group.

> **NOTE**
>
> From version 2.5.4, Colocate Join can be performed on tables from different databases. You only need to specify the same `colocate_with` property when you create tables.

For example:

```
CREATE TABLE tbl (k1 int, v1 int sum)
DISTRIBUTED BY HASH(k1)
BUCKETS 8
PROPERTIES(
    "colocate_with" = "group1"
);
```

If the specified Group does not exist, StarRocks automatically creates a Group that only contains the current table. If the Group exists, StarRocks checks to see if the current table meets the Colocation Group Schema. If so, it creates the table and adds it to the Group. At the same time, the table creates a partition and a tablet based on the data distribution rules of the existing Group.

A Colocation Group belongs to a database. The name of a Colocation Group is unique within a database. In the internal storage, the full name of the Colocation Group is `dbId_groupName`, but you only perceive `groupName`.

> **NOTE**
>
> If you specify the same Colocation Group to associate tables from different databases for Colocate Join, the Colocation Group exists in each of these databases. You can run `show proc "/colocation_group"` to check the Colocation Groups in different databases.

Delete

A complete deletion is a deletion from the Recycle Bin. Normally, after a table is deleted with the `DROP TABLE` command, by default it will stay in the recycle bin for a day before being deleted). When the last table in a Group is completely deleted, the Group will also be deleted automatically.

View group information

The following command allows you to view the Group information that already exists in the cluster.

```
SHOW PROC '/colocation_group';

+-------------+--------------+--------------+-----------+----------
-----+----------+----------+
| GroupId     | GroupName    | TableIds     | BucketsNum |
ReplicationNum | DistCols | IsStable |
+-------------+--------------+--------------+-----------+----------
-----+----------+----------+
| 10005.10008 | 10005_group1 | 10007, 10040 | 10         | 3
| int(11)  | true     |
+-------------+--------------+--------------+-----------+----------
-----+----------+----------+
```

- **GroupId**: The cluster-wide unique identifier of a Group, with the first half being the db id and the second half being the group id.
- **GroupName**: The full name of the Group.
- **TabletIds**: The list of ids of the Tables in the Group.
- **BucketsNum**: The number of buckets.
- **ReplicationNum**: The number of replicas.
- **DistCols**: Distribution columns, i.e. bucketing column type.
- **IsStable**: Whether the Group is stable (for the definition of stability, see the section of Colocation Replica Balancing and Repair).

You can further view the data distribution of a Group with the following command.

```
SHOW PROC '/colocation_group/10005.10008';

+-------------+---------------------+
| BucketIndex | BackendIds          |
```

```
+-------------+---------------------+
| 0           | 10004, 10002, 10001 |
| 1           | 10003, 10002, 10004 |
| 2           | 10002, 10004, 10001 |
| 3           | 10003, 10002, 10004 |
| 4           | 10002, 10004, 10003 |
| 5           | 10003, 10002, 10001 |
| 6           | 10003, 10004, 10001 |
| 7           | 10003, 10004, 10002 |
+-------------+---------------------+
```

- **BucketIndex**: Subscript of the sequence of buckets.
- **BackendIds**: The ids of BE nodes where the bucketing data slices t are located.

> Note: The above command requires AMDIN privilege. Regular users cannot access it.

Modifying Table Group Properties

You can modify the Colocation Group properties of a table. For example:

```
ALTER TABLE tbl SET ("colocate_with" = "group2");
```

If the table has not been assigned to a Group before, the command will check the Schema and add the table to the Group (the Group will be created first if it does not exist). If the table has been previously assigned to another Group, the command will remove the table from the original Group and add it to the new Group (the Group will be created first if it does not exist).

You can also remove the Colocation properties of a table with the following command.

```
ALTER TABLE tbl SET ("colocate_with" = "");
```

Other related operations

When adding a partition using `ADD PARTITION` or modifying the number of copies to a table with the Colocation attribute, StarRocks checks if the operation will violate the Colocation Group Schema and rejects it if it does.

Colocation replica balancing and repair

The replicas distribution of a Colocation table needs to follow the distribution rules specified in the Group schema, so it differs from normal sharding in terms of replica repair and balancing.

The Group itself has a `stable` property. When `stable` is `true`, it means that no changes are being made to table slices in the Group and the Colocation feature is working properly. When `stable` is `false`, it means that some table slices in the current Group are being repaired or migrated, and the Colocate Join of the affected tables will degrade to a normal Join.

Replica repair

Replicas can only be stored on the specified BE node. StarRocks will look for the least loaded BE to replace an unavailable BE (e.g.,down, decommission),. After the replacement, all the bucketing data slices on the old BE are repaired. During migration, the Group is marked as **Unstable**.

Replica balancing

StarRocks tries to distribute the Colocation table slices evenly across all BE nodes. Balancing for normal tables is at the replica level, that is, each replica individually finds a lower-load BE node. Balancing for Colocation tables is at the Bucket level, that is, all replicas within a Bucket are migrated together. We use a simple balancing algorithm that distributes `BucketsSequnce` evenly across all BE nodes without considering the actual size of the replicas but only the number of replicas. The exact algorithm can be found in the code comments in `ColocateTableBalancer.java`.

> Note 1: The current Colocation replica balancing and repair algorithm may not work well for StarRocks clusters with heterogeneous deployment. The so-called heterogeneous deployment means that the disk capacity, number of disk, and disk type (SSD and HDD) of BE nodes are not consistent. In the case of heterogeneous deployment, it may happen that a small-capacity BE node stores the same number of replicas as a large-capacity BE node.
>
> Note 2: When a Group is in the Unstable state, the Join of its tables will degrade to a normal Join, which may significantly degrade the query performance of the cluster. If you do not want the system to be automatically balanced, set the FE configuration

> `disable_colocate_balance` to disable automatic balancing and enable it back at the appropriate time. (See the section Advanced Operations (#Advanced Operations) for details)

Query

The Colocation table is queried in the same way as a normal table. If the Group where the Colocation table is located is in Unstable state, it will automatically degrade to a normal Join, as illustrated by the following example.

Table 1:

```
CREATE TABLE `tbl1` (
    `k1` date NOT NULL COMMENT "",
    `k2` int(11) NOT NULL COMMENT "",
    `v1` int(11) SUM NOT NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
PARTITION BY RANGE(`k1`)
(
    PARTITION p1 VALUES LESS THAN ('2019-05-31'),
    PARTITION p2 VALUES LESS THAN ('2019-06-30')
)
DISTRIBUTED BY HASH(`k2`)  BUCKETS 6
PROPERTIES (
    "colocate_with" = "group1"
);
INSERT INTO tbl1
VALUES
    ("2015-09-12",1000,1),
    ("2015-09-13",2000,2);
```

Table 2:

```
CREATE TABLE `tbl2` (
    `k1` datetime NOT NULL COMMENT "",
    `k2` int(11) NOT NULL COMMENT "",
    `v1` double SUM NOT NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
DISTRIBUTED BY HASH(`k2`)  BUCKETS 6
```

```
PROPERTIES (
    "colocate_with" = "group1"
);
INSERT INTO tbl2
VALUES
    ("2015-09-12 00:00:00",3000,3),
    ("2015-09-12 00:00:00",4000,4);
```

View query plan:

```
EXPLAIN SELECT * FROM tbl1 INNER JOIN tbl2 ON (tbl1.k2 = tbl2.k2);
+-----------------------------------------------------------------------
-----+
| Explain String
|
+-----------------------------------------------------------------------
-----+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:1: k1 | 2: k2 | 3: v1 | 4: k1 | 5: k2 | 6: v1
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   3:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 03
|
```

```
|     UNPARTITIONED
|
|
|
|   2:HASH JOIN
|
|   |  join op: INNER JOIN (COLOCATE)
|
|   |  colocate: true
|
|   |  equal join conjunct: 5: k2 = 2: k2
|
|   |
|
|   |----1:OlapScanNode
|
|   |       TABLE: tbl1
|
|   |       PREAGGREGATION: OFF. Reason: Has can not pre-aggregation
Join |
|   |       partitions=1/2
|
|   |       rollup: tbl1
|
|   |       tabletRatio=6/6
|
|   |       tabletList=15344,15346,15348,15350,15352,15354
|
|   |       cardinality=1
|
|   |       avgRowSize=3.0
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: tbl2
|
|      PREAGGREGATION: OFF. Reason: None aggregate function
|
|      partitions=1/1
|
|      rollup: tbl2
|
|      tabletRatio=6/6
|
```

```
|      tabletList=15373,15375,15377,15379,15381,15383
|
|      cardinality=1
|
|      avgRowSize=3.0
|
+-----------------------------------------------------------------------
-----+
40 rows in set (0.03 sec)
```

If a Colocate Join takes effect, the Hash Join node displays `colocate: true`.

If it doesn't take effect, the query plan is as follows:

```
+----------------------------------------------------+
| Explain String                                     |
+----------------------------------------------------+
| PLAN FRAGMENT 0                                    |
|  OUTPUT EXPRS:`tbl1`.`k1` |                        |
|   PARTITION: RANDOM                                |
|                                                    |
|   RESULT SINK                                      |
|                                                    |
|   2:HASH JOIN                                      |
|   |  join op: INNER JOIN (BROADCAST)               |
|   |  hash predicates:                              |
|   |  colocate: false, reason: group is not stable  |
|   |    `tbl1`.`k2` = `tbl2`.`k2`                   |
|   |  tuple ids: 0 1                                |
|   |                                                |
|   |----3:EXCHANGE                                  |
|   |       tuple ids: 1                             |
|   |                                                |
|   0:OlapScanNode                                   |
|      TABLE: tbl1                                   |
|      PREAGGREGATION: OFF. Reason: No AggregateInfo |
|      partitions=0/2                                |
|      rollup: null                                  |
|      buckets=0/0                                   |
|      cardinality=-1                                |
|      avgRowSize=0.0                                |
|      numNodes=0                                    |
|      tuple ids: 0                                  |
|                                                    |
| PLAN FRAGMENT 1                                    |
```

```
|  OUTPUT EXPRS:                                          |
|   PARTITION: RANDOM                                     |
|                                                         |
|   STREAM DATA SINK                                      |
|     EXCHANGE ID: 03                                     |
|     UNPARTITIONED                                       |
|                                                         |
|   1:OlapScanNode                                        |
|      TABLE: tbl2                                        |
|      PREAGGREGATION: OFF. Reason: null                  |
|      partitions=0/1                                     |
|      rollup: null                                       |
|      buckets=0/0                                        |
|      cardinality=-1                                     |
|      avgRowSize=0.0                                     |
|      numNodes=0                                         |
|      tuple ids: 1                                       |
+---------------------------------------------------------+
```

The HASH JOIN node will show the corresponding reason: `colocate: false, reason: group is not stable`. An EXCHANGE node will be generated at the same time.

Advanced operations

FE configuration items

- **disable_colocate_relocate**

Whether to disable automatic Colocation replica repair for StarRocks. The default is false, which means it is turned on. This parameter only affects replica repair for Colocation tables, not for normal tables.

- **disable_colocate_balance**

Whether to disable automatic Colocation replica balancing for StarRocks. The default is false, which means it is turned on. This parameter only affects the replica balancing of Colocation tables, not for normal tables.

- **disable_colocate_join**

  You can disable Colocate join at session granularity by changing this variable.

- **disable_colocate_join**

  The Colocate join function can be disabled by changing this variable.

HTTP Restful API

StarRocks provides several HTTP Restful APIs related to Colocate Join for viewing and modifying Colocation Groups.

This API is implemented on the FE and can be accessed using `fe_host:fe_http_port` with ADMIN permissions.

1. View all Colocation information of a cluster

```
curl --location-trusted -u<username>:<password>
'http://<fe_host>:<fe_http_port>/api/colocate'
```

```
// Returns the internal Colocation information in Json format.
{
    "colocate_meta": {
        "groupName2Id": {
            "g1": {
                "dbId": 10005,
                "grpId": 10008
            }
        },
        "group2Tables": {},
        "table2Group": {
            "10007": {
                "dbId": 10005,
                "grpId": 10008
            },
            "10040": {
                "dbId": 10005,
                "grpId": 10008
            }
        },
        "group2Schema": {
            "10005.10008": {
                "groupId": {
                    "dbId": 10005,
                    "grpId": 10008
```

```
                },
                "distributionColTypes": [{
                    "type": "INT",
                    "len": -1,
                    "isAssignedStrLenInColDefinition": false,
                    "precision": 0,
                    "scale": 0
                }],
                "bucketsNum": 10,
                "replicationNum": 2
            }
        },
        "group2BackendsPerBucketSeq": {
            "10005.10008": [
                [10004, 10002],
                [10003, 10002],
                [10002, 10004],
                [10003, 10002],
                [10002, 10004],
                [10003, 10002],
                [10003, 10004],
                [10003, 10004],
                [10003, 10004],
                [10002, 10004]
            ]
        },
        "unstableGroups": []
    },
    "status": "OK"
}
```


2. Mark the Group as Stable or Unstable

```
# Mark as Stable
curl -XPOST --location-trusted -u<username>:<password>
'http://<fe_host>:<fe_http_port>/api/colocate/group_stable?db_id=
<dbId>&group_id=<grpId>'
# Mark as Unstable
curl -XPOST --location-trusted -u<username>:<password>
'http://<fe_host>:<fe_http_port>/api/colocate/group_unstable?
db_id=<dbId>&group_id=<grpId>'
```

If the returned result is `200`, the Group is successfully marked as Stable or Unstable.

3. Set the data distribution of a Group

   This interface allows you to force the number distribution of a Group.

   `POST /api/colocate/bucketseq?db_id=10005&group_id= 10008`

   `Body:`

   `[[10004,10002],[10003,10002],[10002,10004],[10003,10002], [10002,10004],[10003,10002],[10003,10004],[10003,10004], [10003,10004],[10002,10004]]`

   `returns: 200`

   Where `Body` is `BucketsSequence` represented as a nested array and the ids of the BEs in which bucketing slices are located.

   > Note that to use this command, you may need to set the FE configuration `disable_colocate_relocate` and `disable_colocate_balance` to true, that is, to disable the system to perform automatic Colocation replica repair and balancing. Otherwise, it may be automatically reset by the system after modification.

Use Lateral Join for column-to-row conversion

Column-to-row conversion is a common operation in ETL processing. Lateral is a special Join keyword that can associate a row with an internal subquery or table function. By using Lateral in conjunction with unnest(), you can expand one row into multiple rows. For more information, see unnest.

Limits

- Currently, Lateral Join is only used with unnest() to achieve column-to-row conversion. Other table functions and UDTFs will be supported later.
- Currently, Lateral Join does not support subqueries.

Use Lateral Join

Syntax:

```
from table_reference join [lateral] table_reference;
```

Examples:

```
SELECT student, score
FROM tests
CROSS JOIN LATERAL UNNEST(scores) AS t (score);

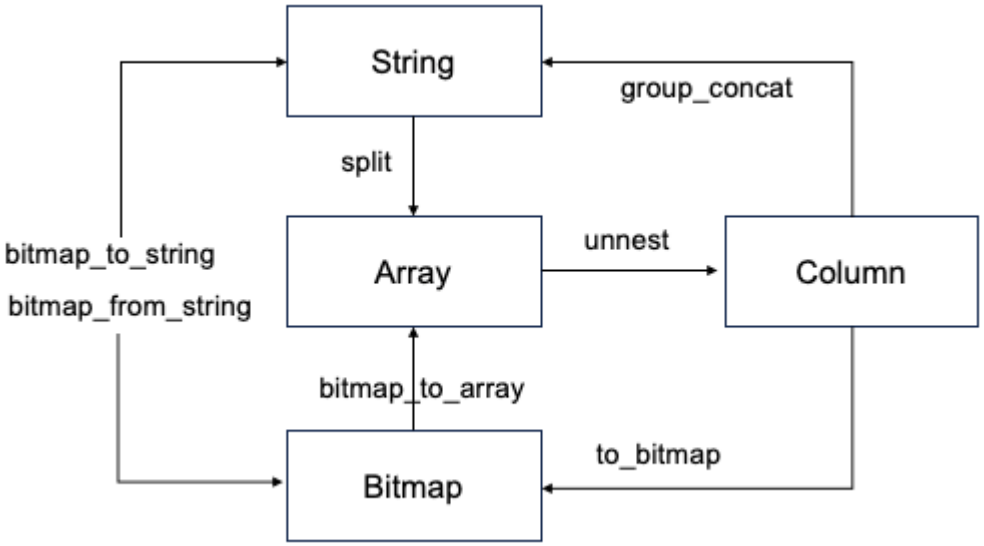
SELECT student, score
FROM tests, UNNEST(scores) AS t (score);
```

The second syntax here is a shortened version of the first one, where the Lateral keyword can be omitted using the UNNEST keyword. The UNNEST keyword is a table function that

converts an array into multiple rows. Together with Lateral Join, it can implement common row expansion logic.

> **NOTE**
>
> If you want to perform unnest on multiple columns, you must specify an alias for each column, for example, `select v1, t1.unnest as v2, t2.unnest as v3 from lateral_test, unnest(v2) t1, unnest(v3) t2;`.

StarRocks supports type conversion among BITMAP, STRING, ARRAY, and Column.



Usage examples

Together with unnest(), you can achieve the following column-to-row conversion features:

Expand a string into multiple rows

1. Create a table and insert data into this table.

```
CREATE TABLE lateral_test2 (
    `v1` bigint(20) NULL COMMENT "",
    `v2` string NULL COMMENT ""
)
DUPLICATE KEY(v1)
DISTRIBUTED BY HASH(`v1`) BUCKETS 1
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
```

```
);

INSERT INTO lateral_test2 VALUES (1, "1,2,3"), (2, "1,3");
```

2. Query data before expansion.

```
select * from lateral_test2;

+------+-------+
| v1   | v2    |
+------+-------+
|    1 | 1,2,3 |
|    2 | 1,3   |
+------+-------+
```

3. Expand `v2` into multiple rows.

```
-- Perform unnest on a single column.

select v1,unnest from lateral_test2, unnest(split(v2, ",")) as
unnest;

+------+--------+
| v1   | unnest |
+------+--------+
|    1 | 1      |
|    1 | 2      |
|    1 | 3      |
|    2 | 1      |
|    2 | 3      |
+------+--------+

-- Perform unnest on multiple columns. You must specify an alias
for each operation.

select v1, t1.unnest as v2, t2.unnest as v3 from lateral_test2,
unnest(split(v2, ",")) t1, unnest(split(v3, ",")) t2;

+------+------+------+
| v1   | v2   | v3   |
+------+------+------+
|    1 | 1    | 1    |
|    1 | 1    | 2    |
```

```
|    1 | 2    | 1    |
|    1 | 2    | 2    |
|    1 | 3    | 1    |
|    1 | 3    | 2    |
|    2 | 1    | 1    |
|    2 | 1    | 3    |
|    2 | 3    | 1    |
|    2 | 3    | 3    |
+------+------+------+
```

Expand an array into multiple rows

From v2.5, unnest() can take multiple arrays of different types and lengths. For more information, see unnest().

1. Create a table and insert data into this table.

```
CREATE TABLE lateral_test (
    `v1` bigint(20) NULL COMMENT "",
    `v2` ARRAY NULL COMMENT ""
)
DUPLICATE KEY(v1)
DISTRIBUTED BY HASH(`v1`) BUCKETS 1
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);

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

2. Query data before expansion.

```
select * from lateral_test;

+------+------------+
| v1   | v2         |
+------+------------+
|    1 | [1,2]      |
|    2 | [1,null,3] |
```

```
|    3 | NULL       |
+------+------------+
```

3. Expand `v2` into multiple rows.

```
select v1,v2,unnest from lateral_test , unnest(v2) as unnest;

+------+-----------+--------+
| v1   | v2        | unnest |
+------+-----------+--------+
|    1 | [1,2]     |      1 |
|    1 | [1,2]     |      2 |
|    2 | [1,null,3] |      1 |
|    2 | [1,null,3] |   NULL |
|    2 | [1,null,3] |      3 |
+------+-----------+--------+
```

Expand Bitmap data

1. Create a table and insert data into this table.

```
CREATE TABLE lateral_test3 (
`v1` bigint(20) NULL COMMENT "",
`v2` Bitmap BITMAP_UNION COMMENT ""
)
AGGREGATE KEY(v1)
DISTRIBUTED BY HASH(`v1`) BUCKETS 1;

INSERT INTO lateral_test3 VALUES (1, bitmap_from_string('1, 2')),
(2, to_bitmap(3));
```

2. Query data before expansion.

```
select v1, bitmap_to_string(v2) from lateral_test3;

+------+-----------------------+
| v1   | bitmap_to_string(`v2`) |
+------+-----------------------+
|    1 | 1,2                   |
|    2 | 3                     |
```

```
+------+-----------------------+
```

3. Insert a new row.

```
insert into lateral_test3 values (1, to_bitmap(3));

select v1, bitmap_to_string(v2) from lateral_test3;

+------+-----------------------+
| v1   | bitmap_to_string(`v2`) |
+------+-----------------------+
|    1 | 1,2,3                 |
|    2 | 3                     |
+------+-----------------------+
```

4. Expand data in `v2` into multiple rows.

```
select v1,unnest from lateral_test3, unnest(bitmap_to_array(v2))
as unnest;

+------+--------+
| v1   | unnest |
+------+--------+
|    1 |      1 |
|    1 |      2 |
|    1 |      3 |
|    2 |      3 |
+------+--------+
```

Query Cache

The query cache is a powerful feature of StarRocks that can greatly enhance the performance of aggregation queries. By storing the intermediate results of local aggregations in memory, the query cache can avoid unnecessary disk access and computation for new queries that are identical or similar to previous ones. With its query cache, StarRocks can deliver fast and accurate results for aggregation queries, saving time and resources and enabling better scalability. The query cache is especially useful for high-concurrency scenarios where many users run similar queries on large and complex data sets.

This feature is supported since v2.5.

Application scenarios

We recommend that you use the query cache in the following scenarios:

- You frequently run aggregate queries on individual flat tables.
- Most of your aggregate queries are non-GROUP BY aggregate queries and low-cardinality GROUP BY aggregate queries.
- Your data is loaded in append mode by time partition and can be categorized as hot data and cold data based on access frequency.

The query cache supports queries that meet the following conditions:

- The query engine is Pipeline. To enable the Pipeline engine, set the session variable `enable_pipeline_engine` to `true`.

  > **NOTE**
  >
  > Other query engines do not support the query cache.

- The queries are on native OLAP tables. The query cache does not support queries on external tables or lake tables. The query cache also supports queries whose plans

require access to synchronous materialized views. However, the query cache does not support queries whose plans require access to asynchronous materialized views.

- The queries are aggregate queries on individual flat tables.

  NOTE

  > The query cache will support aggregate queries on multiple tables that are joined by using colocate joins, broadcast joins, or bucket shuffle joins in the future.

- The queries do not include nondeterminstic functions such as `rand`, `random`, `uuid`, and `sleep`.

The query cache supports queries on tables that use any of the following partition policies: Unpartitioned, Multi-Column Partitioned, and Single-Column Partitioned.

Feature boundaries

- The query cache is based on per-tablet computations of the Pipeline engine. Per-tablet computation means that a pipeline driver can process entire tablets one by one rather than processing a portion of a tablet or many tablets interleaved together. If the number of tablets that need to be processed by each individual BE for a query is greater than or equal to the number of pipeline drivers that are invoked to run this query, the query cache works. The number of pipeline drivers invoked represents the actual degree of parallelism (DOP). If the number of tablets is smaller than the number of pipeline drivers, each pipeline driver processes only a portion of a specific tablet. In this situation, per-tablet computation results cannot be produced, and therefore the query cache does not work.
- In StarRocks, an aggregate query consists of at least four stages. Per-Tablet computation results generated by AggregateNode in the first stage can be cached only when OlapScanNode and AggregateNode compute data from the same fragment. Per-Tablet computation results generated by AggregateNode in the other stages cannot be cached. For some DISTINCT aggregate queries, if the session variable `cbo_cte_reuse` is set to `true`, the query cache does not work when OlapScanNode, which produces data, and the stage-1 AggregateNode, which consumes the produced data, compute data from different fragments and are bridged by an ExchangeNode. The following two examples show scenarios in which CTE optimizations are performed and therefore the query cache does not work:

    - The output columns are computed by using the aggregate function `avg(distinct)`.
    - The output columns are computed by multiple DISTINCT aggregate functions.
- The query cache occupies a small amount of memory provided by the BE to save computation results. The size of the query cache defaults to 512 MB. Therefore, it is unsuitable for the query cache to save large-sized data items. Additionally, after you enable the query cache, query performance is decreased if the cache hit ratio is low. Therefore, if the size of computation results generated for a tablet exceeds the threshold specified by the `query_cache_entry_max_bytes` or `query_cache_entry_max_rows` parameter, the query cache no longer works for the query and the query is switched to Passthrough mode.

Enable query cache

This section describes the parameters and session variables that are used to enable and configure the query cache.

FE session variables

Variable	Default value	Can be dynamically configured	Description
enable_query_cache	false	Yes	Specifies whether to enabl query cache. Valid values: and `false`. `true` specifie enable this feature, and `fa` specifies to disable this fea When the query cache is e it works only for queries tha the conditions specified in "Application scenarios" sec this topic.
query_cache_entry_max_bytes	4194304	Yes	Specifies the threshold for triggering the Passthrough Valid values: `0` to

Variable	Default value	Can be dynamically configured	Description
			`9223372036854775807`. V the number of bytes or row the computation results of specific tablet accessed by query exceeds the thresho specified by the `query_cache_entry_max` or `query_cache_entry_max` parameter, the query is swi to Passthrough mode. If the `query_cache_entry_max` or `query_cache_entry_max` parameter is set to `0`, the Passthrough mode is used when no computation resul generated from the involve tablets.
query_cache_entry_max_rows	409600	Yes	Same as above.

BE parameters

You need to configure the following parameter in the BE configuration file **be.conf**. After you reconfigure this parameter for a BE, you must restart the BE to make the new parameter setting take effect.

Parameter	Required	Description
query_cache_capacity	No	Specifies the size of the query cache. Unit: bytes. The default size is 512 MB. Each BE has its own local query cache in

Parameter	Required	Description
		memory, and it populates and probes only its own query cache. Note that the query cache size cannot be less than 4 MB. If the memory capacity of the BE is insufficient to provision your expected query cache size, you can increase the memory capacity of the BE.

Engineered for maximum cache hit rate in all scenarios

Consider three scenarios where the query cache is still effective even when the queries are not identical literally. These three scenarios are:

- Semantically equivalent queries
- Queries with overlapping scanned partitions
- Queries against data with append-only data changes (no UPDATE or DELETE operations)

Semantically equivalent queries

When two queries are similar, which does not mean that they must be literally equivalent but means that they contain semantically equivalent snippets in their execution plans, they are considered semantically equivalent and can reuse each other's computation results. In a broad sense, two queries are semantically equivalent if they query data from the same source, use the same computation method, and have the same execution plan. StarRocks applies the following rules to evaluate whether two queries are semantically equivalent:

- If the two queries contain multiple aggregations, they are evaluated as semantically equivalent as long as their first aggregations are semantically equivalent. For example, the following two queries, Q1 and Q2, both contain multiple aggregations, but their first aggregations are semantically equivalent. Therefore, Q1 and Q2 are evaluated as semantically equivalent.
    - Q1

```
SELECT
    (
        ifnull(sum(murmur_hash3_32(hour)), 0) +
ifnull(sum(murmur_hash3_32(k0)), 0) +
ifnull(sum(murmur_hash3_32(__c_0)), 0)
    ) AS fingerprint
FROM
    (
        SELECT
            date_trunc('hour', ts) AS hour,
            k0,
            sum(v1) AS __c_0
        FROM
            t0
        WHERE
            ts between '2022-01-03 00:00:00'
            and '2022-01-03 23:59:59'
        GROUP BY
            date_trunc('hour', ts),
            k0
    ) AS t;
```

- Q2

```
SELECT
    date_trunc('hour', ts) AS hour,
    k0,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    k0
```

- If the two queries both belong to one of the following query types, they can be evaluated as semantically equivalent. Note that queries that include a HAVING clause cannot be evaluated as semantically equivalent to queries that do not include a HAVING clause. However, the inclusion of an ORDER BY or LIMIT clause does not affect the evaluation of whether two queries are semantically equivalent.

- GROUP BY aggregations

```
SELECT <GroupByItems>, <AggFunctionItems>
FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
GROUP BY <GroupByItems>
[HAVING <HavingPredicate>]
```

> **NOTE**
>
> In the preceding example, the HAVING clause is optional.

- GROUP BY DISTINCT aggregations

```
SELECT DISTINCT <GroupByItems>, <Items>
FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
GROUP BY <GroupByItems>
HAVING <HavingPredicate>
```

> **NOTE**
>
> In the preceding example, the HAVING clause is optional.

- Non-GROUP BY aggregations

```
SELECT <AggFunctionItems> FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
```

- Non-GROUP BY DISTINCT aggregations

```
SELECT DISTINCT <Items> FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
```

- If either query includes `PartitionColumnRangePredicate`, `PartitionColumnRangePredicate` is removed before the two queries are evaluated for semantic equivalence. `PartitionColumnRangePredicate` specifies one of the following types of predicates that reference a partitioning column:

    - `col between v1 and v2`: The values of the partitioning column fall within the [v1, v2] range, in which `v1` and `v2` are constant expressions.
    - `v1 < col and col < v2`: The values of the partitioning column fall within the (v1, v2) range, in which `v1` and `v2` are constant expressions.
    - `v1 < col and col <= v2`: The values of the partitioning column fall within the (v1, v2] range, in which `v1` and `v2` are constant expressions.
    - `v1 <= col and col < v2`: The values of the partitioning column fall within the [v1, v2) range, in which `v1` and `v2` are constant expressions.
    - `v1 <= col and col <= v2`: The values of the partitioning column fall within the [v1, v2] range, in which `v1` and `v2` are constant expressions.

- If the output columns of the SELECT clauses of the two queries are the same after they are rearranged, the two queries are evaluated as semantically equivalent.

- If the output columns of the GROUP BY clauses of the two queries are the same after they are rearranged, the two queries are evaluated as semantically equivalent.

- If the remaining predicates of the WHERE clauses of the two queries are semantically equivalent after `PartitionColumnRangePredicate` is removed, the two queries are evaluated as semantically equivalent.

- If the predicates in the HAVING clauses of the two queries are semantically equivalent, the two queries are evaluated as semantically equivalent.

Use the following table `lineorder_flat` as an example:

```
CREATE TABLE `lineorder_flat`
(
    `lo_orderdate` date NOT NULL COMMENT "",
    `lo_orderkey` int(11) NOT NULL COMMENT "",
    `lo_linenumber` tinyint(4) 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_orderpriority` varchar(100) NOT NULL COMMENT "",
    `lo_shippriority` tinyint(4) NOT NULL COMMENT "",
    `lo_quantity` tinyint(4) NOT NULL COMMENT "",
    `lo_extendedprice` int(11) NOT NULL COMMENT "",
    `lo_ordtotalprice` int(11) NOT NULL COMMENT "",
    `lo_discount` tinyint(4) NOT NULL COMMENT "",
    `lo_revenue` int(11) NOT NULL COMMENT "",
```

```
    `lo_supplycost` int(11) NOT NULL COMMENT "",
    `lo_tax` tinyint(4) NOT NULL COMMENT "",
    `lo_commitdate` date NOT NULL COMMENT "",
    `lo_shipmode` varchar(100) NOT NULL COMMENT "",
    `c_name` varchar(100) NOT NULL COMMENT "",
    `c_address` varchar(100) NOT NULL COMMENT "",
    `c_city` varchar(100) NOT NULL COMMENT "",
    `c_nation` varchar(100) NOT NULL COMMENT "",
    `c_region` varchar(100) NOT NULL COMMENT "",
    `c_phone` varchar(100) NOT NULL COMMENT "",
    `c_mktsegment` varchar(100) NOT NULL COMMENT "",
    `s_name` varchar(100) NOT NULL COMMENT "",
    `s_address` varchar(100) NOT NULL COMMENT "",
    `s_city` varchar(100) NOT NULL COMMENT "",
    `s_nation` varchar(100) NOT NULL COMMENT "",
    `s_region` varchar(100) NOT NULL COMMENT "",
    `s_phone` varchar(100) NOT NULL COMMENT "",
    `p_name` varchar(100) NOT NULL COMMENT "",
    `p_mfgr` varchar(100) NOT NULL COMMENT "",
    `p_category` varchar(100) NOT NULL COMMENT "",
    `p_brand` varchar(100) NOT NULL COMMENT "",
    `p_color` varchar(100) NOT NULL COMMENT "",
    `p_type` varchar(100) NOT NULL COMMENT "",
    `p_size` tinyint(4) NOT NULL COMMENT "",
    `p_container` varchar(100) NOT NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`lo_orderdate`, `lo_orderkey`)
COMMENT "olap"
PARTITION BY RANGE(`lo_orderdate`)
(PARTITION p1 VALUES [('0000-01-01'), ('1993-01-01')),
PARTITION p2 VALUES [('1993-01-01'), ('1994-01-01')),
PARTITION p3 VALUES [('1994-01-01'), ('1995-01-01')),
PARTITION p4 VALUES [('1995-01-01'), ('1996-01-01')),
PARTITION p5 VALUES [('1996-01-01'), ('1997-01-01')),
PARTITION p6 VALUES [('1997-01-01'), ('1998-01-01')),
PARTITION p7 VALUES [('1998-01-01'), ('1999-01-01')))
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 48
PROPERTIES
(
    "replication_num" = "3",
    "colocate_with" = "groupxx1",
    "in_memory" = "false",
    "storage_format" = "DEFAULT",
    "enable_persistent_index" = "false",
```

```
    "compression" = "LZ4"
);
```

The following two queries, Q1 and Q2, on the table `lineorder_flat` are semantically equivalent after they are processed as follows:

1. Rearrange the output columns of the SELECT statement.
2. Rearrange the output columns of the GROUP BY clause.
3. Remove the output columns of the ORDER BY clause.
4. Rearrange the predicates in the WHERE clause.
5. Add `PartitionColumnRangePredicate`.

- Q1

```
SELECT sum(lo_revenue)), year(lo_orderdate) AS year,p_brand
FROM lineorder_flat
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY year,p_brand
ORDER BY year,p_brand;
```

- Q2

```
SELECT year(lo_orderdate) AS year, p_brand, sum(lo_revenue))
FROM lineorder_flat
WHERE s_region = 'AMERICA' AND p_category = 'MFGR#12' AND
   lo_orderdate >= '1993-01-01' AND lo_orderdate <= '1993-12-31'
GROUP BY p_brand, year(lo_orderdate)
```

Semantic equivalence is evaluated based on the physical plans of queries. Therefore, literal differences in queries do not impact the evaluation for semantic equivalence. Additionally, constant expressions are removed from queries, and `cast` expressions are removed during query optimizations. Therefore, these expressions do not impact the evaluation for semantic equivalence. Thirdly, the aliases of columns and relations do not impact the evaluation for semantic equivalence either.

Queries with overlapping scanned partitions

Query Cache supports predicate-based query splitting.

Splitting queries based on predicate semantics help implement reuse of partial computation results. When a query contains a predicate that references the partitioning column of a table and the predicate specifies a value range, StarRocks can split the range into multiple intervals based on table partitioning. The computation results from each individual interval can be separately reused by other queries.

Use the following table `t0` as an example:

```
CREATE TABLE if not exists t0
(
    ts DATETIME NOT NULL,
    k0 VARCHAR(10) NOT NULL,
    k1 BIGINT NOT NULL,
    v1 DECIMAL64(7, 2) NOT NULL
)
ENGINE=OLAP
DUPLICATE KEY(`ts`, `k0`, `k1`)
COMMENT "OLAP"
PARTITION BY RANGE(ts)
(
  START ("2022-01-01 00:00:00") END ("2022-02-01 00:00:00") EVERY
(INTERVAL 1 day)
)
DISTRIBUTED BY HASH(`ts`, `k0`, `k1`) BUCKETS 1
PROPERTIES
(
    "replication_num" = "3",
    "in_memory" = "false",
    "storage_format" = "default"
);
```

The table `t0` is partitioned by day, and the column `ts` is the table's partitioning column. Among the following four queries, Q2, Q3, and Q4 can reuse portions of the computation results cached for Q1:

- Q1

  ```
  SELECT date_trunc('day', ts) as day, sum(v0)
  FROM t0
  WHERE ts BETWEEN '2022-01-02 12:30:00' AND '2022-01-14 23:59:59'
  GROUP BY day;
  ```

The value range specified by the predicate `ts between '2022-01-02 12:30:00' and '2022-01-14 23:59:59'` of Q1 can be split into the following intervals:

```
1. [2022-01-02 12:30:00, 2022-01-03 00:00:00),
2. [2022-01-03 00:00:00, 2022-01-04 00:00:00),
3. [2022-01-04 00:00:00, 2022-01-05 00:00:00),
...
12. [2022-01-13 00:00:00, 2022-01-14 00:00:00),
13. [2022-01-14 00:00:00, 2022-01-15 00:00:00),
```

- Q2

```
SELECT date_trunc('day', ts) as day, sum(v0)
FROM t0
WHERE ts >= '2022-01-02 12:30:00' AND  ts < '2022-01-05 00:00:00'
GROUP BY day;
```

Q2 can reuse the computation results within the following intervals of Q1:

```
1. [2022-01-02 12:30:00, 2022-01-03 00:00:00),
2. [2022-01-03 00:00:00, 2022-01-04 00:00:00),
3. [2022-01-04 00:00:00, 2022-01-05 00:00:00),
```

- Q3

```
SELECT date_trunc('day', ts) as day, sum(v0)
FROM t0
WHERE ts >= '2022-01-01 12:30:00' AND  ts <= '2022-01-10
12:00:00'
GROUP BY day;
```

Q3 can reuse the computation results within the following intervals of Q1:

```
2. [2022-01-03 00:00:00, 2022-01-04 00:00:00),
3. [2022-01-04 00:00:00, 2022-01-05 00:00:00),
...
8. [2022-01-09 00:00:00, 2022-01-10 00:00:00),
```

- Q4

```
SELECT date_trunc('day', ts) as day, sum(v0)
FROM t0
WHERE ts BETWEEN '2022-01-02 12:30:00' and '2022-01-02 23:59:59'
GROUP BY day;
```

Q4 can reuse the computation results within the following intervals of Q1:

```
1. [2022-01-02 12:30:00, 2022-01-03 00:00:00),
```

The support for reuse of partial computation results varies depending on the partitioning policy used, as described in the following table.

Partitioning policy	Support for reuse of partial computation results
Unpartitioned	Not supported
Multi-Column Partitioned	Not supported **NOTE** This feature may be supported in the future.
Single-Column Partitioned	Supported

Queries against data with append-only data changes

Query Cache supports multi-version caching.

As data loads are made, new versions of tablets are generated. Consequently, the cached computation results that are generated from the previous versions of the tablets become stale and lag behind the latest tablet versions. In this situation, the multi-version caching mechanism tries to merge the stale results saved in the query cache and the incremental versions of the tablets stored on disk into the final results of the tablets so that new queries can carry the latest tablet versions. Multi-version caching is constrained by table types, query types, and data update types.

The support for multi-version caching varies depending on table types and query types, as described in the following table.

Table type	Query type	Support for multi-version caching
Duplicate Key table	• Queries on base tables • Queries on synchronous materialized views	• Queries on base tables: supported in all situations except when incremental tablet versions contain data deletion records. • Queries on synchronous materialized views: supported in all situations except when the GROUP BY, HAVING, or WHERE clauses of queries reference aggregation columns.
Aggregate table	Queries on base tables or queries on synchronous materialized views	Supported in all situations except the following:The schemas of base tables contain the aggregate function `replace`.The GROUP BY, HAVING, or WHERE clauses of queries reference aggregation columns.Incremental tablet versions contain data deletion records.
Unique Key table	N/A	Not supported. However, the query cache is supported.
Primary Key table	N/A	Not supported. However, the query cache is supported.

The impact of data update types on multi-version caching is as follows:

- Data deletions

  Multi-version caching cannot work if incremental versions of tablets contain delete operations.

- Data insertions

  - If an empty version is generated for a tablet, the existing data of the tablet in the query cache remains valid and can still be retrieved.
  - If a non-empty version is generated for a tablet, the existing data of the tablet in the query cache remains valid, but its version lags behind the latest version of the tablet. In this situation, StarRocks reads the incremental data generated from the

version of the existing data to the latest version of the tablet, merges the existing data with the incremental data, and populates the merged data into the query cache.

- Schema changes and tablet truncations

  If the schema of a table is changed or specific tablets of the table are truncated, new tablets are generated for the table. As a result, the existing data of the tablets of the table in the query cache becomes invalid.

Metrics

The profiles of queries for which the query cache works contain `CacheOperator` statistics.

In the source plan of a query, if the pipeline contains `OlapScanOperator`, the names of `OlapScanOperator` and aggregate operators are prefixed with `ML_` to denote that the pipeline uses `MultilaneOperator` to perform per-tablet computations. `CacheOperator` is inserted preceding `ML_CONJUGATE_AGGREGATE` to process the logic that controls how the query cache runs in Passthrough, Populate, and Probe modes. The profile of the query contains the following `CacheOperator` metrics that help you understand the query cache usage.

Metric	Description
CachePassthroughBytes	The number of bytes generated in Passthrough mode.
CachePassthroughChunkNum	The number of chunks generated in Passthrough mode.
CachePassthroughRowNum	The number of rows generated in Passthrough mode.
CachePassthroughTabletNum	The number of tablets generated in Passthrough mode.
CachePassthroughTime:	The amount of computation time taken in Passthrough mode.
CachePopulateBytes	The number of bytes generated in Populate mode.

Metric	Description
CachePopulateChunkNum	The number of chunks generated in Populate mode.
CachePopulateRowNum	The number of rows generated in Populate mode.
CachePopulateTabletNum	The number of tablets generated in Populate mode.
CachePopulateTime	The amount of computation time taken in Populate mode.
CacheProbeBytes	The number of bytes generated for cache hits in Probe mode.
CacheProbeChunkNum	The number of chunks generated for cache hits in Probe mode.
CacheProbeRowNum	The number of rows generated for cache hits in Probe mode.
CacheProbeTabletNum	The number of tablets generated for cache hits in Probe mode.
CacheProbeTime	The amount of computation time taken in Probe mode.

`CachePopulate`*XXX* metrics provide statistics about query cache misses for which the query cache is updated.

`CachePassthrough`*XXX* metrics provide statistics about query cache misses for which the query cache is not updated because the size of per-tablet computation results generated is large.

`CacheProbe`*XXX* metrics provide statistics about query cache hits.

In the multi-version caching mechanism, `CachePopulate` metrics and `CacheProbe` metrics may contain the same tablet statistics, and `CachePassthrough` metrics and `CacheProbe` metrics may also contain the same tablet statistics. For example, when StarRocks computes the data of each tablet, it hits the computation results generated on the historical version of the tablet. In this situation, StarRocks reads the incremental data

generated from the historical version to the latest version of the tablet, computes the data, and merges the incremental data with the cached data. If the size of the computation results generated after the merging does not exceed the threshold specified by the `query_cache_entry_max_bytes` or `query_cache_entry_max_rows` parameter, the statistics of the tablet are collected into `CachePopulate` metrics. Otherwise, the statistics of the tablet are collected into `CachePassthrough` metrics.

RESTful API operations

- `metrics |grep query_cache`

  This API operation is used to query the metrics related to the query cache.

```
curl -s  http://<be_host>:<be_http_port>/metrics |grep
query_cache

# TYPE starrocks_be_query_cache_capacity gauge
starrocks_be_query_cache_capacity 536870912
# TYPE starrocks_be_query_cache_hit_count gauge
starrocks_be_query_cache_hit_count 5084393
# TYPE starrocks_be_query_cache_hit_ratio gauge
starrocks_be_query_cache_hit_ratio 0.984098
# TYPE starrocks_be_query_cache_lookup_count gauge
starrocks_be_query_cache_lookup_count 5166553
# TYPE starrocks_be_query_cache_usage gauge
starrocks_be_query_cache_usage 0
# TYPE starrocks_be_query_cache_usage_ratio gauge
starrocks_be_query_cache_usage_ratio 0.000000
```

- `api/query_cache/stat`

  This API operation is used to query the usage of the query cache.

```
curl  http://<be_host>:<be_http_port>/api/query_cache/stat
{
    "capacity": 536870912,
    "usage": 0,
    "usage_ratio": 0.0,
    "lookup_count": 5025124,
    "hit_count": 4943720,
```

```
    "hit_ratio": 0.983800598751394
}
```

- `api/query_cache/invalidate_all`

  This API operation is used to clear the query cache.

```
curl  -XPUT http://<be_host>:
<be_http_port>/api/query_cache/invalidate_all

{
    "status": "OK"
}
```

The parameters in the preceding API operations are as follows:

- `be_host`: the IP address of the node on which the BE resides.
- `be_http_port`: the HTTP port number of the node on which the BE resides.

Precautions

- StarRocks needs to populate the query cache with the computation results of queries that are initiated for the first time. As a result, the query performance may be slightly lower than expected, and the query latency is increased.
- If you configure a large query cache size, the amount of memory that can be provisioned to query evaluation on the BE is decreased. We recommend that the query cache size do not exceed 1/6 of the memory capacity provisioned to query evaluation.
- If the number of tablets that need to be processed is smaller than the value of `pipeline_dop`, the query cache does not work. To enable the query cache to work, you can set `pipeline_dop` to a smaller value such as `1`.

Examples

Dataset

1. Log in to your StarRocks cluster, go to the destination database, and run the following command to create a table named `t0`:

```
CREATE TABLE t0
(
      `ts` datetime NOT NULL COMMENT "",
      `k0` varchar(10) NOT NULL COMMENT "",
      `k1` char(6) NOT NULL COMMENT "",
      `v0` bigint(20) NOT NULL COMMENT "",
      `v1` decimal64(7, 2) NOT NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`ts`, `k0`, `k1`)
COMMENT "OLAP"
PARTITION BY RANGE(`ts`)
(
    START ("2022-01-01 00:00:00") END ("2022-02-01 00:00:00")
EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(`ts`, `k0`, `k1`) BUCKETS 64
PROPERTIES
(
    "replication_num" = "3",
    "in_memory" = "false",
    "storage_format" = "DEFAULT",
    "enable_persistent_index" = "false"
);
```

2. Insert the following data records into `t0`:

```
INSERT INTO t0
VALUES
    ('2022-01-11 20:42:26', 'n4AGcEqYp', 'hhbawx',
'799393174109549', '8029.42'),
    ('2022-01-27 18:17:59', 'i66lt', 'mtrtzf', '100400167',
'10000.88'),
    ('2022-01-28 20:10:44', 'z6', 'oqkeun', '-58681382337',
'59881.87'),
    ('2022-01-29 14:54:31', 'qQ', 'dzytua', '-19682006834',
'43807.02'),
    ('2022-01-31 08:08:11', 'qQ', 'dzytua',
'79706659299842239 25', '-8947.74'),
    ('2022-01-15 00:40:58', '65', 'hhbawx', '4054945', '156.56'),
    ('2022-01-24 16:17:51', 'onqR3JsK1', 'udtmfp', '-12962',
'-72127.53'),
    ('2022-01-01 22:36:24', 'n4AGcEqYp', 'fabnct', '-50999821',
'17349.85'),
```

```
    ('2022-01-21 08:41:50', 'Nlpz1j3h', 'dzytua', '-60162',
'287.06'),
    ('2022-01-30 23:44:55', '', 'slfght', '62891747919627339',
'8014.98'),
    ('2022-01-18 19:14:28', 'z6', 'dzytua', '-1113001726',
'73258.24'),
    ('2022-01-30 14:54:59', 'z6', 'udtmfp', '111175577438857975',
'-15280.41'),
    ('2022-01-08 22:08:26', 'z6', 'ympyls', '3', '2.07'),
    ('2022-01-03 08:17:29', 'Nlpz1j3h', 'udtmfp', '-234492',
'217.58'),
    ('2022-01-27 07:28:47', 'Pc', 'cawanm', '-1015',
'-20631.50'),
    ('2022-01-17 14:07:47', 'Nlpz1j3h', 'lbsvqu',
'2295574006197343179', '93768.75'),
    ('2022-01-31 14:00:12', 'onqR3JsK1', 'umlkpo', '-227',
'-66199.05'),
    ('2022-01-05 20:31:26', '65', 'lbsvqu', '684307',
'36412.49'),
    ('2022-01-06 00:51:34', 'z6', 'dzytua', '11700309310',
'-26064.10'),
    ('2022-01-26 02:59:00', 'n4AGcEqYp', 'slfght',
'-15320250288446', '-58003.69'),
    ('2022-01-05 03:26:26', 'z6', 'cawanm', '19841055192960542',
'-5634.36'),
    ('2022-01-17 08:51:23', 'Pc', 'ghftus', '35476438804110',
'13625.99'),
    ('2022-01-30 18:56:03', 'n4AGcEqYp', 'lbsvqu',
'3303892099598', '8.37'),
    ('2022-01-22 14:17:18', 'i66lt', 'umlkpo', '-27653110',
'-82306.25'),
    ('2022-01-02 10:25:01', 'qQ', 'ghftus', '-188567166',
'71442.87'),
    ('2022-01-30 04:58:14', 'Pc', 'ympyls', '-9983',
'-82071.59'),
    ('2022-01-05 00:16:56', '7Bh', 'hhbawx', '43712',
'84762.97'),
    ('2022-01-25 03:25:53', '65', 'mtrtzf', '4604107',
'-2434.69'),
    ('2022-01-27 21:09:10', '65', 'udtmfp', '476134823953365199',
'38736.04'),
    ('2022-01-11 13:35:44', '65', 'qmwhvr', '1', '0.28'),
    ('2022-01-03 19:13:07', '', 'lbsvqu', '11', '-53084.04'),
    ('2022-01-20 02:27:25', 'i66lt', 'umlkpo', '3218824416',
'-71393.20'),
    ('2022-01-04 04:52:36', '7Bh', 'ghftus', '-112543071',
```

```
'-78377.93'),
    ('2022-01-27 18:27:06', 'Pc', 'umlkpo', '477', '-98060.13'),
    ('2022-01-04 19:40:36', '', 'udtmfp', '433677211',
'-99829.94'),
    ('2022-01-20 23:19:58', 'Nlpz1j3h', 'udtmfp', '361394977',
'-19284.18'),
    ('2022-01-05 02:17:56', 'Pc', 'oqkeun',
'-552390906075744662', '-25267.92'),
    ('2022-01-02 16:14:07', '65', 'dzytua', '132', '2393.77'),
    ('2022-01-28 23:17:14', 'z6', 'umlkpo', '61', '-52028.57'),
    ('2022-01-12 08:05:44', 'qQ', 'hhbawx', '-9579605666539132',
'-87801.81'),
    ('2022-01-31 19:48:22', 'z6', 'lbsvqu', '9883530877822',
'34006.42'),
    ('2022-01-11 20:38:41', '', 'piszhr', '56108215256366',
'-74059.80'),
    ('2022-01-01 04:15:17', '65', 'cawanm',
'-440061829443010909', '88960.51'),
    ('2022-01-05 07:26:09', 'qQ', 'umlkpo', '-24889917494681901',
'-23372.12'),
    ('2022-01-29 18:13:55', 'Nlpz1j3h', 'cawanm', '-233',
'-24294.42'),
    ('2022-01-10 00:49:45', 'Nlpz1j3h', 'ympyls', '-2396341',
'77723.88'),
    ('2022-01-29 08:02:58', 'n4AGcEqYp', 'slfght', '45212',
'93099.78'),
    ('2022-01-28 08:59:21', 'onqR3JsK1', 'oqkeun', '76',
'-78641.65'),
    ('2022-01-26 14:29:39', '7Bh', 'umlkpo', '176003552517',
'-99999.96'),
    ('2022-01-03 18:53:37', '7Bh', 'piszhr', '3906151622605106',
'55723.01'),
    ('2022-01-04 07:08:19', 'i66lt', 'ympyls',
'-240097380835621', '-81800.87'),
    ('2022-01-28 14:54:17', 'Nlpz1j3h', 'slfght',
'-69018069110121', '90533.64'),
    ('2022-01-22 07:48:53', 'Pc', 'ympyls', '22396835447981344',
'-12583.39'),
    ('2022-01-22 07:39:29', 'Pc', 'uqkghp', '10551305',
'52163.82'),
    ('2022-01-08 22:39:47', 'Nlpz1j3h', 'cawanm', '67905472699',
'87831.30'),
    ('2022-01-05 14:53:34', '7Bh', 'dzytua',
'-779598598706906834', '-38780.41'),
    ('2022-01-30 17:34:41', 'onqR3JsK1', 'oqkeun',
'346687625005524', '-62475.31'),
```

```
    ('2022-01-29 12:14:06', '', 'qmwhvr', '3315', '22076.88'),
    ('2022-01-05 06:47:04', 'Nlpz1j3h', 'udtmfp', '-469',
'42747.17'),
    ('2022-01-19 15:20:20', '7Bh', 'lbsvqu', '347317095885',
'-76393.49'),
    ('2022-01-08 16:18:22', 'z6', 'fghmcd', '2', '90315.60'),
    ('2022-01-02 00:23:06', 'Pc', 'piszhr', '-3651517384168400',
'58220.34'),
    ('2022-01-12 08:23:31', 'onqR3JsK1', 'udtmfp',
'5636394870355729225', '33224.25'),
    ('2022-01-28 10:46:44', 'onqR3JsK1', 'oqkeun',
'-28102078612755', '6469.53'),
    ('2022-01-23 23:16:11', 'onqR3JsK1', 'ghftus',
'-707475035515433949', '63422.66'),
    ('2022-01-03 05:32:31', 'z6', 'hhbawx', '-45', '-49680.52'),
    ('2022-01-27 03:24:33', 'qQ', 'qmwhvr', '3759439060575398 70',
'-66092.96'),
    ('2022-01-25 20:07:22', '7Bh', 'slfght', '1', '72440.21'),
    ('2022-01-04 16:07:24', 'qQ', 'uqkghp', '751213107482249',
'16417.31'),
    ('2022-01-23 19:22:00', 'Pc', 'hhbawx', '-740731249600493',
'88439.40'),
    ('2022-01-05 09:04:20', '7Bh', 'cawanm', '23602', '302.44');
```

Query examples

The statistics of query cache-related metrics in this section are examples and are for reference only.

Query cache works for local aggregations at stage 1

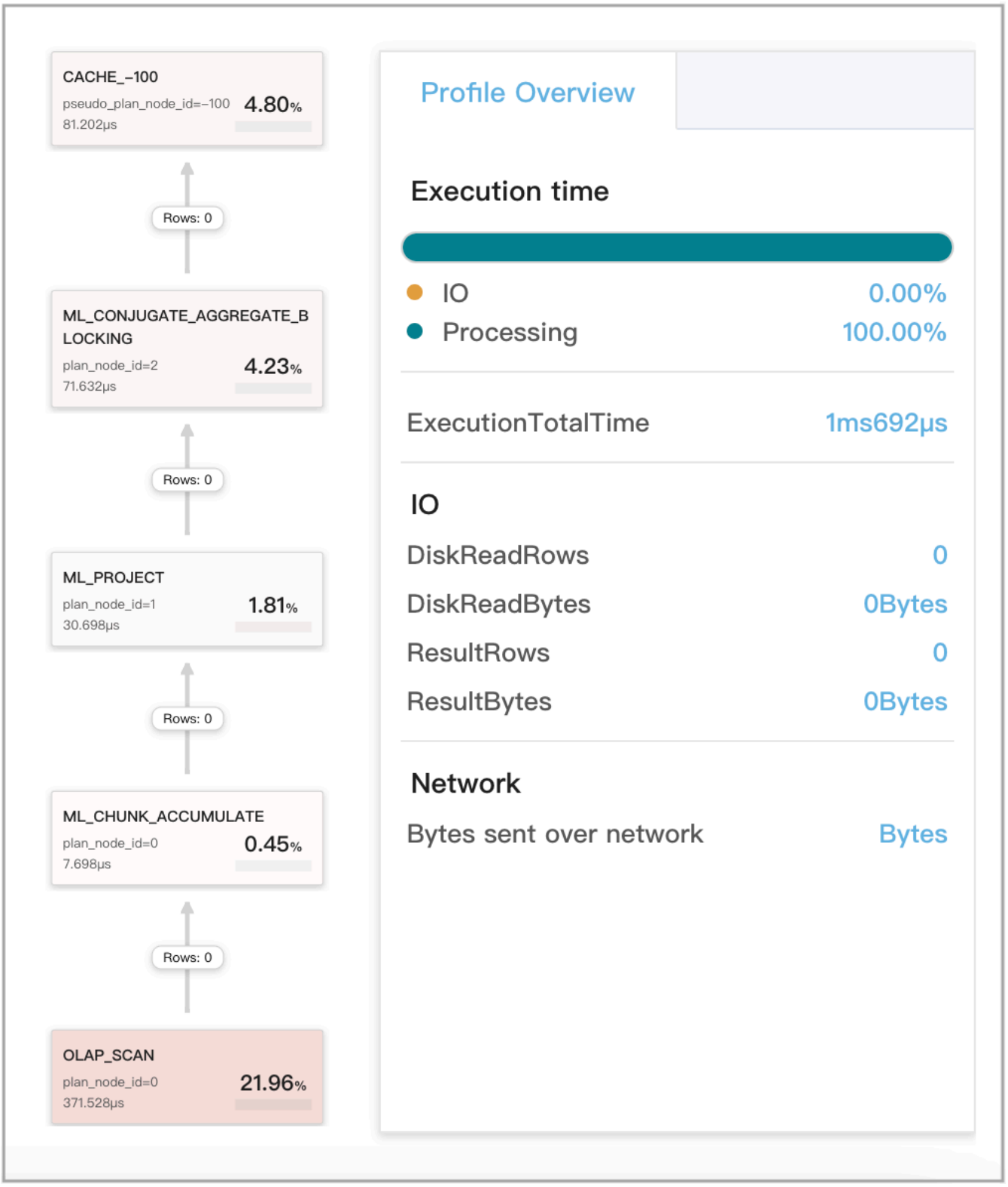
This includes three situations:

- The query accesses only a single tablet.
- The query accesses multiple tablets from multiple partitions of a table that itself comprises a colocated group, and data does not need to be shuffled for aggregations.
- The query accesses multiple tablets from the same partition of a table, and data does not need to be shuffled for aggregations.

Query example:

```
SELECT
    date_trunc('hour', ts) AS hour,
    k0,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    k0
```

The following figure shows the query cache-related metrics in the query profile.



Query cache does not work for remote aggregations at stage 1

When aggregations on multiple tablets are forcibly performed at stage 1, data is first shuffled and then aggregated.

Query example:

```
SET new_planner_agg_stage = 1;

SELECT
    date_trunc('hour', ts) AS hour,
    v0 % 2 AS is_odd,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    is_odd
```

Query cache works for local aggregations at stage 2

This includes three situations:

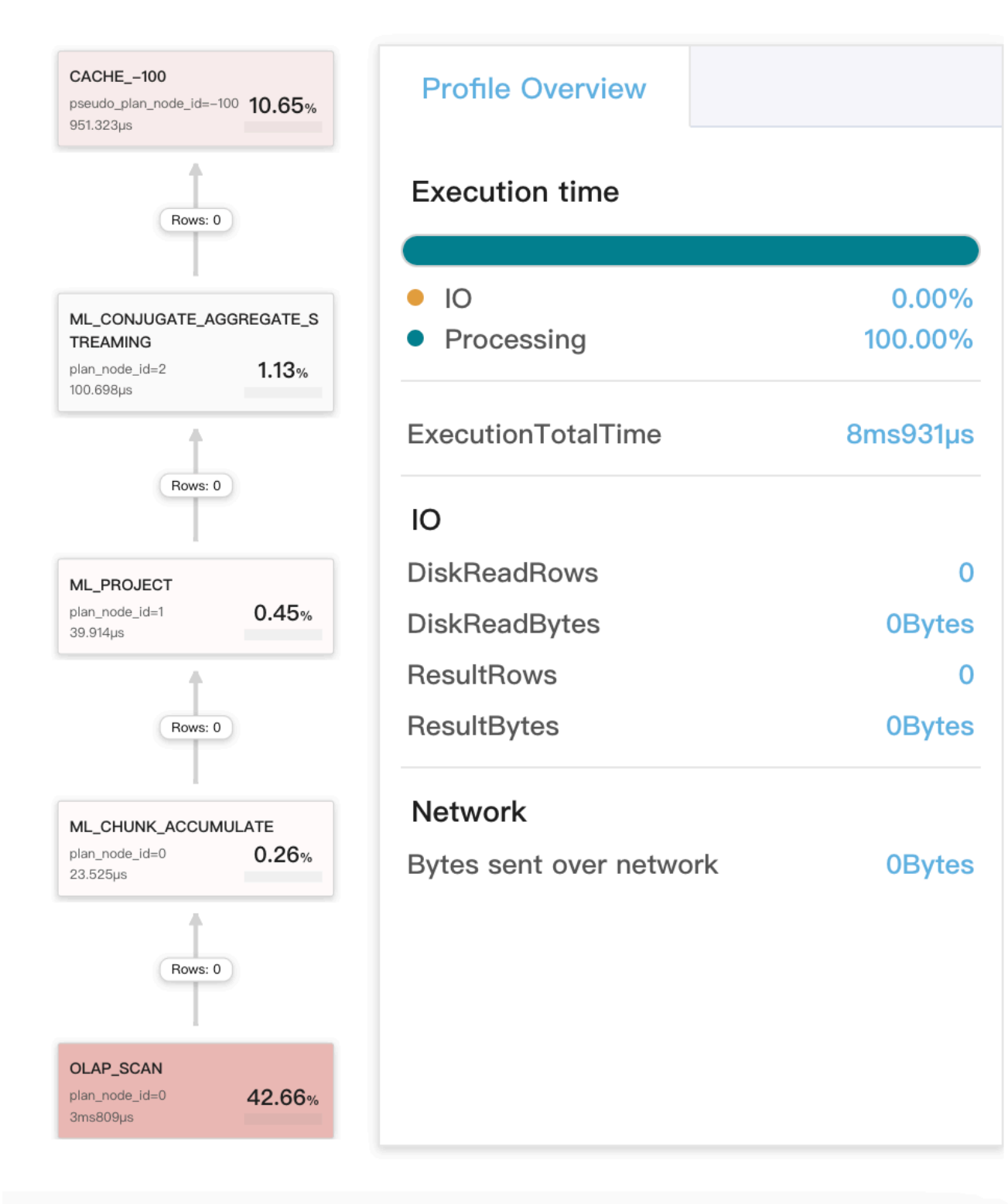
- The aggregations at stage 2 of the query are compiled to compare the same type of data. The first aggregation is a local aggregation. After the first aggregation is complete, the results generated from the first aggregation are computed to perform a second aggregation, which is a global aggregation.
- The query is a SELECT DISTINCT query.
- The query includes one of the following DISTINCT aggregate functions: `sum(distinct)`, `count(distinct)`, and `avg(distinct)`. In most cases, aggregations are performed at stage 3 or 4 for such a query. However, you can run `set new_planner_agg_stage = 1` to forcibly perform aggregations at stage 2 for the query. If the query contains `avg(distinct)` and you want to perform aggregations at stage, you also need to run `set cbo_cte_reuse = false` to disable CTE optimizations.

Query example:

```
SELECT
    date_trunc('hour', ts) AS hour,
    v0 % 2 AS is_odd,
    sum(v1) AS __c_0
FROM
    t0
WHERE
```

```
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    is_odd
```

The following figure shows the query cache-related metrics in the query profile.



Query cache works for local aggregations at stage 3

The query is a GROUP BY aggregate query that includes a single DISTINCT aggregate function.

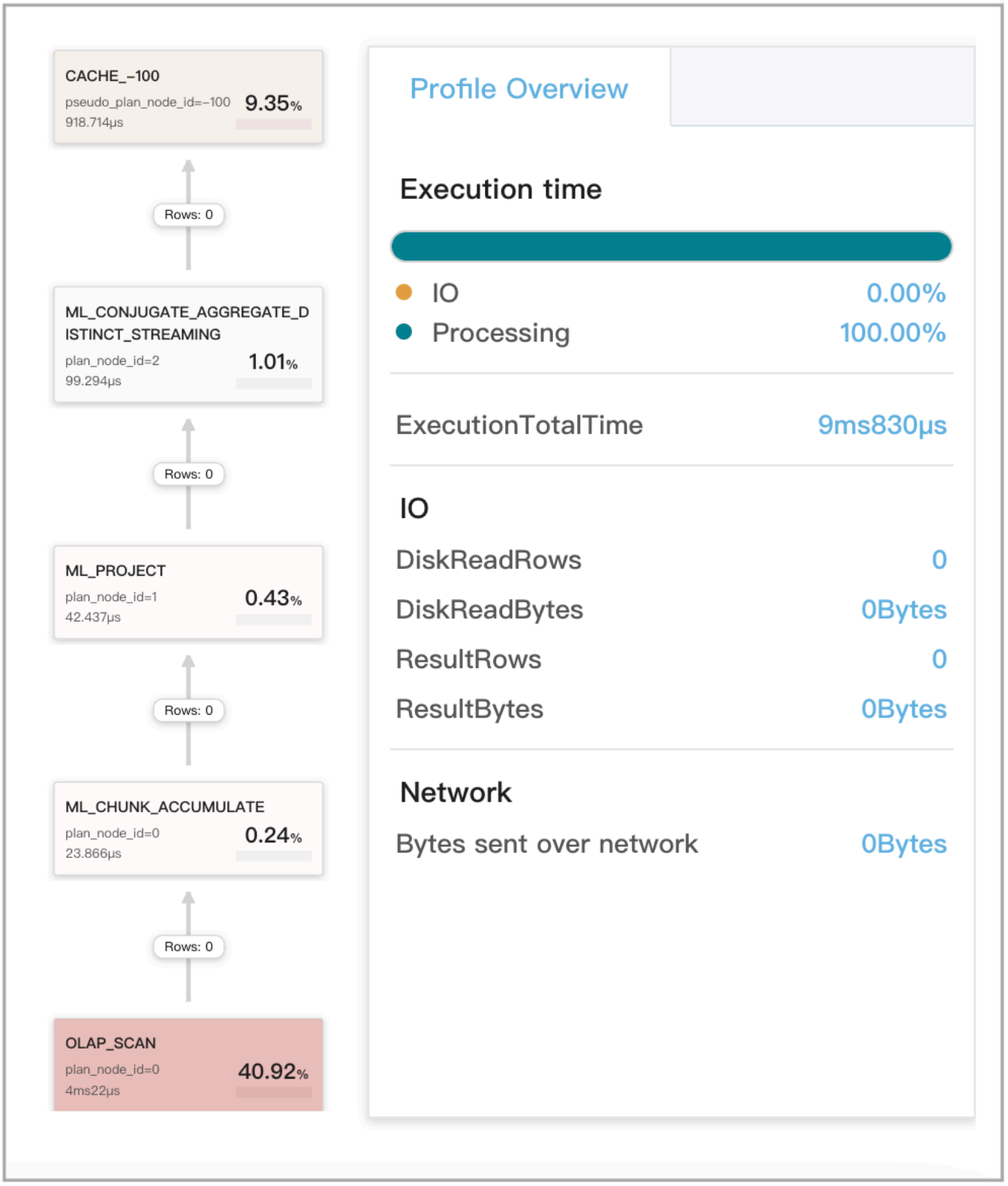
The supported DISTINCT aggregate functions are `sum(distinct)`, `count(distinct)`, and `avg(distinct)`.

> **NOTICE**
>
> If the query includes `avg(distinct)`, you also need to run `set cbo_cte_reuse = false` to disable CTE optimizations.

Query example:

```
SELECT
    date_trunc('hour', ts) AS hour,
    v0 % 2 AS is_odd,
    sum(distinct v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    is_odd;
```

The following figure shows the query cache-related metrics in the query profile.



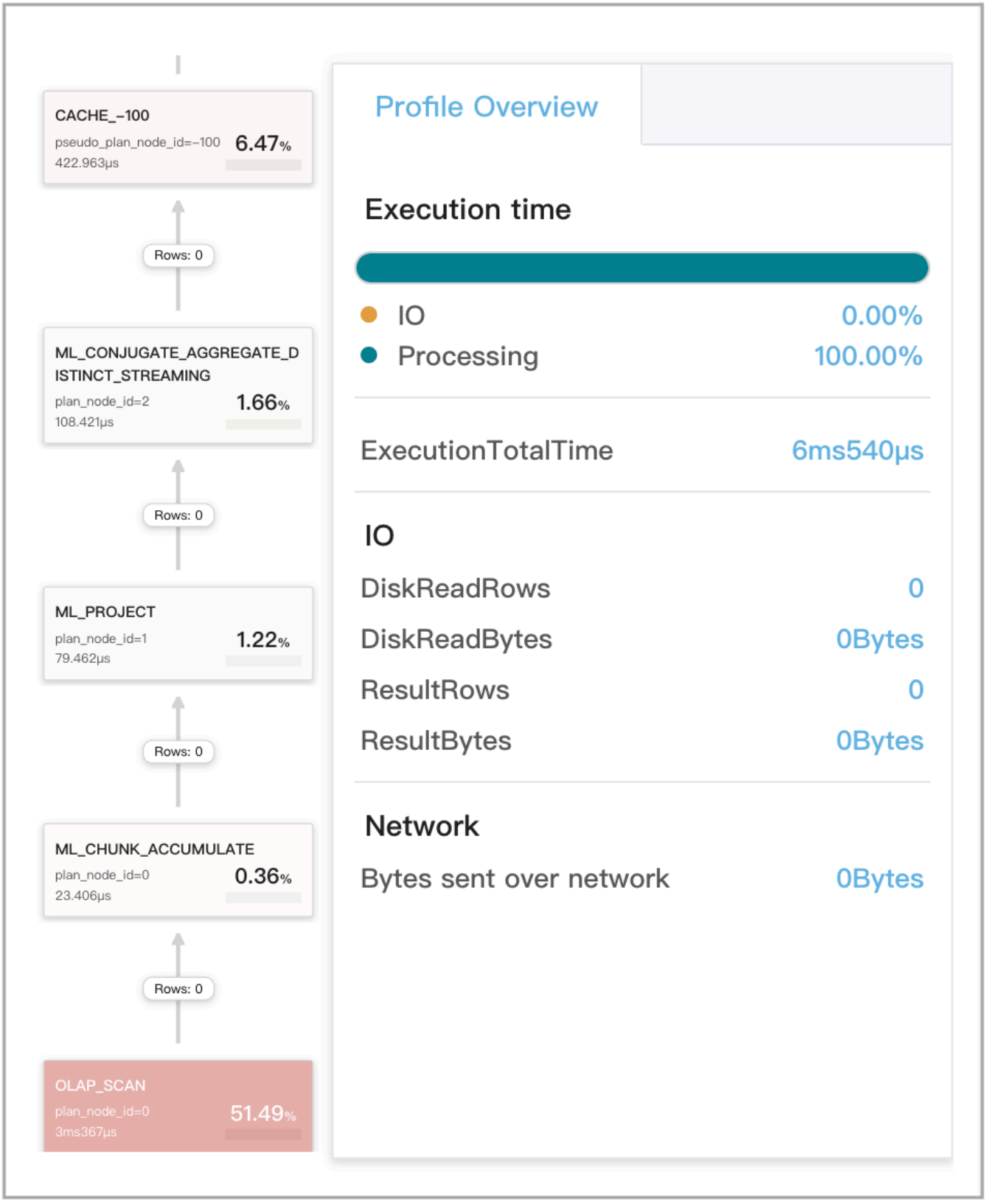
Query cache works for local aggregations at stage 4

The query is a non-GROUP BY aggregate query that includes a single DISTINCT aggregate function. Such queries include classical queries that remove deduplicate data.

Query example:

```
SELECT
    count(distinct v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
```

The following figure shows the query cache-related metrics in the query profile.



Cached results are reused for two queries whose first aggregations are semantically equivalent

Use the following two queries, Q1 and Q2, as an example. Q1 and Q2 both include multiple aggregations, but their first aggregations are semantically equivalent. Therefore, Q1 and Q2 are evaluated as semantically equivalent, and can reuse each other's computation results saved in the query cache.

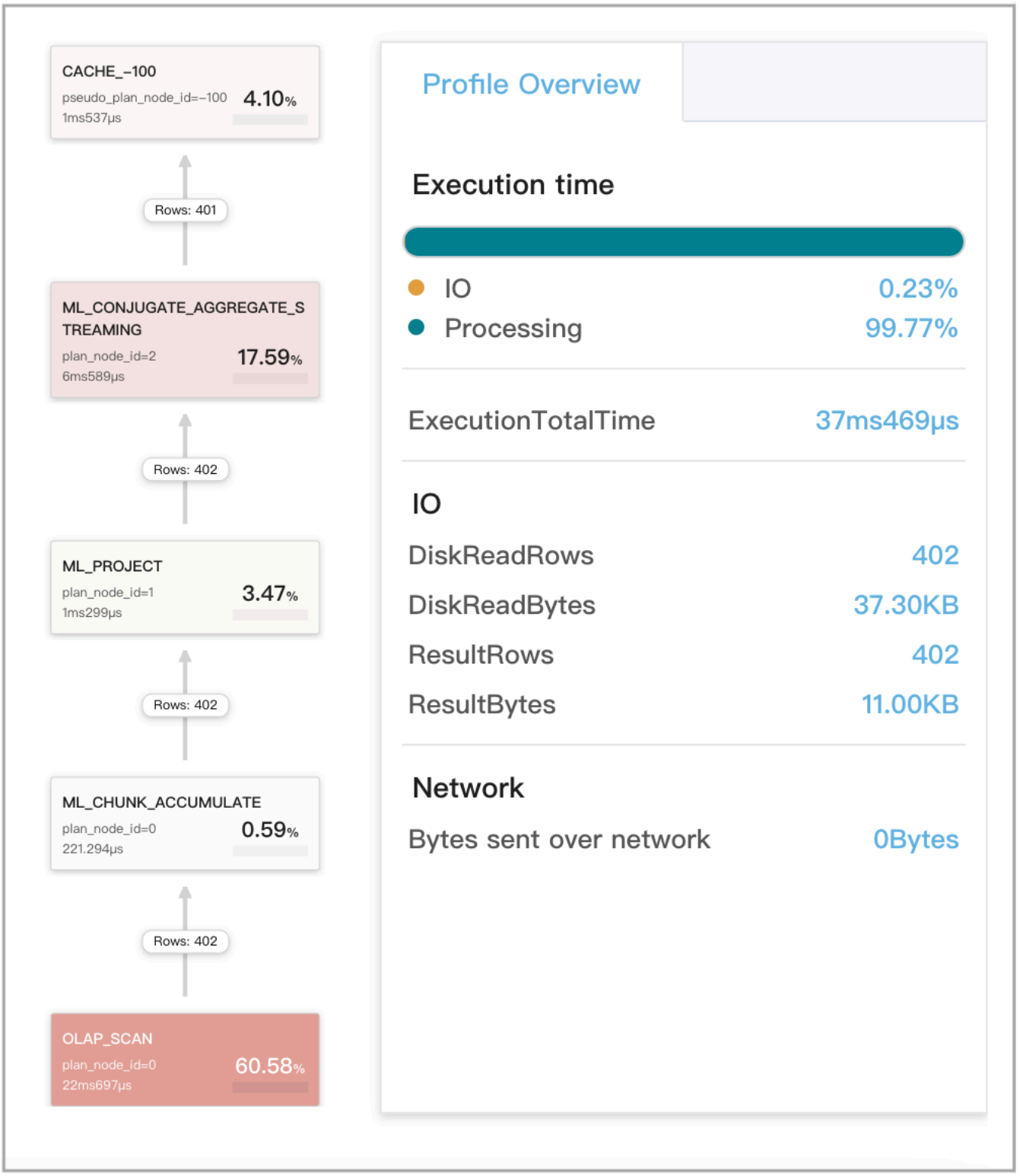
- Q1

```
SELECT
    (
        ifnull(sum(murmur_hash3_32(hour)), 0) +
ifnull(sum(murmur_hash3_32(k0)), 0) +
ifnull(sum(murmur_hash3_32(__c_0)), 0)
      ) AS fingerprint
FROM
    (
        SELECT
            date_trunc('hour', ts) AS hour,
            k0,
            sum(v1) AS __c_0
        FROM
            t0
        WHERE
            ts between '2022-01-03 00:00:00'
            and '2022-01-03 23:59:59'
        GROUP BY
            date_trunc('hour', ts),
            k0
    ) AS t;
```

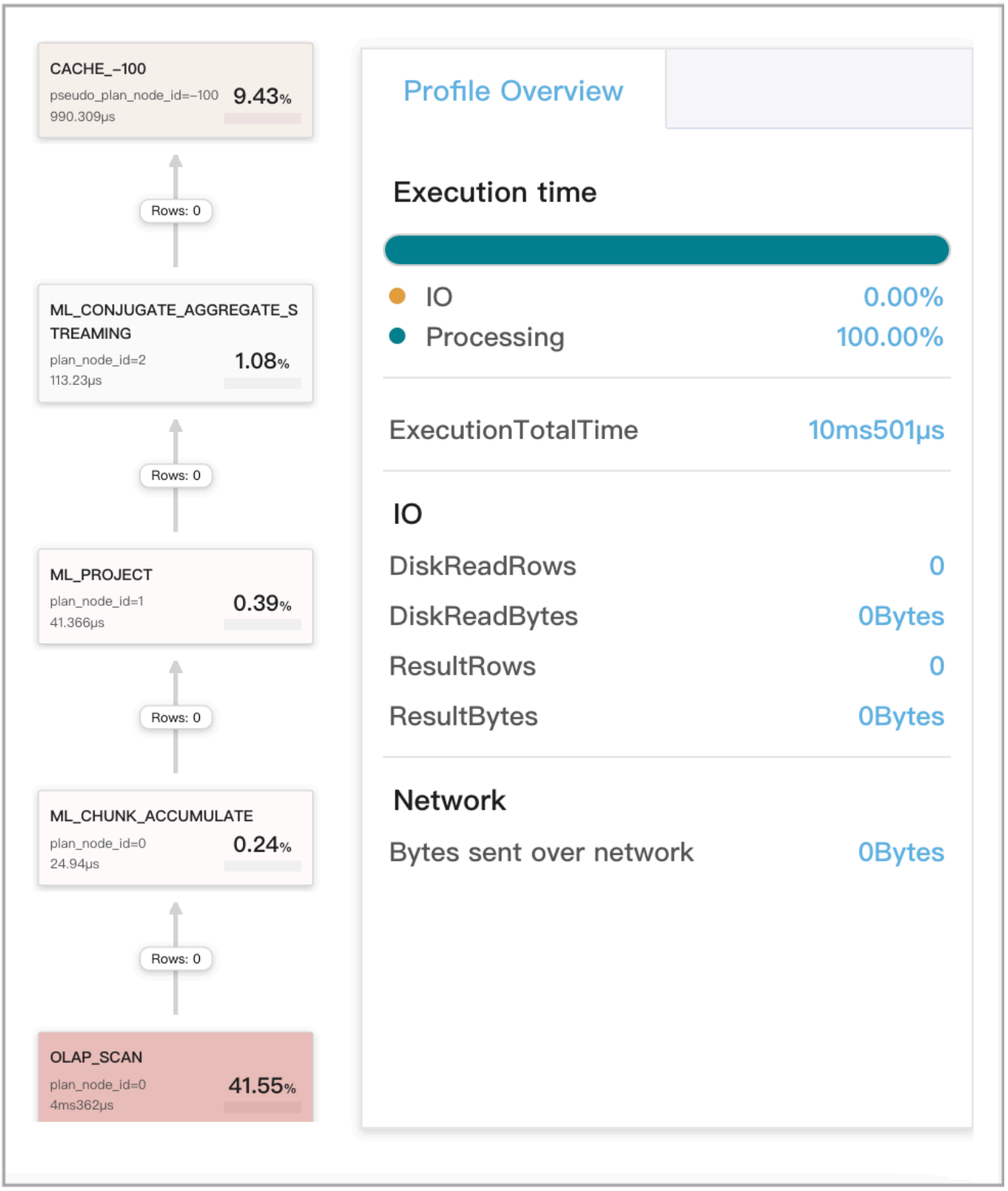
- Q2

```
SELECT
    date_trunc('hour', ts) AS hour,
    k0,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    k0
```

The following figure shows the `CachePopulate` metrics for Q1.



The following figure shows the `CacheProbe` metrics for Q2.



Query cache does not work for DISTINCT queries for which CTE optimizations are enabled

After you run `set cbo_cte_reuse = true` to enable CTE optimizations, the computation results for specific queries that include DISTINCT aggregate functions cannot be cached. A few examples are as follows:

- The query contains a single DISTINCT aggregate function `avg(distinct)`:

```
SELECT
    avg(distinct v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59';
```

- The query contains multiple DISTINCT aggregate functions that reference the same column:

```
SELECT
    avg(distinct v1) AS __c_0,
    sum(distinct v1) AS __c_1,
    count(distinct v1) AS __c_2
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59';
```

- The query contains multiple DISTINCT aggregate functions that each reference a different column:

```
SELECT
    sum(distinct v1) AS __c_1,
    count(distinct v0) AS __c_2
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59';
```

Resource Management

Bitmap indexes

This topic describes how to create and manage a bitmap index, along with usage cases.

Bloom filter indexing

This topic describes how to create and modify bloom filter indexes, along with how they works.

Bitmap indexes

This topic describes how to create and manage a bitmap index, along with usage cases.

Introduction

A bitmap index is a special database index that uses bitmaps, which are an array of bits. A bit is always in one of two values: 0 and 1. Each bit in the bitmap corresponds to a single row in the table. The value of each bit depends upon the value of the corresponding row.

A bitmap index can help improve the query performance on a given column. If a query's filter conditions match a prefix index, it can significantly improve query efficiency and quickly return results. However, a table can only have one prefix index. If the query's filter conditions do not include the prefix of the prefix index, a bitmap index can be created for that column to improve query efficiency.

How to design a bitmap index to accelerate queries

The primary considerations for choosing a bitmap index are **column cardinality and the filtering effect of the bitmap index on queries**. Contrary to popular belief, bitmap indexes in StarRocks are more suitable for **queries on columns with high cardinality and queries on the combination of multiple low cardinality columns. Furthermore, bitmap indexes need to effectively filter out data, potentially filtering out at least 999/1000 of the data, thereby reducing the amount of Page data read.**

For queries on a single low cardinality column, the filtering effect of a bitmap index is poor, because too many rows need to be read and scattered across multiple Pages.

> **INFO**
>
> You need to consider the cost of loading data when evaluating the filtering effect of bitmap indexes on queries. In StarRocks, underlying data is organized and loaded by Pages (default size is 64K). The cost of loading data includes the time to load Pages from disk, decompress Pages, and decode.

However, excessively high cardinality can also cause issues such as **occupying more disk space**, and because bitmap indexes need to be constructed during data loading, frequent data loading can **affect loading performance**.

Additionally, the **overhead of loading bitmap indexes during queries** should be considered. During a query, bitmap indexes are loaded on demand, and the larger the value of `number of column values involved in query conditions/cardinality x bitmap index`, the greater the overhead of loading bitmap indexes during queries.

To determine the appropriate cardinality and query conditions for bitmap indexes, it is recommended to refer to the Performance test on bitmap index in this topic to conduct performance tests. You can use actual business data and queries to **create bitmap indexes on columns of different cardinalities, to analyze the filtering effect of bitmap indexes on queries (at least filtering out 999/1000 of the data), the disk space usage, the impact on loading performance, and the overhead of loading bitmap indexes during queries.**

StarRocks has a built-in adaptive selection mechanism for bitmap indexes. If a bitmap index fails to accelerate queries, for example, if it cannot filter out many Pages, or the overhead of loading bitmap indexes during queries is high, it will not be used during the query, so query performance will not be significantly affected.

Adaptive selection of bitmap indexes

StarRocks can adaptively choose whether to use a bitmap index based on column cardinality and query conditions. If a bitmap index does not effectively filter out many Pages or the overhead of loading bitmap indexes during queries is high, StarRocks will not use the bitmap index by default to avoid degrading query performance.

StarRocks determines whether to use a bitmap index based on the ratio of the number of values involved in the query condition to the column cardinality. Generally, the smaller this ratio, the better the filtering effect of the bitmap index. Thus, StarRocks uses `bitmap_max_filter_ratio/1000` as the threshold. When the `number of values in the filter condition/column cardinality` is less than `bitmap_max_filter_ratio/1000`, the bitmap index will be used. The default value of `bitmap_max_filter_ratio` is `1`.

Take a query based on a single column as an example, such as `SELECT * FROM employees WHERE gender = 'male';`. The `gender` column in the `employees` table has

values 'male' and 'female', so the cardinality is 2 (two distinct values). The query condition involves one value, so the ratio is 1/2, which is greater than 1/1000. Therefore, this query will not use the bitmap index.

Take another query based on a combination of multiple columns as an example, such as `SELECT * FROM employees WHERE gender = 'male' AND city IN ('Beijing', 'Shanghai');`. The cardinality of the `city` column is of 10,000, and the query condition involves two values, so the ratio is calculated as `(1*2)/(2*10000)`, which is less than 1/1000. Therefore, this query will use the bitmap index.

> **INFO**
>
> The value range for `bitmap_max_filter_ratio` is 1-1000. If `bitmap_max_filter_ratio` is set to `1000`, any query on a column with a bitmap index will be forced to use the bitmap index.

Advantages

- Bitmap indexes can quickly locate the row numbers of queried column values, suitable for point queries or small-range queries.
- Bitmap indexes can optimize multi-dimensional queries involving union and intersection operations (OR and AND operations).

Considerations

Queries that can be optimized

Bitmap indexes are suitable for optimizing equality `=` queries, `[NOT] IN` range queries, `>`, `>=`, `<`, `<=` queries, and `IS NULL` queries. They are not suitable for optimizing `!=` and `[NOT] LIKE` queries.

Supported columns and data types

Bitmap indexes can be created on all columns in primary key and duplicate key tables, and on key columns in aggregate and unique key tables. Bitmap indexes can be created on the columns of the following data types:

- Date types: DATE, DATETIME.
- Numeric types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DECIMAL, BOOLEAN.
- String types: CHAR, STRING, VARCHAR.
- Other types: HLL.

Basic operations

Create an index

- Create a bitmap index during table creation.

```
CREATE TABLE `lineorder_partial` (
  `lo_orderkey` int(11) NOT NULL COMMENT "",
  `lo_orderdate` int(11) NOT NULL COMMENT "",
  `lo_orderpriority` varchar(16) NOT NULL COMMENT "",
  `lo_quantity` int(11) NOT NULL COMMENT "",
  `lo_revenue` int(11) NOT NULL COMMENT "",
   INDEX lo_orderdate_index (lo_orderdate) USING BITMAP
) ENGINE=OLAP
DUPLICATE KEY(`lo_orderkey`)
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 1;
```

In this example, a bitmap index named `lo_orderdate_index` is created on the `lo_orderdate` column. Naming requirements for bitmap indexes can be found in System Limits. Identical bitmap indexes cannot be created within the same table.

Multiple bitmap indexes can be created for multiple columns, separated by commas (,).

> **NOTE**
>
> For more parameters of table creation, refer to CREATE TABLE.

- `CREATE INDEX` can be used to create a bitmap index after table creation. For detailed parameter descriptions and examples, refer to CREATE INDEX.

```
CREATE INDEX lo_quantity_index (lo_quantity) USING BITMAP;
```

Progress of creating an index

Creating a bitmap index is an asynchronous process. After executing the index creation statement, you can check the index creation progress using the SHOW ALTER TABLE command. When the `State` field in the returned value shows `FINISHED`, the index is successfully created.

```
SHOW ALTER TABLE COLUMN;
```

> **INFO**
>
> Each table can only have one ongoing Schema Change task at a time. You cannot create a new bitmap index until the current bitmap index is created.

View an index

View all bitmap indexes for a specified table. For detailed parameters and returned results, refer to SHOW INDEX.

```
SHOW INDEXES FROM lineorder_partial;
```

> **NOTE**
>
> Creating a bitmap index is an asynchronous process. Using the above statement, you can only view indexes that have been created successfully.

Delete an index

Delete a bitmap index for a specified table. For detailed parameters and examples, refer to DROP INDEX.

```
DROP INDEX lo_orderdate_index ON lineorder_partial;
```

Verify whether the bitmap index accelerates queries

Check the `BitmapIndexFilterRows` field in the query Profile. For information on viewing the Profile, refer to Query analysis.

Performance test on bitmap index

Objective of testing

To analyze the filtering effect and other impacts, such as disk usage, of bitmap indexes on queries with different cardinalities:

- Query on a single column of low cardinality
- Query on the combination of multiple columns of low cardinality
- Query on a single column of high cardinality

This section also compares the performance between always using bitmap index and using bitmap indexes adaptively to verify the effectiveness of StarRocks' adaptive selection of bitmap indexes.

Create table and bitmap index

> **WARNING**
>
> To avoid caching Page data affecting query performance, ensure that the BE configuration item `disable_storage` is set to `true`.

This section takes the table `lineorder` (SSB 20G) as an example.

- Original table (without bitmap indexes) as a reference

```
CREATE TABLE `lineorder_without_index` (
`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`)
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 1;
```

- Table with bitmap indexes: Bitmap indexes created based on `lo_shipmode`, `lo_quantity`, `lo_discount`, `lo_orderdate`, `lo_tax`, and `lo_partkey`.

```
CREATE TABLE `lineorder_with_index` (
  `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 "",
  INDEX i_shipmode (`lo_shipmode`) USING BITMAP,
  INDEX i_quantity (`lo_quantity`) USING BITMAP,
  INDEX i_discount (`lo_discount`) USING BITMAP,
  INDEX i_orderdate (`lo_orderdate`) USING BITMAP,
  INDEX i_tax (`lo_tax`) USING BITMAP,
  INDEX i_partkey (`lo_partkey`) USING BITMAP
) ENGINE=OLAP
DUPLICATE KEY(`lo_orderkey`)
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 1;
```

Disk space usage of bitmap indexes

- `lo_shipmode`: String type, cardinality 7, occupies 130M
- `lo_quantity`: Integer type, cardinality 50, occupies 291M
- `lo_discount`: Integer type, cardinality 11, occupies 198M
- `lo_orderdate`: Integer type, cardinality 2406, occupies 191M
- `lo_tax`: Integer type, cardinality 9, occupies 160M
- `lo_partkey`: Integer type, cardinality 600,000, occupies 601M

Query 1: Query on single column of low cardinality

Query table without bitmap index

Query:

```
SELECT count(1) FROM lineorder_without_index WHERE
lo_shipmode="MAIL";
```

Query performance analysis: Since the table queried does not have bitmap index, all pages containing the `lo_shipmode` column data need to be read and then predicate filtering is applied.

Total Time: Approximately 0.91 milliseconds, **with data loading taking 0.47 milliseconds**, decoding dictionary for low cardinality optimization taking 0.31 milliseconds, and predicate filtering taking 0.23 milliseconds.

```
PullRowNum: 20.566M (20566493) // Number of rows in the result set.
CompressedBytesRead: 55.283 MB // Total amount of data read.
RawRowsRead: 143.999M (143999468) // Number of rows read. Since there
is no bitmap index, all data in this column is read.
ReadPagesNum: 8.795K (8795) // Number of pages read. Since there is
no bitmap index, all pages containing data for this column are read.
IOTaskExecTime: 914ms // Total time for scanning data.
    BlockFetch: 469ms // Time for loading data.
    DictDecode: 311.612ms // Time for decoding dictionary for low
cardinality optimization.
    PredFilter: 23.182ms // Time for predicate filtering.
    PredFilterRows: 123.433M (123432975) // Number of rows filtered
out.
```

Query table with bitmap index

Use bitmap index compulsorily

> **INFO**
>
> To use the bitmap index compulsorily, according to Starrocks' configuration, `bitmap_max_filter_ratio=1000` needs to be set in the **be.conf** file of each BE node, and then the BE nodes need to be restarted.

Query:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL";
```

Query performance analysis: Since the column queried is of low cardinality, bitmap index does not filter the data efficiently. Even though bitmap index can quickly locate the row numbers of actual data, a large number of rows need to be read, scattered across multiple pages. As a result, it cannot effectively filter out the pages that need to be read. Moreover, additional overhead for loading the bitmap index and using the bitmap index to filter data is incurred, resulting in a longer total time.

Total time: 2.7 seconds, **with 0.93 seconds spent loading data and bitmap index**, 0.33 seconds on decoding dictionary for low cardinality optimization, 0.42 seconds on filtering data with bitmap index, and 0.17 seconds on filtering data with ZoneMap Index.

```
PullRowNum: 20.566M (20566493) // Number of rows in the result set.
CompressedBytesRead: 72.472 MB // Total amount of data read. The size
of the bitmap index for this column is 130MB, with 7 unique values.
The size of a bitmap index for each value is 18MB. The data size of
Pages is 55MB = 73MB.
RawRowsRead: 20.566M (20566493) // Number of rows read. 20 million
rows are actually read.
ReadPagesNum: 8.802K (8802) // Number of Pages read. All Pages are
read because the 20 million rows filtered by bitmap index are
randomly distributed across different Pages, and Page is the smallest
data reading unit. Therefore, bitmap index does not filter out Pages.
IOTaskExecTime: 2s77ms // Total time for scanning data, longer than
without bitmap index.
    BlockFetch: 931.144ms // Time for loading data and bitmap index.
Compared to the previous query, an additional 400ms was spent to load
bitmap index (18MB).
```

```
    DictDecode: 329.696ms // Time for decoding dictionary for low
cardinality optimization is similar since the output row count is the
same.
    BitmapIndexFilter: 419.308ms // Time for filtering data with the
bitmap index.
    BitmapIndexFilterRows: 123.433M (123432975) // Number of rows
filtered by the bitmap index.
    ZoneMapIndexFiter: 171.580ms // Time for filtering data with
ZoneMap Index.
```

Determine whether to use bitmap index based on StarRocks default configuration

Query:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL";
```

Query performance analysis: According to StarRocks default configuration, bitmap index is used only if the number of distinct values in the filter condition column / column cardinality < `bitmap_max_filter_ratio/1000` (default is 1/1000). In this case, the value is greater than 1/1000, so bitmap index is not used, resulting in performance similar to querying a table without bitmap index.

```
PullRowNum: 20.566M (20566493) // Number of rows in the result set.
CompressedBytesRead: 55.283 MB // Total amount of data read.
RawRowsRead: 143.999M (143999468) // Number of rows read. Since the
bitmap index is not used, all data in this column is read.
ReadPagesNum: 8.800K (8800) // Number of Pages read. Since the bitmap
index is not used, all pages containing data for this column are
read.
IOTaskExecTime: 914.279ms // Total time for scanning data.
    BlockFetch: 475.890ms // Time for loading data.
    DictDecode: 312.019ms // Time for decoding dictionary for low
cardinality optimization.
    PredFilterRows: 123.433M (123432975) // Number of rows filtered
out by predicate.
```

Query 2: Query on the combination of multiple columns of low cardinality

Query table without bitmap index

Query:

```
SELECT count(1)
FROM lineorder_without_index
WHERE lo_shipmode = "MAIL"
  AND lo_quantity = 10
  AND lo_discount = 9
  AND lo_tax = 8;
```

Query performance analysis: Since the table queried does not have any bitmap index, all Pages containing data of `lo_shipmode`, `lo_quantity`, `lo_discount`, and `lo_tax` columns are read and then predicate filtering is applied.

Total time taken: 1.76 seconds, **with 1.6 seconds spent loading data (4 columns)**, and 0.1 seconds on predicate filtering.

```
PullRowNum: 4.092K (4092) // Number of rows in the result set.
CompressedBytesRead: 305.346 MB // Total amount of data read. Data of
4 columns is read, totaling 305MB.
RawRowsRead: 143.999M (143999468) // Number of rows read. Since there
is no bitmap index, all data of these 4 columns are read.
ReadPagesNum: 35.168K (35168) // Number of Pages read. Since there is
no bitmap index, all Pages containing data of these 4 columns are
read.
IOTaskExecTime: 1s761ms // Total time for scanning data.
    BlockFetch: 1s639ms // Time for loading data.
    PredFilter: 96.533ms // Time for predicate filtering with 4
filter conditions.
    PredFilterRows: 143.995M (143995376) // Number of rows filtered
out by predicates.
```

Query Table with bitmap index

Use bitmap index compulsorily

> **INFO**
>
> To use bitmap index compulsorily, according to Starrocks' configuration, the `bitmap_max_filter_ratio=1000` must be set in the **be.conf** file of each BE node, then the BE nodes must be restarted.

Query:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL"
AND lo_quantity=10 AND lo_discount=9 AND lo_tax=8;
```

Query performance analysis: Since this is a combination query based on multiple low cardinality columns, bitmap index is effective, filtering out a portion of the Pages and significantly reducing data read time.

Total time taken: 0.68 seconds, **with 0.54 seconds spent loading data and bitmap index**, and 0.14 seconds filtering data with bitmap index.

```
PullRowNum: 4.092K (4092) // Number of rows in the result set.
CompressedBytesRead: 156.340 MB // Total amount of data read. The
bitmap index effectively filters out 2/3 of the data. Out of this
156MB, 60MB is index data, and 90MB is actual data.
ReadPagesNum: 11.325K (11325) // Number of Pages read. The bitmap
index effectively filters out 2/3 of the Pages.
IOTaskExecTime: 683.471ms // Total time for scanning data,
significantly less than the total time spent without bitmap index.
    BlockFetch: 537.421ms // Time for loading data and the bitmap
index. Although additional time is spent loading the bitmap index, it
significantly reduces the time to load data.
    BitmapIndexFilter: 139.024ms // Time for filtering data with
bitmap index.
    BitmapIndexFilterRows: 143.995M (143995376) // Number of rows
filtered out by bitmap index.
```

Determine whether to use bitmap index based on StarRocks default configuration

Query:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL"
AND lo_quantity=10 AND lo_discount=9 AND lo_tax=8;
```

Query performance analysis: According to StarRocks default configuration, the bitmap index is used if the number of distinct values in the filter condition columns / column cardinality < `bitmap_max_filter_ratio/1000` (default is 1/1000). In this case, the value is less than 1/1000, so the bitmap index is used, resulting in performance similar to use the bitmap index compulsorily.

Total time taken: 0.67 seconds, **with 0.54 seconds spent loading data and bitmap index**, and 0.13 seconds filtering data with bitmap index.

```
PullRowNum: 4.092K (4092) // Number of rows in the result set.
CompressedBytesRead: 154.430 MB // Total amount of data read.
ReadPagesNum: 11.209K (11209) // Number of Pages read.
IOTaskExecTime: 672.029ms // Total time for scanning data,
significantly less than without bitmap index.
    BlockFetch: 535.823ms // Time for loading data and bitmap index.
Although loading bitmap index takes a little more time, it greatly
reduces data loading time.
    BitmapIndexFilter: 128.403ms // Time for filtering data with
bitmap index.
    BitmapIndexFilterRows: 143.995M (143995376) // Number of rows
filtered out by bitmap index.
```

Query 3: Query on single Column of high cardinality

Query table without bitmap index

Query:

```
select count(1) from lineorder_without_index where lo_partkey=10000;
```

Query performance analysis: Since the table queried does not have a bitmap index, pages containing the `lo_partkey` column data are read and then predicate filtering is applied.

Total time: approximately 0.43 seconds, **including 0.39 seconds for data loading** and 0.02 seconds for predicate filtering.

```
PullRowNum: 255 // Number of rows in the result set.
CompressedBytesRead: 344.532 MB // Total amount of data read.
RawRowsRead: 143.999M (143,999,468) // Number of rows read. Since
there is no bitmap index, all data in this column is read.
ReadPagesNum: 8.791K (8,791) // Number of pages read. Since there is
no bitmap index, all data of this column are read.
IOTaskExecTime: 428.258ms // Total time for scanning data.
    BlockFetch: 392.434ms // Time for loading data.
    PredFilter: 20.807ms // Time for predicate filtering.
```

```
    PredFilterRows: 143.999M (143,999,213) // Number of rows filtered
out by predicates.
```

Query a Table with bitmap index

Use bitmap index compulsorily

> **INFO**
>
> To use the bitmap index compulsorily, according to StarRocks' configuration, you must set `bitmap_max_filter_ratio=1000` in the **be.conf** file on each BE node, and then restart the BE nodes.

Query:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_partkey=10000;
```

Query performance analysis: Since the queried column is of high cardinality, the bitmap index is effective, allowing for filtering out a portion of the pages and significantly reducing the time for reading data.

Total time: 0.015 seconds, **including 0.009 seconds for loading data and bitmap index**, and 0.003 seconds for bitmap index filtering.

```
PullRowNum: 255 // Number of rows in the result set.
CompressedBytesRead: 13.600 MB // Total amount of data read. Bitmap
Index filtering is effective, filtering out a large amount of data.
RawRowsRead: 255 // Number of rows read.
ReadPagesNum: 225 // Number of pages read. The bitmap index can
effectively filter out data, filtering out a large number of pages.
IOTaskExecTime: 15.354ms // Total time for scanning data,
significantly less than the total time spent without Bitmap Index.
    BlockFetch: 9.530ms // Time for loading data and Bitmap Index.
Although additional time is spent loading the Bitmap Index, it
significantly reduces the time to load data.
    BitmapIndexFilter: 3.450ms // Time for filtering data with Bitmap
Index.
    BitmapIndexFilterRows: 143.999M (143,999,213) // Number of rows
filtered out by Bitmap Index.
```

Determine whether to use bitmap index based on StarRocks default configuration

Query:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_partkey=10000;
```

Query performance analysis: According to StarRocks' default configuration, Bitmap Index is used when the number of distinct values/column cardinality < `bitmap_max_filter_ratio/1000` (default 1/1000). Since this condition is met, the query uses the Bitmap Index, and the performance is similar to that when the Bitmap Index is compulsorily used..

Total time: 0.014 seconds, including **0.008 seconds for loading data and bitmap index**, and 0.003 seconds for Bitmap Index filtering.

```
PullRowNum: 255 // Number of rows in the result set.
CompressedBytesRead: 13.600 MB // Total amount of data read. The
bitmap index can effectively filter out data, filtering out a large
number of data.
RawRowsRead: 255 // Number of rows read.
ReadPagesNum: 225 // Number of pages read. The bitmap index can
effectively filter out data, filtering out a large number of pages.
IOTaskExecTime: 14.387ms // Total time for scanning data,
significantly less than the total time spent without Bitmap Index.
    BitmapIndexFilter: 2.979ms // Time for filtering data with Bitmap
Index.
    BitmapIndexFilterRows: 143.999M (143,999,213) // Number of rows
filtered out by Bitmap Index.
    BlockFetch: 8.988ms // Time for loading data and Bitmap Index.
Although additional time is spent loading the Bitmap Index, it
significantly reduces the time to load data.
```

Bloom filter indexing

This topic describes how to create and modify bloom filter indexes, along with how they works.

A bloom filter index is a space-efficiency data structure that is used to detect the possible presence of filtered data in data files of a table. If the bloom filter index detects that the data to be filtered are not in a certain data file, StarRocks skips scanning the data file. Bloom filter indexes can reduce response time when the column (such as ID) has a relatively high cardinality.

If a query hits a sort key column, StarRocks efficiently returns the query result by using the prefix index. However, the prefix index entry for a data block cannot exceed 36 bytes in length. If you want to improve the query performance on a column, which is not used as a sort key and has a relatively high cardinality, you can create a bloom filter index for the column.

How it works

For example, you create a bloom filter index on a `column1` of a given table `table1` and run a query such as `Select xxx from table1 where column1 = something;`. Then the following situations happen when StarRocks scans the data files of `table1`.

- If the bloom filter index detects that a data file does not contain the data to be filtered, StarRocks skips the data file to improve query performance.
- If the bloom filter index detects that a data file may contain the data to be filtered, StarRocks reads the data file to check whether the data exists. Note that the bloom filter can tell you for sure if a value is not present, but it cannot say for sure that a value is present, only that it may be present. Using a bloom filter index to determine whether a value is present may give false positives, which means that a bloom filter index detects that a data file contains the data to be filtered, but the data file does not actually contain the data.

Usage notes

- You can create bloom filter indexes for all columns of a Duplicate Key or Primary Key table. For an Aggregate table or Unique Key table, you can only create bloom filter indexes for key columns.
- Bloom filter indexes can be created for columns of the following data types:
  - Numeric types: SMALLINT, INT, BIGINT, and LARGEINT.
  - String types: CHAR, STRING, and VARCHAR.
  - Date types: DATE and DATETIME.
- Bloom filter indexes can only improve the performance of queries that contain the `in` and `=` operators, such as `Select xxx from table where x in {}` and `Select xxx from table where column = xxx`.
- You can check whether a query uses bloom filter indexes by viewing the `BloomFilterFilterRows` field of the query's profile.

Create bloom filter indexes

You can create a bloom filter index for a column when you create a table by specifying the `bloom_filter_columns` parameter in `PROPERTIES`. For example, create bloom filter indexes for the `k1` and `k2` columns in `table1`.

```
CREATE TABLE table1
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT DEFAULT "10"
)
ENGINE = olap
PRIMARY KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2) BUCKETS 10
PROPERTIES("bloom_filter_columns" = "k1,k2");
```

You can create bloom filter indexes for multiple columns at a time by specifying these column names. Note that you need to separate these column names with commas (`,`). For other parameter descriptions of the CREATE TABLE statement, see CREATE TABLE.

Display bloom filter indexes

For example, the following statement displays bloom filter indexes of `table1`. For the output description, see SHOW CREATE TABLE.

```
SHOW CREATE TABLE table1;
```

Modify bloom filter indexes

You can add, reduce, and delete bloom filter indexes by using the ALTER TABLE statement.

- The following statement adds a bloom filter index on the `v1` column.

  ```
  ALTER TABLE table1 SET ("bloom_filter_columns" = "k1,k2,v1");
  ```

- The following statement reduces the bloom filter index on the `k2` column.

  ```
  ALTER TABLE table1 SET ("bloom_filter_columns" = "k1");
  ```

- The following statement deletes all bloom filter indexes of `table1`.

  ```
  ALTER TABLE table1 SET ("bloom_filter_columns" = "");
  ```

> Note: Altering an index is an asynchronous operation. You can view the progress of this operation by executing SHOW ALTER TABLE. You can run only one alter index task on a table each time.

Use Bitmap for exact Count Distinct

This topic describes how to use bitmaps to compute the number of distinct values in StarRocks.

Use HLL for approximate count distinct

Background

Use Bitmap for exact Count Distinct

This topic describes how to use bitmaps to compute the number of distinct values in StarRocks.

Bitmaps are a useful tool for computing the number of distinct values in an array. This method takes up less storage space and can accelerate computation when compared to traditional Count Distinct. Assume there is an array named A with a value range of [0, n). By using a bitmap of (n+7)/8 bytes, you can compute the number of distinct elements in the array. To do this, initialize all bits to 0, set the values of the elements as the subscripts of bits, and then set all bits to 1. The number of 1s in the bitmap is the number of distinct elements in the array.

Traditional Count Distinct

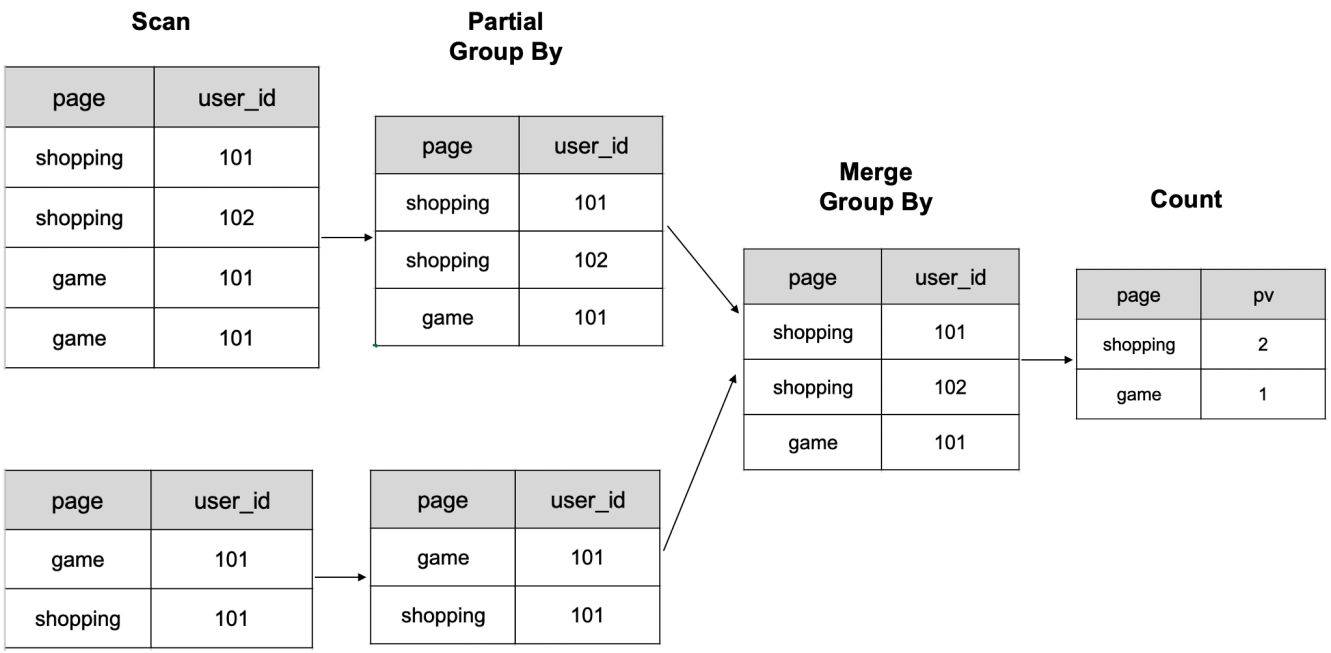
StarRocks uses the MPP architecture, which can retain the detailed data when using Count Distinct. However, the Count Distinct feature requires multiple data shuffles during query processing, which consumes more resources and results in a linear decrease in performance as the data volume increases.

The following scenario calculates UVs based on detailed data in table (dt, page, user_id).

dt	page	user_id
20191206	game	101
20191206	shopping	102
20191206	game	101
20191206	shopping	101

dt	page	user_id
20191206	game	101
20191206	shopping	101

StarRocks computes data according to the following figure. It first groups data by the `page` and `user_id` columns, and then counts the processed result.



- Note: The figure shows a schematic of 6 rows of data computed on two BE nodes.

When dealing with large volumes of data that require multiple shuffle operations, the computational resources needed can increase significantly. This slows queries. However, using the Bitmap technology can help address this issue and improve the query performance in such scenarios.

Count `uv` grouping by `page`:

```
select page, count(distinct user_id) as uv from table group by page;

|  page   |   uv  |
| :---: | :---: |
|   game  |  1   |
|   shopping  |   2  |
```

Benefits of Count Distinct with Bitmap

You can benefit from bitmaps in the following aspects compared with COUNT(DISTINCT expr):

- Less storage space: If you use bitmap to compute the number of distinct values for INT32 data, the required storage space is only 1/32 of COUNT(DISTINCT expr). StarRocks utilizes compressed roaring bitmaps to execute computations, further reducing storage space usage compared to traditional bitmaps.
- Faster computation: Bitmaps use bitwise operations, resulting in faster computation compared to COUNT(DISTINCT expr). In StarRocks, the computation of the number of distinct values can be processed in parallel, leading to further improvements in query performance.

For the implementation of Roaring Bitmap, see specific paper and implementation.

Usage notes

- Both bitmap indexing and bitmap Count Distinct use the bitmap technique. However, the purpose for introducing them and the problem they solve are completely different. The former is used to filter enumerated columns with a low cardinality, while the latter is used to calculate the number of distinct elements in the value columns of a data row.
- StarRocks 2.3 and later versions support defining a value column as BITMAP regardless of the table types (Aggregate table, Duplicate Key table, Primary Key table, or Unique Key table). However, the sort key of a table cannot be of the BITMAP type.
- When creating a table, you can define the value column as BITMAP and the aggregate function as BITMAP_UNION.
- You can only use roaring bitmaps to compute the number of distinct values for data of the following types: TINYINT, SMALLINT, INT, and BIGINT. For data of other types, you need to build global dictionaries.

Count Distinct with Bitmap

Take the calculation of page UVs as an example.

1. Create an Aggregate table with a BITMAP column `visit_users`, which uses the aggregate function BITMAP_UNION.

```
CREATE TABLE `page_uv` (
  `page_id` INT NOT NULL COMMENT 'page ID',
  `visit_date` datetime NOT NULL COMMENT 'access time',
  `visit_users` BITMAP BITMAP_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "3",
  "storage_format" = "DEFAULT"
);
```

2. Load data into this table.

   Use INSET INTO to load data:

```
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));
```

   After data is loaded:

   - In the row `page_id = 1, visit_date = '2020-06-23 01:30:30'`, the `visit_users` field contains three bitmap elements (13, 23, 33).
   - In the row `page_id = 1, visit_date = '2020-06-23 02:30:30'`, the `visit_users` field contains one bitmap element (13).
   - In the row `page_id = 2, visit_date = '2020-06-23 01:30:30'`, the `visit_users` field contains one bitmap element (23).

   Load data from a local file:

```
echo -e '1,2020-06-23 01:30:30,130\n1,2020-06-23
01:30:30,230\n1,2020-06-23 01:30:30,120\n1,2020-06-23
02:30:30,133\n2,2020-06-23 01:30:30,234' > tmp.csv |
curl --location-trusted -u <username>:<password> -H
"label:label_1600960288798" \
    -H "column_separator:," \
    -H "columns:page_id,visit_date,visit_users,
```

```
visit_users=to_bitmap(visit_users)" -T tmp.csv \
    http://StarRocks_be0:8040/api/db0/page_uv/_stream_load
```

3. Calculate page UVs.

```
SELECT page_id, count(distinct visit_users) FROM page_uv GROUP BY
page_id;
+-----------+-----------------------------+
|  page_id  | count(DISTINCT `visit_users`)|
+-----------+-----------------------------+
|         1 |                           3 |
|         2 |                           1 |
+-----------+-----------------------------+
2 row in set (0.00 sec)
```

Global Dictionary

Currently, Bitmap-based Count Distinct mechanism requires the input to be integer. If the user needs to use other data types as input to the Bitmap, then the user needs to build their own global dictionary to map other types of data (such as string types) to integer types. There are several ideas for building a global dictionary.

Hive table-based Global Dictionary

The global dictionary itself in this scheme is a Hive table, which has two columns, one for raw values and one for encoded Int values. The steps to generate the global dictionary are as follows:

1. De-duplicate the dictionary columns of the fact table to generate a temporary table
2. Left join the temporary table and the global dictionary, add `new value` to the temporary table.
3. Encode the `new value` and insert it into the global dictionary.
4. Left join the fact table and the updated global dictionary, replace the dictionary items with IDs.

In this way, the global dictionary can be updated and the value columns in the fact table can be replaced using Spark or MR. Compared with the trie tree-based global dictionary, this approach can be distributed and the global dictionary can be reused.

However, there are a few things to note: the original fact table is read multiple times, and there are two joins that consume a lot of extra resources during the calculation of the global dictionary.

Build a global dictionary based on a trie tree

Users can also build their own global dictionaries using trie trees (aka prefix trees or dictionary trees). The trie tree has common prefixes for the descendants of nodes, which can be used to reduce query time and minimize string comparisons, and therefore is well suited for implementing dictionary encoding. However, the implementation of trie tree is not easy to distribute and can create performance bottlenecks when the data volume is relatively large.

By building a global dictionary and converting other types of data to integer data, you can use Bitmap to perform accurate Count Distinct analysis of non-integer data columns.

Use HLL for approximate count distinct

Background

In a real-world scenario, the pressure to de-duplicate the data increases as the volume of data increases. When the size of data reaches a certain level, the cost of accurate de-duplication is relatively high. In this case, users usually use approximate algorithms to reduce the computational pressure. HyperLogLog (HLL), which will be introduced in this section, is an approximate de-duplication algorithm that has excellent space complexity O(mloglogn) and time complexity O(n). What's more, the error rate of the computation result can be controlled to about 1%-10%, depending on the size of the data set and the hash function used.

What is HyperLogLog

HyperLogLog is an approximate de-duplication algorithm that consumes very little storage space. The **HLL type** is used for implementing the HyperLogLog algorithm. It holds the intermediate results of the HyperLogLog calculation and can only be used as an indicator column type for data tables.

Since the HLL algorithm involves a lot of mathematical knowledge, we will use a practical example to illustrate it. Suppose we design a randomized experiment A, that is to do independent repetitions of coin flips until the first head; record the number of coin flips for the first head as a random variable X, then:

- X=1, P(X=1)=1/2
- X=2, P(X=2)=1/4
- ...
- X=n, P(X=n)=$(1/2)^n$

We use test A to construct randomized test B which is to do N independent repetitions of test A, generating N independent identically distributed random variables X_1, X_2, X_3, ..., X_N.Take the maximum value of the random variables as X_{max}. Leveraging the great likelihood estimation, the estimated value of N is $2^{X_{max}}$.

Now, we simulate the above experiment using the hash function on the given dataset:

- Test A: Calculate the hash value of dataset elements and convert the hash value to binary representation. Record the occurrence of bit=1, starting from the lowest bit of the binary.
- Test B: Repeat the Test A process for dataset elements of Test B. Update the maximum position "m" of the first bit 1 occurrence for each test;
- Estimate the number of non-repeating elements in the dataset as m^2.

In fact, the HLL algorithm divides the elements into $K=2^k$ buckets based on the lower k bits of the element hash. Count the maximum value of the first bit 1 occurrence from the k+1st bit as m_1, m_2,..., m_k, and estimate the number of non-repeating elements in the bucket as 2^{m_1}, 2^{m_2},..., 2^{m_k}. The number of non-repeating elements in the data set is the summed average of the number of buckets multiplied by the number of non-repeating elements in the buckets: $N = K(K/(2^{-m_1}+2^{-m_2},..., 2^{-m_K}))$.

HLL multiplies the correction factor with the estimation result to make the result more accurate.

Refer to the article https://gist.github.com/avibryant/8275649 on implementing HLL de-duplication algorithm with StarRocks SQL statements:

```
SELECT floor((0.721 * 1024 * 1024) / (sum(pow(2, m * -1)) + 1024 -
count(*))) AS estimate
FROM(select(murmur_hash3_32(c2) & 1023) AS bucket,
     max((31 - CAST(log2(murmur_hash3_32(c2) & 2147483647) AS INT)))
AS m
     FROM db0.table0
     GROUP BY bucket) bucket_values
```

This algorithm de-duplicates col2 of db0.table0.

- Use the hash function `murmur_hash3_32` to calculate the hash value of col2 as a 32-signed integer.
- Use 1024 buckets, the correction factor is 0.721, and take the lower 10 bits of the hash value as the subscript of the bucket.
- Ignore the sign bit of the hash value, start from the next highest bit to the lower bit, and determine the position of the first bit 1 occurrence.
- Group the calculated hash values by bucket, and use the `MAX` aggregation to find the maximum position of the first bit 1 occurrence in the bucket.
- The aggregation result is used as a subquery, and the summed average of all bucket estimates is multiplied by the number of buckets and the correction factor.
- Note that the empty bucket count is 1.

The above algorithm has a very low error rate when the data volume is large.

This is the core idea of the HLL algorithm. Please refer to the HyperLogLog paper if you are interested.

How to use HyperLogLog

1. To use HyperLogLog de-duplication, you need to set the target indicator column type to `HLL` and the aggregation function to `HLL_UNION` in the table creation statement.
2. Currently, only the aggregation model supports HLL as indicator column type.
3. When using `count distinct` on columns of the HLL type, StarRocks will automatically convert it to the `HLL_UNION_AGG` calculation.

Example

First, create a table with **HLL** columns, where uv is an aggregated column, the column type is `HLL`, and the aggregation function is HLL_UNION.

```
CREATE TABLE test(
        dt DATE,
        id INT,
        uv HLL HLL_UNION
)
DISTRIBUTED BY HASH(ID) BUCKETS 32;
```

> - Note: When the data volume is large, it is better to create a corresponding rollup table for high frequency HLL queries

Load data using Stream Load:

```
curl --location-trusted -u <username>:<password> -H
"label:label_1600997542287" \
    -H "column_separator:," \
    -H "columns:dt,id,user_id, uv=hll_hash(user_id)" -T
/root/test.csv http://starrocks_be0:8040/api/db0/test/_stream_load
{
    "TxnId": 2504748,
    "Label": "label_1600997542287",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 5,
    "NumberLoadedRows": 5,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 120,
    "LoadTimeMs": 46,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 29,
    "CommitAndPublishTimeMs": 14
}
```

Broker Load mode:

```
LOAD LABEL test_db.label
 (
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/file")
    INTO TABLE `test`
    COLUMNS TERMINATED BY ","
    (dt, id, user_id)
    SET (
      uv = HLL_HASH(user_id)
    )
 );
```

Querying data

- The HLL column does not allow direct query of its original value, use the function HLL_UNION_AGG to query.
- To find the total uv,

```
SELECT HLL_UNION_AGG(uv) FROM test;
```

This statement is equivalent to

```
SELECT COUNT(DISTINCT uv) FROM test;
```

- Query uv of everyday

```
SELECT COUNT(DISTINCT uv) FROM test GROUP BY ID;
```

Cautions

How should I choose between Bitmap and HLL? If the base of the dataset is in the millions or tens of millions, and you have a few dozen machines, use `count distinct`. If the base is in the hundreds of millions and needs to be accurately de-duplicated, use `Bitmap`; if approximate de-duplication is acceptable, use the `HLL` type.

Bitmap only supports TINYINT, SMALLINT, INT, and BIGINT. Note that LARGEINT is not supported. For other types of data sets to be de-duplicated, a dictionary needs to be built to map the original type to an integer type. Building a dictionary is complex, and requires a trade-off between data volume, update frequency, query efficiency, storage, and other issues. HLL does not need a dictionary, but it needs the corresponding data type to support the hash function. Even in an analytical system that does not support HLL internally, it is still possible to use the hash function and SQL to implement HLL de-duplication.

For common columns, users can use the NDV function for approximate de-duplication. This function returns an approximate aggregation of COUNT(DISTINCT col) results, and the underlying implementation converts the data storage type to the HyperLogLog type for calculation. The NDV function consumes a lot of resources when calculating and is therefore not well suited for high concurrency scenarios.

If you want to perform user behavior analysis, you may consider IntersectCount or custom UDAF.

Sorted streaming aggregate

Common aggregation methods in database systems include hash aggregate and sort aggregate.

From v2.5 onwards, StarRocks supports **sorted streaming aggregate**.

Working principles

Aggregate nodes (AGG) are mainly responsible for handling GROUP BY and aggregate functions.

Sorted streaming aggregate can group data by comparing GROUP BY keys according to the sequence of the keys, without the need to create a hash table. This effectively reduces memory resources consumed by aggregation. For queries with a high aggregation cardinality, sorted streaming aggregate improves aggregation performance and reduces memory usage.

You can enable sorted streaming aggregate by setting the following variable:

```
set enable_sort_aggregate=true;
```

Limits

- The keys in GROUP BY must have a sequence. For example, if the sort key is `k1, k2, k3`, then:
  - `GROUP BY k1` and `GROUP BY k1, k2` are allowed.
  - `GROUP BY k1, k3` does not follow the sort key sequence. Therefore, sorted streaming aggregate cannot take effect on such a clause.
- The selected partition must be a single partition (because the same key may be distributed on different machines in different partitions).
- The GROUP BY keys must have the same distribution as the bucket keys specified when you create a table. For example, if a table has three columns `k1, k2, k3`, the

bucket key can be `k1` or `k1, k2`.
    - If the bucket key is `k1`, the `GROUP BY` key can be `k1`, `k1, k2`, or `k1, k2, k3`.
    - If the bucket key is `k1, k2`, the GROUP BY key can be `k1, k2` or `k1, k2, k3`.
    - If the query plans do not meet this requirement, the sorted streaming aggregate feature cannot take effect even if this feature is enabled.
- Sorted streaming aggregate works only for first-stage aggregation (that is, there is only one Scan node under the AGG node).

Examples

1. Create a table and insert data.

```
CREATE TABLE `test_sorted_streaming_agg_basic`
(
    `id_int` int(11) NOT NULL COMMENT "",
    `id_string` varchar(100) NOT NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`id_int`)COMMENT "OLAP"
DISTRIBUTED BY HASH(`id_int`) BUCKETS 10
PROPERTIES
("replication_num" = "3");

INSERT INTO test_sorted_streaming_agg_basic VALUES
(1, 'v1'),
(2, 'v2'),
(3, 'v3'),
(1, 'v4');
```

2. Enable sorted streaming aggregate and use EXPLAIN to query the SQL profile.

```
set enable_sort_aggregate = true;

explain costs select id_int, max(id_string)
from test_sorted_streaming_agg_basic
group by id_int;
```

Check whether sorted streaming aggregate is enabled

View the results of `EXPLAIN costs`. If the `sorted streaming` field is `true` in the AGG node, this feature is enabled.

```
|
|
|   1:AGGREGATE (update finalize)
|
|   |  aggregate: max[([2: id_string, VARCHAR, false]); args: VARCHAR;
result: VARCHAR; args nullable: false; result nullable: true] |
|   |  group by: [1: id_int, INT, false]
|
|   |  sorted streaming: true
|
|   |  cardinality: 1
|
|   |  column statistics:
|
|   |  * id_int-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
|   |  * max-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
|   |
|
|   0:OlapScanNode
|
|      table: test_sorted_streaming_agg_basic, rollup:
test_sorted_streaming_agg_basic
|
|      preAggregation: on
|
|      partitionsRatio=1/1, tabletsRatio=10/10
|
|
tabletList=30672,30674,30676,30678,30680,30682,30684,30686,30688,30690
|
|      actualRows=0, avgRowSize=2.0
|
|      cardinality: 1
|
|      column statistics:
|
```

```
|      * id_int-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
|      * id_string-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
```

Integrations

Authenticate to AWS resources

StarRocks supports using three authentication methods to integrate with AWS resources: instance ...

Hive Bitmap UDF

Hive Bitmap UDF provides UDFs that can be directly used in Hive. They can be used to generate Bi...

Other tools and systems

This topic introduces the other tools and systems with which StarRocks can integrate.

Authenticate to AWS resources

StarRocks supports using three authentication methods to integrate with AWS resources: instance profile-based authentication, assumed role-based authentication, and IAM user-based authentication. This topic describes how to configure AWS credentials by using these authentication methods.

Authentication methods

Instance profile-based authentication

The instance profile-based authentication method allows your StarRocks cluster to inherit the privileges specified in the instance profile of the EC2 instance on which the cluster runs. In theory, any cluster user who can log in to the cluster can perform permitted actions on your AWS resources according to the AWS IAM policies you have configured. The typical scenario for this use case is that you do not need any AWS resource access control between multiple cluster users in the cluster. This authentication method means no isolation is required within the same cluster.

However, this authentication method still can be seen as a cluster-level safe access control solution, because whoever can log in to the cluster is controlled by the cluster administrator.

Assumed role-based authentication

Unlike instance profile-based authentication, the assumed role-based authentication method supports assuming an AWS IAM role to gain access to your AWS resources. For more information, see Assuming a role.

IAM user-based authentication

The IAM user-based authentication method supports using IAM user credentials to gain access to your AWS resources. For more information, see IAM users.

Preparations

First, find the IAM role associated with the EC2 instance on which your StarRocks cluster runs (that role is referred to as the EC2 instance role hereinafter in this topic), and obtain the role's ARN. You will need the EC2 instance role for instance profile-based authentication and need the EC2 instance role and its ARN for assumed role-based authentication.

Next step, create an IAM policy based on the type of AWS resource you want to access and the specific operation scenario within StarRocks. A policy in AWS IAM declares a set of permissions on a specific AWS resource. After creating a policy, you need to attach it to an IAM role or user. As such, the IAM role or user is assigned the permissions declared in the policy to access the specified AWS resource.

> **NOTICE**
>
> To make these preparations, you must have permission to sign in to the AWS IAM console and edit IAM users and roles.

For the IAM policy you will need to access a specific AWS resource, see the following sections:

- Batch load data from AWS S3
- Read/write AWS S3
- Integrate with AWS Glue

Preparation for instance profile-based authentication

Attach the IAM policies for accessing the required AWS resources to the EC2 instance role.

Preparation for assumed role-based authentication

Create IAM roles and attach policies to them

Create one or more IAM roles, depending on the AWS resources you want to access. See Creating IAM roles. Then, attach the IAM policies for accessing the required AWS resources to the IAM roles you create.

For example, you want your StarRocks cluster to access AWS S3 and AWS Glue. In this situation, you can choose to create one IAM role (for example, `s3_assumed_role`), and attach the policy for accessing AWS S3 and the policy for accessing AWS Glue both to that

role. Alternatively, you can choose to create two different IAM roles (for example, `s3_assumed_role` and `glue_assumed_role`), and attach these policies to the two different roles respectively (that is, attach the policy for accessing AWS S3 to `s3_assumed_role` and the policy for accessing AWS Glue to `glue_assumed_role`).

The IAM roles you create will be assumed by the EC2 instance role of the StarRocks cluster to access the specified AWS resources.

This section assumes that you have created only one assumed role, `s3_assumed_role`, and have added the policy for accessing AWS S3 and the policy for accessing AWS Glue both to that role.

Configure a trust relationship

Configure your assumed role as follows:

1. Sign in to the AWS IAM console.
2. In the left-side navigation pane, choose **Access management** > **Roles**.
3. Find the assumed role (`s3_assumed_role`) and click its name.
4. On the role's details page, click the **Trust relationships** tab, and on the **Trust relationships** tab click **Edit trust policy**.
5. On the **Edit trust policy** page, delete the existing JSON policy document, and paste the following IAM policy, in which you must replace `<cluster_EC2_iam_role_ARN>` with the EC2 instance role's ARN you have obtained above. Then, click **Update policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "<cluster_EC2_iam_role_ARN>"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

If you have created different assumed roles for accessing different AWS resources, you need to repeat the preceding steps to configure your other assumed roles. For example, you have created `s3_assumed_role` and `glue_assumed_role` for accessing AWS S3 and AWS Glue respectively. In this situation, you need to repeat the preceding steps to configure `glue_assumed_role`.

Configure your EC2 instance role as follows:

1. Sign in to the AWS IAM console.
2. In the left-side navigation pane, choose **Access management** > **Roles**.
3. Find the EC2 instance role and click its name.
4. In the **Permissions policies** section of the role's details page, click **Add permissions** and choose **Create inline policy**.
5. In the **Specify permissions** step, click the **JSON** tab, delete the existing JSON policy document, and paste the following IAM policy, in which you must replace `<s3_assumed_role_ARN>` with the ARN of the assumed role `s3_assumed_role`. Then, click **Review policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["sts:AssumeRole"],
            "Resource": [
                "<s3_assumed_role_ARN>"
            ]
        }
    ]
}
```

   If you have created different assumed roles for accessing different AWS resources, you need to fill the ARNs of all these assumed roles in the **Resource** element of the preceding IAM policy and separate them with a comma (,). For example, you have created `s3_assumed_role` and `glue_assumed_role` for accessing AWS S3 and AWS Glue respectively. In this situation, you need to fill the ARN of `s3_assumed_role` and

the ARN of `glue_assumed_role` in the **Resource** element by using the following format: `"<s3_assumed_role_ARN>","<glue_assumed_role_ARN>"`.

6. In the **Review Policy** step, enter a policy name and click **Create policy**.

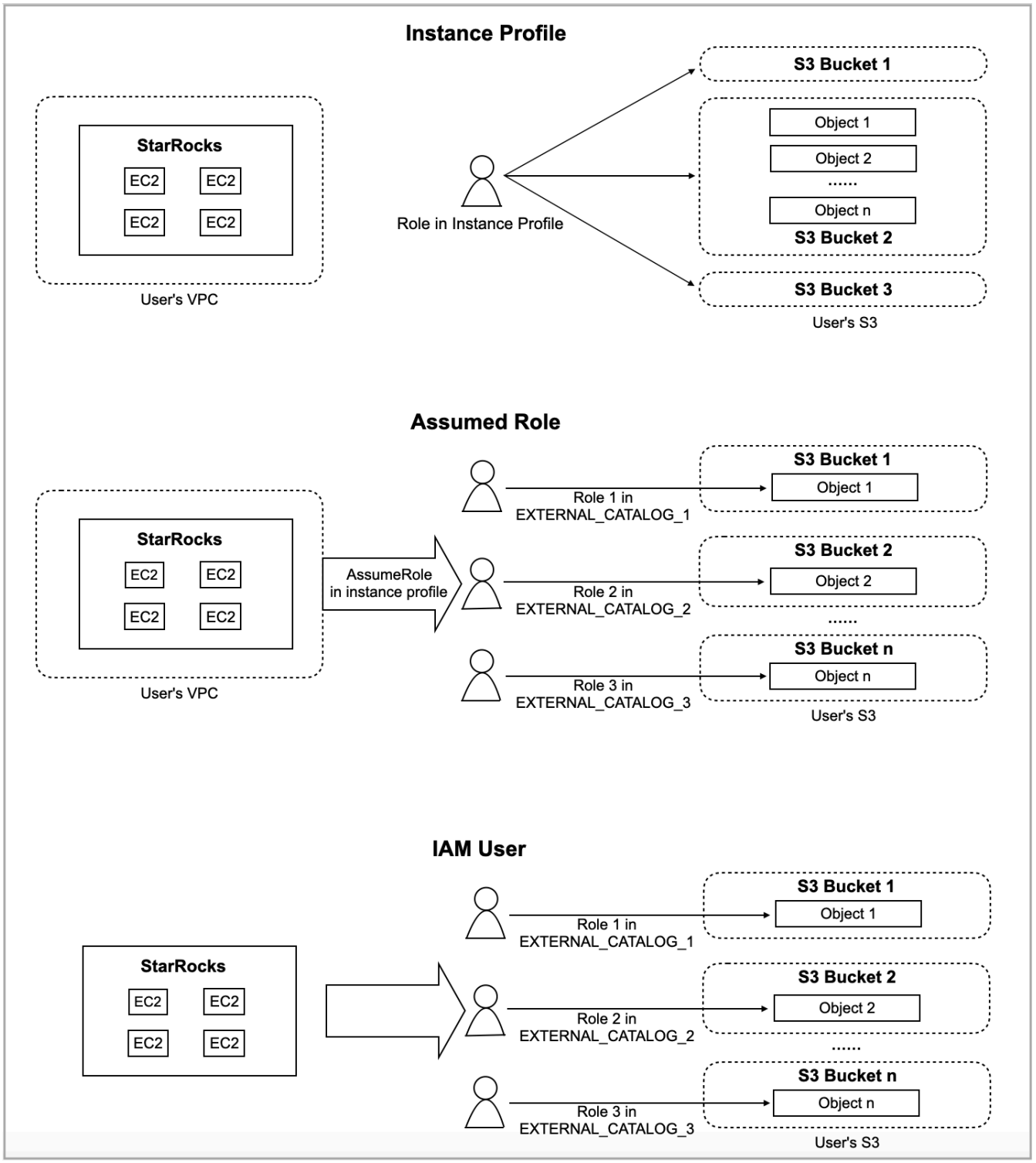
Preparation for IAM user-based authentication

Create an IAM user. See Creating an IAM user in your AWS account.

Then, attach the IAM policies for accessing the required AWS resources to the IAM user you create.

Comparison between authentication methods

The following figure provides a high-level explanation of the differences in mechanism between instance profile-based authentication, assumed role-based authentication, and IAM user-based authentication in StarRocks.



Build connections with AWS resources

Authentication parameters for accessing AWS S3

In various scenarios in which StarRocks needs to integrate with AWS S3, for example, when you create external catalogs or file external tables or when you ingest, back up, or restore data from AWS S3, configure the authentication parameters for accessing AWS S3 as follows:

- For instance profile-based authentication, set `aws.s3.use_instance_profile` to `true`.
- For assumed role-based authentication, set `aws.s3.use_instance_profile` to `true` and configure `aws.s3.iam_role_arn` as the assumed role's ARN that you use to access AWS S3 (for example, the ARN of the assumed role `s3_assumed_role` you have created above).
- For IAM user-based authentication, set `aws.s3.use_instance_profile` to `false` and configure `aws.s3.access_key` and `aws.s3.secret_key` as the access key and secret key of your AWS IAM user.

The following table describes the parameters.

Parameter	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication method. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you use the assumed role-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.access_key	No	The access key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.
aws.s3.secret_key	No	The secret key of your IAM user. If you use the IAM user-based authentication method to access AWS S3, you must specify this parameter.

Authentication parameters for accessing AWS Glue

In various scenarios in which StarRocks needs to integrate with AWS Glue, for example, when you create external catalogs, configure the authentication parameters for accessing AWS Glue as follows:

- For instance profile-based authentication, set `aws.glue.use_instance_profile` to `true`.
- For assumed role-based authentication, set `aws.glue.use_instance_profile` to `true` and configure `aws.glue.iam_role_arn` as the assumed role's ARN that you use to access AWS Glue (for example, the ARN of the assumed role `glue_assumed_role` you have created above).
- For IAM user-based authentication, set `aws.glue.use_instance_profile` to `false` and configure `aws.glue.access_key` and `aws.glue.secret_key` as the access key and secret key of your AWS IAM user.

The following table describes the parameters.

Parameter	Required	Description
aws.glue.use_instance_profile	Yes	Specifies whether to enable the instance profile-based authentication method and the assumed role-based authentication. Valid values: `true` and `false`. Default value: `false`.
aws.glue.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS Glue Data Catalog. If you use the assumed role-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.access_key	No	The access key of your AWS IAM user. If you use the IAM user-based authentication method to access AWS Glue, you must specify this parameter.
aws.glue.secret_key	No	The secret key of your AWS IAM user. If you use the IAM user-based

Parameter	Required	Description
		authentication method to access AWS Glue, you must specify this parameter.

Integration examples

External catalog

Creating an external catalog in your StarRocks cluster means building integration with the target data lake system, which is composed of two key components:

- File storage like AWS S3 to store table files
- Metastore like Hive metastore or AWS Glue to store the metadata and locations of table files

StarRocks supports the following types of catalogs:

- Hive catalog
- Iceberg catalog
- Hudi catalog
- Delta Lake catalog

The following examples create a Hive catalog named `hive_catalog_hms` or `hive_catalog_glue`, depending on the type of metastore you use, to query data from your Hive cluster. For detailed syntax and parameters, see Hive catalog.

Instance profile-based authentication

- If you use Hive metastore in your Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2",
```

```
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

- If you use AWS Glue in your Amazon EMR Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2"
);
```

Assumed role-based authentication

- If you use Hive metastore in your Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/s3_assumed_role",
    "aws.s3.region" = "us-west-2",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

- If you use AWS Glue in your Amazon EMR Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/s3_assumed_role",
    "aws.s3.region" = "us-west-2",
    "hive.metastore.type" = "glue",
```

```
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/glue_assumed_role",
    "aws.glue.region" = "us-west-2"
);
```

IAM user-based authentication

- If you use Hive metastore in your Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

- If you use AWS Glue in your Amazon EMR Hive cluster, run a command like below:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "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" = "us-west-2",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2"
);
```

File external table

File external tables must be created in your internal catalog named `default_catalog`.

The following examples create a file external table named `file_table` on an existing database named `test_s3_db`. For detailed syntax and parameters, see File external table.

Instance profile-based authentication

Run a command like below:

```
CREATE EXTERNAL TABLE test_s3_db.file_table
(
    id varchar(65500),
    attributes map<varchar(100), varchar(2000)>
)
ENGINE=FILE
PROPERTIES
(
    "path" = "s3://starrocks-test/",
    "format" = "ORC",
    "aws.s3.use_instance_profile"="true",
    "aws.s3.region"="us-west-2"
);
```

Assumed role-based authentication

Run a command like below:

```
CREATE EXTERNAL TABLE test_s3_db.file_table
(
    id varchar(65500),
    attributes map<varchar(100), varchar(2000)>
)
ENGINE=FILE
PROPERTIES
(
    "path" = "s3://starrocks-test/",
    "format" = "ORC",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/s3_assumed_role",
    "aws.s3.region" = "us-west-2"
);
```

IAM user-based authentication

Run a command like below:

```
CREATE EXTERNAL TABLE test_s3_db.file_table
(
    id varchar(65500),
    attributes map<varchar(100), varchar(2000)>
)
ENGINE=FILE
PROPERTIES
(
    "path" = "s3://starrocks-test/",
    "format" = "ORC",
    "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"="us-west-2"
);
```

Ingestion

You can use LOAD LABEL to load data from AWS S3.

The following examples load the data from all Parquet data files stored in the `s3a://test-bucket/test_brokerload_ingestion` path into the `test_ingestion_2` table in an existing database named `test_s3_db`. For detailed syntax and parameters, see BROKER LOAD.

Instance profile-based authentication

Run a command like below:

```
LOAD LABEL test_s3_db.test_credential_instanceprofile_7
(
    DATA INFILE("s3a://test-bucket/test_brokerload_ingestion/*")
    INTO TABLE test_ingestion_2
    FORMAT AS "parquet"
)
WITH BROKER
(
    "aws.s3.use_instance_profile"= "true",
    "aws.s3.region"="us-west-1"
)
```

```
PROPERTIES
(
    "timeout"="1200"
);
```

Assumed role-based authentication

Run a command like below:

```
LOAD LABEL test_s3_db.test_credential_instanceprofile_7
(
    DATA INFILE("s3a://test-bucket/test_brokerload_ingestion/*")
    INTO TABLE test_ingestion_2
    FORMAT AS "parquet"
)
WITH BROKER
(
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/s3_assumed_role",
    "aws.s3.region" = "us-west-1"
)
PROPERTIES
(
    "timeout"="1200"
);
```

IAM user-based authentication

Run a command like below:

```
LOAD LABEL test_s3_db.test_credential_instanceprofile_7
(
    DATA INFILE("s3a://test-bucket/test_brokerload_ingestion/*")
    INTO TABLE test_ingestion_2
    FORMAT AS "parquet"
)
WITH BROKER
(
    "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"="us-west-1"
```

```
)
PROPERTIES
(
    "timeout"="1200"
);
```

Hive Bitmap UDF

Hive Bitmap UDF provides UDFs that can be directly used in Hive. They can be used to generate Bitmap data and perform Bitmap-related calculations.

The Bitmap format defined by Hive Bitmap UDF is consistent with the format in StarRocks and can be directly used for loading Bitmap data into StarRocks and unloading Bitmap data from StarRocks to Hive.

Applicable scenarios:

- The amount of raw data is large and directly loading these data into StarRocks for computing will cause tremendous pressure on StarRocks clusters. The desired solution is generating Bitmap data in Hive and then loading Bitmap into StarRocks.
- Export the Bitmap data generated in StarRocks to Hive for other systems to use.

Supported source and target data types:

- v3.1 and later support loading and unloading data of these types: String, Base64, and Binary.
- v2.5 and v3.0 only support loading and unloading of String and Base64 data.

Hive Bitmap UDFs that can be generated

- com.starrocks.hive.udf.UDAFBitmapAgg

  Combines multiple rows of non-null values in a column into one row of Bitmap values, which is equivalent to StarRocks' built-in aggregate function bitmap_agg.

- com.starrocks.hive.udf.UDAFBitmapUnion

  Calculates the union of a set of bitmaps, which is equivalent to StarRocks' built-in aggregate function bitmap_union.

- com.starrocks.hive.udf.UDFBase64ToBitmap

Converts a base64-encoded string into a bitmap, which is equivalent to StarRocks' built-in function base64_to_bitmap.

- com.starrocks.hive.udf.UDFBitmapAnd

  Calculates the intersection of two bitmaps, which is equivalent to StarRocks' built-in function bitmap_and.

- com.starrocks.hive.udf.UDFBitmapCount

  Counts the number of values in the bitmap, which is equivalent to StarRocks' built-in function bitmap_count.

- com.starrocks.hive.udf.UDFBitmapFromString

  Converts a comma-separated string to a bitmap, equivalent to StarRocks' built-in function bitmap_from_string.

- com.starrocks.hive.udf.UDFBitmapOr

  Calculates the union of two bitmaps, equivalent to StarRocks' built-in function bitmap_or.

- com.starrocks.hive.udf.UDFBitmapToBase64

  Converts Bitmap to Base64 string, equivalent to StarRocks' built-in function bitmap_to_base64.

- com.starrocks.hive.udf.UDFBitmapToString

  Converts a bitmap to a comma-separated string, equivalent to StarRocks' built-in function bitmap_to_string.

- com.starrocks.hive.udf.UDFBitmapXor

  Calculates the set of unique elements in two Bitmaps, which is equivalent to StarRocks' built-in function bitmap_xor.

How to use

1. Compile and generate Hive UDF on the FE.

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

A JAR package `hive-udf-1.0.0.jar` will be generated in the `fe/hive-udf/` directory.

2. Upload the JAR package to HDFS.

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

3. Load the JAR package to Hive.

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

4. Load UDF functions.

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

Usage examples

Generate Bitmap in Hive and load it into StarRocks in Binary format

1. Create a Hive source table.

```
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. Create a Hive bitmap table.

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

   Hive generates bitmap through UDFBitmapAgg and writes it into the bitmap table.

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

3. Create a StarRocks Bitmap table.

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

4. Load Bitmap data into StarRocks via Hive Catalog.

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

5. View the results.

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

Export Bitmap from StarRocks to Hive

1. Create a Bitmap table in StarRocks and write data into this table.

```
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. Create a Bitmap table in Hive.

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

3. Export data in different ways.

   - Export data via INSERT INTO FILES (Binary format).

```
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;
```

   - Export data via Hive Catalog (Binary format).

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

4. View results in Hive.

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

Other tools and systems

This topic introduces the other tools and systems with which StarRocks can integrate.

Data integration

DBT

Apache Hive

Apache Iceberg

Apache Hudi

Delta Lake

Apache Flink Connector

Apache Spark Connector

Apache Kafka

Cluster management

Kubernetes

Helm

Monitor

DataDog

Administration

Management

9 items

Data Recovery

StarRocks supports data recovery for mistakenly deleted databases/tables/partitions. After drop ta...

Metadata Recovery

This topic describes how to recover the metadata in your StarRocks clusters when FE nodes encou...

User Privilege

Based on MySQL's permission management mechanism, StarRocks permission management syste...

Authentication methods

In addition to the authentication method of "username+password", StarRocks also supports LDAP.

Performance Tuning

3 items

Administration Management

Scale in and out

This topic describes how to scale in and out the node of StarRocks.

Back up and restore data

This topic describes how to back up and restore data in StarRocks, or migrate data to a new StarR...

Parameter configuration

This topic describes FE, BE, and system parameters. It also provides suggestions on how to config...

Monitor and Alerting

You can build your own monitoring services, or use the Prometheus + Grafana solution. StarRocks ...

Manage audit logs within StarRocks via AuditLoader

This topic describes how to manage StarRocks audit logs within a table via the plugin - AuditLoader.

Enable FQDN access

This topic describes how to enable cluster access by using a fully qualified domain name (FQDN). ...

Configure a time zone

This topic describes how to configure a time zone and the impacts of time zone settings.

Information Schema

The StarRocks informationschema is a database within each StarRocks instance. informationsche...

Resource management

8 items

Scale in and out

This topic describes how to scale in and out the node of StarRocks.

Scale FE in and out

StarRocks has two types of FE nodes: Follower and Observer. Followers are involved in election voting and writing. Observers are only used to synchronize logs and extend read performance.

> - The number of follower FEs (including leader) must be odd, and it is recommended to deploy 3 of them to form a High Availability (HA) mode.
> - When the FE is in high availability deployment (1 leader, 2 followers), it is recommended to add Observer FEs for better read performance. * Typically one FE node can work with 10-20 BE nodes. It is recommended that the total number of FE nodes be less than 10. Three is sufficient in most cases.

Scale FE out

After deploying the FE node and starting the service, run the following command to scale FE out.

```
alter system add follower "fe_host:edit_log_port";
alter system add observer "fe_host:edit_log_port";
```

Scale FE in

FE scale-in is similar to the scale-out. Run the following command to scale FE in.

```
alter system drop follower "fe_host:edit_log_port";
alter system drop observer "fe_host:edit_log_port";
```

After the expansion and contraction, you can view the node information by running `show proc '/frontends';`.

Scale BE in and out

After BE is scaled in or out, StarRocks will automatically perform load-balancing without affecting the overall performance.

Scale BE out

Run the following command to scale BE out.

```
alter system add backend 'be_host:be_heartbeat_service_port';
```

Run the following command to check the BE status.

```
show proc '/backends';
```

Scale BE in

There are two ways to scale in a BE node – `DROP` and `DECOMMISSION`.

`DROP` will delete the BE node immediately, and the lost duplicates will be made up by FE scheduling. `DECOMMISSION` will make sure the duplicates are made up first, and then drop the BE node. `DECOMMISSION` is a bit more friendly and is recommended for BE scale-in.

The commands of both methods are similar:

- `alter system decommission backend "be_host:be_heartbeat_service_port";`
- `alter system drop backend "be_host:be_heartbeat_service_port";`

Drop backend is a dangerous operation, so you need to confirm it twice before executing it

- `alter system drop backend "be_host:be_heartbeat_service_port";`

Back up and restore data

This topic describes how to back up and restore data in StarRocks, or migrate data to a new StarRocks cluster.

StarRocks supports backing up data as snapshots into a remote storage system, and restoring the data to any StarRocks clusters.

StarRocks supports the following remote storage systems:

- Apache™ Hadoop® (HDFS) cluster
- AWS S3
- Google GCS

Back up data

StarRocks supports FULL backup on the granularity level of database, table, or partition.

If you have stored a large amount of data in a table, we recommend that you back up and restore data by partition. This way, you can reduce the cost of retries in case of job failures. If you need to back up incremental data on a regular basis, you can strategize a dynamic partitioning plan (by a certain time interval, for example) for your table, and back up only new partitions each time.

Create a repository

Before backing up data, you need to create a repository, which is used to store data snapshots in a remote storage system. You can create multiple repositories in a StarRocks cluster. For detailed instructions, see CREATE REPOSITORY.

- Create a repository in HDFS

The following example creates a repository named `test_repo` in an HDFS cluster.

```
CREATE REPOSITORY test_repo
WITH BROKER
ON LOCATION "hdfs://<hdfs_host>:<hdfs_port>/repo_dir/backup"
PROPERTIES(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

- Create a repository in AWS S3

  You can choose IAM user-based credential (Access Key and Secret Key), Instance Profile, or Assumed Role as the credential method for accessing AWS S3.

  - The following example creates a repository named `test_repo` in the AWS S3 bucket `bucket_s3` using IAM user-based credential as the credential method.

```
CREATE REPOSITORY test_repo
WITH BROKER
ON LOCATION "s3a://bucket_s3/backup"
PROPERTIES(
    "aws.s3.access_key" = "XXXXXXXXXXXXXXXXX",
    "aws.s3.secret_key" = "yyyyyyyyyyyyyyyyyyyyyyyyy",
    "aws.s3.endpoint" = "s3.us-east-1.amazonaws.com"
);
```

  - The following example creates a repository named `test_repo` in the AWS S3 bucket `bucket_s3` using Instance Profile as the credential method.

```
CREATE REPOSITORY test_repo
WITH BROKER
ON LOCATION "s3a://bucket_s3/backup"
PROPERTIES(
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-east-1"
);
```

  - The following example creates a repository named `test_repo` in the AWS S3 bucket `bucket_s3` using Assumed Role as the credential method.

```
CREATE REPOSITORY test_repo
WITH BROKER
ON LOCATION "s3a://bucket_s3/backup"
```

```
PROPERTIES(
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::xxxxxxxxxx:role/yyyyyyyy",
    "aws.s3.region" = "us-east-1"
);
```

> **NOTE**
>
> StarRocks supports creating repositories in AWS S3 only according to the S3A protocol. Therefore, when you create repositories in AWS S3, you must replace `s3://` in the S3 URI you pass as a repository location in `ON LOCATION` with `s3a://`.

- Create a repository in Google GCS

The following example creates a repository named `test_repo` in the Google GCS bucket `bucket_gcs`.

```
CREATE REPOSITORY test_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"
);
```

> **NOTE**
>
> - StarRocks supports creating repositories in Google GCS only according to the S3A protocol. Therefore, when you create repositories in Google GCS, you must replace the prefix in the GCS URI you pass as a repository location in `ON LOCATION` with `s3a://`.
> - Do not specify `https` in the endpoint address.

After the repository is created, you can check the repository via SHOW REPOSITORIES. After restoring data, you can delete the repository in StarRocks using DROP REPOSITORY. However, data snapshots backed up in the remote storage system cannot be deleted through StarRocks. You need to delete them manually in the remote storage system.

Back up a data snapshot

After the repository is created, you need to create a data snapshot and back up it in the remote repository. For detailed instructions, see BACKUP.

The following example creates a data snapshot `sr_member_backup` for the table `sr_member` in the database `sr_hub` and backs up it in the repository `test_repo`.

```
BACKUP SNAPSHOT sr_hub.sr_member_backup
TO test_repo
ON (sr_member);
```

> **TIP**
>
> StarRocks supports BACKUP and RESTORE operations on the following levels of granularity:
>
> - Partition level: You need to specify the ON clause in the format `ON (<table_name> PARTITION (<partition_name>, ...))`.
> - Table level: You need to specify the ON clause in the format `ON (<table_name>)`.
> - Database level: You do not need to specify the ON clause. This will back up or restore the entire database.

BACKUP is an asynchronous operation. You can check the status of a BACKUP job using SHOW BACKUP, or cancel a BACKUP job using CANCEL BACKUP.

Restore or migrate data

You can restore the data snapshot backed up in the remote storage system to the current or other StarRocks clusters to restore or migrate data.

(Optional) Create a repository in the new cluster

To migrate data to another StarRocks cluster, you need to create a repository with the same **repository name** and **location** in the new cluster, otherwise you will not be able to view the previously backed up data snapshots. See Create a repository for details.

Check the snapshot

Before restoring data, you can check the snapshots in a specified repository using SHOW SNAPSHOT.

The following example checks the snapshot information in `test_repo`.

```
mysql> SHOW SNAPSHOT ON test_repo;
+-----------------+-------------------------+--------+
| Snapshot        | Timestamp               | Status |
+-----------------+-------------------------+--------+
| sr_member_backup | 2023-02-07-14-45-53-143 | OK     |
+-----------------+-------------------------+--------+
1 row in set (1.16 sec)
```

Restore data via the snapshot

You can use the RESTORE statement to restore data snapshots in the remote storage system to the current or other StarRocks clusters.

The following example restores the data snapshot `sr_member_backup` in `test_repo` on the table `sr_member`. It only restores ONE data replica.

```
RESTORE SNAPSHOT sr_hub.sr_member_backup
FROM test_repo
ON (sr_member)
PROPERTIES (
    "backup_timestamp"="2023-02-07-14-45-53-143",
    "replication_num" = "1"
);
```

> **TIP**
>
> StarRocks supports BACKUP and RESTORE operations on the following levels of granularity:
>
> - Partition level: You need to specify the ON clause in the format `ON (<table_name> PARTITION (<partition_name>, ...))`.
> - Table level: You need to specify the ON clause in the format `ON (<table_name>)`.

- Database level: You do not need to specify the ON clause. This will back up or restore the entire database.

RESTORE is an asynchronous operation. You can check the status of a RESTORE job using SHOW RESTORE, or cancel a RESTORE job using CANCEL RESTORE.

Configure BACKUP or RESTORE jobs

You can optimize the performance of BACKUP or RESTORE jobs by modifying the following configuration items in the BE configuration file **be.conf**:

Configuration item	Description
upload_worker_count	The maximum number of threads for the upload tasks of BACKUP jobs on a BE node. Default: `1`. Increase the value of this configuration item to increase the concurrency of the upload task.
download_worker_count	The maximum number of threads for the download tasks of RESTORE jobs on a BE node. Default: `1`. Increase the value of this configuration item to increase the concurrency of the download task.
max_download_speed_kbps	The upper limit of the download speed on a BE node. Default: `50000`. Unit: KB/s. Usually, the speed of the download tasks in RESTORE jobs will not exceed the default value. If this configuration is limiting the performance of RESTORE jobs, you can increase it according to your bandwidth.

Usage notes

- Only users with the ADMIN privilege can back up or restore data.
- In each database, only one running BACKUP or RESTORE job is allowed each time. Otherwise, StarRocks returns an error.

- Because BACKUP and RESTORE jobs occupy many resources of your StarRocks cluster, you can back up and restore your data while your StarRocks cluster is not heavily loaded.
- StarRocks does not support specifying data compression algorithm for data backup.
- Because data is backed up as snapshots, the data loaded upon snapshot generation is not included in the snapshot. Therefore, if you load data into the old cluster after the snapshot is generated and before the RESTORE job is completed, you also need to load the data into the cluster that data is restored into. It is recommended that you load data into both clusters in parallel for a period of time after the data migration is complete, and then migrate your application to the new cluster after verifying the correctness of the data and services.
- Before the RESTORE job is completed, you cannot operate the table to be restored.
- Primary Key tables cannot be restored to a StarRocks cluster earlier than v2.5.
- You do not need to create the table to be restored in the new cluster before restoring it. The RESTORE job automatically creates it.
- If there is an existing table that has a duplicated name with the table to be restored, StarRocks first checks whether or not the schema of the existing table matches that of the table to be restored. If the schemas match, StarRocks overwrites the existing table with the data in the snapshot. If the schema does not match, the RESTORE job fails. You can either rename the table to be restored using the keyword `AS`, or delete the existing table before restoring data.
- If the RESTORE job overwrites an existing database, table, or partition, the overwritten data cannot be restored after the job enters the COMMIT phase. If the RESTORE job fails or is canceled at this point, the data may be corrupted and inaccessible. In this case, you can only perform the RESTORE operation again and wait for the job to complete. Therefore, we recommend that you do not restore data by overwriting unless you are sure that the current data is no longer used. The overwrite operation first checks metadata consistency between the snapshot and the existing database, table, or partition. If an inconsistency is detected, the RESTORE operation cannot be performed.
- During a BACKUP or a RESTORE job, StarRocks automatically backs up or restores the Synchronous materialized view, which can still accelerate or rewrite your queries after data restoration. Currently, StarRocks does not support backing up views and Asynchronous materialized views. You can only back up the physical table of the materialized view, which cannot be used for query acceleration or query rewriting.
- Currently, StarRocks does not support backing up the configuration data related to user accounts, privileges, and resource groups.

- Currently, StarRocks does not support backing up and restoring the Colocate Join relationship among tables.

Parameter configuration

This topic describes FE, BE, and system parameters. It also provides suggestions on how to configure and tune these parameters.

FE configuration items

FE parameters are classified into dynamic parameters and static parameters.

- Dynamic parameters can be configured and adjusted by running SQL commands, which is very convenient. But the configurations become invalid if you restart your FE. Therefore, we recommend that you also modify the configuration items in the `fe.conf` file to prevent the loss of modifications.
- Static parameters can only be configured and adjusted in the FE configuration file **fe.conf**. **After you modify this file, you must restart your FE for the changes to take effect.**

Whether a parameter is a dynamic parameter is indicated by the `IsMutable` column in the output of ADMIN SHOW CONFIG. `TRUE` indicates a dynamic parameter.

Note that both dynamic and static FE parameters can be configured in the **fe.conf** file.

View FE configuration items

After your FE is started, you can run the ADMIN SHOW FRONTEND CONFIG command on your MySQL client to check the parameter configurations. If you want to query the configuration of a specific parameter, run the following command:

```
ADMIN SHOW FRONTEND CONFIG [LIKE "pattern"];
```

For detailed description of the returned fields, see ADMIN SHOW CONFIG.

> **NOTE**

> You must have administrator's privilege to run cluster administration-related commands.

Configure FE dynamic parameters

You can configure or modify the settings of FE dynamic parameters using ADMIN SET FRONTEND CONFIG.

```
ADMIN SET FRONTEND CONFIG ("key" = "value");
```

> **NOTE**
>
> The configurations will be restored to the default values in the `fe.conf` file after the FE restarts. Therefore, we recommend that you also modify the configuration items in `fe.conf` to prevent the loss of modifications.

Logging

Parameter	Unit	Default	Description
qe_slow_log_ms	ms	5000	The threshold used to determine whether a query is a slow query. If the response time of a query exceeds this threshold, it is recorded as a slow query in `fe.audit.log`.

Metadata and cluster management

Parameter	Unit	Default
catalog_try_lock_timeout_ms	ms	5000
edit_log_roll_num	-	50000

Parameter	Unit	Default
ignore_unknown_log_id	-	FALSE
ignore_materialized_view_error	-	FALSE

Parameter	Unit	Default
ignore_meta_check	-	FALSE

Parameter	Unit	Default
meta_delay_toleration_second	s	300
drop_backend_after_decommission	-	TRUE

Parameter	Unit	Default
enable_collect_query_detail_info	-	FALSE
enable_background_refresh_connector_metadata	-	`true` in v3.0 `false` in v2.5

Parameter	Unit	Default
background_refresh_metadata_interval_millis	ms	600000
background_refresh_metadata_time_secs_since_last_access_secs	s	86400

Parameter	Unit	Default	

Query engine

Parameter	Unit	Default	Description
max_allowed_in_element_num_of_delete	-	10000	The maximum number of elements allowed for the IN predicate in a DELETE statement.
enable_materialized_view	-	TRUE	Whether to enable the creation of materialized views.
enable_decimal_v3	-	TRUE	Whether to support the DECIMAL V3 data type.
expr_children_limit	-	10000	The maximum number of child expressions allowed in an expression.

Parameter	Unit	Default	Description
enable_sql_blacklist	-	FALSE	Whether to enable blacklist check for SQL queries. When this feature is enabled, queries in the blacklist cannot be executed.
dynamic_partition_check_interval_seconds	s	600	The interval at which new data is checked. If new data is detected, StarRocks automatically creates partitions for the data.
dynamic_partition_enable	-	TRUE	Whether to enable the dynamic partitioning feature. When this feature is enabled, StarRocks dynamically creates partitions for new data and automatically deletes expired partitions to

Parameter	Unit	Default	Description
			ensure the freshness of data.
max_partitions_in_one_batch	-	4096	The maximum number of partitions that can be created when you bulk create partitions.
max_query_retry_time	-	2	The maximum number of query retries on an FE.
max_create_table_timeout_second	s	600	The maximum timeout duration for creating a table, in seconds.
create_table_max_serial_replicas	-	128	The maximum number of replicas to create serially. If the actual replica count exceeds this, replicas will be created concurrently. Try to reduce this config if table creation takes a long

Parameter	Unit	Default	Description
			time to complete.
max_running_rollup_job_num_per_table	-	1	The maximum number of rollup jobs can run in parallel for a table.
max_planner_scalar_rewrite_num	-	100000	The maximum number of times that the optimizer can rewrite a scalar operator.
enable_statistic_collect	-	TRUE	Whether to collect statistics for the CBO. This feature is enabled by default.
enable_collect_full_statistic	-	TRUE	Whether to enable automatic full statistics collection. This feature is enabled by default.
statistic_auto_collect_ratio	-	0.8	The threshold for determining whether the statistics for automatic

Parameter	Unit	Default	Description
			collection are healthy. If statistics health is below this threshold, automatic collection is triggered.
statistic_max_full_collect_data_size	GB	100	The size of the largest partition for automatic collection to collect data. Unit: GB.If a partition exceeds this value, full collection is discarded and sampled collection is performed instead.
statistic_collect_interval_sec	s	300	The interval for checking data updates during automatic collection. Unit: seconds.
statistic_auto_analyze_start_time	STRING	00:00:00	The start time of automatic collection. Value range:

Parameter	Unit	Default	Description
			`00:00:00` - `23:59:59`.
statistic_auto_analyze_end_time	STRING	23:59:59	The end time of automatic collection. Value range: `00:00:00` - `23:59:59`.
statistic_sample_collect_rows	-	200000	The minimum number of rows to collect for sampled collection. If the parameter value exceeds the actual number of rows in your table, full collection is performed.
histogram_buckets_size	-	64	The default bucket number for a histogram.
histogram_mcv_size	-	100	The number of most common values (MCV) for a histogram.
histogram_sample_ratio	-	0.1	The sampling ratio for a histogram.

Parameter	Unit	Default	Description
histogram_max_sample_row_count	-	10000000	The maximum number of rows to collect for a histogram.
statistics_manager_sleep_time_sec	s	60	The interval at which metadata is scheduled. Unit: seconds. The system performs the following operations based on this interval: • Create tables for storing statistics. • Delete statistics that have been deleted. • Delete expired statistics.
statistic_update_interval_sec	s	24 * 60 * 60	The interval at which the cache of statistical information is updated. Unit: seconds.

Parameter	Unit	Default	Description
statistic_analyze_status_keep_second	s	259200	The duration to retain the history of collection tasks. The default value is 3 days. Unit: seconds.
statistic_collect_concurrency	-	3	The maximum number of manual collection tasks that can run in parallel. The value defaults to 3, which means you can run a maximum of three manual collection tasks in parallel. If the value is exceeded, incoming tasks will be in the PENDING state, waiting to be scheduled.
enable_local_replica_selection	-	FALSE	Whether to select local replicas for queries. Local replicas reduce the network transmission

Parameter	Unit	Default	Description
			cost. If this parameter is set to TRUE, the CBO preferentially selects tablet replicas on BEs that have the same IP address as the current FE. If this parameter is set to FALSE, both local replicas and non-local replicas can be selected. The default value is FALSE.
max_distribution_pruner_recursion_depth	-	100	The maximum recursion depth allowed by the partition pruner. Increasing the recursion depth can prune more elements but also increases CPU consumption.
enable_udf	-	FALSE	Whether to enable UDF .

Loading and unloading

Parameter	Unit	Default
max_broker_load_job_concurrency	-	5
load_straggler_wait_second	s	300
desired_max_waiting_jobs	-	1024
max_load_timeout_second	s	259200

Parameter	Unit	Default
min_load_timeout_second	s	1
max_running_txn_num_per_db	-	100
load_parallel_instance_num	-	1
disable_load_job	-	FALSE

Parameter	Unit	Default
history_job_keep_max_second	s	604800
label_keep_max_num	-	1000
label_keep_max_second	s	259200
max_routine_load_job_num	-	100
max_routine_load_task_concurrent_num	-	5
max_routine_load_task_num_per_be	-	5
max_routine_load_batch_size	Byte	4294967296

Parameter	Unit	Default
routine_load_task_consume_second	s	15
routine_load_task_timeout_second	s	60
routine_load_unstable_threshold_second	s	3600
max_tolerable_backend_down_num	-	0
period_of_auto_resume_min	Min	5
spark_load_default_timeout_second	s	86400
spark_home_default_dir	-	StarRocksFE.STARROCKS_HOME_DI + "/lib/spark2x"
stream_load_default_timeout_second	s	600
max_stream_load_timeout_second	s	259200

Parameter	Unit	Default
insert_load_default_timeout_second	s	3600
broker_load_default_timeout_second	s	14400
min_bytes_per_broker_scanner	Byte	67108864
max_broker_concurrency	-	100
export_max_bytes_per_be_per_task	Byte	268435456
export_running_job_num_limit	-	5
export_task_default_timeout_second	s	7200

Parameter	Unit	Default
empty_load_as_error	-	TRUE
external_table_commit_timeout_ms	ms	10000

Storage

Parameter	Unit	Default	
enable_strict_storage_medium_check	-	FALSE	Whether the medium of BE parameter is storage medi and returns a BE is differen parameter sp statement. Fc specified in th but the actua a result, the t is `FALSE`, the medium of BE

Parameter	Unit	Default	
enable_auto_tablet_distribution	-	TRUE	Whether to au buckets. • If this par need to s you creat StarRock number c automati see Dete • If this par manually when you you do no adding a partition creation manually the new p Starting from setting this pa
storage_usage_soft_limit_percent	%	90	Soft limit of th directory. If th the BE storag the remaining `storage_usa` tablets canno
storage_usage_soft_limit_reserve_bytes	Byte	200 * 1024 * 1024 * 1024	Soft limit of th directory. If th storage direc storage usag `storage_usa` cannot be clo

Parameter	Unit	Default	
storage_usage_hard_limit_percent	%	95	Hard limit of t BE directory. of the BE stor and the rema `storage_usa` Load and Res set this item t item `storage` allow the con
storage_usage_hard_limit_reserve_bytes	Byte	100 * 1024 * 1024 * 1024	Hard limit of t directory. If th storage direc storage usage `storage_usa` and Restore j this item toge `storage_fl` to allow the c
catalog_trash_expire_second	s	86400	The longest c retained after this duration cannot be rec
alter_table_timeout_second	s	86400	The timeout c operation (AL
recover_with_empty_tablet	-	FALSE	Whether to re replica with a or corrupted, healthy tablet corrupted tab ensures that t However, the data is lost. T

Parameter	Unit	Default	
			means lost or replaced with
tablet_create_timeout_second	s	10	The timeout d seconds.
tablet_delete_timeout_second	s	2	The timeout d seconds.
check_consistency_default_timeout_second	s	600	The timeout d check. You ca size of your ta
tablet_sched_slot_num_per_path	-	8	The maximum can run concu The alias is `s` From v2.5 onv parameter is
tablet_sched_max_scheduling_tablets	-	10000	The maximum scheduled at exceeded, tab be skipped.
tablet_sched_disable_balance	-	FALSE	Whether to di indicates that indicates that alias is `disab`
tablet_sched_disable_colocate_balance	-	FALSE	Whether to di Table. `TRUE` i disabled. `FAL` enabled. The `disable_co`
tablet_sched_max_balancing_tablets	-	500	The maximum balanced at th

Parameter	Unit	Default	
			exceeded, tab The alias is `m`

tablet_sched_balance_load_disk_safe_threshold	-	0.5	The threshold
			disk usage is
			effect only wh
			`tablet_sch`
			`disk_and_ta`
			lower than 50
			balanced. For
			the difference
			disk usage is
			considered un
			triggered. The
			`balance_loa`

tablet_sched_balance_load_score_threshold	-	0.1	The threshold load is balanc when `tablet` set to `be_loa` lower than the and a BE who average load `balance_loa`
tablet_sched_repair_delay_factor_second	s	60	The interval a seconds. The `tablet_repa`
tablet_sched_min_clone_task_timeout_sec	s	3 * 60	The minimum tablet, in seco
tablet_sched_max_clone_task_timeout_sec	s	2 * 60 * 60	The maximum tablet, in seco `max_clone_t`

Parameter	Unit	Default	
tablet_sched_max_not_being_scheduled_interval_ms	ms	15 * 60 * 100	When the tab if a tablet has specified time it a higher pri possible.
tablet_sched_be_down_tolerate_time_s	s	900	The maximum BE node to re threshold is r be migrated t

Other FE dynamic parameters

Parameter	Unit	Default	Description
plugin_enable	-	TRUE	Whether plugins can be installed on FEs. Plugins can be installed or uninstalled only on the Leader FE.
max_small_file_number	-	100	The maximum number of small files that can be stored on an FE directory.
max_small_file_size_bytes	Byte	1024 * 1024	The maximum size of a small file, in bytes.

Parameter	Unit	Default	Description
agent_task_resend_wait_time_ms	ms	5000	The duration the FE must wait before it can resend an agent task. An agent task can be resent only when the gap between the task creation time and the current time exceeds the value of this parameter. This parameter is used to prevent repetitive sending of agent tasks. Unit: ms.
backup_job_default_timeout_ms	ms	86400*1000	The timeout duration of a backup job, in ms. If this value is exceeded, the backup job fails.
report_queue_size	-	100	The maximum number of

Parameter	Unit	Default	Description
			jobs that can wait in a report queue. The report is about disk, task, and tablet information of BEs. If too many report jobs are piling up in a queue, OOM will occur.
enable_experimental_mv	-	TRUE	Whether to enable the asynchronous materialized view feature. `TRUE` indicates this feature is enabled. From v2.5.2 onwards, this feature is enabled by default. For versions earlier than v2.5.2, this feature is disabled by default.

Parameter	Unit	Default	Description
authentication_ldap_simple_bind_base_dn	-	Empty string	The base DN, which is the point from which the LDAP server starts to search for users' authentication information.
authentication_ldap_simple_bind_root_dn	-	Empty string	The administrator DN used to search for users' authentication information.
authentication_ldap_simple_bind_root_pwd	-	Empty string	The password of the administrator used to search for users' authentication information.
authentication_ldap_simple_server_host	-	Empty string	The host on which the LDAP server runs.
authentication_ldap_simple_server_port	-	389	The port of the LDAP server.

Parameter	Unit	Default	Description
authentication_ldap_simple_user_search_attr	-	uid	The name of the attribute that identifies users in LDAP objects.
max_upload_task_per_be	-	0	In each BACKUP operation, the maximum number of upload tasks StarRocks assigned to a BE node. When this item is set to less than or equal to 0, no limit is imposed on the task number. This item is supported from v2.5.7 onwards.
max_download_task_per_be	-	0	In each RESTORE operation, the maximum number of download tasks StarRocks

Parameter	Unit	Default	Description
			assigned to a BE node. When this item is set to less than or equal to 0, no limit is imposed on the task number. This item is supported from v2.5.7 onwards.
allow_system_reserved_names	FALSE	Whether to allow users to create columns whose names are initiated with `__op` and `__row`. To enable this feature, set this parameter to `TRUE`. Please note that these name formats are reserved for special purposes in StarRocks and creating	

Parameter	Unit	Default	Description
		such columns may result in undefined behavior. Therefore this feature is disabled by default. This item is supported from v2.5.15 onwards.	
default_mv_refresh_immediate	TRUE	Whether to refresh an asynchronous materialized view immediately after creation. When this item is set to `true`, newly created materialized view will be refreshed immediately. This item is supported from v2.5.18 onwards.	

Configure FE static parameters

This section provides an overview of the static parameters that you can configure in the FE configuration file **fe.conf**. After you reconfigure these parameters for an FE, you must restart the FE for the changes to take effect.

Logging

Parameter	Default	Descr
log_roll_size_mb	1024	The size per log fil default value `1024` size per log file as
sys_log_dir	StarRocksFE.STARROCKS_HOME_DIR + "/log"	The directory that files.
sys_log_level	INFO	The severity levels system log entries Valid values: `INFO` and `FATAL`.
sys_log_verbose_modules	Empty string	The modules for w generates system parameter is set to `org.apache.sta` StarRocks generat only for the catalo

Parameter	Default	Descr
sys_log_roll_interval	DAY	The time interval a rotates system log values: `DAY` and `H` • If this parame a suffix in the is added to th system log file • If this parame a suffix in the format is adde system log file
sys_log_delete_age	7d	The retention perio files. The default v that each system l retained for 7 days checks each syste deletes those that days ago.
sys_log_roll_num	10	The maximum num files that can be re retention period sp `sys_log_roll_i` parameter.
audit_log_dir	StarRocksFE.STARROCKS_HOME_DIR + "/log"	The directory that files.
audit_log_roll_num	90	The maximum num files that can be re retention period sp `audit_log_roll` parameter.

Parameter	Default	Descr
audit_log_modules	slow_query, query	The modules for w generates audit lo default, StarRocks logs for the slow_c the query module. module names wit a space.
audit_log_roll_interval	DAY	The time interval a rotates audit log e values: `DAY` and `H` • If this parame a suffix in the is added to th log files. • If this parame a suffix in the format is adde audit log files.
audit_log_delete_age	30d	The retention perio files. The default v specifies that each be retained for 30 checks each audit deletes those that 30 days ago.
dump_log_dir	StarRocksFE.STARROCKS_HOME_DIR + "/log"	The directory that files.
dump_log_modules	query	The modules for w generates dump lo default, StarRocks logs for the the qu

Parameter	Default	Descr
		Separate the modu comma (,) and a sp
dump_log_roll_interval	DAY	The time interval a rotates dump log e values: `DAY` and `H` • If this parame a suffix in the is added to th log files. • If this parame a suffix in the format is adde dump log files
dump_log_roll_num	10	The maximum num files that can be re retention period sp `dump_log_roll_` parameter.
dump_log_delete_age	7d	The retention perio files. The default v that each dump lo retained for 7 days checks each dump deletes those that days ago.

Server

Parameter	Default	Description
frontend_address	0.0.0.0	The IP address of the FE node.

Parameter	Default	Description
priority_networks	Empty string	Declares a selection strategy fo that have multiple IP addresses at most one IP address must ma list specified by this parameter. of this parameter is a list that co entries, which are separated wi semicolons (;) in CIDR notation, 10.10.10.0/24. If no IP address m the entries in this list, an IP add be randomly selected.
http_port	8030	The port on which the HTTP se FE node listens.
http_worker_threads_num	0	Number of worker threads for h to deal with http requests. For a or 0 value, the number of threa twice the number of cpu cores. Introduced in: 2.5.18， 3.0.10， 3 3.2.2.
http_backlog_num	1024	The length of the backlog queu the HTTP server in the FE node
cluster_name	StarRocks Cluster	The name of the StarRocks clus which the FE belongs. The clust is displayed for `Title` on the w
rpc_port	9020	The port on which the Thrift ser FE node listens.
thrift_backlog_num	1024	The length of the backlog queu the Thrift server in the FE node.
thrift_server_max_worker_threads	4096	The maximum number of worke that are supported by the Thrift

Parameter	Default	Description
		the FE node.
thrift_client_timeout_ms	5000	The length of time after which i connections time out. Unit: ms.
thrift_server_queue_size	4096	The length of queue where requ pending. If the number of threa are being processed in the thrif exceeds the value specified in `thrift_server_max_worker_` new requests are added to the queue.
brpc_idle_wait_max_time	10000	The maximum length of time fo bRPC clients wait as in the idle Unit: ms.
query_port	9030	The port on which the MySQL s the FE node listens.
mysql_service_nio_enabled	TRUE	Specifies whether asynchronou enabled for the FE node.
mysql_service_io_threads_num	4	The maximum number of threac can be run by the MySQL serve node to process I/O events.
mysql_nio_backlog_num	1024	The length of the backlog queu the MySQL server in the FE nod
max_mysql_service_task_threads_num	4096	The maximum number of threac can be run by the MySQL serve node to process tasks.
mysql_server_version	5.1.0	The MySQL server version retu the client. Modifying this param affect the version information ir

Parameter	Default	Description
		following situations: 1. `select version();` 2. Handshake pac version 3. Value of the global va `version` (`show variables l 'version';`)
max_connection_scheduler_threads_num	4096	The maximum number of thread supported by the connection sc
qe_max_connection	1024	The maximum number of conne that can be established by all us FE node.
check_java_version	TRUE	Specifies whether to check vers compatibility between the execu compiled Java programs. If the are incompatible, StarRocks rep errors and aborts the startup of programs.

Metadata and cluster management

Parameter	Default	D
meta_dir	StarRocksFE.STARROCKS_HOME_DIR + "/meta"	The dire metadat
heartbeat_mgr_threads_num	8	The nun that can Heartbe run hea
heartbeat_mgr_blocking_queue_size	1024	The size queue th heartbe the Hea

Parameter	Default	D
metadata_failure_recovery	FALSE	Specifie forcibly metadat Exercise you set
edit_log_port	9010	The por commu the lead observe StarRoc
edit_log_type	BDB	The typ can be the valu
bdbje_heartbeat_timeout_second	30	The am after wh heartbe leader, f observe StarRoc out. Uni
bdbje_lock_timeout_second	1	The am after wh BDB JE- out. Uni
max_bdbje_clock_delta_ms	5000	The ma offset th betwee and the observe StarRoc ms.

Parameter	Default	D
txn_rollback_limit	100	The ma of trans be rolle
bdbje_replica_ack_timeout_second	10	The ma of time leader F ACK me specifie follower metada the lead follower second. amount being w follower long tim can retu messag FE, caus timeout metada the FE p recomm increase this para prevent
master_sync_policy	SYNC	The poli which th flushes parame when th leader F

Parameter	Default	D
		• `SYN` trar con ent and sim • `NO_` ger flus ent at t whe is c • `WRI` Wh is c ent sim is n disk If you ha only one recomm this para If you ha three or FEs, we you set and the `replic` both to `WRITE_`
replica_sync_policy	SYNC	The poli which th

Parameter	Default	D
		flushes paramet when th follower • `SYN` trar con ent and sim • `NO_` ger flus ent at t whe is c • `WRI` Wh is c ent sim is n disk
replica_ack_policy	SIMPLE_MAJORITY	The poli which a conside default `SIMPLE` specifie is consi majority return A

Parameter	Default	D
cluster_id	-1	The ID c cluster t belongs have the belong t StarRoc values: integer. value `-1` StarRoc a rando the Star the time leader F is starte time.

Query engine

Parameter	Default	Description
publish_version_interval_ms	10	The time interval at which release validation tasks are issued. Unit: ms.
statistic_cache_columns	100000	The number of rows that can be cached for the statistics table.
statistic_cache_thread_pool_size	10	The size of the thread-pool which will be used to refresh statistic caches.

Loading and unloading

Parameter	Default	

async_load_task_pool_size	2	The size o
		This parar
		Load. The
		`max_runn`
		v2.5 onwa
		renamed
		`max_brok`
		which is a
		the defaul
		`2`.

load_checker_interval_second	5	The time i processec second.
transaction_clean_interval_second	30	The time i transactio second. W specify a s that finish cleaned u
label_clean_interval_second	14400	The time i cleaned u recommer time interv labels can manner.
spark_dpp_version	1.0.0	The versic Pruning (D
spark_resource_path	Empty string	The root d dependen

Parameter	Default	
spark_launcher_log_dir	sys_log_dir + "/spark_launcher_log"	The direct files.
yarn_client_path	StarRocksFE.STARROCKS_HOME_DIR + "/lib/yarn-client/hadoop/bin/yarn"	The root d package.
yarn_config_dir	StarRocksFE.STARROCKS_HOME_DIR + "/lib/yarn-config"	The direct configurat
export_checker_interval_second	5	The time i scheduled
export_task_pool_size	5	The size o

Storage

Parameter	Default	Description
default_storage_medium	HDD	The default storage media is used for a table or partit at the time of table or part creation if no storage med specified. Valid values: `HD` and `SSD`. When you creat table or partition, the defa storage media specified b parameter is used if you d specify a storage media ty for the table or partition.
tablet_sched_balancer_strategy	disk_and_tablet	The policy based on which balancing is implemented among tablets. The alias o parameter is `tablet_balancer_strat` Valid values:

Parameter	Default	Description
		`disk_and_tablet` and `be_load_score`.
tablet_sched_storage_cooldown_second	-1	The latency of automatic cooling starting from the t of table creation. The alias this parameter is `storage_cooldown_sec` Unit: second. The default `-1` specifies that automat cooling is disabled. If you to enable automatic coolin set this parameter to a val greater than `-1`.
tablet_stat_update_interval_second	300	The time interval at which FE retrieves tablet statistic from each BE. Unit: secon

Other FE static parameters

Parameter	Default	Descri
plugin_dir	STARROCKS_HOME_DIR/plugins	The dire that stor plugin installati package
small_file_dir	StarRocksFE.STARROCKS_HOME_DIR + "/small_files"	The root directory small file
max_agent_task_threads_num	4096	The max number threads

Parameter	Default	Descri
		are allow the ager thread p
auth_token	Empty string	The toke that is us for ident authenti within th StarRocl cluster t which th belongs. this paramet left unspecif StarRocl generate random for the c at the tir when th leader F the clust started f first time
tmp_dir	StarRocksFE.STARROCKS_HOME_DIR + "/temp_dir"	The dire that stor tempora files suc files generate during backup a restore

Parameter	Default	Descri
		procedu After the procedu finish, th generate tempora files are deleted.
locale	zh_CN.UTF-8	The cha set that used by FE.
hive_meta_load_concurrency	4	The max number concurre threads are supp for Hive metadat
hive_meta_cache_refresh_interval_s	7200	The time interval a which th cached metadat Hive ext tables is updated second.
hive_meta_cache_ttl_s	86400	The amo of time a which th cached metadat

Parameter	Default	Descri
		Hive ext tables expires. second.
hive_meta_store_timeout_s	10	The amc of time a which a connect a Hive metasto times ou Unit: sec
es_state_sync_interval_second	10	The time interval a which th obtains Elasticse indexes synchro the meta of StarR external tables. U second.
enable_auth_check	TRUE	Specifie whether enable t authenti check fe Valid val `TRUE` ar `FALSE`. specifie enable t

Parameter	Default	Descri
		feature, `FALSE` specifies disable t feature.
enable_metric_calculator	TRUE	Specifies whether enable th feature t used to periodic collect metrics. values: `T` and `FAL` `TRUE` specifies enable th feature, `FALSE` specifies disable t feature.

BE configuration items

Some BE configuration items are dynamic parameters which you can set them by commands when BE nodes are still online. The rest of them are static parameters. You can only set the static parameters of a BE node by changing them in the corresponding configuration file **be.conf**, and restart the BE node to allow the change to take effect.

View BE configuration items

You can view the BE configuration items using the following command:

```
curl http://<BE_IP>:<BE_HTTP_PORT>/varz
```

Configure BE dynamic parameters

You can configure a dynamic parameter of a BE node by using the `curl` command.

```
curl -XPOST http://be_host:http_port/api/update_config?
<configuration_item>=<value>
```

BE dynamic parameters are as follows.

Configuration item	Default	Unit
enable_stream_load_verbose_log	false	N/A
report_task_interval_seconds	10	Second
report_disk_state_interval_seconds	60	Second
report_tablet_interval_seconds	60	Second
report_workgroup_interval_seconds	5	Second
max_download_speed_kbps	50000	KB/s
download_low_speed_limit_kbps	50	KB/s

Configuration item	Default	Unit
download_low_speed_time	300	Second
status_report_interval	5	Second
scanner_thread_pool_thread_num	48	N/A
thrift_client_retry_interval_ms	100	ms
scanner_thread_pool_queue_size	102400	N/A
scanner_row_num	16384	N/A
max_scan_key_num	1024	N/A
max_pushdown_conditions_per_column	1024	N/A
exchg_node_buffer_size_bytes	10485760	Byte

Configuration item	Default	Unit
memory_limitation_per_thread_for_schema_change	2	GB
update_cache_expire_sec	360	Second
file_descriptor_cache_clean_interval	3600	Second
disk_stat_monitor_interval	5	Second
unused_rowset_monitor_interval	30	Second
max_percentage_of_error_disk	0	%
default_num_rows_per_column_file_block	1024	N/A
pending_data_expire_time_sec	1800	Second
inc_rowset_expired_sec	1800	Second
tablet_rowset_stale_sweep_time_sec	1800	Second
snapshot_expire_time_sec	172800	Second

Configuration item	Default	Unit
trash_file_expire_time_sec	86,400	Second
base_compaction_check_interval_seconds	60	Second
min_base_compaction_num_singleton_deltas	5	N/A
max_base_compaction_num_singleton_deltas	100	N/A
base_compaction_interval_seconds_since_last_operation	86400	Second
cumulative_compaction_check_interval_seconds	1	Second
update_compaction_check_interval_seconds	60	Second
min_compaction_failure_interval_sec	120	Second
max_compaction_concurrency	-1	N/A
periodic_counter_update_period_ms	500	ms

Configuration item	Default	Unit
pindex_major_compaction_limit_per_disk	1	N/A
load_error_log_reserve_hours	48	Hour
streaming_load_max_mb	10240	MB
streaming_load_max_batch_size_mb	100	MB
memory_maintenance_sleep_time_s	10	Second
write_buffer_size	104857600	Byte
tablet_stat_cache_update_interval_second	300	Second
result_buffer_cancelled_interval_time	300	Second
thrift_rpc_timeout_ms	5000	ms
txn_commit_rpc_timeout_ms	20000	ms
max_consumer_num_per_group	3	N/A

Configuration item	Default	Unit
max_memory_sink_batch_count	20	N/A
scan_context_gc_interval_min	5	Minute
path_gc_check_step	1000	N/A
path_gc_check_step_interval_ms	10	ms
path_scan_interval_second	86400	Second
storage_flood_stage_usage_percent	95	%
storage_flood_stage_left_capacity_bytes	107374182400	Byte
tablet_meta_checkpoint_min_new_rowsets_num	10	N/A

Configuration item	Default	Unit
tablet_meta_checkpoint_min_interval_secs	600	Second
max_runnings_transactions_per_txn_map	100	N/A
tablet_max_pending_versions	1000	N/A
tablet_max_versions	1000	N/A
max_hdfs_file_handle	1000	N/A
parquet_buffer_stream_reserve_size	1048576	Byte
be_exit_after_disk_write_hang_second	60	second
min_cumulative_compaction_failure_interval_sec	30	second
size_tiered_level_num	7	N/A
size_tiered_level_multiple	5	N/A

Configuration item	Default	Unit
size_tiered_min_level_size	131072	Byte
storage_page_cache_limit	20%	N/A

Configure BE static parameters

You can only set the static parameters of a BE by changing them in the corresponding configuration file **be.conf**, and restart the BE to allow the changes to take effect.

BE static parameters are as follows.

be_port

- **Default**: 9060
- **Unit**: N/A
- **Description**: The BE thrift server port, which is used to receive requests from FEs.

brpc_port

- **Default**: 8060
- **Unit**: N/A
- **Description**: The BE bRPC port, which is used to view the network statistics of bRPCs.

brpc_num_threads

- **Default**: -1
- **Unit**: N/A

- **Description**: The number of bthreads of a bRPC. The value -1 indicates the same number with the CPU threads.

priority_networks

- **Default**: Empty string
- **Unit**: N/A
- **Description**: The CIDR-formatted IP address that is used to specify the priority IP address of a BE node if the machine that hosts the BE node has multiple IP addresses.

heartbeat_service_port

- **Default**: 9050
- **Unit**: N/A
- **Description**: The BE heartbeat service port, which is used to receive heartbeats from FEs.

heartbeat_service_thread_count

- **Default**: 1
- **Unit**: N/A
- **Description**: The thread count of the BE heartbeat service.

create_tablet_worker_count

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads used to create a tablet.

drop_tablet_worker_count

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads used to drop a tablet.

push_worker_count_normal_priority

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads used to handle a load task with NORMAL priority.

push_worker_count_high_priority

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads used to handle a load task with HIGH priority.

transaction_publish_version_worker_count

- **Default**: 0
- **Unit**: N/A
- **Description**: The maximum number of threads used to publish a version. When this value is set to less than or equal to 0, the system uses half of the CPU core count as the value, so as to avoid insufficient thread resources when import concurrency is high but only a fixed number of threads are used. From v2.5, the default value has been changed from 8 to 0.

clear_transaction_task_worker_count

- **Default**: 1
- **Unit**: N/A
- **Description**: The number of threads used for clearing transaction.

alter_tablet_worker_count

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads used for schema change.

clone_worker_count

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads used for clone.

storage_medium_migrate_count

- **Default**: 1
- **Unit**: N/A
- **Description**: The number of threads used for storage medium migration (from SATA to SSD).

check_consistency_worker_count

- **Default**: 1
- **Unit**: N/A
- **Description**: The number of threads used for checking the consistency of tablets.

sys_log_dir

- **Default**: `${STARROCKS_HOME}/log`
- **Unit**: N/A
- **Description**: The directory that stores system logs (including INFO, WARNING, ERROR, and FATAL).

user_function_dir

- **Default**: `${STARROCKS_HOME}/lib/udf`
- **Unit**: N/A
- **Description**: The directory used to store User-defined Functions (UDFs).

small_file_dir

- **Default**: `${STARROCKS_HOME}/lib/small_file`
- **Unit**: N/A
- **Description**: The directory used to store the files downloaded by the file manager.

sys_log_level

- **Default**: INFO
- **Unit**: N/A
- **Description**: The severity levels into which system log entries are classified. Valid values: INFO, WARN, ERROR, and FATAL.

sys_log_roll_mode

- **Default**: SIZE-MB-1024
- **Unit**: N/A
- **Description**: The mode in which system logs are segmented into log rolls. Valid values include `TIME-DAY`, `TIME-HOUR`, and `SIZE-MB-size`. The default value indicates that logs are segmented into rolls, each of which is 1 GB.

sys_log_roll_num

- **Default**: 10
- **Unit**: N/A
- **Description**: The number of log rolls to reserve.

sys_log_verbose_modules

- **Default**: Empty string
- **Unit**: N/A
- **Description**: The module of the logs to be printed. For example, if you set this configuration item to OLAP, StarRocks only prints the logs of the OLAP module. Valid values are namespaces in BE, including `starrocks`, `starrocks::vectorized`, and `pipeline`

sys_log_verbose_level

- **Default**: 10
- **Unit**: N/A
- **Description**: The level of the logs to be printed. This configuration item is used to control the output of logs initiated with VLOG in codes.

log_buffer_level

- **Default**: Empty string
- **Unit**: N/A
- **Description**: The strategy for flushing logs. The default value indicates that logs are buffered in memory. Valid values are `-1` and `0`. `-1` indicates that logs are not buffered in memory.

num_threads_per_core

- **Default**: 3
- **Unit**: N/A
- **Description**: The number of threads started on each CPU core.

compress_rowbatches

- **Default**: TRUE
- **Unit**: N/A

- **Description**: A boolean value to control whether to compress the row batches in RPCs between BEs. TRUE indicates compressing the row batches, and FALSE indicates not compressing them.

serialize_batch

- **Default**: FALSE
- **Unit**: N/A
- **Description**: A boolean value to control whether to serialize the row batches in RPCs between BEs. TRUE indicates serializing the row batches, and FALSE indicates not serializing them.

storage_root_path

- **Default**: `${STARROCKS_HOME}/storage`
- **Unit**: N/A
- **Description**: The directory and medium of the storage volume.
  - Multiple volumes are separated by semicolons (`;`).
  - If the storage medium is SSD, add `medium:ssd` at the end of the directory.
  - If the storage medium is HDD, add `medium:hdd` at the end of the directory.

max_tablet_num_per_shard

- **Default**: 1024
- **Unit**: N/A
- **Description**: The maximum number of tablets in each shard. This configuration item is used to restrict the number of tablet child directories under each storage directory.

max_garbage_sweep_interval

- **Default**: 3600
- **Unit**: Second
- **Description**: The maximum time interval for garbage collection on storage volumes.

min_garbage_sweep_interval

- **Default**: 180
- **Unit**: Second
- **Description**: The minimum time interval for garbage collection on storage volumes.

row_nums_check

- **Default**: True
- **Unit**: N/A
- **Description**: A boolean value to control whether to check row counts before and after compaction. The value true indicates to enable the row count check. The value false indicates disable the row count check.

file_descriptor_cache_capacity

- **Default**: 16384
- **Unit**: N/A
- **Description**: The number of file descriptors that can be cached.

min_file_descriptor_number

- **Default**: 60000
- **Unit**: N/A
- **Description**: The minimum number of file descriptors in the BE process.

index_stream_cache_capacity

- **Default**: 10737418240
- **Unit**: Byte
- **Description**: The cache capacity for the statistical information of BloomFilter, Min, and Max.

disable_storage_page_cache

- **Default**: FALSE
- **Unit**: N/A
- **Description**: A boolean value to control whether to disable PageCache.
    - When PageCache is enabled, StarRocks caches the recently scanned data.
    - PageCache can significantly improve the query performance when similar queries are repeated frequently.
    - TRUE indicates disabling PageCache.
    - The default value of this item has been changed from TRUE to FALSE since StarRocks v2.4.

base_compaction_num_threads_per_disk

- **Default**: 1
- **Unit**: N/A
- **Description**: The number of threads used for Base Compaction on each storage volume.

base_cumulative_delta_ratio

- **Default**: 0.3
- **Unit**: N/A
- **Description**: The ratio of cumulative file size to base file size. The ratio reaching this value is one of the conditions that triggers Base Compaction.

compaction_trace_threshold

- **Default**: 60
- **Unit**: Second
- **Description**: The time threshold for each compaction. If a compaction takes more time than the time threshold, StarRocks prints the corresponding trace.

webserver_port

- **Default**: 8040
- **Unit**: N/A
- **Description**: The HTTP server port.

webserver_num_workers

- **Default**: 48
- **Unit**: N/A
- **Description**: The number of thread used by the HTTP server.

load_data_reserve_hours

- **Default**: 4
- **Unit**: Hour
- **Description**: The reservation time for the files produced by small-scale loadings.

number_tablet_writer_threads

- **Default**: 16
- **Unit**: N/A
- **Description**: The number of threads used for Stream Load.

streaming_load_rpc_max_alive_time_sec

- **Default**: 1200
- **Unit**: Second
- **Description**: The RPC timeout for Stream Load.

fragment_pool_thread_num_min

- **Default**: 64
- **Unit**: N/A
- **Description**: The minimum number of threads used for query.

fragment_pool_thread_num_max

- **Default**: 4096
- **Unit**: N/A
- **Description**: The maximum number of threads used for query.

fragment_pool_queue_size

- **Default**: 2048
- **Unit**: N/A
- **Description**: The upper limit of the number of queries that can be processed on each BE node.

enable_partitioned_aggregation

- **Default**: TRUE
- **Unit**: N/A
- **Description**: A boolean value to control whether to enable Partition Aggregation. The value true indicates to enable the Partition Aggregation. The value false indicates to disable the Partition Aggregation. |

enable_token_check

- **Default**: TRUE

- **Unit**: N/A
- **Description**: A boolean value to control whether to enable the token check. TRUE indicates enabling the token check, and FALSE indicates disabling it.

enable_prefetch

- **Default**: TRUE
- **Unit**: N/A
- **Description**: A boolean value to control whether to enable the pre-fetch of the query. TRUE indicates enabling pre-fetch, and FALSE indicates disabling it.

load_process_max_memory_limit_bytes

- **Default**: 107374182400
- **Unit**: Byte
- **Description**: The maximum size limit of memory resources that can be taken up by all load processes on a BE node.

load_process_max_memory_limit_percent

- **Default**: 30
- **Unit**: %
- **Description**: The maximum percentage limit of memory resources that can be taken up by all load processes on a BE node.

sync_tablet_meta

- **Default**: FALSE
- **Unit**: N/A
- **Description**: A boolean value to control whether to enable the synchronization of the tablet metadata. TRUE indicates enabling synchronization, and FALSE indicates disabling it.

routine_load_thread_pool_size

- **Default**: 10
- **Unit**: N/A
- **Description**: The thread pool size for Routine Load on each BE.

internal_service_async_thread_num

- **Default:** 10 (Number of Threads)
- **Description:** The thread pool size allowed on each BE for interacting with Kafka. Currently, the FE responsible for processing Routine Load requests depends on BEs to interact with Kafka, and each BE in StarRocks has its own thread pool for interactions with Kafka. If a large number of Routine Load tasks are distributed to a BE, the BE's thread pool for interactions with Kafka may be too busy to process all tasks in a timely manner. In this situation, you can adjust the value of this parameter to suit your needs.

brpc_max_body_size

- **Default**: 2147483648
- **Unit**: Byte
- **Description**: The maximum body size of a bRPC.

tablet_map_shard_size

- **Default**: 32
- **Unit**: N/A
- **Description**: The tablet map shard size. The value must be a power of two.

enable_bitmap_union_disk_format_with_set

- **Default**: FALSE
- **Unit**: N/A
- **Description**: A boolean value to control whether to enable the new storage format of the BITMAP type, which can improve the performance of bitmap_union. TRUE indicates enabling the new storage format, and FALSE indicates disabling it.

mem_limit

- **Default**: 90%
- **Unit**: N/A
- **Description**: BE process memory upper limit. You can set it as a percentage ("80%") or a physical limit ("100G"). The default hard limit is 90% of the server's memory size, and the soft limit is 80%. You need to configure this parameter if you want to deploy StarRocks with other memory-intensive services on a same server.

flush_thread_num_per_store

- **Default**: 2

- **Unit**: N/A
- **Description**: Number of threads that are used for flushing MemTable in each store.

block_cache_enable

- **Default**: false
- **Unit**: N/A
- **Description**: Whether to enable Data Cache. TRUE indicates Data Cache is enabled, and FALSE indicates Data Cache is disabled.

block_cache_disk_path

- **Default**: N/A
- **Unit**: N/A
- **Description**: The paths of disks. We recommend that the number of paths you configure for this parameter is the same as the number of disks on your BE machine. Multiple paths need to be separated with semicolons (;). After you add this parameter, StarRocks automatically creates a file named cachelib_data to cache blocks.

block_cache_meta_path

- **Default**: N/A
- **Unit**: N/A
- **Description**: The storage path of block metadata. You can customize the storage path. We recommend that you store the metadata under the `$STARROCKS_HOME` path.

block_cache_mem_size

- **Default**: 2147483648
- **Unit**: Bytes
- **Description**: The maximum amount of data that can be cached in memory. Unit: bytes. The default value is 2147483648, which is 2 GB. We recommend that you set the value of this parameter to at least 20 GB. If StarRocks reads a large amount of data from disks after Data Cache is enabled, consider increasing the value.

block_cache_disk_size

- **Default**: 0
- **Unit**: Bytes

- **Description**: The maximum amount of data that can be cached on a single disk. For example, if you configure two disk paths for the block_cache_disk_path parameter and set the value of the block_cache_disk_size parameter as 21474836480 (20 GB), a maximum of 40 GB data can be cached on these two disks. The default value is 0, which indicates that only memory is used to cache data. Unit: bytes.

jdbc_connection_pool_size

- **Default**: 8
- **Unit**: N/A
- **Description**: The JDBC connection pool size. On each BE node, queries that access the external table with the same jdbc_url share the same connection pool.

jdbc_minimum_idle_connections

- **Default**: 1
- **Unit**: N/A
- **Description**: The minimum number of idle connections in the JDBC connection pool.

jdbc_connection_idle_timeout_ms

- **Default**: 600000
- **Unit**: N/A
- **Description**: The length of time after which an idle connection in the JDBC connection pool expires. If the connection idle time in the JDBC connection pool exceeds this value, the connection pool closes idle connections beyond the number specified in the configuration item jdbc_minimum_idle_connections.

query_cache_capacity

- **Default**: 536870912
- **Unit**: N/A
- **Description**: The size of the query cache in the BE. Unit: bytes. The default size is 512 MB. The size cannot be less than 4 MB. If the memory capacity of the BE is insufficient to provision your expected query cache size, you can increase the memory capacity of the BE.

enable_event_based_compaction_framework

- **Default**: TRUE

- **Unit**: N/A
- **Description**: Whether to enable the Event-based Compaction Framework. TRUE indicates Event-based Compaction Framework is enabled, and FALSE indicates it is disabled. Enabling Event-based Compaction Framework can greatly reduce the overhead of compaction in scenarios where there are many tablets or a single tablet has a large amount of data.

enable_size_tiered_compaction_strategy

- **Default**: TRUE
- **Unit**: N/A
- **Description**: Whether to enable the Size-tiered Compaction strategy. TRUE indicates the Size-tiered Compaction strategy is enabled, and FALSE indicates it is disabled.

update_compaction_size_threshold

- **Default**: 67108864
- **Unit**: bytes
- **Description**: The Compaction Score of Primary Key tables is calculated based on the file size, which is different from other table types. This parameter can be used to make the Compaction Score of Primary Key tables similar to that of other table types, making it easier for users to understand. Since v2.5.20, the default value of this parameter is changed from `268435456` (256 MB) to `67108864` (64 MB) to accelerate compaction.

Monitor and Alerting

You can build your own monitoring services, or use the Prometheus + Grafana solution. StarRocks provides a Prometheus-compatible interface that directly links to the HTTP port of the BE and FE to obtain monitoring information from the cluster.

Monitoring Indicators

The available metrics are:

Indicator	Unit	Type
be_broker_count	pcs	average
be_brpc_endpoint_count	pcs	average
be_bytes_read_per_second	bytes/s	average
be_bytes_written_per_second	bytes/s	average
be_base_compaction_bytes_per_second	bytes/s	average
be_cumulative_compaction_bytes_per_second	bytes/s	average

Indicator	Unit	Type
be_base_compaction_rowsets_per_second	rowsets/s	average
be_cumulative_compaction_rowsets_per_second	rowsets/s	average
be_base_compaction_failed	pcs/s	average
be_clone_failed	pcs/s	average
be_create_rollup_failed	pcs/s	average
be_create_tablet_failed	pcs/s	average
be_cumulative_compaction_failed	pcs/s	average
be_delete_failed	pcs/s	average
be_finish_task_failed	pcs/s	average

Indicator	Unit	Type
be_publish_failed	pcs/s	average
be_report_tables_failed	pcs/s	average
be_report_disk_failed	pcs/s	average
be_report_tablet_failed	pcs/s	average
be_report_task_failed	pcs/s	average
be_schema_change_failed	pcs/s	average
be_base_compaction_requests	pcs/s	average
be_clone_total_requests	pcs/s	average
be_create_rollup_requests	pcs/s	average
be_create_tablet_requests	pcs/s	average

Indicator	Unit	Type
be_cumulative_compaction_requests	pcs/s	average
be_delete_requests	pcs/s	average
be_finish_task_requests	pcs/s	average
be_publish_requests	pcs/s	average
be_report_tablets_requests	pcs/s	average
be_report_disk_requests	pcs/s	average
be_report_tablet_requests	pcs/s	average
be_report_task_requests	pcs/s	average
be_schema_change_requests	pcs/s	average
be_storage_migrate_requests	pcs/s	average
be_fragment_endpoint_count	pcs	average

Indicator	Unit	Type
be_fragment_request_latency_avg	ms	average
be_fragment_requests_per_second	pcs/s	average
be_http_request_latency_avg	ms	average
be_http_requests_per_second	pcs/s	average
be_http_request_send_bytes_per_second	bytes/s	average
fe_connections_per_second	connections/s	average
fe_connection_total	connections	cumulative
fe_edit_log_read	operations/s	average
fe_edit_log_size_bytes	bytes/s	average

Indicator	Unit	Type
fe_edit_log_write	bytes/s	average
fe_checkpoint_push_per_second	operations/s	average
fe_pending_hadoop_load_job	pcs	average
fe_committed_hadoop_load_job	pcs	average
fe_loading_hadoop_load_job	pcs	average
fe_finished_hadoop_load_job	pcs	average
fe_cancelled_hadoop_load_job	pcs	average
fe_pending_insert_load_job	pcs	average
fe_loading_insert_load_job	pcs	average
fe_committed_insert_load_job	pcs	average

Indicator	Unit	Type
fe_finished_insert_load_job	pcs	average
fe_cancelled_insert_load_job	pcs	average
fe_pending_broker_load_job	pcs	average
fe_loading_broker_load_job	pcs	average
fe_committed_broker_load_job	pcs	average
fe_finished_broker_load_job	pcs	average
fe_cancelled_broker_load_job	pcs	average
fe_pending_delete_load_job	pcs	average

Indicator	Unit	Type
fe_loading_delete_load_job	pcs	average
fe_committed_delete_load_job	pcs	average
fe_finished_delete_load_job	pcs	average
fe_cancelled_delete_load_job	pcs	average
fe_rollup_running_alter_job	pcs	average
fe_schema_change_running_job	pcs	average
cpu_util	percentage	average
cpu_system	percentage	average
cpu_user	percentage	average
cpu_idle	percentage	average

Indicator	Unit	Type
cpu_guest	percentage	average
cpu_iowait	percentage	average
cpu_irq	percentage	average
cpu_nice	percentage	average
cpu_softirq	percentage	average
cpu_steal	percentage	average
disk_free	bytes	average
disk_io_svctm	Ms	average
disk_io_util	percentage	average
disk_used	bytes	average
starrocks_fe_meta_log_count	pcs	Instantaneous

Indicator	Unit	Type
starrocks_fe_query_resource_group	pcs	cumulative
starrocks_fe_query_resource_group_latency	second	average
starrocks_fe_query_resource_group_err	pcs	cumulative
starrocks_be_resource_group_cpu_limit_ratio	percentage	Instantaneous

Indicator	Unit	Type
starrocks_be_resource_group_cpu_use_ratio	percentage	average
starrocks_be_resource_group_mem_limit_bytes	Byte	Instantaneous
starrocks_be_resource_group_mem_allocated_bytes	Byte	Instantaneous

Monitoring Alarm Best Practices

Background information on the monitoring system:

1. The system collects information every 15 seconds.
2. Some indicators are divided by 15 seconds and the unit is pcs/sec. Some indicators are not divided, and the count is still 15 seconds.
3. P90, P99 and other quantile values are currently counted within 15 seconds. When calculating at a greater granularity (1 minute, 5 minutes, etc.), use "how many alarms greater than a certain value" rather than "what is the average value".

References

1. The purpose of monitoring is to only alert on abnormal conditions, not normal conditions.
2. Different clusters have different resources (e.g., memory, disk), different usage, and need to be set to different values; however, "percentage" is universal as a measurement unit.
3. For indicators such as `number of failures`, it is necessary to monitor the change of the total number, and calculate the alarm boundary value according to a certain proportion (for example, for the amount of P90, P99, P999).
4. `A value of 2x or more` or `a value higher than the peak` can generally be used as a warning value for the growth of used/query.

Alarm settings

Low frequency alarms

Trigger the alarm if one or more failures occur. Set a more advanced alarm if there are multiple failures.

For operations (e.g.,schema change) that are not frequently performed, "alarm on failure" is sufficient.

No task started

Once the monitoring alarm is turned on, there may be a lot of successful and failed tasks. You can set `failed > 1` to alert and modify it later.

Fluctuation

Large fluctuations

Need to focus on data with different time granularity, as the peaks and valleys in data with large granularity may be averaged out. Generally, you need to look at 15 days, 3 days, 12 hours, 3 hours, and 1 hour (for different time ranges).

The monitoring interval may need to be slightly longer (e.g. 3 minutes, 5 minutes, or even longer) to shield the alarm caused by fluctuations.

Small fluctuations

Set shorter intervals to quickly get alarms when problems occur.

High spikes

It depends on whether the spikes need to be alarmed or not. If there are too many spikes, setting longer intervals may help smooth out the spikes.

Resource usage

High resource usage

You can set the alarm to reserve a little resource.For example, set the memory alert to `mem_avaliable<=20%`.

Low resource usage

You can set a stricter value than "high resource usage".For example, for a CPU with low usage (less than 20%), set the alarm to `cpu_idle<60%`.

Caution

Usually FE/BE are monitored together, but there are some values that only FE or BE has.

There may be some machines that need to be set up in batches for monitoring.

Additional information

P99 Batch calculation rules

The node collects data every 15 seconds and calculates a value, the 99th percentile is the 99th percentile in those 15 seconds. When the QPS is not high (e.g. QPS is below 10), these percentiles are not very accurate. Also, it is meaningless to aggregate four values generated in one minute (4 x 15 seconds) whether using sum or average function.

The same applies to P50, P90, and so on.

Cluster Monitoring for errors

> Some undesired cluster errors need to be found and resolved in time to keep the cluster stable. If the errors are less critical (e.g. SQL syntax errors, etc.) but **can't be stripped out from the important error items**, it’s recommended to monitor first and distinguish those at a later stage.

Using Prometheus+Grafana

StarRocks can use Prometheus to monitor data storage and use Grafana to visualize results.

Components

> This document describes StarRocks' visual monitoring solution based on Prometheus and Grafana implementations. StarRocks is not responsible for maintaining or developing these components. For more detailed information about Prometheus and Grafana, please refer to their official websites.

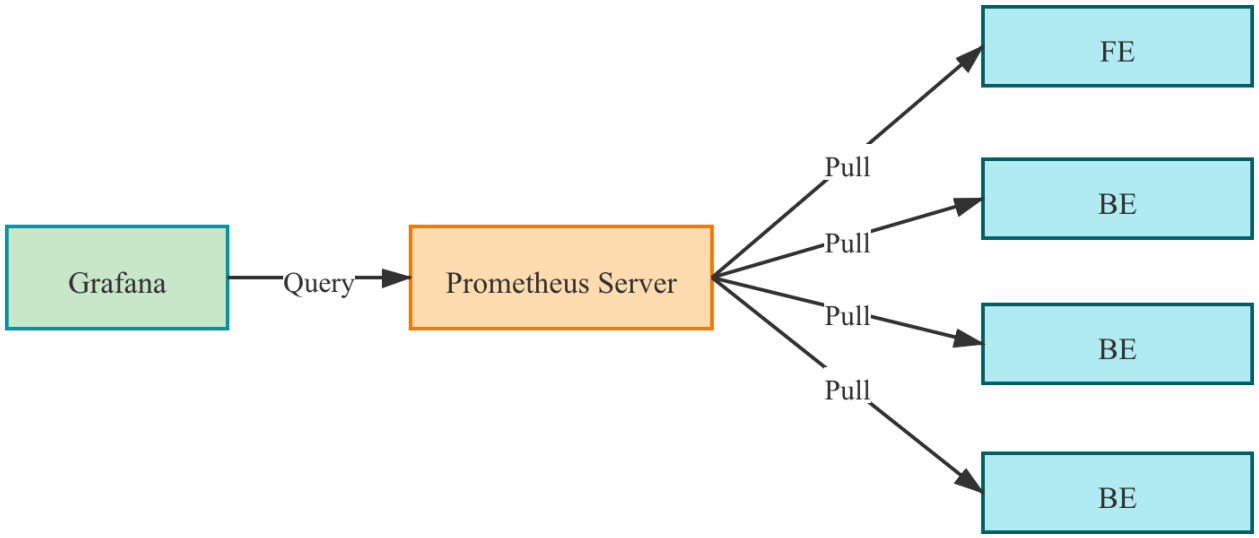
Prometheus

Prometheus is a temporal database with multi-dimensional data models and flexible query statements. It collects data by pulling or pushing them from monitored systems and stores these data in its temporal database. It meets different user needs through its rich multi-dimensional data query language.

Grafana

Grafana is an open-source metric analysis and visualization system that supports a variety of data sources. Grafana retrieves data from data sources with corresponding query statements. It allows users to create charts and dashboards to visualize data.

Monitoring architecture



Prometheus pulls the metrics from the FE/BE interface and then stores the data into its temporal database.

In Grafana, users can configure Prometheus as a data source to customize the Dashboard.

Deployment

Prometheus

1. Download the latest version of Prometheus from the Prometheus official website. Take the prometheus-2.29.1.linux-amd64 version for example.

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.29.1/prom
2.29.1.linux-amd64.tar.gz
tar -xf prometheus-2.29.1.linux-amd64.tar.gz
```

2. Add configuration in `vi prometheus.yml`

```
# my global config
global:
  scrape_interval: 15s # global acquisition interval, 1m by default,
here set to 15s
  evaluation_interval: 15s # global rule trigger interval, 1m by
default, here set to 15s

scrape_configs:
  # The job name is added as a label `job=<job_name>` to any
timeseries scraped from this config.
  - job_name: 'StarRocks_Cluster01' # Each cluster is called a job,
job name is customizable
    metrics_path: '/metrics' # Specify the Restful API to get metrics

    static_configs:
      - targets:
['fe_host1:http_port','fe_host3:http_port','fe_host3:http_port']
        labels:
          group: fe # Here the group of FE is configured which
contains 3 Frontends

      - targets: ['be_host1:http_port', 'be_host2:http_port',
'be_host3:http_port']
```

```
        labels:
          group: be # The group of BE is configured here which
contains three Backends
  - job_name: 'StarRocks_Cluster02' # Multiple StarRocks clusters can
be monitored in Prometheus
metrics_path: '/metrics'

    static_configs:
      - targets:
['fe_host1:http_port','fe_host3:http_port','fe_host3:http_port']
        labels:
          group: fe

      - targets: ['be_host1:http_port', 'be_host2:http_port',
'be_host3:http_port']
        labels:
          group: be
```

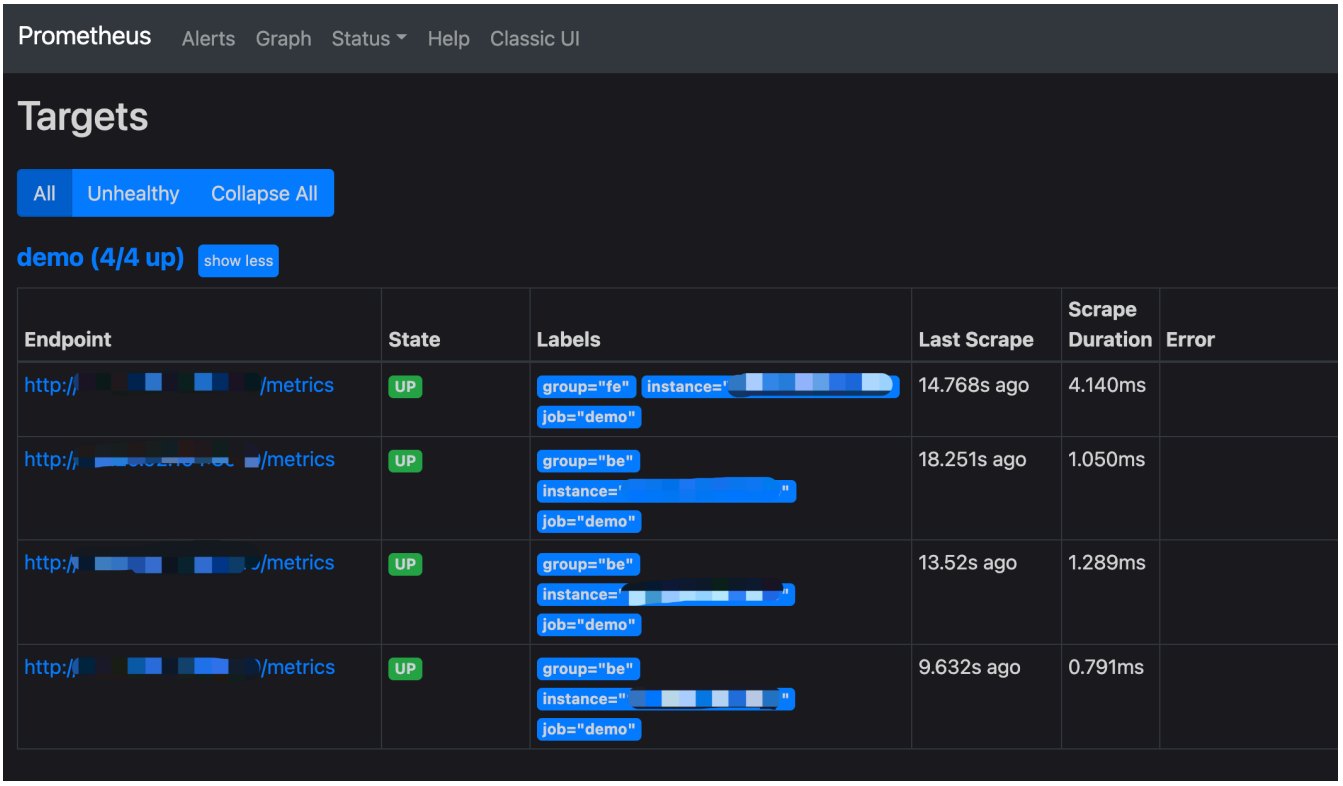
3. Start Prometheus

```
nohup ./prometheus \
    --config.file="./prometheus.yml" \
    --web.listen-address=":9090" \
    --log.level="info" &
```

This command runs Prometheus in the background and specifies its web port as 9090. Once set up, Prometheus starts collecting data and stores it in the `. /data` directory.

4. Accessing Prometheus

Prometheus can be accessed via BUI. You simply need to open port 9090 in your browser. Go to `Status -> Targets` to see the monitored host nodes for all grouped jobs. Under normal circumstances, all nodes should be `UP`. If the node status is not `UP`, you can visit the StarRocks metrics (`http://fe_host:fe_http_port/metrics` or `http://be_host:be_http_port/metrics`) interface first to check if it is accessible, or check the Prometheus documentation for troubleshooting.



A simple Prometheus has been built and configured. For more advanced usage, please refer to the official documentation

Grafana

1. Download the latest version of Grafana from Grafana official website. Take thegrafana-8.0.6.linux-amd64 version for example.

```
wget https://dl.grafana.com/oss/release/grafana-8.0.6.linux-amd64.tar.gz
tar -zxf grafana-8.0.6.linux-amd64.tar.gz
```

2. Add configuration in `vi . /conf/defaults.ini`

```
...
[paths]
data = ./data
logs = ./data/log
plugins = ./data/plugins
[server]
http_port = 8000
domain = localhost
...
```

3. Start Grafana

```
nohup ./bin/grafana-server \
    --config="./conf/grafana.ini" &
```

Dashboard

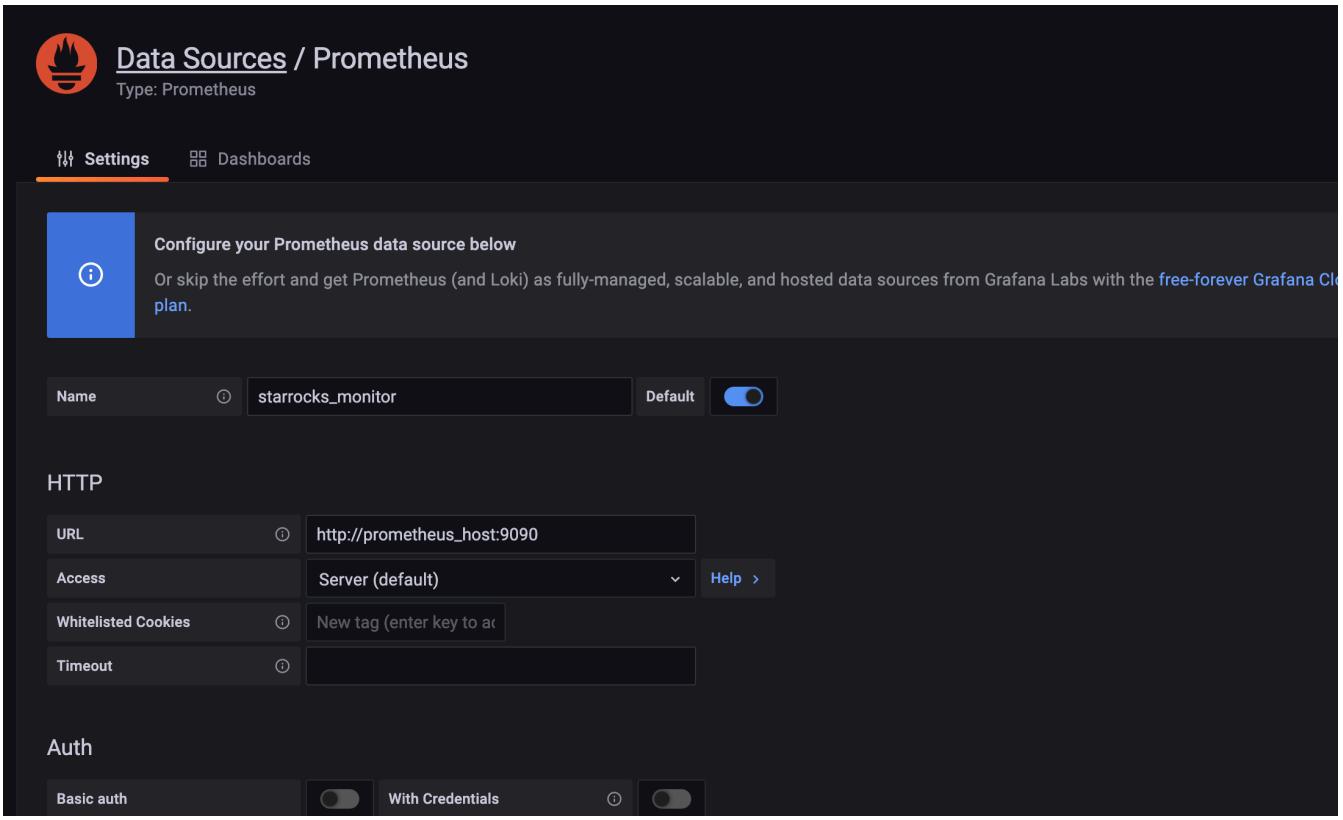
DashBoard Configuration

Log in to Grafana through the address configured in the previous step `http://grafana_host:8000` with the default username,password (i.e. admin,admin).

1. Add a data source.

Configuration path: `Configuration-->Data sources-->Add data source-->Prometheus`

Data Source Configuration Introduction



- Name: Name of the data source. Can be customized, e.g. starrocks_monitor
- URL: The web address of Prometheus, e.g. `http://prometheus_host:9090`

- Access: Select the Server method, i.e., the server where Grafana is located for Prometheus to access. The rest of the options are default.

Click Save & Test at the bottom, if it shows `Data source is working`, it means the data source is available.

2. Add a dashboard.

Download a dashboard.

> **NOTE**
>
> Metric names in StarRocks v1.19.0 and v2.4.0 are changed. You must download a dashboard template based on your StarRocks version:
>
> - Dashboard template for versions earlier than v1.19.0
> - Dashboard template for v1.19.0 to v2.4.0 (exclusive)
> - Dashboard template for v2.4.0 and later

Dashboard templates will be updated from time to time.

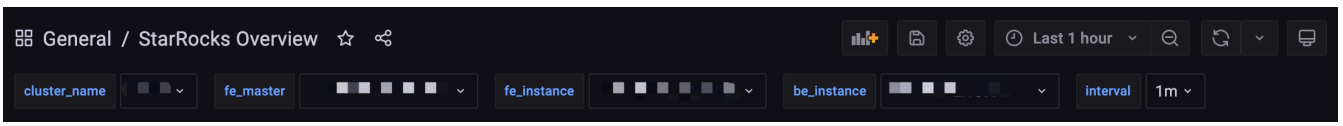
After confirming the data source is available, click on the `+` sign to add a new Dashboard, here we use the StarRocks Dashboard template downloaded above. Go to Import -> Upload Json File to load the downloaded json file.

After loading, you can name the Dashboard. The default name is StarRocks Overview. Then select `starrocks_monitor` as the data source. Click `Import` to complete the import. Then you should see the Dashboard.

Dashboard Description

Add a description for your dashboard. Update the description for each version.

1. Top bar

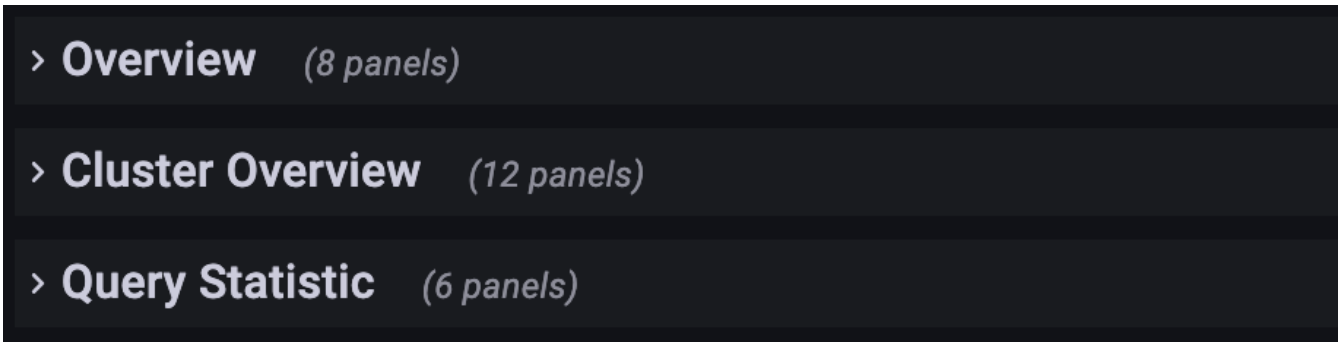


The top left corner shows the Dashboard name. The top right corner shows the current time range. Use the drop down to select a different time range and specify an interval for page refresh. cluster_name: The `job_name` of each job in the Prometheus configuration file,

representing a StarRocks cluster. You can select a cluster and view its monitoring information in the chart.

- fe_master: The leader node of the cluster.
- fe_instance: All frontend nodes of the corresponding cluster. Select to view the monitoring information in the chart.
- be_instance: All backend nodes of the corresponding cluster. Select to view the monitoring information in the chart.
- interval: Some charts show intervals related to monitoring items. Interval is customizable(Note: 15s interval may cause some charts not to display).

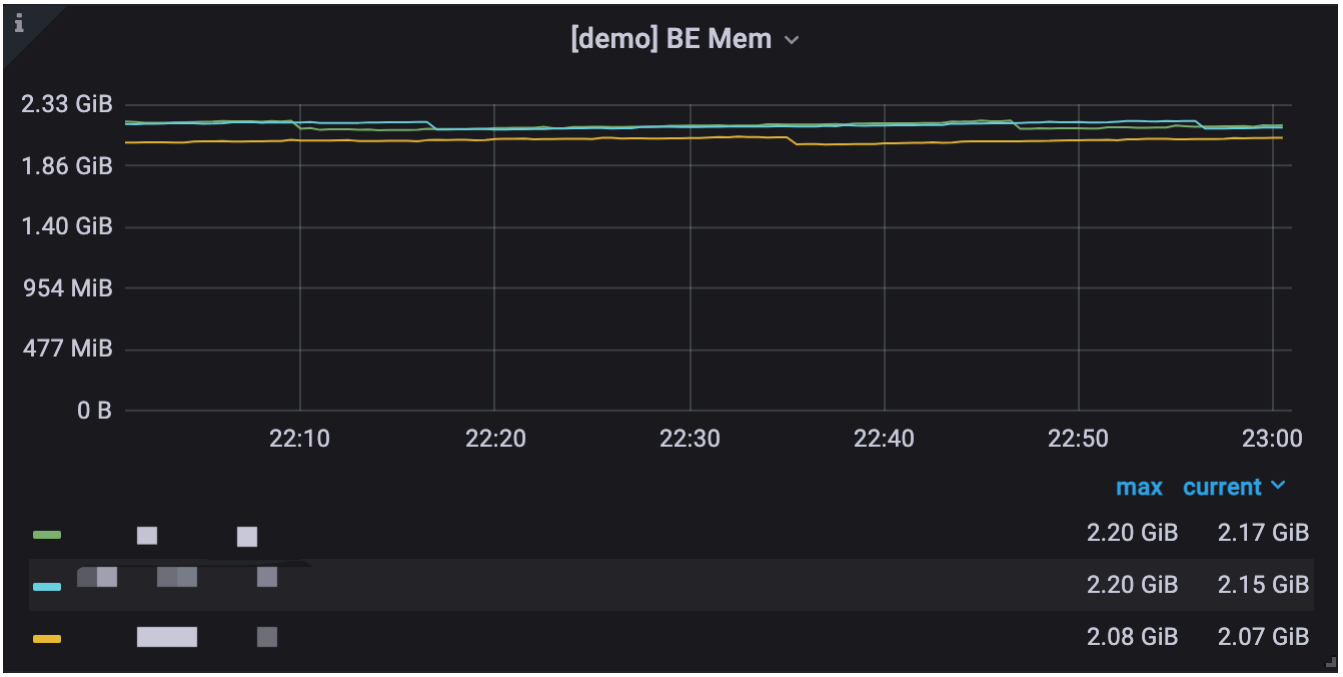
2. Row



In Grafana, the concept of a `Row` is a collection of diagrams. You can collapse a `Row` by clicking on it. The current Dashboard has the following `Rows` :

- Overview: Display of all StarRocks clusters.
- Cluster Overview: Display of selected clusters.
- Query Statistic: Monitoring for Queries of selected clusters.
- Jobs: Monitoring for Import jobs.
- Transaction: Monitoring for Transactions.
- FE JVM: Monitoring for JVM of selected Frontend.
- BE: Display of Backends of selected clusters.
- BE Task: Display of Backends tasks of selected clusters.

3. A typical chart is divided into the following parts.



- Hover over the i icon in the upper left corner to see the chart description.
- Click on the legend below to view a particular item. Click again to display all.
- Drag and drop in the chart to select a time range.
- The name of the selected cluster is displayed in [] of the title.
- Values may correspond to the left Y-axis or the right Y-axis, which can be distinguished by the -right at the end of the legend.
- Click on the chart name to edit the name.

Other

If you need to access the monitoring data in your own Prometheus system, access it through the following interface.

- FE: fe_host:fe_http_port/metrics
- BE: be_host:be_web_server_port/metrics

If JSON format is required, access the following instead.

- FE: fe_host:fe_http_port/metrics?type=json
- BE: be_host:be_web_server_port/metrics?type=json

Manage audit logs within StarRocks via AuditLoader

This topic describes how to manage StarRocks audit logs within a table via the plugin - AuditLoader.

StarRocks stores its audit logs in the local file **fe/log/fe.audit.log** rather than an internal database. The plugin AuditLoader allows you to manage audit logs directly within your cluster. Once installed, AuditLoader reads logs from the file, and loads them into StarRocks via HTTP PUT. You can then query the audit logs in StarRocks using SQL statements.

Create a table to store audit logs

Create a database and a table in your StarRocks cluster to store its audit logs. See CREATE DATABASE and CREATE TABLE for detailed instructions.

Because the fields of audit logs vary among different StarRocks versions, you must choose among the following examples to create a table that is compatible with your StarRocks.

> **CAUTION**
>
> - DO NOT change the table schema in the examples, or the log loading will fail.
> - Because the fields of audit logs vary among different StarRocks versions, the new version AuditLoader collects the common fields among them from all available StarRocks versions.

```
CREATE DATABASE starrocks_audit_db__;

CREATE TABLE starrocks_audit_db__.starrocks_audit_tbl__ (
  `queryId`           VARCHAR(64)               COMMENT "Unique
query ID",
  `timestamp`         DATETIME         NOT NULL  COMMENT "Query start
time",
  `queryType`         VARCHAR(12)               COMMENT "Query type
(query, slow_query, connection) ",
```

```
  `clientIp`          VARCHAR(32)                COMMENT "Client IP
address",
  `user`              VARCHAR(64)                COMMENT "User who
initiates the query",
  `authorizedUser`    VARCHAR(64)                COMMENT
"user_identity",
  `resourceGroup`     VARCHAR(64)                COMMENT "Resource
group name",
  `catalog`           VARCHAR(32)                COMMENT "Catalog
name",
  `db`                VARCHAR(96)                COMMENT "Database
that the query scans",
  `state`             VARCHAR(8)                 COMMENT "Query state
(EOF, ERR, OK)",
  `errorCode`         VARCHAR(512)               COMMENT "Error
code",
  `queryTime`         BIGINT                     COMMENT "Query
latency in milliseconds",
  `scanBytes`         BIGINT                     COMMENT "Size of the
scanned data in bytes",
  `scanRows`          BIGINT                     COMMENT "Row count
of the scanned data",
  `returnRows`        BIGINT                     COMMENT "Row count
of the result",
  `cpuCostNs`         BIGINT                     COMMENT "CPU
resources consumption time for query in nanoseconds",
  `memCostBytes`      BIGINT                     COMMENT "Memory cost
for query in bytes",
  `stmtId`            INT                        COMMENT "Incremental
SQL statement ID",
  `isQuery`           TINYINT                    COMMENT "If the SQL
is a query (0 and 1)",
  `feIp`              VARCHAR(128)               COMMENT "IP address
of FE that executes the SQL",
  `stmt`              VARCHAR(1048576)           COMMENT "Original
SQL statement",
  `digest`            VARCHAR(32)                COMMENT "Slow SQL
fingerprint",
  `planCpuCosts`      DOUBLE                     COMMENT "CPU
resources consumption time for planning in nanoseconds",
  `planMemCosts`      DOUBLE                     COMMENT "Memory cost
for planning in bytes"
) ENGINE = OLAP
DUPLICATE KEY (`queryId`, `timestamp`, `queryType`)
COMMENT "Audit log table"
PARTITION BY RANGE (`timestamp`) ()
```

```
DISTRIBUTED BY HASH (`queryId`) BUCKETS 3
PROPERTIES (
  "dynamic_partition.time_unit" = "DAY",
  "dynamic_partition.start" = "-30",       -- Keep the audit logs
from the latest 30 days. You can adjust this value based on you
business demand.
  "dynamic_partition.end" = "3",
  "dynamic_partition.prefix" = "p",
  "dynamic_partition.buckets" = "3",
  "dynamic_partition.enable" = "true",
  "replication_num" = "3"                  -- Keep three replicas of
audit logs. It is recommended to keep three replicas in a production
environment.
);
```

`starrocks_audit_tbl__` is created with dynamic partitions. By default, the first dynamic partition is created 10 minutes after the table is created. Audit logs can then be loaded into the table. You can check the partitions in the table using the following statement:

```
SHOW PARTITIONS FROM starrocks_audit_db__.starrocks_audit_tbl__;
```

After a partition is created, you can move on to the next step.

Download and configure AuditLoader

1. Download the AuditLoader installation package. The package is compatible with all available versions of StarRocks.

2. Unzip the installation package.

   ```
   unzip auditloader.zip
   ```

   The following files are inflated:

   - **auditloader.jar**: the JAR file of AuditLoader.
   - **plugin.properties**: the properties file of AuditLoader. You do not need to modify this file.
   - **plugin.conf**: the configuration file of AuditLoader. In most cases, you only need to modify the `user` and `password` fields in the file.

3. Modify **plugin.conf** to configure AuditLoader. You must configure the following items to make sure AuditLoader can work properly:

   - `frontend_host_port`: FE IP address and HTTP port, in the format `<fe_ip>:<fe_http_port>`. The default value is `127.0.0.1:8030`.
   - `database`: name of the database you created to host audit logs.
   - `table`: name of the table you created to host audit logs.
   - `user`: your cluster username. You MUST have the privilege to load data (LOAD_PRIV) into the table.
   - `password`: your user password.

4. Zip the files back into a package.

```
zip -q -m -r auditloader.zip auditloader.jar plugin.conf
plugin.properties
```

5. Dispatch the package to all machines that host FE nodes. Make sure all packages are stored in an identical path. Otherwise, the installation fails. Remember to copy the absolute path to the package after you dispatched the package.

Install AuditLoader

Execute the following statement along with the path you copied to install AuditLoader as a plugin in StarRocks:

```
INSTALL PLUGIN FROM "<absolute_path_to_package>";
```

See INSTALL PLUGIN for detailed instructions.

Verify the installation and query audit logs

1. You can check if the installation is successful via SHOW PLUGINS.

   In the following example, the `Status` of the plugin `AuditLoader` is `INSTALLED`, meaning installation is successful.

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
    Name: __builtin_AuditLogBuilder
    Type: AUDIT
Description: builtin audit logger
    Version: 0.12.0
JavaVersion: 1.8.31
ClassName: com.starrocks.qe.AuditLogBuilder
    SoName: NULL
    Sources: Builtin
    Status: INSTALLED
Properties: {}
*************************** 2. row ***************************
    Name: AuditLoader
    Type: AUDIT
Description: Available for versions 2.3+. Load audit log to
starrocks, and user can view the statistic of queries.
    Version: 4.0.0
JavaVersion: 1.8.0
ClassName: com.starrocks.plugin.audit.AuditLoaderPlugin
    SoName: NULL
    Sources: /x/xx/xxx/xxxxx/auditloader.zip
    Status: INSTALLED
Properties: {}
2 rows in set (0.01 sec)
```

2. Execute some random SQLs to generate audit logs, and wait for 60 seconds (or the time you have specified in the item `max_batch_interval_sec` when you configure AuditLoader) to allow AuditLoader to load audit logs into StarRocks.

3. Check the audit logs by querying the table.

```
SELECT * FROM starrocks_audit_db__.starrocks_audit_tbl__;
```

The following example shows when audit logs are loaded into the table successfully:

```
mysql> SELECT * FROM starrocks_audit_db__.starrocks_audit_tbl__\G
*************************** 1. row ***************************
     queryId: 84a69010-d47e-11ee-9647-024228044898
   timestamp: 2024-02-26 16:10:35
   queryType: query
    clientIp: xxx.xx.xxx.xx:65283
```

```
        user: root
authorizedUser: 'root'@'%'
resourceGroup: default_wg
     catalog: default_catalog
          db:
       state: EOF
   errorCode:
   queryTime: 3
   scanBytes: 0
    scanRows: 0
  returnRows: 1
   cpuCostNs: 33711
memCostBytes: 4200
      stmtId: 102
     isQuery: 1
        feIp: xxx.xx.xxx.xx
        stmt: SELECT * FROM
starrocks_audit_db__.starrocks_audit_tbl__
      digest:
planCpuCosts: 0
planMemCosts: 0
1 row in set (0.01 sec)
```

Troubleshooting

If no audit logs are loaded to the table after the dynamic partition is created and the plugin is installed, you can check whether **plugin.conf** is configured properly or not. To modify it, you must first uninstall the plugin:

```
UNINSTALL PLUGIN AuditLoader;
```

After all configurations are set correctly, you can follow the above steps to install AuditLoader again.

Enable FQDN access

This topic describes how to enable cluster access by using a fully qualified domain name (FQDN). An FQDN is a **complete domain name** for a specific entity that can be accessed over the Internet. The FQDN consists of two parts: the hostname and the domain name.

Before 2.4, StarRocks supports access to FEs and BEs via IP address only. Even if an FQDN is used to add a node to a cluster, it is transformed into an IP address eventually. This causes a huge inconvenience for DBAs because changing the IP addresses of certain nodes in a StarRocks cluster can lead to access failures to the nodes. In version 2.4, StarRocks decouples each node from its IP address. You can now manage nodes in StarRocks solely via their FQDNs.

Prerequisites

To enable FQDN access for a StarRocks cluster, make sure the following requirements are satisfied:

- Each machine in the cluster must have a hostname.
- In the file **/etc/hosts** on each machine, you must specify the corresponding IP addresses and FQDNs of other machines in the cluster.
- IP addresses in the **/etc/hosts** file must be uniqule.

Set up a new cluster with FQDN access

By default, FE nodes in a new cluster are started via IP address access. To start a new cluster with FQDN access, you must start the FE nodes by running the following commands **when you start the cluster for the first time**:

```
./bin/start_fe.sh --host_type FQDN --daemon
```

The property `--host_type` specifies the access method that is used to start the node. Valid values include `FQDN` and `IP`. You only need to specify this property ONCE when you start the node for the first time.

See Deploy StarRocks for detailed instructions on how to install StarRocks.

Each BE node identifies itself with `BE Address` defined in the FE metadata. Therefore, you DO NOT need to specify `--host_type` when you start BE nodes. If the `BE Address` defines a BE node with an FQDN, the BE node identifies itself with this FQDN.

Enable FQDN access in an existing cluster

To enable FQDN access in an existing cluster that was previously started via IP addresses, you must first **upgrade** StarRocks to version 2.4.0 or later.

Enable FQDN access for FE nodes

You need to enable FQDN access for all the non-Leader Follower FE nodes before enabling that for the Leader FE node.

> **CAUTION**
>
> Make sure that the cluster has at least three Follower FE nodes before you enable FQDN access for FE nodes.

Enable FQDN access for non-Leader Follower FE nodes

1. Navigate to the deployment directory of the FE node, and run the following command to stop the FE node:

   ```
   ./bin/stop_fe.sh
   ```

2. Execute the following statement via your MySQL client to check the `Alive` status of the FE node that you have stopped. Wait until the `Alive` status becomes `false`.

   ```
   SHOW PROC '/frontends'\G
   ```

3. Execute the following statement to replace the IP address with FQDN.

```
ALTER SYSTEM MODIFY FRONTEND HOST "<fe_ip>" TO "<fe_hostname>";
```

4. Run the following command to start the FE node with FQDN access.

   ```
   ./bin/start_fe.sh --host_type FQDN --daemon
   ```

   The property `--host_type` specifies the access method that is used to start the node. Valid values include `FQDN` and `IP`. You only need to specify this property ONCE when you restart the node after you modify the node.

5. Check the `Alive` status of the FE node. Wait until the `Alive` status becomes `true`.

   ```
   SHOW PROC '/frontends'\G
   ```

6. Repeat the above steps to enable FQDN access for the other non-Leader Follower FE nodes one after another when the `Alive` status of the current FE node is `true`.

Enable FQDN access for the Leader FE node

After all the non-Leader FE nodes have been modified and restarted successfully, you can now enable FQDN access for the Leader FE node.

> **NOTE**
>
> Before the Leader FE node is enabled with FQDN access, the FQDNs used to add nodes to a cluster are still transformed into the corresponding IP addresses. After a Leader FE node with FQDN access enabled is elected for the cluster, the FQDNs will not be transformed into IP addresses.

1. Navigate to the deployment directory of the Leader FE node, and run the following command to stop the Leader FE node.

   ```
   ./bin/stop_fe.sh
   ```

2. Execute the following statement via your MySQL client to check whether a new Leader FE node has been elected for the cluster.

   ```
   
   ```

```
SHOW PROC '/frontends'\G
```

Any FE node with status `Alive` and `isMaster` being `true` is a Leader FE that is running.

3. Execute the following statement to replace the IP address with FQDN.

```
ALTER SYSTEM MODIFY FRONTEND HOST "<fe_ip>" TO "<fe_hostname>";
```

4. Run the following command to start the FE node with FQDN access.

```
./bin/start_fe.sh --host_type FQDN --daemon
```

The property `--host_type` specifies the access method that is used to start the node. Valid values include `FQDN` and `IP`. You only need to specify this property ONCE when you restart the node after you modify the node.

5. Check the `Alive` status of the FE node.

```
SHOW PROC '/frontends'\G
```

If the `Alive` status becomes `true`, the FE node is successfully modified and added to the cluster as a Follower FE node.

Enable FQDN access for BE nodes

Execute the following statement via your MySQL client to replace the IP address with FQDN to enable FQDN access for the BE node.

```
ALTER SYSTEM MODIFY BACKEND HOST "<be_ip>" TO "<be_hostname>";
```

> **NOTE**
>
> You DO NOT need to restart the BE node after FQDN access is enabled.

Rollback

To rollback an FQDN access-enabled StarRocks cluster to an earlier version that does not support FQDN access, you must first enable IP address access for all nodes in the cluster. You can refer Enable FQDN access in an existing cluster for the general guidance except that you need to change the SQL commands to the following ones:

- Enable IP address access for an FE node:

```
ALTER SYSTEM MODIFY FRONTEND HOST "<fe_hostname>" TO "<fe_ip>";
```

- Enable IP address access for a BE node:

```
ALTER SYSTEM MODIFY BACKEND HOST "<be_hostname>" TO "<be_ip>";
```

The modification takes effect after your cluster is successfully restarted.

FAQ

Q: An error occurs when I enable FQDN access for an FE node: "required 1 replica. But none were active with this master". What should I do?

A: Make sure the cluster has at least three Follower FE nodes before you enable FQDN access for FE nodes.

Q: Can I add a new node by using IP address to a cluster with FQDN access enabled?

A: Yes.

Configure a time zone

This topic describes how to configure a time zone and the impacts of time zone settings.

Configure a session-level time zone or a global time zone

You can configure a session-level time zone or a global time zone for your StarRocks cluster using the `time_zone` parameter.

- To configure a session-level time zone, execute the command `SET time_zone = 'xxx';`. You can configure different time zones for different sessions. The time zone setting becomes invalid if you disconnect with FEs.
- To configure a global time zone, execute the command `SET global time_zone = 'xxx';`. The time zone setting is persisted in FEs and is valid even if you disconnect with FEs.

> **Note**
>
> Before you load data into StarRocks, modify the global time zone of your StarRocks cluster to the same value of the `system_time_zone` parameter. Otherwise, after data loading, data of the DATE type are incorrect. The `system_time_zone` parameter refers to the time zone of the machines that are used to host FEs. When the machines are started, the time zone of the machines is recorded as the value of this parameter. You cannot manually configure this parameter.

Time zone format

The value of the `time_zone` parameter is not case-sensitive. The value of the parameter can be in one of the following formats.

Format	Example
UTC offset	`SET time_zone = '+10:00';` `SET global time_zone = '-6:00';`
Time zone name	`SET time_zone = 'Asia/Shanghai';` `SET global time_zone = 'America/Los_Angeles';`

For more information about time zone format, see List of tz database time zones.

> **Note**
>
> Time zone abbreviations are not supported except for CST. If you set the value of `time_zone` to `CST`, StarRocks converts `CST` into `Asia/Shanghai`.

Default time zone

The default value of the `time_zone` parameter is `Asia/Shanghai`.

View time zone settings

To view the time zone setting, run the following command.

```
SHOW VARIABLES LIKE '%time_zone%';
```

Impacts of time zone settings

- Time zone settings affect the time values returned by the SHOW LOAD and SHOW BACKENDS statements. However, the settings do not affect the value specified in the `LESS THAN` clause when the partitioning columns specified in CREATE TABLE statement are of the DATE or DATETIME type. The settings also do not affect data of the DATE and DATETIME types.
- Time zone settings affect the display and storage of the following functions:
  - **from_unixtime**: returns a date and time of your specified time zone based on a specified UTC timestamp. For example, if the global time zone of your StarRocks

cluster is `Asia/Shanghai`, `select FROM_UNIXTIME(0);` returns `1970-01-01 08:00:00`.

- **unix_timestamp**: returns a UTC timestamp based on the date and time of your specified time zone. For example, if the global time zone of your StarRocks cluster is `Asia/Shanghai`, `select UNIX_TIMESTAMP('1970-01-01 08:00:00');` returns `0`.
- **curtime**: returns the current time of your specified time zone. For example, if the current time of a specified time zone is 16:34:05. `select CURTIME();` returns `16:34:05`.
- **now**: returns the current date and time of your specified time zone. For example, if the current date and time of a specified time zone is 2021-02-11 16:34:13, `select NOW();` returns `2021-02-11 16:34:13`.
- **convert_tz**: converts the date and time from one time zone to another. For example, `select CONVERT_TZ('2021-08-01 11:11:11', 'Asia/Shanghai', 'America/Los_Angeles');` returns `2021-07-31 20:11:11`.

Information Schema

The StarRocks `information_schema` is a database within each StarRocks instance. `information_schema` contains several read-only, system-defined tables which store extensive metadata information of all objects that the StarRocks instance maintains.

View metadata via Information Schema

You can view the metadata information within a StarRocks instance by querying the content of tables in `information_schema`.

The following example views metadata information about a table named `sr_member` in StarRocks by querying the table `tables`.

```
mysql> SELECT * FROM information_schema.tables WHERE TABLE_NAME like
'sr_member'\G
*************************** 1. row ***************************
  TABLE_CATALOG: def
   TABLE_SCHEMA: sr_hub
     TABLE_NAME: sr_member
     TABLE_TYPE: BASE TABLE
         ENGINE: StarRocks
        VERSION: NULL
     ROW_FORMAT: NULL
     TABLE_ROWS: 6
 AVG_ROW_LENGTH: 542
    DATA_LENGTH: 3255
MAX_DATA_LENGTH: NULL
   INDEX_LENGTH: NULL
      DATA_FREE: NULL
 AUTO_INCREMENT: NULL
    CREATE_TIME: 2022-11-17 14:32:30
    UPDATE_TIME: 2022-11-17 14:32:55
     CHECK_TIME: NULL
TABLE_COLLATION: utf8_general_ci
       CHECKSUM: NULL
 CREATE_OPTIONS: NULL
```

```
  TABLE_COMMENT: OLAP
1 row in set (1.04 sec)
```

Information Schema tables

StarRocks have optimized the metadata information provided by the following tables in `information_schema`:

Information Schema table name	Description
tables	Provides general metadata information of tables.
tables_config	Provides additional table metadata information that is unique to StarRocks.

tables

The following fields are provided in `tables`:

Field	Description
TABLE_CATALOG	Name of the catalog that stores the table.
TABLE_SCHEMA	Name of the database that stores the table.
TABLE_NAME	Name of the table.
TABLE_TYPE	Type of the table. Valid values: "BASE TABLE" or "VIEW".
ENGINE	Engine type of the table. Valid values: "StarRocks", "MySQL", "MEMORY" or an empty string.
VERSION	Applies to a feature not available in StarRocks.
ROW_FORMAT	Applies to a feature not available in StarRocks.

Field	Description
TABLE_ROWS	Row count of the table.
AVG_ROW_LENGTH	Average row length (size) of the table. It is equivalent to `DATA_LENGTH` / `TABLE_ROWS`. Unit: Byte.
DATA_LENGTH	Data length (size) of the table. Unit: Byte.
MAX_DATA_LENGTH	Applies to a feature not available in StarRocks.
INDEX_LENGTH	Applies to a feature not available in StarRocks.
DATA_FREE	Applies to a feature not available in StarRocks.
AUTO_INCREMENT	Applies to a feature not available in StarRocks.
CREATE_TIME	The time when the table is created.
UPDATE_TIME	The last time when the table is updated.
CHECK_TIME	The last time when a consistency check is performed on the table.
TABLE_COLLATION	The default collation of the table.
CHECKSUM	Applies to a feature not available in StarRocks.
CREATE_OPTIONS	Applies to a feature not available in StarRocks.
TABLE_COMMENT	Comment on the table.

tables_config

The following fields are provided in `tables_config`:

Field	Description
TABLE_SCHEMA	Name of the database that stores the table.
TABLE_NAME	Name of the table.
TABLE_ENGINE	Engine type of the table.
TABLE_MODEL	Table type. Valid values: "DUP_KEYS", "AGG_KEYS", "UNQ_KEYS" or "PRI_KEYS".
PRIMARY_KEY	The primary key of a Primary Key table or a Unique Key table. An empty string is returned if the table is not a Primary Key table or a Unique Key table.
PARTITION_KEY	The partitioning columns of the table.
DISTRIBUTE_KEY	The bucketing columns of the table.
DISTRIBUTE_TYPE	The data distribution method of the table.
DISTRIBUTE_BUCKET	Number of buckets in the table.
SORT_KEY	Sort keys of the table.
PROPERTIES	Properties of the table.
TABLE_ID	ID of the table.

load_tracking_logs

This feature is supported since StarRocks v3.0.

The following fields are provided in `load_tracking_logs`:

Field	Description
JOB_ID	The ID of the load job.
LABEL	The label of the load job.
DATABASE_NAME	The database that the load job belongs to.
TRACKING_LOG	Error logs (if any) of the load job.
Type	The type of the load job. Valid values: BROKER, INSERT, ROUTINE_LOAD and STREAM_LOAD.

materialized_views

The following fields are provided in `materialized_views`:

Field	Description
MATERIALIZED_VIEW_ID	ID of the materialized view
TABLE_SCHEMA	Database in which the materialized view resides
TABLE_NAME	Name of the materialized view
TABLE_ROWS	Number of data rows in the materialized view, based on approximate background statistics
MATERIALIZED_VIEW_DEFINITION	SQL definition of the materialized view

Resource group

This topic describes the resource group feature of StarRocks.

Query queues

This topic describes how to manage query queues in StarRocks.

Query Management

Number of user connections

Memory Management

This section briefly introduces memory classification and StarRocks' methods of managing memory.

Load Balancing

When deploying multiple FE nodes, users can deploy a load balancing layer on top of the FEs to ac...

Replica management

Terminology

Blacklist Management

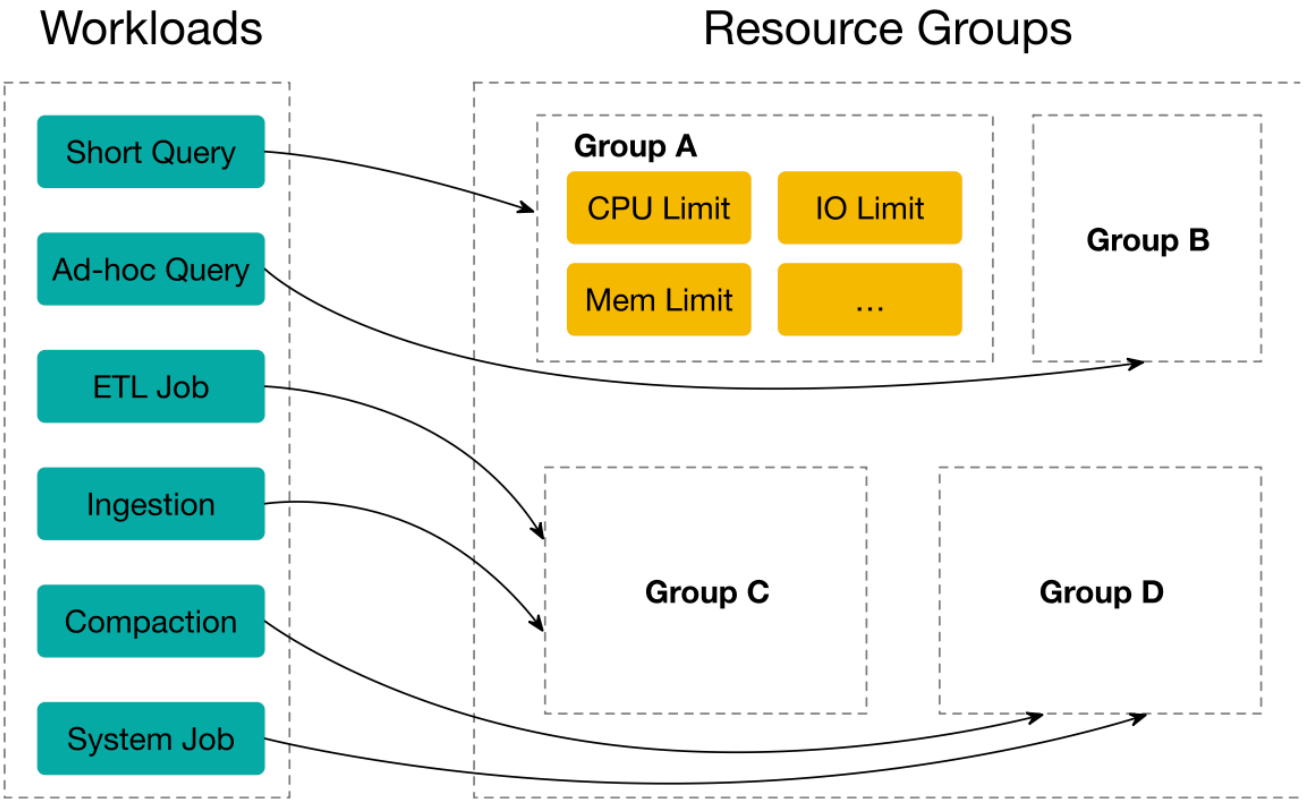
In some cases, administrators need to disable certain patterns of SQL to avoid SQL from triggering...

File manager

With file manager, you can create, view, and delete files, such as the files that are used to access e...

Resource group

This topic describes the resource group feature of StarRocks.



With this feature, you could simultaneously run several workloads in a single cluster, including short query, ad-hoc query, ETL jobs, to save extra cost of deploying multiple clusters. From technical perspective, the execution engine would schedule concurrent workloads according to users' specification and isolate the interference among them.

Roadmap of Resource Group feature:

- Since v2.2, StarRocks supports limiting resource consumption for queries and implementing isolation and efficient use of resources among tenants in the same cluster.
- In StarRocks v2.3, you can further restrict the resource consumption for big queries, and prevent the cluster resources from getting exhausted by oversized query requests, to guarantee the system stability.
- StarRocks v2.5 supports limiting computation resource consumption for data loading (INSERT).

	Internal Table	External Table	Big Query Restriction	Short Query	Data Ingestion	Schema Change	INSERT
2.2	√	×	×	×	×	×	×
2.3	√	√	√	√	×	×	×
2.4	√	√	√	√	×	×	×
2.5	√	√	√	√	√	×	√

Terms

This section describes the terms that you must understand before you use the resource group feature.

resource group

Each resource group is a set of computing resources from a specific BE. You can divide each BE of your cluster into multiple resource groups. When a query is assigned to a resource group, StarRocks allocates CPU and memory resources to the resource group based on the resource quotas that you specified for the resource group.

You can specify CPU and memory resource quotas for a resource group on a BE by using the following parameters:

- `cpu_core_limit`

  This parameter specifies the soft limit for the number of CPU cores that can be allocated to the resource group on the BE. Valid values: any non-zero positive integer.

  In actual business scenarios, CPU cores that are allocated to the resource group proportionally scale based on the availability of CPU cores on the BE.

  > **NOTE**
  >
  > For example, you configure three resource groups on a BE that provides 16 CPU cores: rg1, rg2, and rg3. The values of `cpu_core_limit` for the three resource

> groups are `2`, `6`, and `8`, respectively.
>
> If all CPU cores of the BE are occupied, the number of CPU cores that can be allocated to each of the three resource groups are 2, 6, and 8, respectively, based on the following calculations:
>
> - Number of CPU cores for rg1 = Total number of CPU cores on the BE × (2/16) = 2
> - Number of CPU cores for rg2 = Total number of CPU cores on the BE × (6/16) = 6
> - Number of CPU cores for rg3 = Total number of CPU cores on the BE × (8/16) = 8
>
> If not all CPU cores of the BE are occupied, as when rg1 and rg2 are loaded but rg3 is not, the number of CPU cores that can be allocated to rg1 and rg2 are 4 and 12, respectively, based on the following calculations:
>
> - Number of CPU cores for rg1 = Total number of CPU cores on the BE × (2/8) = 4
> - Number of CPU cores for rg2 = Total number of CPU cores on the BE × (6/8) = 12

- `mem_limit`

  This parameter specifies the percentage of memory that can be used for queries in the total memory that is provided by the BE. Valid values: (0, 1).

  > **NOTE**
  >
  > The amount of memory that can be used for queries is indicated by the `query_pool` parameter. For more information about the parameter, see Memory management.

- `concurrency_limit`

  This parameter specifies the upper limit of concurrent queries in a resource group. It is used to avoid system overload caused by too many concurrent queries. This parameter takes effect only when it is set greater than 0. Default: 0.

On the basis of the above resource consumption restrictions, you can further restrict the resource consumption for big queries with the following parameters:

- `big_query_cpu_second_limit`: This parameter specifies the upper time limit of CPU occupation for a big query. Concurrent queries add up the time. The unit is second. This parameter takes effect only when it is set greater than 0. Default: 0.
- `big_query_scan_rows_limit`: This parameter specifies the upper limit of row count that a big query can scan. This parameter takes effect only when it is set greater than 0. Default: 0.
- `big_query_mem_limit`: This parameter specifies the upper limit of memory usage of a big query. The unit is byte. This parameter takes effect only when it is set greater than 0. Default: 0.

> **NOTE**
>
> When a query running in a resource group exceeds the above big query limit, the query will be terminated with an error. You can also view error messages in the `ErrorCode` column of the FE node **fe.audit.log**.

You can set the resource group `type` to `short_query`, or `normal`.

- The default value is `normal`. You do not need specify `normal` in the parameter `type`.
- When queries hit a `short_query` resource group, the BE node reserves the CPU resource specified in `short_query.cpu_core_limit`. The CPU resource reserved for queries that hit `normal` resource group is limited to `BE core - short_query.cpu_core_limit`.
- When no query hits the `short_query` resource group, no limit is imposed to the resource of `normal` resource group.

> **CAUTION**
>
> - You can create at most ONE short query resource group in a StarRocks Cluster.
> - StarRocks does not set set a hard upper limit of CPU resource for `short_query` resource group.

classifier

Each classifier holds one or more conditions that can be matched to the properties of queries. StarRocks identifies the classifier that best matches each query based on the match conditions and assigns resources for running the query.

Classifiers support the following conditions:

- `user`: the name of the user.
- `role`: the role of the user.
- `query_type`: the type of the query. `SELECT` and `INSERT` (from v2.5) are supported. When INSERT tasks hit a resource group with `query_type` as `insert`, the BE node reserves the specified CPU resources for the tasks.
- `source_ip`: the CIDR block from which the query is initiated.
- `db`: the database which the query accesses. It can be specified by strings separated by commas `,`.

A classifier matches a query only when one or all conditions of the classifier match the information about the query. If multiple classifiers match a query, StarRocks calculates the degree of matching between the query and each classifier and identifies the classifier with the highest degree of matching.

> **NOTE**
>
> You can view the resource group to which a query belongs in the `ResourceGroup` column of the FE node **fe.audit.log**.
>
> If a query does not hit any classifiers, the default resource group `default_wg` is used. The resource limits of `default_wg` are as follows:
>
> - `cpu_core_limit`: 1 (`<=` v2.3.7) or the number of CPU cores in BE (`>` v2.3.7)
> - `mem_limit`: 100%
> - `concurrency_limit`: 0
> - `big_query_cpu_second_limit`: 0
> - `big_query_scan_rows_limit`: 0
> - `big_query_mem_limit`: 0

StarRocks calculates the degree of matching between a query and a classifier by using the following rules:

- If the classifier has the same value of `user` as the query, the degree of matching of the classifier increases by 1.
- If the classifier has the same value of `role` as the query, the degree of matching of the classifier increases by 1.
- If the classifier has the same value of `query_type` as the query, the degree of matching of the classifier increases by 1 plus the number obtained from the following

calculation: 1/Number of `query_type` fields in the classifier.

- If the classifier has the same value of `source_ip` as the query, the degree of matching of the classifier increases by 1 plus the number obtained from the following calculation: (32 - `cidr_prefix`)/64.
- If the classifier has the same value of `db` as the query, the degree of matching of the classifier increases by 10.

If multiple classifiers match a query, the classifier with a larger number of conditions has a higher degree of matching.

```
-- Classifier B has more conditions than Classifier A. Therefore,
Classifier B has a higher degree of matching than Classifier A.


classifier A (user='Alice')


classifier B (user='Alice', source_ip = '192.168.1.0/24')
```

If multiple matching classifiers have the same number of conditions, the classifier whose conditions are described more accurately has a higher degree of matching.

```
-- The CIDR block that is specified in Classifier B is smaller in
range than Classifier A. Therefore, Classifier B has a higher degree
of matching than Classifier A.
classifier A (user='Alice', source_ip = '192.168.1.0/16')
classifier B (user='Alice', source_ip = '192.168.1.0/24')

-- Classifier C has fewer query types specified in it than Classifier
D. Therefore, Classifier C has a higher degree of matching than
Classifier D.
classifier C (user='Alice', query_type in ('select'))
classifier D (user='Alice', query_type in ('insert','select'))
```

Isolate computing resources

You can isolate computing resources among queries by configuring resource groups and classifiers.

Enable resource groups

Execute the following statement to enable resource groups:

```
SET enable_resource_group = true;
```

If you want to globally enable resource groups, execute the following statement:

```
SET GLOBAL enable_resource_group = true;
```

Create resource groups and classifiers

Execute the following statement to create a resource group, associate the resource group with a classifier, and allocate computing resources to the resource group:

```
CREATE RESOURCE GROUP <group_name>
TO (
    user='string',
    role='string',
    query_type in ('select'),
    source_ip='cidr'
) --Create a classifier. If you create more than one classifier,
separate the classifiers with commas (`,`).
WITH (
    "cpu_core_limit" = "INT",
    "mem_limit" = "m%",
    "concurrency_limit" = "INT",
    "type" = "str" --The type of the resource group. Set the value to
normal.
);
```

Example:

```
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_core_limit' = '10',
    'mem_limit' = '20%',
    'big_query_cpu_second_limit' = '100',
    'big_query_scan_rows_limit' = '100000',
    'big_query_mem_limit' = '1073741824'
);
```

Specify resource group (Optional)

You can specify resource group for the current session directly.

```
SET resource_group = 'group_name';
```

View resource groups and classifiers

Execute the following statement to query all resource groups and classifiers:

```
SHOW RESOURCE GROUPS ALL;
```

Execute the following statement to query the resource groups and classifiers of the logged-in user:

```
SHOW RESOURCE GROUPS;
```

Execute the following statement to query a specified resource group and its classifiers:

```
SHOW RESOURCE GROUP group_name;
```

Example:

```
mysql> SHOW RESOURCE GROUPS ALL;
+------+--------+--------------+----------+-----------------+-------
-+-------------------------------------------------------------------
-----------------------------------------------+
```

```
| Name | Id     | CPUCoreLimit | MemLimit | ConcurrencyLimit | Type
| Classifiers
|
+------+--------+--------------+----------+------------------+-------
-+-------------------------------------------------------------------
-------------------------------------------------+
| rg1  | 300039 | 10           | 20.0%    | 11               | NORMAL
| (id=300040, weight=4.409375, user=rg1_user1, role=rg1_role1,
query_type in (SELECT), source_ip=192.168.2.1/24)         |
| rg1  | 300039 | 10           | 20.0%    | 11               | NORMAL
| (id=300041, weight=3.459375, user=rg1_user2, query_type in
(SELECT), source_ip=192.168.3.1/24)                         |
| rg1  | 300039 | 10           | 20.0%    | 11               | NORMAL
| (id=300042, weight=2.359375, user=rg1_user3,
source_ip=192.168.4.1/24)
|
| rg1  | 300039 | 10           | 20.0%    | 11               | NORMAL
| (id=300043, weight=1.0, user=rg1_user4)
|
+------+--------+--------------+----------+------------------+-------
-+-------------------------------------------------------------------
-------------------------------------------------+
```

> **NOTE**
>
> In the preceding example, `weight` indicates the degree of matching.

Manage resource groups and classifiers

You can modify the resource quotas for each resource group. You can also add or delete classifiers from resource groups.

Execute the following statement to modify the resource quotas for an existing resource group:

```
ALTER RESOURCE GROUP group_name WITH (
    'cpu_core_limit' = 'INT',
    'mem_limit' = 'm%'
);
```

Execute the following statement to delete a resource group:

```
DROP RESOURCE GROUP group_name;
```

Execute the following statement to add a classifier to a resource group:

```
ALTER RESOURCE GROUP <group_name> ADD (user='string', role='string',
query_type in ('select'), source_ip='cidr');
```

Execute the following statement to delete a classifier from a resource group:

```
ALTER RESOURCE GROUP <group_name> DROP (CLASSIFIER_ID_1,
CLASSIFIER_ID_2, ...);
```

Execute the following statement to delete all classifiers of a resource group:

```
ALTER RESOURCE GROUP <group_name> DROP ALL;
```

Monitor resource group

You can set monitor and alert for your resource groups.

Metrics you can monitor regarding resource groups include:

- FE
  - `starrocks_fe_query_resource_group`: The number of queries in each resource group.
  - `starrocks_fe_query_resource_group_latency`: The query latency percentile for each resource group.
  - `starrocks_fe_query_resource_group_err`: The number of terminated-with-error queries in each resource group.
- BE
  - `starrocks_be_resource_group_cpu_limit_ratio`: Instantaneous value of resource group CPU quota ratio.
  - `starrocks_be_resource_group_cpu_use_ratio`: Instantaneous value of resource group CPU usage ratio.

- `starrocks_be_resource_group_mem_limit_bytes`: Instantaneous value of resource group memory quota.
- `starrocks_be_resource_group_mem_allocated_bytes`: Instantaneous value of resource group memory usage.

What to do next

After you configure resource groups, you can manage memory resources and queries. For more information, see the following topics:

- Memory management
- Query management

Query queues

This topic describes how to manage query queues in StarRocks.

From v2.5, StarRocks supports query queues. With query queues enabled, StarRocks automatically queues the incoming queries when the concurrency threshold or resource limit is reached, thereby avoiding the overload deteriorating. Pending queries wait in a queue until there is enough compute resources available to begin execution.

You can set thresholds on CPU usage, memory usage, and query concurrency to trigger query queues.

Enable query queues

Query queues are disabled by default. You can enable query queues for INSERT loading, SELECT queries, and statistics queries by setting corresponding global session variables.

- Enable query queues for loading tasks:

```
SET GLOBAL enable_query_queue_load = true;
```

- Enable query queues for SELECT queries:

```
SET GLOBAL enable_query_queue_select = true;
```

- Enable query queues for statistics queries:

```
SET GLOBAL enable_query_queue_statistic = true;
```

Specify resource thresholds

You can set the thresholds that trigger query queues via the following global session variables:

Variable	Default	Description
query_queue_concurrency_limit	0	The upper limit of concurrent queries on a BE. It takes effect only after being set greater than `0`.
query_queue_mem_used_pct_limit	0	The upper limit of memory usage percentage on a BE. It takes effect only after being set greater than `0`. Range: [0, 1]
query_queue_cpu_used_permille_limit	0	The upper limit of CPU usage permille (CPU usage * 1000) on a BE. It takes effect only after being set greater than `0`. Range: [0, 1000]

> **NOTE**
>
> By default, BE reports resource usage to FE at one-second intervals. You can change this interval by setting the BE configuration item `report_resource_usage_interval_ms`.

Configure query queues

You can set the capacity of a query queue and the maximum timeout of queries in queues via the following global session variables:

Variable	Default	Description
query_queue_max_queued_queries	1024	The upper limit of queries in a queue. When this threshold is reached, incoming queries are

Variable	Default	Description
		rejected. It takes effect only after being set greater than `0`.
query_queue_pending_timeout_second	300	The maximum timeout of a pending query in a queue. When this threshold is reached, the corresponding query is rejected. Unit: second.

View query queue statistics

You can view the statistics of query queues via the following ways:

- Check the number of running queries, and memory and CPU usages in BE nodes using SHOW PROC:

```
mysql> SHOW PROC '/backends'\G
*************************** 1. row ***************************
...
    NumRunningQueries: 0
           MemUsedPct: 0.79 %
           CpuUsedPct: 0.0 %
```

- Check if a query is in a queue (when `IsPending` is `true`) using SHOW PROCESSLIST:

```
mysql> SHOW PROCESSLIST;
+------+------+--------------------+------+---------+-------------
-------+------+-------+-----------------+-----------+
| Id   | User | Host               | Db    | Command |
ConnectionStartTime | Time | State | Info            | IsPending |
+------+------+--------------------+------+---------+-------------
-------+------+-------+-----------------+-----------+
|    2 | root | xxx.xx.xxx.xx:xxxxx |       | Query   | 2022-11-24
18:08:29 |    0 | OK    | SHOW PROCESSLIST  | false     |
+------+------+--------------------+------+---------+-------------
-------+------+-------+-----------------+-----------+
```

- Check the FE audit log file **fe.audit.log**. The field `PendingTimeMs` indicates the time that a query is in a queue, and its unit is milliseconds.
- Check the following monitoring metrics:

Metrics	Description
starrocks_fe_query_queue_pending	Number of pending queries in the queue.
starrocks_fe_query_queue_total	Total number of queries that have been queued.
starrocks_fe_query_queue_timeout	Number of timeout queries in the queue.

Query Management

Number of user connections

`Property` is set for user granularity. To set the maximum number of connections between Client and FE, use the following command.

```
SET PROPERTY [FOR 'user'] 'key' = 'value' [, 'key' = 'value']
```

User properties include the resources assigned to the user. The properties set here are for the user, not `user_identity`. That is, if two users `jack'@'%` and `jack'@'192.%` are created by the `CREATE USER` statement, then the `SET PROPERTY` statement can work on the user `jack`, not `jack'@'%` or `jack'@'192.%`.

Example 1:

```
For the user `jack`, change the maximum number of connections to 1000
SET PROPERTY FOR 'jack' 'max_user_connections' = '1000';

Check the connection limit for the root user
SHOW PROPERTY FOR 'root';
```

Query-related session variables

The session variables can be set by 'key' = 'value', which can limit the concurrency, memory and other query parameters in the current session. For example:

- parallel_fragment_exec_instance_num

  The parallelism of the query with a default value of 1. It indicates the number of fragment instances on each BE. You can set this to half the number of CPU cores of the BE to improve query performance.

- exec_mem_limit

Memory limit of a query on each BE node, can be adjusted when a query reports insufficient memory.

- load_mem_limit

  Memory limit for import, can be adjusted when an import job reports insufficient memory.

Example 2:

```
set parallel_fragment_exec_instance_num  = 8;
set exec_mem_limit  = 137438953472;
```

capacity quota of database storage

The capacity quota of database storage is unlimited by default. And you can change quota value by using `alter database`.

```
ALTER DATABASE db_name SET DATA QUOTA quota;
```

The quota units are: B/K/KB/M/MB/G/GB/T/TB/P/PB

Example 3:

```
ALTER DATABASE example_db SET DATA QUOTA 10T;
```

Kill queries

To terminate a query on a particular connection with the following command:

```
kill connection_id;
```

The `connection_id` can be seen by `show processlist;` or `select connection_id();`.

```
 show processlist;
+------+------------+--------------------+-----------------+--------
-------+---------+------+-------+------+
| Id   | User       | Host                | Cluster         | Db
| Command | Time | State | Info |
+------+------------+--------------------+-----------------+--------
-------+---------+------+-------+------+
|    1 | starrocksmgr | 172.26.34.147:56208 | default_cluster |
starrocks_monitor | Sleep   |    8 |       |      |
|  129 | root       | 172.26.92.139:54818 | default_cluster |
| Query   |    0 |       |      |
|  114 | test       | 172.26.34.147:57974 | default_cluster |
ssb_100g      | Query   |    3 |       |      |
|    3 | starrocksmgr | 172.26.34.147:57268 | default_cluster |
starrocks_monitor | Sleep   |    8 |       |      |
|  100 | root       | 172.26.34.147:58472 | default_cluster | ssb_100
| Sleep   |  637 |       |      |
|  117 | starrocksmgr | 172.26.34.147:33790 | default_cluster |
starrocks_monitor | Sleep   |    8 |       |      |
|    6 | starrocksmgr | 172.26.34.147:57632 | default_cluster |
starrocks_monitor | Sleep   |    8 |       |      |
|  119 | starrocksmgr | 172.26.34.147:33804 | default_cluster |
starrocks_monitor | Sleep   |    8 |       |      |
|  111 | root       | 172.26.92.139:55472 | default_cluster |
| Sleep   | 2758 |       |      |
+------+------------+--------------------+-----------------+--------
-------+---------+------+-------+------+
9 rows in set (0.00 sec)

mysql> select connection_id();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|              98 |
+-----------------+


mysql> kill 114;
Query OK, 0 rows affected (0.02 sec)
```

Memory Management

This section briefly introduces memory classification and StarRocks' methods of managing memory.

Memory Classification

Explanation:

Metric	Name	Description
process	Total memory used of BE	
query_pool	Memory used by data querying	Consists of two parts: memory used by the execution layer and memory used by the storage layer.
load	Memory used by data loading	Generally MemTable
table_meta	Metadata memory	S Schema, Tablet metadata, RowSet metadata, Column metadata, ColumnReader, IndexReader
compaction	Multi-version memory compaction	compaction that happens after data import is complete
snapshot	Snapshot memory	Generally used for clone, little memory usage
column_pool	Column pool memory	Request to release column cache for accelerated column
page_cache	BE's own PageCache	The default is off, the user can turn it on by modifying the BE file

Memory-related configuration

- **BE Configuration**

Name	Default	De
vector_chunk_size	4096	Number of chunk ro
mem_limit	90%	BE process memory as a percentage ("8 ("100G"). The defau server's memory si 80%. You need to c you want to deploy memory-intensive s
disable_storage_page_cache	false	The boolean value t disable PageCache enabled, StarRocks scanned data. Page improve the query p queries are repeate indicates to disable together with `stor` you can accelerate scenarios with suffi and much data scar item has been chan since StarRocks v2.
write_buffer_size	104857600	The capacity limit o exceeding which a performed.
load_process_max_memory_limit_bytes	107374182400	The upper limit of n be taken up by all lo node. Its value is th `mem_limit *`

Name	Default	De
		`load_process_ma` `/ 100` and `load_process_ma` If this threshold is e backpressure will b
load_process_max_memory_limit_percent	30	The maximum perc resources that can processes on a BE smaller one betwee `load_process_ma` `/ 100` and `load_process_ma` If this threshold is e backpressure will b
default_load_mem_limit	2147483648	If the memory limit reached for a single swipe will be trigge modified with the S `load_mem_limit`
max_compaction_concurrency	-1	The maximum conc (both Base Compac Compaction). The v limit is imposed on
cumulative_compaction_check_interval_seconds	1	Interval of compact

- **Session variables**

Name	Default	Description
query_mem_limit	0	Memory limit of a query on each BE node

Name	Default	Description
load_mem_limit	0	Memory limit of a single import task. If the value is 0, `exec_mem_limit` will be taken

View memory usage

- **mem_tracker**

```
//View the overall memory statistics
<http://be_ip:be_web_port/mem_tracker>

// View fine-grained memory statistics
<http://be_ip:be_web_port/mem_tracker?type=query_pool&upper_level=3>
```

- **tcmalloc**

```
<http://be_ip:be_web_port/memz>
```

```
------------------------------------------------
MALLOC:      777276768 (  741.3 MiB) Bytes in use by application
MALLOC: +   8851890176 ( 8441.8 MiB) Bytes in page heap freelist
MALLOC: +    143722232 (  137.1 MiB) Bytes in central cache freelist
MALLOC: +     21869824 (   20.9 MiB) Bytes in transfer cache freelist
MALLOC: +    832509608 (  793.9 MiB) Bytes in thread cache freelists
MALLOC: +     58195968 (   55.5 MiB) Bytes in malloc metadata
MALLOC:   ------------
MALLOC: =  10685464576 (10190.5 MiB) Actual memory used (physical +
swap)
MALLOC: +  25231564800 (24062.7 MiB) Bytes released to OS (aka
unmapped)
MALLOC:   ------------
MALLOC: =  35917029376 (34253.1 MiB) Virtual address space used
MALLOC:
MALLOC:         112388              Spans in use
MALLOC:            335              Thread heaps in use
MALLOC:           8192              Tcmalloc page size
------------------------------------------------
Call ReleaseFreeMemory() to release freelist memory to the OS (via
```

```
madvise()).
Bytes released to the OS take up virtual address space but no
physical memory.
```

The memory queried by this method is accurate. However, some memory in StarRocks is reserved but not in use. TcMalloc counts the memory that is reserved, not the memory used.

Here `Bytes in use by application` refers to the memory currently in use.

- **metrics**

```
curl -XGET http://be_ip:be_web_port/metrics | grep 'mem'
curl -XGET http://be_ip:be_web_port/metrics | grep 'column_pool'
```

The value of metrics is updated every 10 seconds. It is possible to monitor some of the memory statistics with older versions.

Load Balancing

When deploying multiple FE nodes, users can deploy a load balancing layer on top of the FEs to achieve high availability.

The following are some high availability options:

Code approach

One way is to implement code at the application layer to perform retry and load balancing . For example, if a connection is broken, it will automatically retry on other connections. This approach requires users to configure multiple FE node addresses.

JDBC Connector

JDBC connector supports automatic retry:

```
jdbc:mysql:loadbalance://[host1][:port],[host2][:port][,[host3]
[:port]]...[/[database]][?
propertyName1=propertyValue1[&propertyName2=propertyValue2]...]
```

ProxySQL

ProxySQL is a MySQL proxy layer that supports read/write separation, query routing, SQL caching, dynamic load configuration, failover, and SQL filtering.

StarRocks FE is responsible for receiving connection and query requests, and it's horizontally scalable and highly available. However FE requires users to set up a proxy layer on top of it to achieve automatic load balancing. See the following steps for setup:

1. Install relevant dependencies

```
yum install -y gnutls perl-DBD-MySQL perl-DBI perl-devel
```

2. Download the installation package

```
wget
https://github.com/sysown/proxysql/releases/download/v2.0.14/proxysql-
2.0.14-1-centos7.x86_64.rpm
```

3. Decompress to the current directory

```
rpm2cpio proxysql-2.0.14-1-centos7.x86_64.rpm | cpio -ivdm
```

4. Modify the configuration file

```
vim ./etc/proxysql.cnf
```

Direct to a directory that the user has privilege to access (absolute path):

```
datadir="/var/lib/proxysql"
errorlog="/var/lib/proxysql/proxysql.log"
```

5. Start

```
./usr/bin/proxysql -c ./etc/proxysql.cnf --no-monitor
```

6. Log in

```
mysql -u admin -padmin -h 127.0.0.1 -P6032
```

7. Configure the global log

```
SET mysql-eventslog_filename='proxysql_queries.log';
SET mysql-eventslog_default_log=1;
SET mysql-eventslog_format=2;
LOAD MYSQL VARIABLES TO RUNTIME;
SAVE MYSQL VARIABLES TO DISK;
```

8. Insert into the leader node

```
insert into mysql_servers(hostgroup_id, hostname, port) values(1,
'172.26.92.139', 8533);
```

9. Insert the observer nodes

```
insert into mysql_servers(hostgroup_id, hostname, port) values(2,
'172.26.34.139', 9931);
insert into mysql_servers(hostgroup_id, hostname, port) values(2,
'172.26.34.140', 9931);
```

10. Load the configuration

```
load mysql servers to runtime;
save mysql servers to disk;
```

11. Configure the username and password

```
insert into mysql_users(username, password, active,
default_hostgroup, backend, frontend) values('root',
'*94BDCEBE19083CE2A1F959FD02F964C7AF4CFC29', 1, 1, 1, 1);
```

12. Load the configuration

```
load mysql users to runtime;
save mysql users to disk;
```

13. Write to the proxy rules

```
insert into mysql_query_rules(rule_id, active, match_digest,
destination_hostgroup, mirror_hostgroup, apply) values(1, 1, '.', 1,
2, 1);
```

14. Load the configuration

```
load mysql query rules to runtime;
save mysql query rules to disk;
```

Replica management

Terminology

- Tablet: Logical slice of a table, a table has multiple tablets.
- Replica: Copy of a tablet, 3 replicas per tablet by default.
- Healthy Replica: A replica whose backend is alive and version is complete.
- TabletChecker (TC): A resident background thread that periodically scans all Tablets and decides whether to send tablets to TabletScheduler based on their status.
- TabletScheduler (TS): A resident background thread that processes tablets sent by TabletChecker, and also performs cluster replica balancing.
- TabletSchedCtx (TSC): A wrapper for a tablet. When a tablet is selected by TC, it is encapsulated as a TSC and sent to TS.
- Storage Medium: StarRocks supports different storage media for partition granularity, including SSD and HDD. The scheduling of replicas varies for different storage media.

Status

Replica Status

The health states of a Replica are as follows.

- **BAD**

The replica is corrupted. This includes, but is not limited to, unrecoverable damage to the replica caused by disk failures, bugs, etc.

- **VERSION_MISSING**

Each batch import corresponds to one version of data. You should see several consecutive versions in one replica. Due to import errors, some replicas may have incomplete versions of data.

- **HEALTHY**

The replica has correct data and the BE node where it is located is in normal state (with a normal heartbeat and is not in the process of being taken offline).

Tablet Status

The health state of a Tablet is determined by the status of all its replicas:

- **REPLICA_MISSING**

Replica missing. That is, the number of surviving replicas is less than the expected number of replicas.

- **VERSION_INCOMPLETE**

The number of surviving replicas is greater than or equal to the expected number, but the number of healthy replicas is less than the expected number.

- **REPLICA_RELOCATING**

The number of surviving replicas that have full version is equal to `replication num`. But the BE node where some of the replicas are located is unavailable (e.g., in decommission status).

- **REPLICA_MISSING_IN_CLUSTER**

When using multiple clusters, the number of healthy replicas is greater than or equal to the expected number, but the number of replicas within the corresponding cluster is less than the expected number.

- **REDUNDANT**

Replica redundancy. All healthy replicas are in the corresponding cluster, but the total number of replicas is greater than the expected numbers due to redundant unavailable replicas.

- **FORCE_REDUNDANT**

This is a special status. It only occurs when the expected number of replicas is greater than or equal to the number of available nodes, and the Tablet is in the replica missing state. In this case, one replica needs to be deleted first to ensure that there are available nodes for creating new replicas.

- **COLOCATE_MISMATCH**

Tablet status for the Colocation attribute, indicating that the sharded replica does not match the distribution specified by the Colocation Group.

- **COLOCATE_REDUNDANT**

Tablet status for the Colocation attribute, indicating that the tablet replica of the Colocation table is redundant.

- **HEALTHY**

Healthy tablets.

Replica Repair

TabletChecker periodically checks the status of all tablets. Unhealthy tablets are handed over to TabletScheduler for scheduling and repairing. The repair is done on the BE as a clone task. The FE is responsible for generating the clone task.

> Note 1: The main idea of replica repair is to first create or patch new replicas to reach the desired value, and then delete any redundant replicas. Note 2: A clone task is to copy target data from a remote BE to a destination BE.

For different states, we use different methods to repair.

1. REPLICA_MISSING/REPLICA_RELOCATING Select an available BE node as the destination BE. Select a healthy replica as the source. The clone task will copy a full replica from the source to the destination BE. For replica replenishment, an available BE node will be selected regardless of the storage medium.

2. VERSION_INCOMPLETE Select a relatively complete replica as the destination. Select a healthy replica as the source. The clone task will copy the missing version from the source to the destination.

3. REPLICA_MISSING_IN_CLUSTER This status is handled in the same way as REPLICA_MISSING.

4. REDUNDANT Usually after a replica repair, the tablet will have redundant replicas. Choose a redundant replica and delete it. The selection of redundant replicas follows

this order:

   - The BE where the replica is located is already offline
   - The replica is corrupted
   - The BE where the replica is located is out of connection or in the process of being taken offline
   - The replica is in CLONE state, which is an intermediate state during clone task execution)
   - The replica has a missing version
   - The cluster where the replica is located is incorrect
   - The BE node where the replica is located is heavily loaded

5. FORCE_REDUNDANT Although the Tablet has a missing replica, there are no available nodes to create a new one. In this case, a replica must be deleted first to free up a node for creating a new replica. The order of deleting replicas is the same as REDUNDANT.

6. COLOCATE_MISMATCH Select one of the BE nodes specified in the Colocation Group as the destination node for replica replenishment.

7. COLOCATE_REDUNDANT Delete a replica on a BE node that is not specified in the Colocation Group.

StarRocks does not deploy replicas of the same Tablet on different BEs of the same host, which ensures that even if all BEs of the same host fail, some replicas are still alive.

Replica Scheduling

Scheduling priority

Tablets waiting to be scheduled in TabletScheduler are given different priorities depending on their status. Tablets with higher priority will be scheduled first. Priority levels are as follows:

- VERY_HIGH

- In `REDUNDANT` state. Tablets with redundant replicas are given very high priority. Although the replica redundancy is the least urgent, it is the fastest to process and free up resources (such as disk space).

- In `FORCE_REDUNDANT` state. Same as above.
- HIGH
- In `REPLICA_MISSING` state and most replicas are missing (e.g., 2 of 3 replicas are missing)
- In `VERSION_INCOMPLETE` state and most replicas have missing versions
- In `COLOCATE_MISMATCH` state. Tablets associated with the Colocation table will be fixed as soon as possible.
- In `COLOCATE_REDUNDANT` state.
- NORMAL
- In `REPLICA_MISSING` state but most replicas survive (e.g. 1 of 3 replicas is missing)
- In `VERSION_INCOMPLETE` state but most replicas have complete versions
- In `REPLICA_RELOCATING` state and most replicas need to be relocated (e.g. 3 copies with 2)
- LOW
- In `REPLICA_MISSING_IN_CLUSTER` state
- In `REPLICA_RELOCATING` state but most replicas are stable

Manually determined priority

The system will automatically determine the scheduling priority. However, there are times when users want certain tablets to be repaired faster. Therefore, we provide a command where the user can specify tablets of a certain table or partition to be repaired first.

ADMIN REPAIR TABLE tbl [PARTITION (p1, p2, ...)] ;

This command tells TC to give `VERY_HIGH` priority to the problematic Tablet so it can be repaired first.

> Note: This command does not guarantee a successful repair, and the priority will change with the scheduling of TS. This information will be lost when the leader FE is

> changed or restarted.

Prioritization can be canceled with the following command.

ADMIN CANCEL REPAIR TABLE tbl [PARTITION (p1, p2, ...)] ;

Priority Scheduling

Prioritization ensures that severely damaged tablets are repaired first. However, if a high-priority repair task keeps failing, it will result in low-priority tasks remaining unscheduled. Therefore, we dynamically adjust the priority based on the status of each task to ensure that all tasks have a chance to be scheduled.

- If the scheduling fails for 5 consecutive times (e.g., unable to acquire resources, unable to find a suitable source or destination, etc.), the task will be deprioritized.
- A task will be prioritized if it has not been scheduled for 30 minutes.
- The priority of a task can't be adjusted twice within five minutes.

To ensure that the initial priority level is weighted, `VERY_HIGH` tasks can only be deprioritized to `NORMAL` at most, and `LOW` tasks can only be prioritized to `HIGH` at most. The priority adjustment also applies to tasks whose priority is manually set by users.

Replica Balancing

StarRocks automatically performs replica balancing within a cluster. The main idea of balancing is to create replicas on a low-load node and delete replicas on the high-load node. More than one storage medium may exist on different BE nodes within the same cluster. To keep tablets in their original storage medium after balancing, BE nodes are divided based on storage media , allowing the load balancing to be wisely scheduled .

Similarly, replica balancing ensures that no replicas of the same Tablet are deployed on BEs of the same host.

BE Node Load

ClusterLoadStatistics (CLS) shows the load balancing of each Backend in a cluster and triggersTabletScheduler to balance the cluster. Currently, we `**Disk Utilization**` and

`**Number of Replicas**` to calculate `loadScore` for each BE. The higher the score, the heavier the load on the BE.

`capacityCoefficient` and `replicaNumCoefficient` (sum to 1) are the weighting factors for `Disk Utilization` and `Number of Replicas` respectively. The `capacityCoefficient` is dynamically adjusted according to the actual disk usage. When the overall disk utilization of a BE is below 50%, the `capacityCoefficient` value is 0.5. When the disk utilization is above 75% (configurable via the FE configuration item `capacity_used_percent_high_water`), the value is 1. If the utilization is between 50% and 75%, the weight factor increases smoothly based on this formula:

`capacityCoefficient= 2 * disk utilization - 0.5`

This weighting factor ensures that when the disk usage is too high, the load score of this Backend will be higher, forcing to reduce the load on this BE as soon as possible.

TabletScheduler will update the CLS every 1 minute.

Balancing Policy

TabletScheduler selects a certain number of healthy tablets as candidates for balancing through LoadBalancer in each scheduling round and adjust future scheduling based on these candidate slices.

Resource Control

Both replica repair and balancing are done by copying across BEs. Too many tasks simultaneously performed on the same BE lead to an increase in IO pressure. Therefore, StarRocks controls the number of tasks that can be executed on each node during scheduling. The smallest unit of resource control is the disk (i.e., a data path specified in `be.conf`). By default, two slots for each disk are allocated for replica repair. A clone task occupies one slot on the source and one slot on the destination. If the disk has zero slots, no more tasks will be assigned to it. This number of slots can be configured with the `schedule_slot_num_per_path` parameter of FE.

In addition, by default, two slots for each disk are allocated for replica balancing. The purpose is to prevent heavily loaded nodes from being occupied by repair tasks and not being able to free up space through balancing.

View replica status

The replica status is **only present in the leader FE node**. Therefore, the following commands need to be executed by connecting directly to the leader FE.

View replica status

1. Global Status Check
   The `SHOW PROC '/statistic';` command allows you to view the replica status of the entire cluster.

```
+----------+-----------------------------+----------+------------
--+----------+-----------+------------+-------------------+-----
------------------+
| DbId     | DbName                         | TableNum |
PartitionNum | IndexNum | TabletNum | ReplicaNum |
UnhealthyTabletNum | InconsistentTabletNum |
+----------+-----------------------------+----------+------------
--+----------+-----------+------------+-------------------+-----
------------------+
| 35153636 | default_cluster:DF_Newrisk  | 3        | 3
| 3        | 96        | 288        | 0                    | 0
|
| 48297972 | default_cluster:PaperData   | 0        | 0
| 0        | 0         | 0          | 0                    | 0
|
| 5909381  | default_cluster:UM_TEST     | 7        | 7
| 10       | 320       | 960        | 1                    | 0
|
| Total    | 240                         | 10       | 10
| 13       | 416       | 1248       | 1                    | 0
|
+----------+-----------------------------+----------+------------
--+----------+-----------+------------+-------------------+-----
------------------+
```

   The `UnhealthyTabletNum` column shows how many unhealthy Tablets are in the corresponding Database. The `InconsistentTabletNum` column shows how many Tablets in the corresponding Database are in a replica inconsistent state. The last `Total` row gives the statistics of the entire cluster. Normally `UnhealthyTabletNum` and `InconsistentTabletNum` should be zero. If they are not zero, you can further

check the tablet's information. As shown above, there is one tablet in the `UM_TEST` database with unhealthy status, then you can use the following command to see which tablet it is.

`SHOW PROC '/statistic/5909381';`
where `5909381` is the corresponding DbId.

```
+------------------+--------------------+
| UnhealthyTablets | InconsistentTablets |
+------------------+--------------------+
| [40467980]       | []                 |
+------------------+--------------------+
```

The above result shows the specific unhealthy Tablet ID (40467980). Next, we will describe how to check the status of each replica of a specific Tablet.

2. Table (Partition) Level Status Check
   Users can check the status of replicas of a specific table or partition with the following command and can filter the status by the WHERE statement. For example, to view the status of NORMAL replicas of `p1` and `p2` in `tbl1`.
   `ADMIN SHOW REPLICA STATUS FROM tbl1 PARTITION (p1, p2) WHERE STATUS = "OK";`

```
+----------+-----------+-----------+---------+------------------
+-------------------+-----------------+------------+-----------
-+-------+--------+--------+
| TabletId | ReplicaId | BackendId | Version | LastFailedVersion
| LastSuccessVersion | CommittedVersion | SchemaHash | VersionNum
| IsBad | State  | Status |
+----------+-----------+-----------+---------+------------------
+-------------------+-----------------+------------+-----------
-+-------+--------+--------+
| 29502429 | 29502432  | 10006     | 2       | -1
| 2                 | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502429 | 36885996  | 10002     | 2       | -1
| -1                | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502429 | 48100551  | 10007     | 2       | -1
| -1                | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502433 | 29502434  | 10001     | 2       | -1
```

```
| 2                  | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502433 | 44900737  | 10004     | 2       | -1
| -1                 | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502433 | 48369135  | 10006     | 2       | -1
| -1                 | 1               | -1         | 2
| false | NORMAL | OK     |
+----------+----------+----------+---------+-----------------
+------------------+-----------------+-----------+----------
-+-------+--------+--------+
```

If `IsBad = true`, that means that the replica is corrupted. The `Status` column shows additional information. The `ADMIN SHOW REPLICA STATUS` command is mainly used to view the health status of a replica. Users can also view additional information with the following command.

`SHOW TABLET FROM tbl1;`

```
+----------+----------+----------+-----------+---------+------
-------+------------------+----------------------+-------------
-----+--------------------+--------------+----------+---------
-+--------+----------------------+-------------+-------------
---------+-------------+--------------------+----------------
-----+---------------------+
| TabletId | ReplicaId | BackendId | SchemaHash | Version |
VersionHash | LstSuccessVersion | LstSuccessVersionHash |
LstFailedVersion | LstFailedVersionHash | LstFailedTime |
DataSize | RowCount | State  | LstConsistencyCheckTime |
CheckVersion |     CheckVersionHash | VersionCount | PathHash
| MetaUrl               | CompactionStatus     |
+----------+----------+----------+-----------+---------+------
-------+------------------+----------------------+-------------
-----+--------------------+--------------+----------+---------
-+--------+----------------------+-------------+-------------
---------+-------------+--------------------+----------------
-----+---------------------+
| 29502429 | 29502432  | 10006     | 1421156361 | 2       | 0
| 2                  | 0                    | -1                |
0                    | N/A           | 784      | 0        |
NORMAL | N/A                     | -1           |     -1
| 2            | -5822326203532286804 | url                  |
url                  |
| 29502429 | 36885996  | 10002     | 1421156361 | 2       | 0
```

```
| -1                | 0                     | -1                |
0                    | N/A           | 784       | 0         |
NORMAL | N/A                      | -1            |     -1
| 2             | -1441285706148429853 | url                    |
url                    |
| 29502429 | 48100551  | 10007     | 1421156361 | 2        | 0
| -1                | 0                     | -1                |
0                    | N/A           | 784       | 0         |
NORMAL | N/A                      | -1            |     -1
| 2             | -4784691547051455525 | url                    |
url                    |
+----------+-----------+-----------+------------+---------+------
-------+------------------+----------------------+-------------
-----+--------------------+--------------+----------+---------
-+--------+-----------------------+-------------+-------------
---------+-------------+--------------------+----------------
-----+---------------------+
```

Additional information includes replica size, number of rows, number of versions, data path where it is located, etc.

> Note: The `State` column does not represent the health status of the replica itself, but the status of the replica under certain tasks (e.g., `CLONE`, `SCHEMA_CHANGE`, `ROLLUP`, etc.).

In addition, users can check whether the replicas are evenly distributed with the following command
`ADMIN SHOW REPLICA DISTRIBUTION FROM tbl1;`

```
+-----------+------------+-------+---------+
| BackendId | ReplicaNum | Graph | Percent |
+-----------+------------+-------+---------+
| 10000     | 7          |       | 7.29 %  |
| 10001     | 9          |       | 9.38 %  |
| 10002     | 7          |       | 7.29 %  |
| 10003     | 7          |       | 7.29 %  |
| 10004     | 9          |       | 9.38 %  |
| 10005     | 11         | >     | 11.46 % |
| 10006     | 18         | >     | 18.75 % |
| 10007     | 15         | >     | 15.62 % |
| 10008     | 13         | >     | 13.54 % |
+-----------+------------+-------+---------+
```

Information includes the number of replicas on each BE node, percentage, and simple graphical display is shown above.

Tablet Level Status Check

Users can use the following command to check the status of a specific Tablet. For example, to check the status of the tablet with ID 29502553.

`SHOW TABLET 29502553;`

```
+----------------------+-----------+---------------+-----------+---
-------+----------+-------------+----------+--------+---------------
-------------------------------------------------------+
| DbName                 | TableName | PartitionName | IndexName |
DbId     | TableId  | PartitionId | IndexId  | IsSync | DetailCmd
|
+----------------------+-----------+---------------+-----------+---
-------+----------+-------------+----------+--------+---------------
-------------------------------------------------------+
| default_cluster:test   | test      | test          | test      |
29502391 | 29502428 | 29502427    | 29502428 | true   | SHOW PROC
'/dbs/29502391/29502428/partitions/29502427/29502428/29502553'; |
+----------------------+-----------+---------------+-----------+---
-------+----------+-------------+----------+--------+---------------
-------------------------------------------------------+
```

Above shows the database, table, partition, index and other information corresponding to the tablet. Users can copy the command in `DetailCmd` to check out the details `SHOW PROC '/dbs/29502391/29502428/partitions/29502427/29502428/29502553';`

```
+-----------+-----------+---------+-------------+-------------------
+----------------------+-----------------+--------------------+--
-------------+------------+----------+----------+--------+-------+---
-----------+---------------------+----------+-----------------+
| ReplicaId | BackendId | Version | VersionHash | LstSuccessVersion |
LstSuccessVersionHash | LstFailedVersion | LstFailedVersionHash |
LstFailedTime | SchemaHash | DataSize | RowCount | State  | IsBad |
VersionCount | PathHash              | MetaUrl  | CompactionStatus |
+-----------+-----------+---------+-------------+-------------------
+----------------------+-----------------+--------------------+--
-------------+------------+----------+----------+--------+-------+---
-----------+---------------------+----------+-----------------+
| 43734060  | 10004     | 2       | 0           | -1                |
```

```
0                    | -1               | 0                    | N/A
| -1         | 784      | 0        | NORMAL | false | 2            |
-8566523878520798656 | url      | url              |
| 29502555  | 10002     | 2       | 0           | 2                 |
0                    | -1               | 0                    | N/A
| -1         | 784      | 0        | NORMAL | false | 2            |
1885826196444191611  | url      | url              |
| 39279319  | 10007     | 2       | 0           | -1                |
0                    | -1               | 0                    | N/A
| -1         | 784      | 0        | NORMAL | false | 2            |
1656508631294397870  | url      | url              |
+----------+----------+--------+------------+-----------------
+---------------------+-----------------+--------------------+--
------------+-----------+---------+---------+-------+------+---
----------+---------------------+---------+-----------------+
```

Above shows all replicas of the corresponding Tablet. The content shown here is the same as `SHOW TABLET FROM tbl1;`. But here you can clearly see the status of all replicas of a specific Tablet.

Scheduling tasks for replicas

View the tasks that are waiting to be scheduled by `SHOW PROC '/cluster_balance/pending_tablets';`

```
+----------+--------+----------------+---------+----------+---------
-+-------+---------+--------+----------+---------+-----------------
--+--------------------+--------------------+----------+------+----
---------+--------------+--------------------+-----------+--------
------------+--------+--------------------+----------------------
-------+
| TabletId | Type   | Status           | State   | OrigPrio | DynmPrio
| SrcBe | SrcPath | DestBe | DestPath | Timeout | Create
| LstSched            | LstVisit            | Finished | Rate |
FailedSched | FailedRunning | LstAdjPrio          | VisibleVer |
VisibleVerHash      | CmtVer | CmtVerHash          | ErrMsg
|
+----------+--------+----------------+---------+----------+---------
-+-------+---------+--------+----------+---------+-----------------
--+--------------------+--------------------+----------+------+----
---------+--------------+--------------------+-----------+--------
------------+--------+--------------------+----------------------
-------+
```

```
| 4203036  | REPAIR | REPLICA_MISSING | PENDING | HIGH     | LOW
| -1    | -1      | -1     | -1       | 0       | 2019-02-21 15:00:20
| 2019-02-24 11:18:41 | 2019-02-24 11:18:41 | N/A      | N/A  | 2
| 0             | 2019-02-21 15:00:43 | 1          | 0
| 2      | 0                  | unable to find source replica |
+----------+--------+-----------------+---------+----------+---------
-+-------+---------+--------+----------+---------+------------------
--+--------------------+--------------------+----------+------+----
---------+--------------+--------------------+-----------+--------
------------+--------+--------------------+----------------------
-------+
```

- A breakdown of all the properties is as follows.
    - TabletId: The ID of the Tablet that is waiting for scheduling. A task for each Tablet
    - Type: The type of task, either REPAIR or BALANCE
    - Status: The current status of the Tablet, such as `REPLICA_MISSING`
    - State: the state of this scheduling task, may be `PENDING`/`RUNNING`/`FINISHED`/`CANCELLED`/`TIMEOUT`/`UNEXPECTED`
    - OrigPrio: The initial priority
    - DynmPrio: The dynamic priority
    - SrcBe: The ID of the source BE node
    - SrcPath: The path of the source BE node
    - DestBe: The ID of the destination BE node
    - DestPath: The path of the destination BE node
    - Timeout: When the task is scheduled successfully, the timeout of the task is shown here in seconds
    - Create: The time when the task was created
    - LstSched: The time the task was last scheduled
    - LstVisit: The time the task was last accessed. Here " accessed " refers to scheduling, task execution reporting, etc.
    - Finished: The time when the task was finished.
    - Rate: The data copy rate of the clone task
    - FailedSched: The number of times the task scheduling failed
    - FailedRunning: The number of times the task failed to execute
    - LstAdjPrio: The time the task priority was last adjusted

- CmtVer/CmtVerHash/VisibleVer/VisibleVerHash: Version information used to execute the clone task
- ErrMsg: The error message when the task is scheduled and run

View the running tasks

`SHOW PROC '/cluster_balance/running_tablets';`
Same properties are used here. See `pending tablets` for details.

View the finished tasks

`SHOW PROC '/cluster_balance/history_tablets';`
By default, we only keep the last 1000 completed tasks. Same properties are used here. See `pending tablets` for details. If the `State` is `FINISHED`, that means the task is completed properly. Otherwise, check `ErrMsg` for the reason of a failed task.

Blacklist Management

In some cases, administrators need to disable certain patterns of SQL to avoid SQL from triggering cluster crashes or unexpected high concurrent queries.

StarRocks allows users to add, view, and delete SQL blacklists.

Syntax

EnableSQL blacklisting via `enable_sql_blacklist`. The default is False (off).

```
admin set frontend config ("enable_sql_blacklist" = "true")
```

The admin user who has ADMIN_PRIV privileges can manage blacklists by executing the following commands:

```
ADD SQLBLACKLIST "<sql>"
DELETE SQLBLACKLIST <sql_index_number>
SHOW SQLBLACKLISTS
```

- When `enable_sql_blacklist` is true, every SQL query needs to be filtered by sqlblacklist. If it matches, the user will be informed that theSQL is in the blacklist. Otherwise, the SQL will be executed normally. The message may be as follows when the SQL is blacklisted:

`ERROR 1064 (HY000): Access denied; sql 'select count (*) from test_all_type_select_2556' is in blacklist`

Add blacklist

```
ADD SQLBLACKLIST "<sql>"
```

sql is a regular expression for a certain type of SQL.

> **TIP**
>
> Currently, StarRocks supports adding SELECT statements to the SQL Blacklist.

Since SQL itself contains the common characters `(`, `)`, `*`, `.` that may be mixed up with the semantics of regular expressions, so we need to distinguish those by using escape characters. Given that `(` and `)` are used too often in SQL, there is no need to use escape characters. Other special characters need to use the escape character `\` as a prefix. For example:

- Prohibit `count(\*)`:

```
ADD SQLBLACKLIST "select count(\\*) from .+"
```

- Prohibit `count(distinct)`:

```
ADD SQLBLACKLIST "select count(distinct .+) from .+"
```

- Prohibit order by limit `x`, `y`, `1 <= x <=7`, `5 <=y <=7`:

```
ADD SQLBLACKLIST "select id_int from test_all_type_select1 order by
id_int limit [1-7], [5-7]"
```

- Prohibit complex SQL:

```
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"
```

View blacklist

```
SHOW SQLBLACKLIST
```

Result format: `Index | Forbidden SQL`

For example:

```
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 .+
|
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
---------------------------+
```

The SQL shown in `Forbidden SQL` is escaped for all SQL semantic characters.

Delete blacklist

```
DELETE SQLBLACKLIST <sql_index_number>
```

`<sql_index_number>` is a list of SQL IDs separated by comma (,).

For example, delete the No.3 and No.4 SQLs in the above blacklist:

```
delete sqlblacklist  3, 4;
```

Then, the remaining sqlblacklist is as follows:

```
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 |
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
---------------------------+
```

File manager

With file manager, you can create, view, and delete files, such as the files that are used to access external data sources: public key files, private key files, and certificate files. You can reference or access the created files by using commands.

Basic concepts

File: refers to the file that is created and saved in StarRocks. After a file is created and stored in StarRocks, StarRocks assigns a unique ID to the file. You can find a file based on the database name, catalog, and file name. In a database, only an admin user can create and delete files, and all users who have permissions to access a database can use the files that belong to the database.

Before you begin

- Configure the following parameters for each FE.
  - `small_file_dir`: the path in which the uploaded files are stored. The default path is `small_files/`, which is in the runtime directory of the FE. You need to specify this parameter in the **fe.conf** file and then restart the FE to allow the change to take effect.
  - `max_small_file_size_bytes`: the maximum size of a single file. The default value of this parameter is 1 MB. If the size of a file exceeds the value of this parameter, the file cannot be created. You can specify this parameter by using the ADMIN SET CONFIG statement.
  - `max_small_file_number`: the maximum number of files that can be created within a cluster. The default value of this parameter is 100. If the number of files that you have created reaches the value of this parameter, you cannot continue creating files. You can specify this parameter by using the ADMIN SET CONFIG statement.

> Note: Increasing the values of the two parameters causes an increase in the memory usage of the FE. Therefore, we recommend that you do not increase the values of the

> two parameters unless necessary.

- Configure the following parameters for each BE.

`small_file_dir`: the path in which the downloaded files are stored. The default path is `lib/small_files/`, which is in the runtime directory of the BE. You can specify this parameter in the **be.conf** file.

Create a file

You can execute the CREATE FILE statement to create a file. For more information, see CREATE FILE. After a file is created, the file is uploaded and persisted in StarRocks.

View a file

You can execute the SHOW FILE statement to view the information about a file stored in a database. For more information, see SHOW FILE.

Delete a file

You can execute the DROP FILE statement to delete a file. For more information, see DROP FILE.

How an FE and a BE use a file

- **FE**: The SmallFileMgr class stores the data related to the file in the specified directory of the FE. Then the SmallFileMgr class returns a local file path for the FE to use the file.
- **BE**: The BE calls the **/api/get_small_file API** (HTTP) to download the file to its specified directory and record the information of the file. When the BE requests to use the file, the BE checks whether the file has been downloaded and then verifies the file. If the file pass the verification, the path of the file is returned. If the file fails the verification, the file is deleted and re-downloaded from the FE. When a BE restarts, it preloads the downloaded file into its memory.

Data Recovery

StarRocks supports data recovery for mistakenly deleted databases/tables/partitions. After `drop table` or `drop database`, StarRocks will not physically delete the data immediately, but keep it in Trash for a period of time (1 day by default). Administrators can recover the mistakenly deleted data with the `RECOVER` command.

Related commands

Syntax:

```
-- 1) Recover database
RECOVER DATABASE db_name;
-- 2) Restore table
RECOVER TABLE [db_name.]table_name;
-- 3) Recover partition
RECOVER PARTITION partition_name FROM [db_name.]table_name;
```

Notes

1. This operation can only restore the deleted meta information. The default time is 1 day, which can be configured by the `catalog_trash_expire_second` parameter in `fe.conf`.
2. If new meta information of the same name and type is created after the meta information is deleted, the previously deleted meta information cannot be restored.

Examples

1. Recover the database named `example_db`

```
RECOVER DATABASE example_db;
~~~ 2.
```

2. Recover the table named `example_tbl`

```
RECOVER TABLE example_db.example_tbl;
~~~ 3.
```

3. Recover the partition named `p1` in the table `example_tbl`

```
RECOVER PARTITION p1 FROM example_tbl;
```

Metadata Recovery

This topic describes how to recover the metadata in your StarRocks clusters when FE nodes encounter different exceptions.

Generally, you may have to resort to metadata recovery when and only when one of the following issues occurs:

- FE fails to restart
- FE fails to provide services
- Recover metadata on a new FE node with the metadata backup.

Check the issue you encountered, follow the solution provided in the corresponding section, and perform any recommended actions.

FE fails to restart

FE nodes may fail to restart if the metadata is damaged or incompatible with the cluster after rollback.

Metadata incompatibility after rollback

When you downgrade your StarRocks cluster, FEs may fail to restart if the metadata is incompatible with that before the downgrade.

You can identify this issue if you encounter the following exception while downgrading the cluster:

```
UNKNOWN Operation Type xxx
```

You can follow these steps to recover the metadata and start FE:

1. Stop all FE nodes.
2. Back up the metadata directories `meta_dir` of all FE nodes.

3. Add the configuration `ignore_unknown_log_id = true` to the configuration files **fe.conf** of all FE nodes.

4. Start all FE nodes, and check whether your data and metadata are intact.

5. If both your data and metadata are intact, execute the following statement to create an image file for your metadata:

   ```
   ALTER SYSTEM CREATE IMAGE;
   ```

6. After the new image file is transmitted to the directory **meta/image** of all FE nodes, you can remove the configuration `ignore_unknown_log_id = true` from all FE configuration files and restart the FE nodes.

Metadata damage

Both damage to BDBJE metadata and StarRocks metadata will cause failures to restart.

BDBJE metadata damage

VLSN Bug

You can identify the VLSN bug based on the following error message:

```
recoveryTracker should overlap or follow on disk last VLSN of
6,684,650 recoveryFirst= 6,684,652 UNEXPECTED_STATE_FATAL: Unexpected
internal state, unable to continue. Environment is invalid and must
be closed.
```

You can follow these steps to fix this issue:

1. Clear the metadata directory `meta_dir` of the FE node that throws this exception.

2. Restart the FE node using the Leader FE node as the helper.

   ```
   # Replace <leader_ip> with the IP address (priority_networks)
   # of the Leader FE node, and replace <leader_edit_log_port>
   (Default: 9010) with
   # the Leader FE node's edit_log_port.
   ```

```
./fe/bin/start_fe.sh --helper <leader_ip>:<leader_edit_log_port>
--daemon
```

> **TIP**
>
> - This bug has been fixed in StarRocks v3.1. You can avoid this issue by upgrading your cluster to v3.1 or later.
> - This solution does not apply if more than half of the FE nodes have encountered this issue. You must follow the instructions provided in Measure of the last resort to fix the issue.

RollbackException

You can identify this issue based on the following error message:

```
must rollback 1 total commits(1 of which were durable) to the
earliest point indicated by transaction id=-14752149 time=2022-01-12
14:36:28.21 vlsn=28,069,415 lsn=0x1174/0x16e durable=false in order
to rejoin the replication group. All existing ReplicatedEnvironment
handles must be closed and reinstantiated. Log files were truncated
to file 0x4467, offset 0x269, vlsn 28,069,413 HARD_RECOVERY: Rolled
back past transaction commit or abort. Must run recovery by re-
opening Environment handles Environment is invalid and must be
closed.
```

This issue occurs when the Leader FE node writes BDBJE metadata but fails to synchronize it to Follower FE nodes before it hangs. After being restarted, the original Leader becomes a Follower, corrupting the metadata.

To solve this issue, you only need to restart this node again to wipe out the dirty metadata.

ReplicaWriteException

You can identify this issue based on the keyword `removeReplicaDb` from the FE log **fe.log**.

```
Caused by: com.sleepycat.je.rep.ReplicaWriteException: (JE 18.3.16) Pro
transaction 25000090. The current state is:REPLICA. The node transition
Feb 23 01:31:00 UTC 2024 Problem seen replaying entry NameLN_TX/14 vlsn
isReplicated="1"  txn=-953505106 dbop=REMOVE Originally thrown by HA th
10.233.132.23_9010_1684154162022(6)
        at com.sleepycat.je.rep.txn.ReadonlyTxn.disallowReplicaWrite(Re
```

```
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.dbi.DbTree.checkReplicaWrite(DbTree.java:88
18.3.16.jar:?]
        at com.sleepycat.je.dbi.DbTree.doCreateDb(DbTree.java:579) ~[st
18.3.16.jar:?]
        at com.sleepycat.je.dbi.DbTree.createInternalDb(DbTree.java:507
18.3.16.jar:?]
        at com.sleepycat.je.cleaner.ExtinctionScanner.openDb(Extinction
[starrocks-bdb-je-18.3.16.jar:?]
        at
com.sleepycat.je.cleaner.ExtinctionScanner.prepareForDbExtinction(Extin
~[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.dbi.DbTree.doRemoveDb(DbTree.java:1208) ~[s
18.3.16.jar:?]
        at com.sleepycat.je.dbi.DbTree.removeReplicaDb(DbTree.java:1261
18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.Replay.applyNameLN(Replay.jav
bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.Replay.replayEntry(Replay.jav
bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.Replica$ReplayThread.run(Repl
[starrocks-bdb-je-18.3.16.jar:?]
```

This issue occurs when the BDBJE version of the failed FE node (v18.3.16) mismatches that of the Leader FE node (v7.3.7).

You can follow these steps to fix this issue:

1. Drop the failed Follower or Observer node.

```
-- To drop a Follower node, replace <follower_host> with the IP
address (priority_networks)
-- of the Follower node, and replace <follower_edit_log_port>
(Default: 9010) with
-- the Follower node's edit_log_port.
ALTER SYSTEM DROP FOLLOWER "<follower_host>:
<follower_edit_log_port>";

-- To drop an Observer node, replace <observer_host> with the IP
address (priority_networks)
-- of the Observer node, and replace <observer_edit_log_port>
(Default: 9010) with
-- the Observer node's edit_log_port.
```

```
ALTER SYSTEM DROP OBSERVER "<observer_host>:
<observer_edit_log_port>";
```

2. Add the failed node back to the cluster.

```
-- Add the Follower node:
ALTER SYSTEM ADD FOLLOWER "<follower_host>:
<follower_edit_log_port>";

-- Add the Observer node:
ALTER SYSTEM ADD OBSERVER "<observer_host>:
<observer_edit_log_port>";
```

3. Clear the metadata directory `meta_dir` of the failed node.

4. Restart the failed node using the Leader FE node as the helper.

```
# Replace <leader_ip> with the IP address (priority_networks)
# of the Leader FE node, and replace <leader_edit_log_port>
(Default: 9010) with
# the Leader FE node's edit_log_port.
./fe/bin/start_fe.sh --helper <leader_ip>:<leader_edit_log_port>
--daemon
```

5. After the failed node recovers to a healthy status, you need to upgrade the BDBJE packages in your cluster to **starrocks-bdb-je-18.3.16.jar** (or upgrade your StarRocks cluster to v3.0 or later), following the order of Followers first and then the Leader.

InsufficientReplicasException

You can identify this issue based on the following error message:

```
com.sleepycat.je.rep.InsufficientReplicasException: (JE 7.3.7) Commit
policy: SIMPLE_MAJORITY required 1 replica. But none were active with
this master.
```

This issue occurs when the Leader node or Follower nodes use excessive memory resources, leading to Full GC.

To solve this issue, you can either increase the JVM heap size or use the G1 GC algorithm.

InsufficientLogException

You can identify this issue based on the following error message:

```
xxx INSUFFICIENT_LOG: Log files at this node are obsolete.
Environment is invalid and must be closed.
```

This issue occurs when the Follower node requires full metadata synchronization. It may occur when one of the following situations happens:

- The metadata on the Follower node lags behind that of the Leader node, which has already done a CheckPoint of metadata within itself. The Follower node cannot perform incremental updates on its metadata, thus full metadata synchronization is required.
- The original Leader node writes and checkpoints its metadata, but fails to synchronize it to Follower FE nodes before it hangs. After being restarted, it becomes a Follower node. With dirty metadata checkpointed, the Follower node cannot perform incremental deletion of its metadata, thus full metadata synchronization is required.

Please note that this exception will be thrown when a new Follower node is added to the cluster. In this case, you do not need to take any action. If this exception is thrown for an existing Follower node or the Leader node, you only need to restart the node.

HANDSHAKE_ERROR: Error during the handshake between two nodes

You can identify this issue based on the following error message:

```
2023-11-13 21:51:55,271 WARN (replayer|82) [BDBJournalCursor.wrapDataba
get DB names for 1 times!Got EnvironmentFailureExce
com.sleepycat.je.EnvironmentFailureException: (JE 18.3.16) Environment
com.sleepycat.je.EnvironmentFailureException: Environment invalid becau
18.3.16) 10.26.5.115_9010_1697071897979(1):/data1/meta/bdb A replica wi
10.26.5.115_9010_1697071897979(1) is already active with the Feeder:nul
during the handshake between two nodes. Some validity or compatibility
further communication between the nodes. Environment is invalid and mus
        at
com.sleepycat.je.EnvironmentFailureException.wrapSelf(EnvironmentFailur
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.dbi.EnvironmentImpl.checkIfInvalid(Environm
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.dbi.EnvironmentImpl.checkOpen(EnvironmentIm
je-18.3.16.jar:?]
        at com.sleepycat.je.Environment.checkOpen(Environment.java:2697
```

```
18.3.16.jar:?]
        at com.sleepycat.je.Environment.getDatabaseNames(Environment.ja
18.3.16.jar:?]
        at
com.starrocks.journal.bdbje.BDBEnvironment.getDatabaseNamesWithPrefix(B
[starrocks-fe.jar:?]
        at com.starrocks.journal.bdbje.BDBJournalCursor.refresh(BDBJour
[starrocks-fe.jar:?]
        at com.starrocks.server.GlobalStateMgr$5.runOneCycle(GlobalStat
fe.jar:?]
        at com.starrocks.common.util.Daemon.run(Daemon.java:115) ~[star
        at com.starrocks.server.GlobalStateMgr$5.run(GlobalStateMgr.jav
Caused by: com.sleepycat.je.EnvironmentFailureException: Environment in
exception: (JE 18.3.16) 10.26.5.115_9010_1697071897979(1):/data1/meta/b
10.26.5.115_9010_1697071897979(1) is already active with the Feeder:nul
during the handshake between two nodes. Some validity or compatibility
further communication between the nodes. Environment is invalid and mus
by HA thread: UNKNOWN 10.26.5.115_9010_1697071897979(1) Originally thro
10.26.5.115_9010_1697071897979(1)
        at
com.sleepycat.je.rep.stream.ReplicaFeederHandshake.negotiateProtocol(Re
~[starrocks-bdb-je-18.3.16.jar:?]
        at
com.sleepycat.je.rep.stream.ReplicaFeederHandshake.execute(ReplicaFeede
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.Replica.initReplicaLoop(Repli
je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.Replica.runReplicaLoopInterna
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.Replica.runReplicaLoop(Replic
je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.RepNode.run(RepNode.java:1869
18.3.16.jar:?]
```

This issue occurs when the original Leader node hangs and becomes alive again while the surviving Follower nodes are trying to elect a new Leader node. The Follower nodes will try to establish a new connection with the original Leader node. However, the Leader node will reject the connection request because the old connection still exists. Once the request is rejected, the Follower node will set the environment as invalid and throw this exception.

To solve this issue, you can either increase the JVM heap size or use the G1 GC algorithm.

Latch timeout. com.sleepycat.je.log.LogbufferPool_FullLatch

You can identify this issue based on the following error message:

```
Environment invalid because of previous exception: xxx Latch timeout.
com.sleepycat.je.log.LogbufferPool_FullLatch xxx'
        at
com.sleepycat.je.EnvironmentFailureException.unexpectedState(Environmen
        at com.sleepycat.je.latch.LatchSupport.handleTimeout(LatchSuppo
        at com.sleepycat.je.latch.LatchWithStatsImpl.acquireExclusive(L
        at com.sleepycat.je.log.LogBufferPool.bumpCurrent(LogBufferPool
        at com.sleepycat.je.log.LogManager.flushInternal(LogManager.jav
        at com.sleepycat.je.log.LogManager.flushNoSync(LogManager.java:
        at com.sleepycat.je.log.LogFlusher$FlushTask.run(LogFlusher.jav
        at java.util.TimerThread.mainLoop(Timer.java:555)
        at java.util.TimerThread.run(Timer.java:505)
```

This issue occurs when there is excessive pressure on the local disk of the FE node.

To solve this issue, you can allocate a dedicated disk for the FE metadata, or replace the disk with a high-performance one.

DatabaseNotFoundException

You can identify this issue based on the following error message:

```
2024-01-05 12:47:21,087 INFO (main|1) [BDBEnvironment.ensureHelperInLoc
check local environment because helper node and local node are identica
2024-01-05 12:47:21,339 ERROR (MASTER 172.17.0.1_9112_1704430041062(-1)
[StarRocksFE.start():186] StarRocksFE start failed
com.sleepycat.je.DatabaseNotFoundException: (JE 18.3.16) _jeRepGroupDB
        at com.sleepycat.je.rep.impl.RepImpl.openGroupDb(RepImpl.java:1
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.RepImpl.getGroupDb(RepImpl.java:19
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.RepGroupDB.reinitFirstNode(RepGrou
~[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.RepNode.reinitSelfElect(RepNo
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.RepNode.startup(RepNode.java:
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.node.RepNode.joinGroup(RepNode.jav
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.impl.RepImpl.joinGroup(RepImpl.java:618
bdb-je-18.3.16.jar:?]
```

```
        at
com.sleepycat.je.rep.ReplicatedEnvironment.joinGroup(ReplicatedEnvironm
[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.ReplicatedEnvironment.<init>
(ReplicatedEnvironment.java:619) ~[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.ReplicatedEnvironment.<init>
(ReplicatedEnvironment.java:464) ~[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.ReplicatedEnvironment.<init>
(ReplicatedEnvironment.java:538) ~[starrocks-bdb-je-18.3.16.jar:?]
        at com.sleepycat.je.rep.util.DbResetRepGroup.reset(DbResetRepGr
[starrocks-bdb-je-18.3.16.jar:?]
        at
com.starrocks.journal.bdbje.BDBEnvironment.initConfigs(BDBEnvironment.j
[starrocks-fe.jar:?]
        at com.starrocks.journal.bdbje.BDBEnvironment.setup(BDBEnvironm
[starrocks-fe.jar:?]
        at
com.starrocks.journal.bdbje.BDBEnvironment.initBDBEnvironment(BDBEnviro
~[starrocks-fe.jar:?]
        at com.starrocks.journal.JournalFactory.create(JournalFactory.j
[starrocks-fe.jar:?]
        at com.starrocks.server.GlobalStateMgr.initJournal(GlobalStateM
[starrocks-fe.jar:?]
        at com.starrocks.server.GlobalStateMgr.initialize(GlobalStateMg
[starrocks-fe.jar:?]
        at com.starrocks.StarRocksFE.start(StarRocksFE.java:129) ~[star
        at com.starrocks.StarRocksFE.main(StarRocksFE.java:83) ~[starro
```

This issue occurs when you add the configuration `metadata_failure_recovery = true` in the FE configuration file **fe.conf**.

To solve this issue, you need to remove the configuration and restart the node.

StarRocks metadata damage

You can identify the StarRocks metadata damage issue based on one of the following error messages:

```
failed to load journal type xxx
```

Or

```
catch exception when replaying
```

> **WARNING**
>
> Before proceeding to recover the metadata by following the solution provided below, you are strongly advised to seek assistance from the technical experts in the StarRocks community, because this solution may lead to **data loss**.

You can follow these steps to fix this issue:

1. Stop all FE nodes.
2. Back up the metadata directories `meta_dir` of all FE nodes.
3. Add the configuration `metadata_enable_recovery_mode = true` to the configuration files **fe.conf** of all FE nodes. Note that data loading is forbidden in this mode.
4. Start all FE nodes, and query tables in the cluster to check whether your data is intact.

   You must wait until metadata recovery is completed if the following error is returned when you query these tables:

   ```
   ERROR 1064 (HY000): capture_consistent_versions error: version
   already been compacted.
   ```

   You can execute the following statement from the Leader FE node to view the progress of metadata recovery:

   ```
   SHOW PROC '/meta_recovery';
   ```

   This statement will show the partitions that failed to be recovered. You can follow the advice returned to recover the partitions. If nothing is returned, it indicates the recovery is successful.
5. If both your data and metadata are intact, execute the following statement to create an image file for your metadata:

```
ALTER SYSTEM CREATE IMAGE;
```

6. After the new image file is transmitted to the directory **meta/image** of all FE nodes, you can remove the configuration `metadata_enable_recovery_mode = true` from all FE configuration files and restart the FE nodes.

FE fails to provide services

FE will not provide services when Follower FE nodes fail to perform Leader election. When this issue occurs, you may find the following log record repeating:

```
wait globalStateMgr to be ready. FE type: INIT. is ready: false
```

A variety of exceptions may cause this issue. You are strongly advised to troubleshoot the issue step by step following the sections below. Applying inappropriate solutions will deteriorate the problem and probably lead to data loss.

1. The majority of Follower nodes are not running

If the majority of the Follower nodes are not running, the FE group will not provide services. Here, 'majority' indicates `1 + (Follower node count/2)`. Please note that the Leader FE node itself is a Follower, but Observer nodes are not Followers.

- You can identify the role of each FE node from the **fe/meta/image/ROLE** file:

```
cat fe/meta/image/ROLE

#Fri Jan 19 20:03:14 CST 2024
role=FOLLOWER
hostType=IP
name=172.26.92.154_9312_1705568349984
```

- You can view the total number of Follower nodes from the BDBJE log:

```
grep "Current group size" fe/meta/bdb/je.info.0

# The example output indicates there are three Follower nodes in
```

```
the cluster.
2024-01-24 08:21:44.754 UTC INFO
[172.26.92.139_29917_1698226672727] Current group size: 3
```

To solve this issue, you need to start all Follower nodes in the cluster. If they cannot be restarted, please refer to The measure of last resort.

2. Node IP is changed

If the `priority_networks` of the node is not configured, the FE node will randomly select an available IP address once it is restarted. If the IP address recorded in the BDBJE metadata is different from that used to start the node, the FE will not provide services.

- You can view the IP address recorded in the BDBJE metadata from the **fe/meta/image/ROLE** file:

  ```
  cat fe/meta/image/ROLE

  #Fri Jan 19 20:03:14 CST 2024
  role=FOLLOWER
  hostType=IP
  name=172.26.92.154_9312_1705568349984
  ```

  The value `172.26.92.154` before the first underscore is the IP address recorded in the BDBJE metadata.

- You can view the IP address used to start the node from the FE log:

  ```
  grep "IP:" fe/log/fe.log

  2024-02-06 14:33:58,211 INFO (main|1)
  [FrontendOptions.initAddrUseIp():249] Use IP init local addr, IP:
  /172.17.0.1
  2024-02-06 14:34:27,689 INFO (main|1)
  [FrontendOptions.initAddrUseIp():249] Use IP init local addr, IP:
  /172.17.0.1
  ```

To solve this issue, you need to set the `priority_networks` of the node in the FE configuration file **fe.conf** to the IP address recorded in **fe/meta/image/ROLE**, and restart the node.

3. System clock among nodes is not synchronized

You can identify this issue based on the following error message from **fe.out**, **fe.log** or **fe/meta//bdb/je.info.0**:

```
com.sleepycat.je.EnvironmentFailureException: (JE 7.3.7) Environment
must be closed, caused by:
com.sleepycat.je.EnvironmentFailureException: Environment invalid
because of previous exception: (JE 7.3.7)
172.26.92.139_29917_1631006307557(2180):xxx Clock delta: 11020 ms.
between Feeder: 172.26.92.154_29917_1641969377236 and this Replica
exceeds max permissible delta: 5000 ms. HANDSHAKE_ERROR: Error during
the handshake between two nodes. Some validity or compatibility check
failed, preventing further communication between the nodes.
Environment is invalid and must be closed. fetchRoot of
0x1278/0x1fcbb8 state=0 expires=never
```

You must synchronize the system clock among all nodes.

4. Available disk space is insufficient

After you upgrade StarRocks to v3.0 or later, or upgrade BDBJE to v18 or later, the node may fail to restart when the available space of the disk that stores `meta_dir` is less than 5 GB.

You can view the BDBJE version from the **.jar** package under the directory **fe/lib**.

To solve this issue, you can scale up the disk, or allocate a dedicated disk with larger capacity for the FE metadata.

5. `edit_log_port` is changed

If the `edit_log_port` recorded in the BDBJE metadata is different from that configured in **fe.conf**, the FE will not provide services.

You can view the `edit_log_port` recorded in the BDBJE metadata from the **fe/meta/image/ROLE** file:

```
cat fe/meta/image/ROLE
```

```
#Fri Jan 19 20:03:14 CST 2024
role=FOLLOWER
hostType=IP
name=172.26.92.154_9312_1705568349984
```

The value `9312` before the second underscore is the `edit_log_port` recorded in the BDBJE metadata.

To solve this issue, you need to set the `edit_log_port` of the node in the FE configuration file **fe.conf** to the `edit_log_port` recorded in **fe/meta/image/ROLE**, and restart the node.

6. JVM heap size is insufficient

You can view the JVM memory usage using the `jstat` command:

```
jstat -gcutil pid 1000 1000

  S0     S1     E      O      M     CCS    YGC     YGCT    FGC
FGCT     GCT
  0.00 100.00  27.78  95.45  97.77  94.45     24    0.226     1
0.065    0.291
  0.00 100.00  44.44  95.45  97.77  94.45     24    0.226     1
0.065    0.291
  0.00 100.00  55.56  95.45  97.77  94.45     24    0.226     1
0.065    0.291
  0.00 100.00  72.22  95.45  97.77  94.45     24    0.226     1
0.065    0.291
  0.00 100.00  88.89  95.45  97.77  94.45     24    0.226     1
0.065    0.291
  0.00 100.00   5.26  98.88  97.80  94.45     25    0.231     1
0.065    0.297
  0.00 100.00  21.05  98.88  97.80  94.45     25    0.231     1
0.065    0.297
  0.00 100.00  31.58  98.88  97.80  94.45     25    0.231     1
0.065    0.297
  0.00 100.00  47.37  98.88  97.80  94.45     25    0.231     1
0.065    0.297
  0.00 100.00  63.16  98.88  97.80  94.45     25    0.231     1
0.065    0.297
  0.00 100.00  73.68  98.88  97.80  94.45     25    0.231     1
0.065    0.297
```

If the percentages shown in field `0` remain high, it indicates that the JVM heap size is insufficient.

To solve this issue, you must increase the JVM heap size.

7. Measure of the last resort

> **WARNING**
>
> You can try the following solution as the last resort only when none of the preceding solutions works.

This solution is designed only for extreme cases, such as:

- The majority of Follower nodes cannot be restarted.
- Follower nodes cannot perform Leader election due to bugs of BDBJE.
- Exceptions other than those mentioned in preceding sections.

Follow these steps to recover the metadata:

1. Stop all FE nodes.
2. Back up the metadata directories `meta_dir` of all FE nodes.
3. Run the following commands on **all the servers that host the FE nodes** to identify the node with the latest metadata.
    - For StarRocks v2.5 and earlier:

      ```
      java -jar fe/lib/je-7.3.7.jar DbPrintLog -h meta/bdb/ -vd
      ```

    - For StarRocks v3.0 and later:

      ```
      # You need to specify the exact .jar package the node uses in the command
      # as the package varies according to the StarRocks version.
      java -jar fe/lib/starrocks-bdb-je-18.3.16.jar DbPrintLog -h meta/bdb/ -vd
      ```

Example output:

```
<DbPrintLog>
file 0x3b numRepRecords = 24479 firstVLSN = 1,434,126 lastVLSN =
1,458,604
file 0x3c numRepRecords = 22541 firstVLSN = 1,458,605 lastVLSN =
1,481,145
file 0x3d numRepRecords = 25176 firstVLSN = 1,481,146 lastVLSN =
1,506,321
......
file 0x74 numRepRecords = 26903 firstVLSN = 2,927,458 lastVLSN =
2,954,360
file 0x75 numRepRecords = 26496 firstVLSN = 2,954,361 lastVLSN =
2,980,856
file 0x76 numRepRecords = 18727 firstVLSN = 2,980,857 lastVLSN =
2,999,583
... 0 files at end
First file: 0x3b
Last file: 0x76
</DbPrintLog>
```

The node with the largest `lastVLSN` value has the latest metadata.

4. Find the role (Follower or Observer) of the FE node that has the latest metadata from the **fe/meta/image/ROLE** file:

```
cat fe/meta/image/ROLE

#Fri Jan 19 20:03:14 CST 2024
role=FOLLOWER
hostType=IP
name=172.26.92.154_9312_1705568349984
```

If multiple nodes have the latest metadata, it is recommended to proceed with a Follower node. If multiple Follower nodes have the latest metadata, you can proceed with any of them.

5. Perform the corresponding operations based on the role of the FE node you chose in the previous step.

Proceed with Follower Node **Proceed with Observer Node**

If a Follower node has the latest metadata, perform the following operations:

1. Add the following configurations to **fe.conf**:
    - For StarRocks v2.5, v3.0, v3.1.9 and earlier patch versions, and v3.2.4 and earlier patch versions:

      ```
      metadata_failure_recovery = true
      ```

    - For StarRocks v3.1.10 and later patch versions, v3.2.5 and later patch versions, and v3.3 and later:

      ```
      bdbje_reset_election_group = true
      ```

2. Restart the node, and check whether your data and metadata are intact.
3. Check whether the current FE node is the Leader FE node.

   ```
   SHOW FRONTENDS;
   ```

    - If the field `Alive` is `true`, this FE node is properly started and added to the cluster.
    - If the field `Role` is `LEADER`, this FE node is the Leader FE node.

4. If the data and metadata are intact, and the role of the node is Leader, you can remove the configuration you added earlier and restart the node.
6. The surviving Follower node is now essentially the Leader node of the cluster. Drop all FE nodes except for the current node.

   ```
   -- To drop a Follower node, replace <follower_host> with the IP
   address (priority_networks)
   -- of the Follower node, and replace <follower_edit_log_port>
   (Default: 9010) with
   -- the Follower node's edit_log_port.
   ALTER SYSTEM DROP FOLLOWER "<follower_host>:
   <follower_edit_log_port>";

   -- To drop an Observer node, replace <observer_host> with the IP
   ```

```
address (priority_networks)
-- of the Observer node, and replace <observer_edit_log_port>
(Default: 9010) with
-- the Observer node's edit_log_port.
ALTER SYSTEM DROP OBSERVER "<observer_host>:
<observer_edit_log_port>";
```

7. Clear the metadata directories `meta_dir` of the FE nodes that you want to add back to the cluster.

8. Start new Follower nodes using the new Leader FE node as the helper.

```
# Replace <leader_ip> with the IP address (priority_networks)
# of the Leader FE node, and replace <leader_edit_log_port>
(Default: 9010) with
# the Leader FE node's edit_log_port.
./fe/bin/start_fe.sh --helper <leader_ip>:<leader_edit_log_port>
--daemon
```

9. Add the Follower nodes back to the cluster.

```
ALTER SYSTEM ADD FOLLOWER "<new_follower_host>:
<new_follower_edit_log_port>";
```

After all nodes are added back to the cluster, the metadata is successfully recovered.

Recover metadata on a new FE node with metadata backup

Follow these steps if you want to start a new FE node with the metadata backup:

1. Copy the backup metadata directory `meta_dir` to the new FE node.

2. In the configuration file of the FE node, set `bdbje_reset_election_group` to `true`.

```
bdbje_reset_election_group = true
```

3. Start the FE node.

```
# Replace <fe_ip> with the IP address (priority_networks)
# of the new FE node, and replace <fe_edit_log_port> (Default:
9010) with
# the new FE node's edit_log_port.
./fe/bin/start_fe.sh --helper <fe_ip>:<fe_edit_log_port> --daemon
```

4. Check whether the current FE node is the Leader FE node.

   ```
   SHOW FRONTENDS;
   ```

   If the field `Role` is `LEADER`, this FE node is the Leader FE node. Make sure its IP address is the that of the current FE node.

5. If the data and metadata are intact, and the role of the node is Leader, you must remove the configuration `bdbje_reset_election_group` and restart the node.

6. Now you have successfully start a new Leader FE node with the metadata backup. You can add new Follower nodes using the new Leader FE node as the helper.

   ```
   # Replace <leader_ip> with the IP address (priority_networks)
   # of the Leader FE node, and replace <leader_edit_log_port>
   (Default: 9010) with
   # the Leader FE node's edit_log_port.
   ./fe/bin/start_fe.sh --helper <leader_ip>:<leader_edit_log_port>
   --daemon
   ```

Metadata recovery-related configurations

> **TIP**
>
> You must remove the following configuration once the metadata recovery is completed.

- `bdbje_reset_election_group`
- `metadata_enable_recovery_mode`
- `ignore_unknown_log_id`

User Privilege

Based on MySQL's permission management mechanism, StarRocks permission management system supports fine-grained permission control at the table level, role-based access control, and whitelisting.

Terminology

- User Identity

In the privilege system, a user is identified as a User Identity. The user identity consists of two parts: username and userhost. `user_identity` is presented as `username@'userhost'`. The username is made of English characters. The userhost is the IP from which the user is linked.

`user_identity` can also be presented as `username@['domain'],` where `domain` is a domain name that can be resolved into a set of IPs by DNS. The final expression is a set of username@'userhost', so we use username@'userhost' to represent it throughout this chapter.

- Privilege

Users can set privileges on nodes, databases, or tables. Different privileges represent different permissions to operate.

- Role

StarRocks can create roles with customized names. A role is a collection of permissions. A newly created user can be given a role with corresponding permissions. Subsequent changes to the permissions apply to all users granted the role.

- User Property

User properties are attached directly to a user, not to a user identity That is, both `abc@'192.%'` and `abc@['domain']` have the same set of user properties that belong to

the user `abc`, not `abc@'192.%'` or `abc@['domain']`. User attributes include, but are not limited to maximum number of user connections, import cluster configuration, and so on.

Supported operations

- Create user: CREATE USER
- Delete user: DROP USER
- Authorization: GRANT
- Revoke authority: REVOKE
- Create Role: CREATE ROLE
- Delete Role: DROP ROLE
- View current user permissions: SHOW GRANTS
- View all user permissions: SHOW ALL GRANTS
- View created roles: SHOW ROLES
- View user properties: SHOW PROPERTY

Privilege Types

StarRocks currently supports the following types of privileges.

- NODE_PRIV

  Privileges for Node changes. This allows you to add, delete, and disable FE, BE, and BROKER nodes. Currently, this privilege can only be granted to Root users.

- GRANT_PRIV

  Privileges for privilege changes. This allows you to grant, revoke, and modify users/roles.

- SELECT_PRIV

  Read-only access to databases and tables.

- LOAD_PRIV

Write access to databases and tables. This allows you to Load, Insert, and Delete databases or tables.

- ALTER_PRIV

  Privileges to change databases and tables. This allows you to rename, add/remove/change columns, add/remove partitions.

- CREATE_PRIV

  Privileges to create databases, tables, and views.

- DROP_PRIV

  Privileges to delete databases, tables, and views.

- USAGE_PRIV

  Privileges to use resources.

Privilege Hierarchy

StarRocks classifies database and table privileges into three levels:

- GLOBAL LEVEL: Global privileges. This privilege is granted on `.` by the GRANT statement. The privilege granted applies to any table in any database.
- DATABASE LEVEL: Database level privileges. This privilege is granted on `db.*` by the `GRANT` statement. The privilege granted applies to any table in a specific database.
- TABLE LEVEL: Table level privileges. This privilege is granted on `db.tbl` by the `GRANT` statement. The privilege granted applies to a specific table in a specific database.

StarRocks classifies the resource privileges into two levels.

- GLOBAL LEVEL: Global privileges. This privilege is granted on `*` by the `GRANT` statement. The privilege granted applies to all resources.
- RESOURCE LEVEL: Resource level privileges. This privilege is granted on `resource_name` by the `GRANT` statement. The privilege granted applies to a specific resource.

Privilege Description

The `ADMIN_PRIV` and `GRANT_PRIV` privileges are special because they both have the privilege to grant privileges.

- CREATE USER
- Users with the ADMIN privilege or GRANT privilege at any level can create new users.
- DROP USER
- Users with ADMIN privilege can delete users.
- CREATE/DROP ROLE
- Users with ADMIN privilege can create roles.
- GRANT/REVOKE
- Users with the ADMIN privilege or GLOBAL level GRANT privilege can grant or revoke privileges to any user.
- Users with DATABASE level GRANT privilege can grant or revoke privileges to any user for any table in a specific database.
- Users with TABLE level GRANT privilege can grant or revoke privileges to any user for a specific table in a specific database.
- SET PASSWORD
- Users with the ADMIN privilege or GLOBAL level GRANT privilege can set the user password.
- Regular users can set passwords for their User Identity. Use the `SELECT CURRENT_USER();` command to view User Identity.
- Users with TABLE or DATABASE level GRANT privileges cannot set passwords for existing users, but can set passwords for new users.

Other notes

- When StarRocks is initialized, the following users and roles are automatically created.
- operator role: This role has `NODE_PRIV` and `ADMIN_PRIV`, i.e. all privileges. In a later upgrade, this role may be restricted to `NODE_PRIV` (privileges for node changes). This is to meet certain deployment requirements on the cloud.
- admin role: This role has `ADMIN_PRIV`, i.e., all privileges except for the one for node changes.
- root@'%': Root user, allowed to log in from any node as `operator`.
- admin@'%': Admin user, allowed to log in from any node as `admin`.
- It is not supported to delete or change default privileges set for roles or users.
- Only one user can be granted the operator role. Multiple users can be granted the admin role.
- Description of some potentially conflicting operations:
- Domain name and IP conflict

Suppose the following user is created.
`CREATE USER abc@['domain'];`
and authorized:
`GRANT SELECT_PRIV ON *. * TO abc@['domain']`
The domain is resolved to two IPs: ip1 and ip2. Suppose we then authorize abc@'ip1' separately: `GRANT ALTER_PRIV ON *. * TO abc@'ip1';`
Then the privileges of abc@'ip1' will be changed to `SELECT_PRIV`, `ALTER_PRIV`.Any future privileges changes to abc@['domain'] will not apply to abc@'ip1'.

- Duplicate IP conflict

Suppose the following user is created.
`CREATE USER abc@'%' IDENTIFIED BY "12345";`
`CREATE USER abc@'192.%' IDENTIFIED BY "abcde";`
'192.%' takes precedence over '%'. When the user requests to log in to StarRocks from 192.168.1.1 with the password '12345', the request will be denied.

- Forgotten password
  If you forget your password, you can log in to StarRocks using the following command.

`mysql-client -h 127.0.0.1 -P query_port -uroot`
After logging in, you can reset the password with the `SET PASSWORD` command.

- Only root users can reset their password, other users cannot reset root user's password.
- The ADMIN_PRIV privilege can only be granted or revoked at the GLOBAL level.
- `GRANT_PRIV` at the GLOBAL level is similar to `ADMIN_PRIV`. `GRANT_PRIV` at the GLOBAL level has the privilege to grant arbitrary privileges, so please use it carefully.
- current_user() and user()

Users can use `SELECT current_user();` and `SELECT user();` to see `current_user` and `user` respectively. `current_user` indicates the current user's identity under the authentication system, and `user` is the actual `user_identity` of the user.

For example, suppose user1@'192.%' is created, and then user1 logs into the system from 192.168.10.1. In this case, `current_user` is user1@'192.%', and `user` is user1@'192.168.10.1'.

All privileges are granted to `current_user`, and the real user has all the privileges of the corresponding `current_user`.

Best Practices

Here are some scenarios for using the StarRocks permission management system.

Scenario 1: Permissions Assignment

StarRocks users are divided into Administrator (Admin), Development Engineer (RD), and User (Client). The administrators have all the privileges and are mainly responsible for cluster construction, and node management. The development engineers are responsible for business modeling, including building databases and tables, importing and modifying data, and so on. Users access databases and tables to get data.

In this scenario, Admins are granted ADMIN privileges or GRANT privileges. RDs are granted privileges to CREATE, DROP, ALTER, LOAD, and SELECT any or specific database tables.

Clients are granted privileges to SELECT any or specific database tables. When there are multiple users, it is also possible to simplify the authorization by creating different roles.

Scenario 2: Multiple lines of business

There may be multiple use cases within a cluster, each of which may use one or more databases. Each use case needs to manage its own users. In this scenario, an ADMIN can grant one user with DATABASE level GRANT privilege for each database. This user can only authorize users for this specific database.

Scenario 3: Blacklisting

StarRocks does not support blacklisting (only whitelisting is supported), but we can emulate blacklisting in some way. Suppose that the user named user@'192.%' is created first, indicating that the user f is allowed to log in from 192.*. At this point, if you want to prevent the user from logging in from 192.168.10.1, you can create another user abc@'192.168.10.1' and set a new password. Because 192.168.10.1 has a higher priority than 192.%, the user will no longer be able to log in from 192.168.10.1 with the old password.

Troubleshooting

Reset password for root user

If you have lost the password for the user `root`, you can reset it by following these procedures:

1. Add the following configuration item to the configuration files **fe/conf/fe.conf** of **all FE nodes** to disable authentication:

   ```
   enable_auth_check = false
   ```

2. Restart **all FE nodes**.

   ```
   ./fe/bin/stop_fe.sh
   ./fe/bin/start_fe.sh
   ```

3. Launch a MySQL client, and connect to StarRocks using the user `root` without the password.

```
mysql -h <fe_ip> -P<fe_query_port> -uroot
```

4. Reset the password for the user `root`.

```
SET PASSWORD for root = PASSWORD('xxxxxx');
```

5. Re-enable authentication by configuring the configuration item `enable_auth_check` to `true` in the configuration files **fe/conf/fe.conf** of **all FE nodes**.

```
enable_auth_check = true
```

6. Restart **all FE nodes**.

```
./fe/bin/stop_fe.sh
./fe/bin/start_fe.sh
```

7. Launch a MySQL client, and connect to StarRocks using the user `root` and the new password to verify whether the password is reset successfully.

```
mysql -h <fe_ip> -P<fe_query_port> -uroot -p<xxxxxx>
```

Authentication methods

In addition to the authentication method of "username+password", StarRocks also supports LDAP.

LDAP Authentication

To use LDAP authentication, you need to add the LDAP service into the FE node configuration first.

- `authentication_ldap_simple_server_host`: Specify the service IP.
- `authentication_ldap_simple_server_port`: Specify the service port, with a default value of 389.

When creating a user, specify the authentication method as LDAP authentication by `IDENTIFIED WITH authentication_ldap_simple AS 'xxx'`. xxx is the DN (Distinguished Name) of the user in LDAP.

Example 1:

```
CREATE USER tom IDENTIFIED WITH authentication_ldap_simple AS
'uid=tom,ou=company,dc=example,dc=com'
```

It is possible to create a user without specifying the user's DN in LDAP. When the user logs in, StarRocks will go to the LDAP system to retrieve the user information. if there is one and only one match, the authentication is successful.

Example 2:

```
CREATE USER tom IDENTIFIED WITH authentication_ldap_simple
```

In this case, additional configuration needs to be added to the FE

- `authentication_ldap_simple_bind_base_dn`: The base DN of the user, specifying the retrieval range of the user.
- `authentication_ldap_simple_user_search_attr`: The name of the attribute in the LDAP object that identifies the user, uid by default.
- `authentication_ldap_simple_bind_root_dn`: The DN of the administrator account used to retrieve the user information.
- `authentication_ldap_simple_bind_root_pwd`: The password of the administrator account used when retrieving the user information.

LDAP authentication requires the client to pass on a clear-text password to StarRocks. There are three ways to pass on a clear-text password:

- **MySQL command line**

Add `--default-auth mysql_clear_password --enable-cleartext-plugin` when executing:

```
mysql -utom -P8030 -h127.0.0.1 -p --default-auth mysql_clear_password
--enable-cleartext-plugin
```

- **JDBC**

Since JDBC's default MysqlClearPasswordPlugin requires SSL transport, a custom plugin is required.

```
public class MysqlClearPasswordPluginWithoutSSL extends
MysqlClearPasswordPlugin {
    @Override
    public boolean requiresConfidentiality() {
        return false;
    }
}
```

Once connected, configure the custom plugin into the property.

```
...
Properties properties = new Properties();// replace xxx.xxx.xxx to your
properties.put("authenticationPlugins", "xxx.xxx.xxx.MysqlClearPassword
properties.put("defaultAuthenticationPlugin",
```

```
"xxx.xxx.xxx.MysqlClearPasswordPluginWithoutSSL");
properties.put("disabledAuthenticationPlugins",
"com.mysql.jdbc.authentication.MysqlNativePasswordPlugin");DriverManage
properties);
```

- **ODBC**

Add `default\_auth=mysql_clear_password` and `ENABLE_CLEARTEXT\_PLUGIN=1` in the DSN of ODBC: , along with username and password.

Query analysis

How to optimize query performance is a frequently asked question. Slow queries impair user experi...

Analyze query profile

This topic describes how to check the query profile. A query profile records the execution informati...

Performance Optimization

Table Type Selection

Query analysis

How to optimize query performance is a frequently asked question. Slow queries impair user experience as well as cluster performance. It is important to analyze and optimize query performance.

You can view query information in `fe/log/fe.audit.log`. Each query corresponds to a `QueryID` that can be used to search for the `QueryPlan` and `Profile` of a query. `QueryPlan` is the execution plan generated by FE by parsing SQL statements. `Profile` is BE execution result and contains information such as the time consumed by each step and the volume of data processed at each step.

Plan analysis

In StarRocks, the lifecycle of an SQL statement can be divided into three phases: query parsing, query planning, and query execution. Query parsing is generally not a bottleneck because the required QPS of analytical workloads is not high.

Query performance in StarRocks is determined by query planning and query execution. Query planning is responsible for coordinating operators (Join/Order/Aggregate), and query execution is responsible for running specific operations.

A query plan provides the DBA with a macro perspective to access query information. A query plan is the key to query performance and a good resource for the DBA to reference. The following code snippet uses `TPCDS query96` as an example to show how to view a query plan.

```
-- query96.sql
select  count(*)
from store_sales
    ,household_demographics
    ,time_dim
    , store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
```

```
    and time_dim.t_hour = 8
    and time_dim.t_minute >= 30
    and household_demographics.hd_dep_count = 5
    and store.s_store_name = 'ese'
order by count(*) limit 100;
```

There are two types of query plans – logical query plan and physical query plan. The query plan described here refers to the logical query plan. The query plan corresponding to `TPCDS query96.sql` is shown below.

```
+-----------------------------------------------------------------------
----------+
| Explain String
|
+-----------------------------------------------------------------------
----------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:<slot 11>
|
|   PARTITION: UNPARTITIONED
|
|   RESULT SINK
|
|   12:MERGING-EXCHANGE
|
|      limit: 100
|
|      tuple ids: 5
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 12
|
|     UNPARTITIONED
|
```

```
|
|
|   8:TOP-N
|
|   |  order by: <slot 11> ASC
|
|   |  offset: 0
|
|   |  limit: 100
|
|   |  tuple ids: 5
|
|   |
|
|   7:AGGREGATE (update finalize)
|
|   |  output: count(*)
|
|   |  group by:
|
|   |  tuple ids: 4
|
|   |
|
|   6:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  hash predicates:
|
|   |  colocate: false, reason: left hash join node can not do
colocate        |
|   |  equal join conjunct: `ss_store_sk` = `s_store_sk`
|
|   |  tuple ids: 0 2 1 3
|
|   |
|
|   |----11:EXCHANGE
|
|   |       tuple ids: 3
|
|   |
|
|   4:HASH JOIN
|
```

```
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  hash predicates:
|
|   |  colocate: false, reason: left hash join node can not do
colocate        |
|   |  equal join conjunct:
`ss_hdemo_sk`=`household_demographics`.`hd_demo_sk`|
|   |  tuple ids: 0 2 1
|
|   |
|
|   |----10:EXCHANGE
|
|   |       tuple ids: 1
|
|   |
|
|   2:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  hash predicates:
|
|   |  colocate: false, reason: table not in same group
|
|   |  equal join conjunct: `ss_sold_time_sk` =
`time_dim`.`t_time_sk`          |
|   |  tuple ids: 0 2
|
|   |
|
|   |----9:EXCHANGE
|
|   |       tuple ids: 2
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: store_sales
|
|      PREAGGREGATION: OFF. Reason: `ss_sold_time_sk` is value column
|
|      partitions=1/1
|
```

```
|      rollup: store_sales
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 0
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 11
|
|     UNPARTITIONED
|
|
|
|   5:OlapScanNode
|
|      TABLE: store
|
|      PREAGGREGATION: OFF. Reason: null
|
|      PREDICATES: `store`.`s_store_name` = 'ese'
|
|      partitions=1/1
|
|      rollup: store
|
|      tabletRatio=0/0
|
```

```
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 3
|
|
|
| PLAN FRAGMENT 3
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 10
|
|     UNPARTITIONED
|
|
|
|   3:OlapScanNode
|
|      TABLE: household_demographics
|
|      PREAGGREGATION: OFF. Reason: null
|
|      PREDICATES: `household_demographics`.`hd_dep_count` = 5
|
|      partitions=1/1
|
|      rollup: household_demographics
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
```

```
|      numNodes=0
|
|      tuple ids: 1
|
|
|
| PLAN FRAGMENT 4
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 09
|
|     UNPARTITIONED
|
|
|
|   1:OlapScanNode
|
|      TABLE: time_dim
|
|      PREAGGREGATION: OFF. Reason: null
|
|      PREDICATES: `time_dim`.`t_hour` = 8, `time_dim`.`t_minute` >=
30        |
|      partitions=1/1
|
|      rollup: time_dim
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 2
|
+----------------------------------------------------------------------
```

```
----------+
128 rows in set (0.02 sec)
```

Query 96 shows a query plan that involves several StarRocks concepts.

Name	Explanation
avgRowSize	The average size of the scanned data rows
cardinality	The total number of data rows in the scanned table
colocate	Whether the table is in colocate mode
numNodes	The number of nodes to be scanned
rollup	Materialized view
preaggregation	Pre-aggregation
predicates	Predicates, the query filters

The query plan of Query 96 is divided into five fragments, numbered from 0 to 4. The query plan can be read one by one in a bottom-up manner.

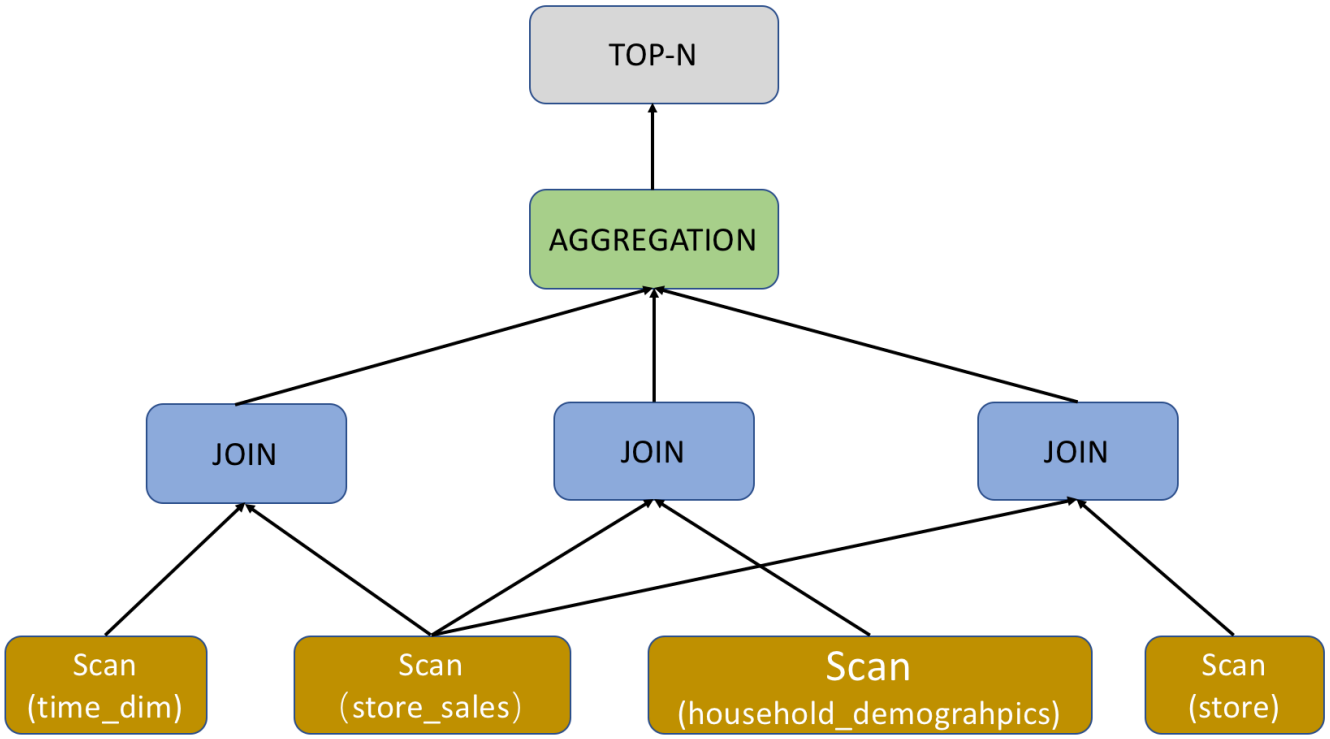
Fragment 4 is responsible for scanning the `time_dim` table and executing the related query condition (i.e. `time_dim.t_hour = 8 and time_dim.t_minute >= 30`) in advance. This step is also known as predicate pushdown. StarRocks decides whether to enable `PREAGGREGATION` for aggregation tables. In the previous figure, preaggregation of `time_dim` is disabled. In this case, all dimension columns of `time_dim` are read, which may negatively affect performance if there are many dimension columns in the table. If the `time_dim` table selects `range partition` for data division, several partitions will be hit in the query plan and irrelevant partitions will be automatically filtered out. If there is a materialized view, StarRocks will automatically select the materialized view based on the query. If there is no materialized view, the query will automatically hit the base table (for example, `rollup: time_dim` in the previous figure).

When the scan is complete, Fragment 4 ends. Data will be passed to other fragments, as indicated by EXCHANGE ID : 09 in the previous figure, to the receiving node labeled 9.

For the query plan of Query 96, Fragment 2, 3, and 4 have similar functions but they are responsible for scanning different tables. Specifically, the `Order/Aggregation/Join` operations in the query are performed in Fragment 1.

Fragment 1 uses the `BROADCAST` method to perform `Order/Aggregation/Join` operations i, that is, to broadcast the small table to the large table. If both tables are large, we recommend that you use the `SHUFFLE` method. Currently, StarRocks only supports `HASH JOIN`. The `colocate` field is used to show that the two joined tables are partitioned and bucketed in the same way, so that the join operation can be performed locally without migrating the data. When the Join operation is complete, the upper-level `aggregation`, `order by`, and `top-n` operations will be performed.

By removing the specific expressions (only keep the operators), the query plan can be presented in a more macroscopic view, as shown in the following figure.



SQL fingerprint

SQL fingerprint is used to optimize slow queries and improve system resource utilization. StarRocks uses the SQL fingerprint feature to normalize SQL statements in the slow query log (`fe.audit.log.slow_query`), categorizes the SQL statements into different types, and calculates the MD5 hash value of each SQL type to identify slow queries. The MD5 hash value is specified by the field `Digest`.

```
2021-12-27 15:13:39,108 [slow_query] |Client=172.26.xx.xxx:54956|User=r
00163e04d4c2|IsQuery=true|feIp=172.26.92.195|Stmt=select count(*) from
```

SQL statement normalization transforms a statement text into a more normalized format and preserves only important statement structure.

- Preserves object identifiers, such as database and table names.
- Converts constants into a question mark (?).
- Deletes comments and formats spaces.

For example, the following two SQL statements belong to the same type after normalization.

- SQL statements before normalization

```
SELECT * FROM orders WHERE customer_id=10 AND quantity>20


SELECT * FROM orders WHERE customer_id = 20 AND quantity > 100
```

- SQL statement after normalization

```
SELECT * FROM orders WHERE customer_id=? AND quantity>?
```

Analyze query profile

This topic describes how to check the query profile. A query profile records the execution information of all worker nodes involved in a query. Query profiles help you quickly identify the bottlenecks that affect the query performance of your StarRocks cluster.

Enable query profile

For StarRocks versions earlier than v2.5, you can enable query profile by setting the variable `is_report_success` to `true`:

```
set is_report_success = true;
```

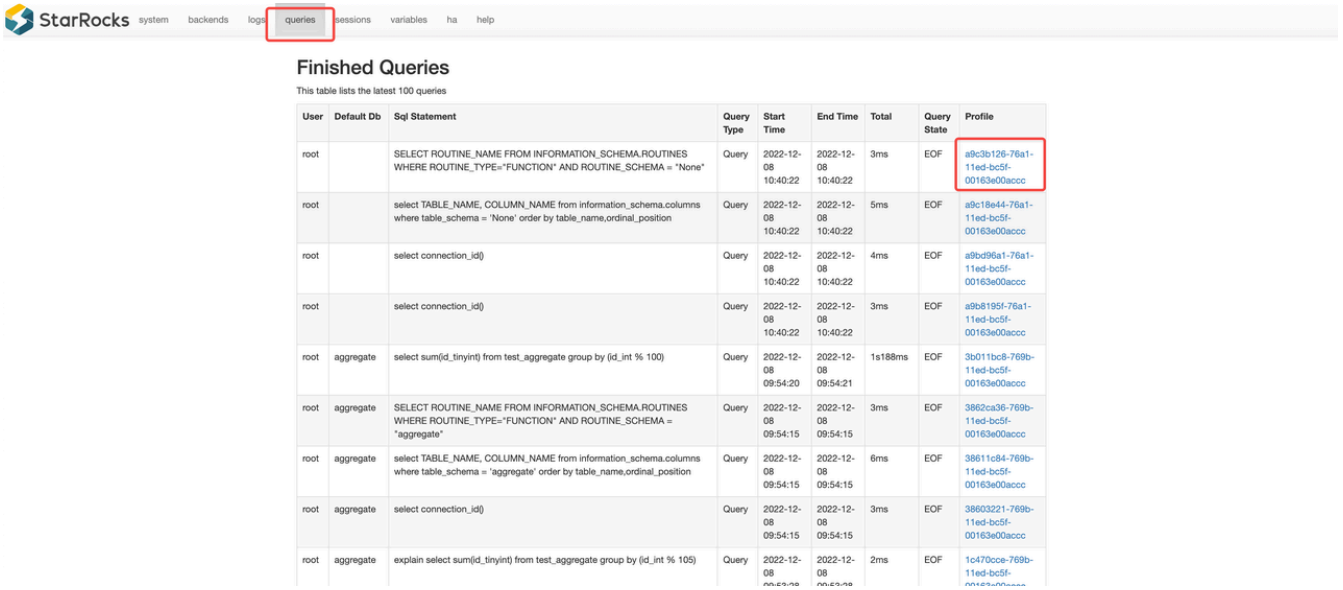
For StarRocks v2.5 or later versions, you can enable query profile by setting the variable `enable_profile` to `true`:

```
set enable_profile = true;
```

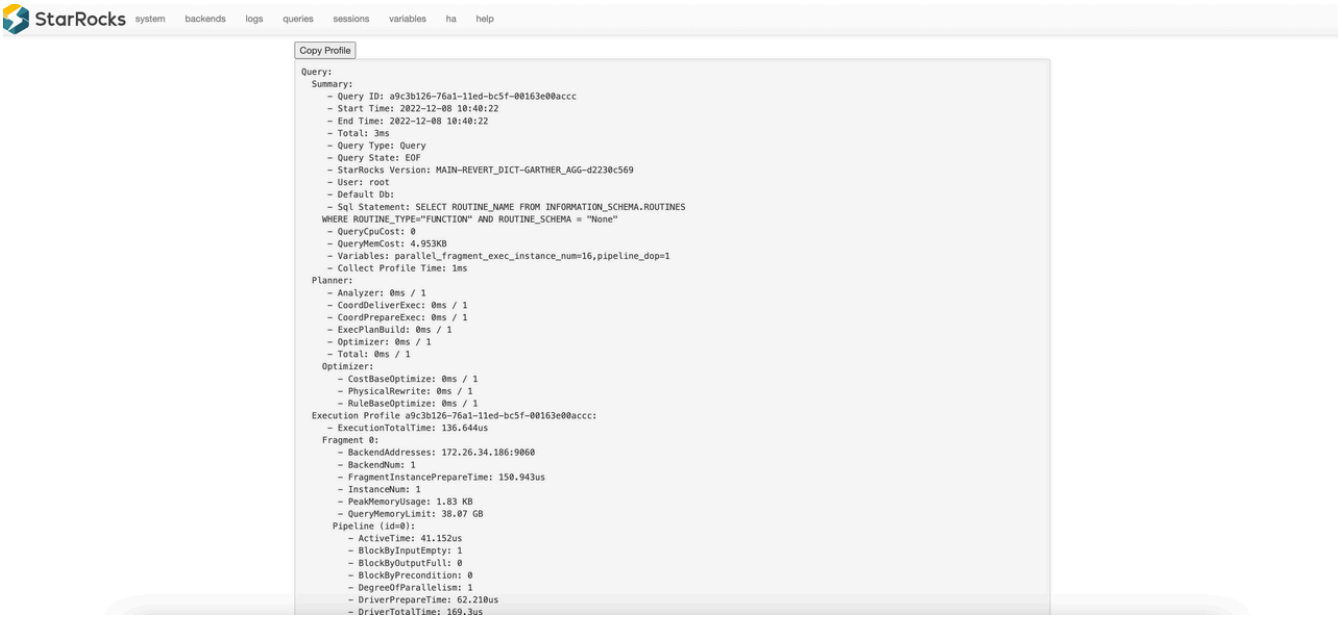
Access query profiles

Follow these steps to access your query profiles:

1. Enter `http://<fe_ip>:<fe_http_port>` in your browser.
2. On the page this is displayed, click **queries** on the top navigation pane.
3. In the **Finished Queries** list, choose the query you want to check and click the link in the **Profile** column.



The browser redirects you to a new page that contains the corresponding query profile.



Interpret a query profile

Structure of a query profile

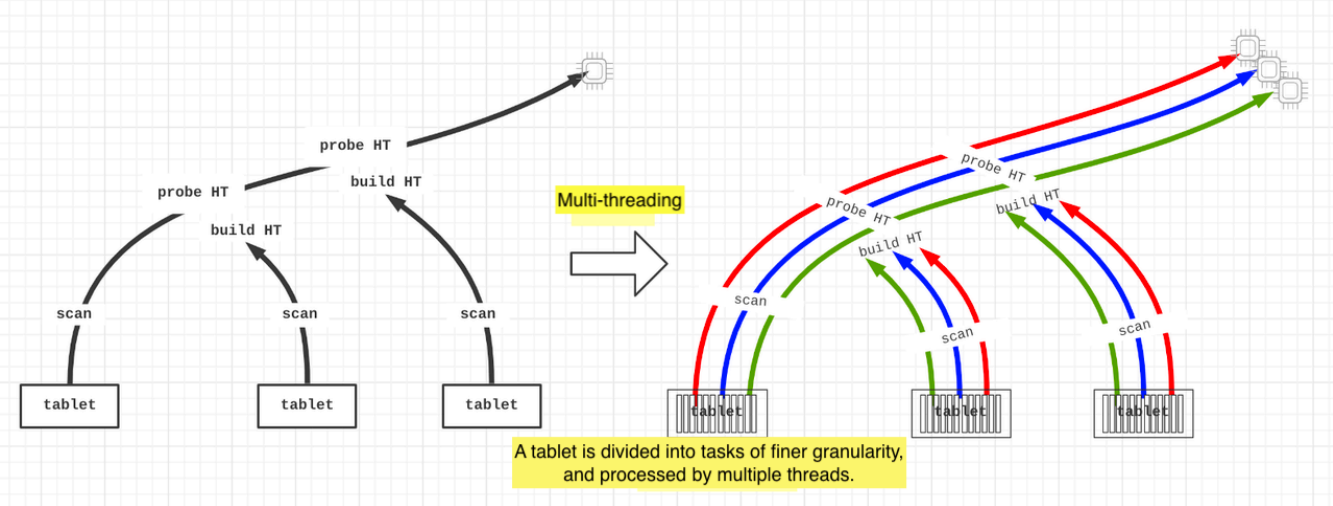
Following is an example query profile:

```
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:
      ...
```

A query profile consists of three sections:

- Fragment: the execution tree. One query can be divided into one or more fragments.
- Pipeline: the execution chain. An execution chain does not have branches. A fragment can be split into several pipelines.
- Operator: A pipeline consists of a number of operators.



A fragment consists of several pipelines.

Key metrics

A query profile encompasses a mass of metrics that show the details of the query execution. In most cases, you only need to observe the execution time of the operators and the size of the data that they processed. After finding the bottlenecks, you can solve them accordingly.

Summary

Metric	Description
Total	Total time consumed by the query, including the time spent in planning, executing, and profiling.
QueryCpuCost	Total CPU time cost of the query. CPU time costs are aggregated for concurrent processes. As a result, the value of this metric may be greater than the actual execution time of the query.
QueryMemCost	Total memory cost of the query.

Universal metrics for operators

Metric	Description
OperatorTotalTime	Total CPU time cost of the operator.
PushRowNum	Total row count of the data that the operator pushed.
PullRowNum	Total row count of the data that the operator pulled.

Scan operator

Metric	Description
Table	Table name.
ScanTime	Total scan time. Scans are performed in the asynchronous I/O thread pool.
TabletCount	Number of tablets.
PushdownPredicates	Number of the predicates that were pushed down.
BytesRead	Size of the data read by StarRocks.

Metric	Description
CompressedBytesRead	Size of the compressed data read by StarRocks.
IOTime	Total I/O time.
BitmapIndexFilterRows	Row count of the data that were filtered out by Bitmap index.
BloomFilterFilterRows	Row count of the data that were filtered out by Bloomfilter.
SegmentRuntimeZoneMapFilterRows	Row count of the data that were filtered out by runtime Zone Map.
SegmentZoneMapFilterRows	Row count of the data that were filtered out by Zone Map.
ShortKeyFilterRows	Row count of the data that were filtered out by Short Key.
ZoneMapIndexFilterRows	Row count of the data that were filtered out by Zone Map index.

Exchange operator

Metric	Description
PartType	Data distribution type. Valid values: `UNPARTITIONED`, `RANDOM`, `HASH_PARTITIONED`, and `BUCKET_SHUFFLE_HASH_PARTITIONED`.
BytesSent	Size of the data that were sent.
OverallThroughput	Overall throughput.
NetworkTime	Data package transmission time (excluding the time of post-receiving processing). See FAQ below for more about how this

Metric	Description
	metric is calculated and how it can hit an exception.
WaitTime	Time to wait because the queue at the sender side is full.

Aggregate operator

Metric	Description
GroupingKeys	Name of the grouping keys (GROUP BY columns).
AggregateFunctions	Aggregate functions.
AggComputeTime	Compute time consumed by aggregate functions.
ExprComputeTime	Compute time consumed by the expression.
HashTableSize	Size of the hash table.

Join operator

Metric	Description
JoinPredicates	Predicates of the JOIN operation.
JoinType	The JOIN type.
BuildBuckets	Bucket number of the hash table.
BuildHashTableTime	Time used to build the hash table.
ProbeConjunctEvaluateTime	Time consumed by Probe Conjunct.
SearchHashTableTimer	Time used to search the hash table.

Window Function operator

Metric	Description
ComputeTime	Compute time consumed by the window function.
PartitionKeys	Name of the partitioning keys (PARTITION BY columns).
AggregateFunctions	Aggregate functions.

Sort operator

Metric	Description
SortKeys	Name of the sort keys (ORDER BY columns).
SortType	Result sorting type: to list all results, or to list top n results.

TableFunction operator

Metric	Description
TableFunctionExecTime	Compute time consumed by the table function.
TableFunctionExecCount	Number of times the table function is executed.

Project operator

Metric	Description
ExprComputeTime	Compute time consumed by the expression.
CommonSubExprComputeTime	Compute time consumed by the common sub-expression.

LocalExchange operator

metric	Description
Type	Local Exchange type. Valid values: `Passthrough`, `Partition`, and `Broadcast`.
ShuffleNum	Number of shuffles. This metric is valid only when `Type` is `Partition`.

Time consumed by operators

- For the OlapScan and ConnectorScan operators, their time consumption is equivalent to `OperatorTotalTime + ScanTime`. Because the Scan operators perform I/O operations in the asynchronous I/O thread pool, ScanTime represents asynchronous I/O time.
- The time consumption of the Exchange operator is equivalent to `OperatorTotalTime + NetworkTime`. Because the Exchange operator sends and receives data packages in the bRPC thread pool, NetworkTime represents the time consumed by network transmission.
- For all other operators, their time cost is `OperatorTotalTime`.

Metric merging and MIN/MAX

Pipeline engine is a parallel computing engine. Each fragment is distributed to multiple machines for parallel processing, and pipelines on each machine are executed in parallel as multiple concurrent instances. Therefore, while profiling, StarRocks merges same metrics, and records the minimum and maximum values of each metric among all the concurrent instances.

Different merging strategies are adopted for different types of metrics:

- Time metrics are averages. For example:
  - `OperatorTotalTime` represents the average time cost of all the concurrent instances.
  - `__MAX_OF_OperatorTotalTime` is the maximum time cost among all the concurrent instances.
  - `__MIN_OF_OperatorTotalTime` is the minimum time cost among all the concurrent instances.

```
- OperatorTotalTime: 2.192us
  - __MAX_OF_OperatorTotalTime: 2.502us
  - __MIN_OF_OperatorTotalTime: 1.882us
```

- Non-time metrics are totaled. For example:
  - `PullChunkNum` represents the total number of all concurrent instances.
  - `__MAX_OF_PullChunkNum` is the maximum value among all the concurrent instances.
  - `__MIN_OF_PullChunkNum` is the minimum value among all the concurrent instances.

```
- PullChunkNum: 146.66K (146660)
  - __MAX_OF_PullChunkNum: 24.45K (24450)
  - __MIN_OF_PullChunkNum: 24.435K (24435)
```

- Some special metrics, which do not have minimum and maximum values, have identical values among all the concurrent instances (for example, `DegreeOfParallelism`).

Sharp difference between MIN and MAX

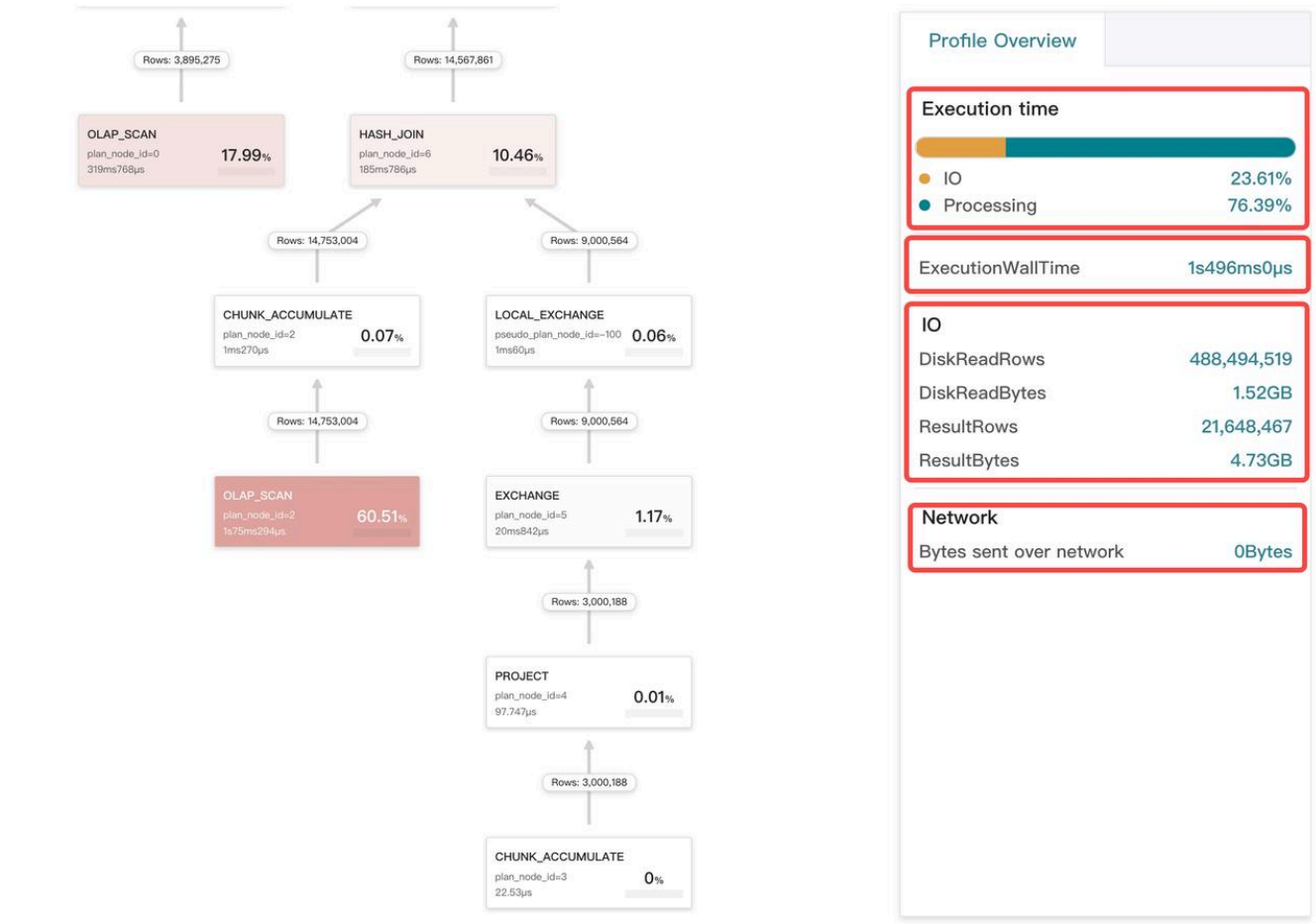
Usually, a noticeable difference between MIN and MAX values indicates the data is skewed. It possibly happens during aggregate or JOIN operations.

```
- OperatorTotalTime: 2m48s
  - __MAX_OF_OperatorTotalTime: 10m30s
  - __MIN_OF_OperatorTotalTime: 279.170us
```

Visualize a query profile

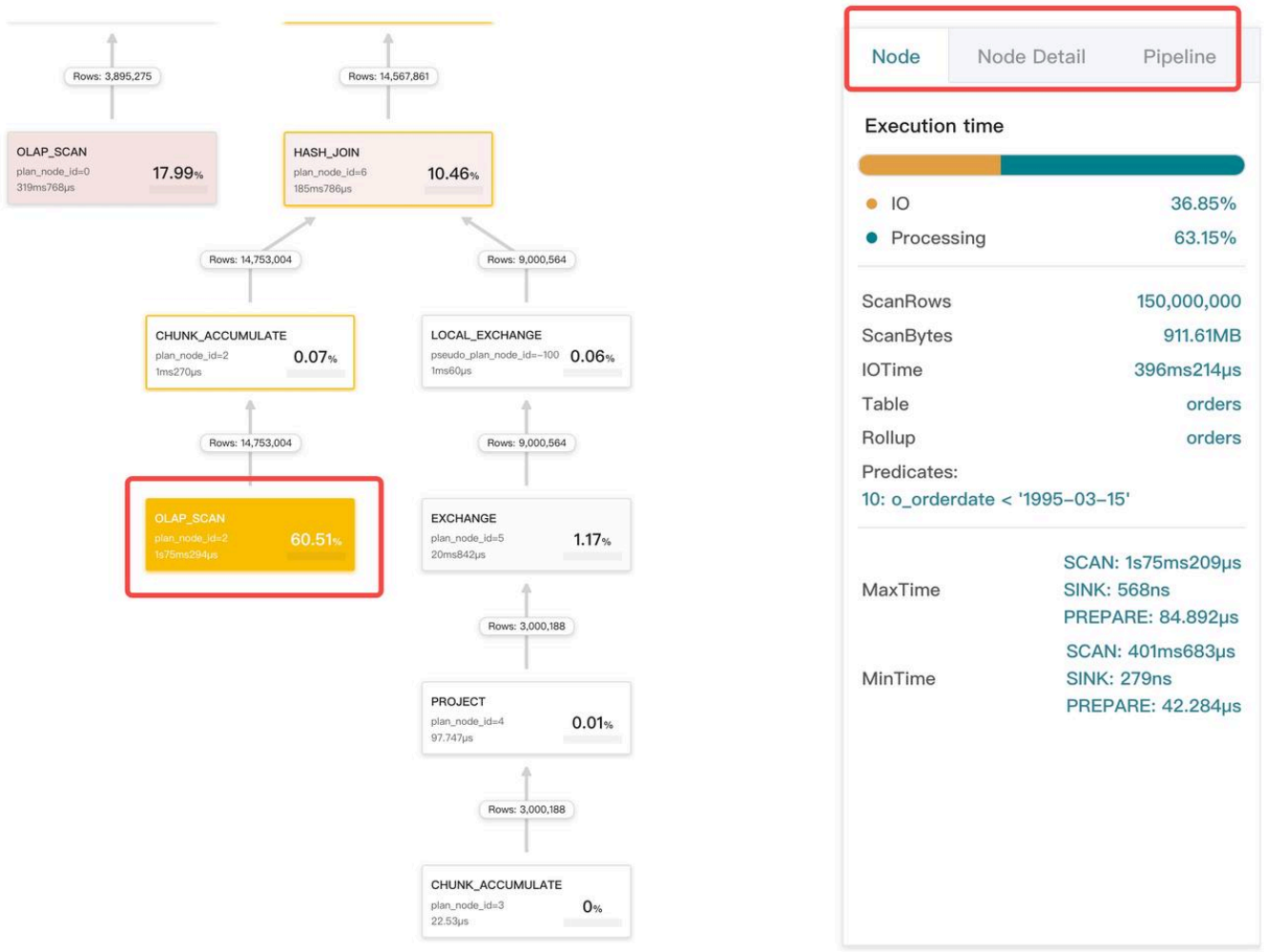
If you are a user of StarRocks Enterprise Edition, you can visualize your query profiles via StarRocks Manager.

The **Profile Overview** page displays some summary metrics, including the total execution time `ExecutionWallTime`, I/O metrics, network transmission size, and the proportion of CPU and I/O time.



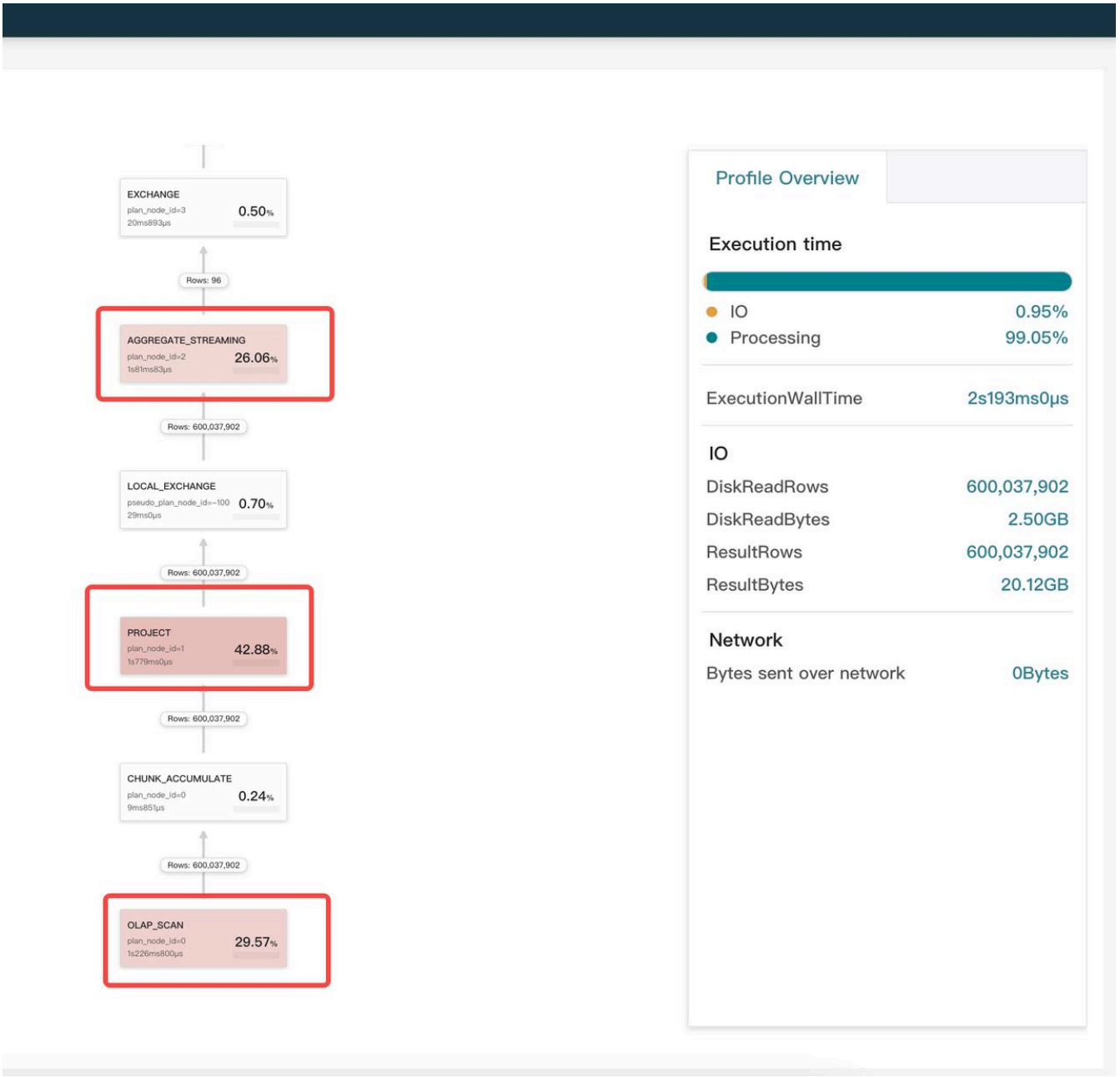
By clicking the card of an operator (a node), you can view its detailed information in the right pane of the page. There are three tabs:

- **Node**: core metrics of this operator.
- **Node Detail**: all metrics of this operator.
- **Pipeline**: metrics of the pipeline to which the operator belongs. You do not need to pay much attention to this tab because it is related only to scheduling.



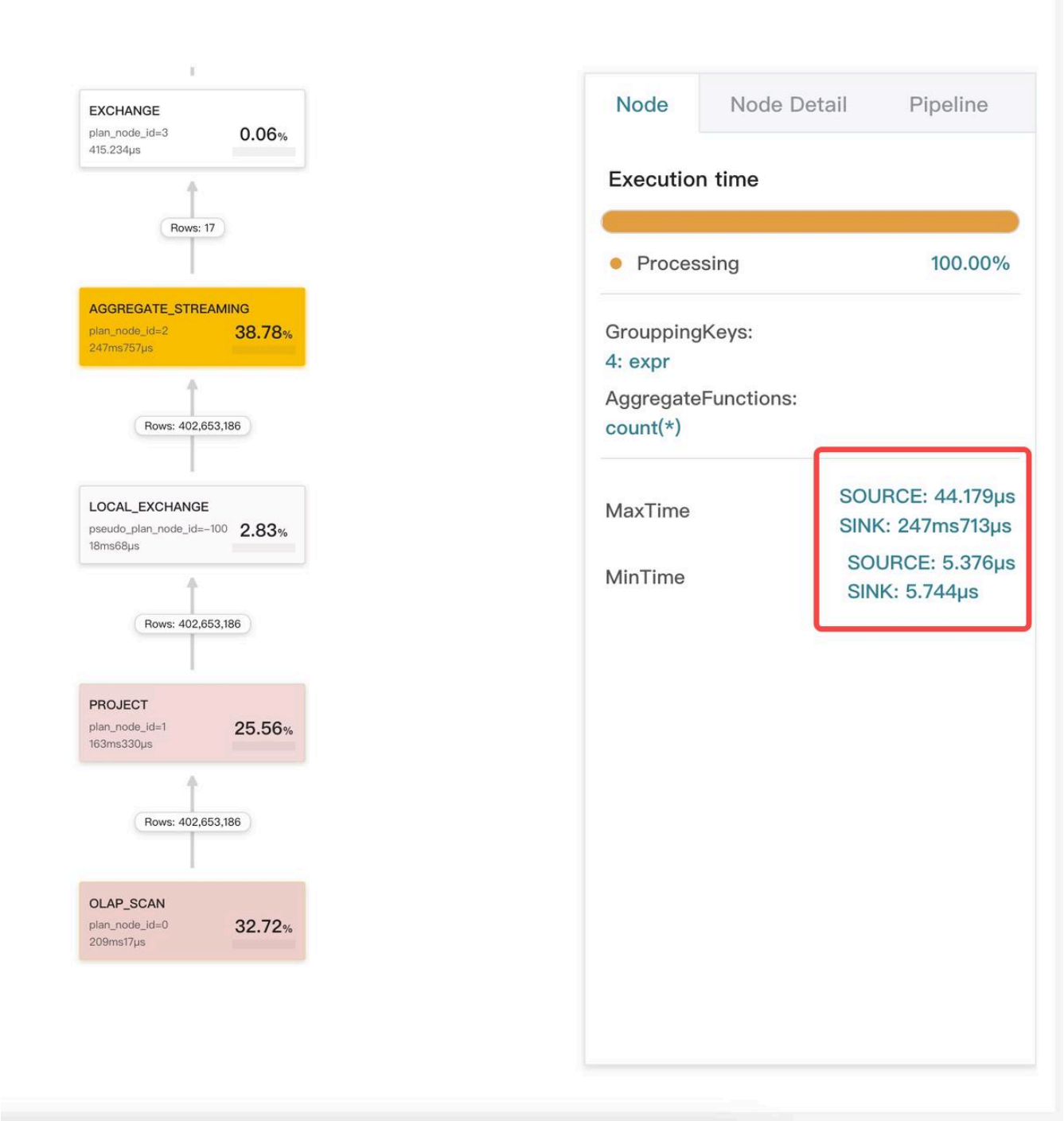
Identify bottlenecks

The larger the proportion of time taken by an operator, the darker color its card becomes. This helps you easily identify bottlenecks of the query.

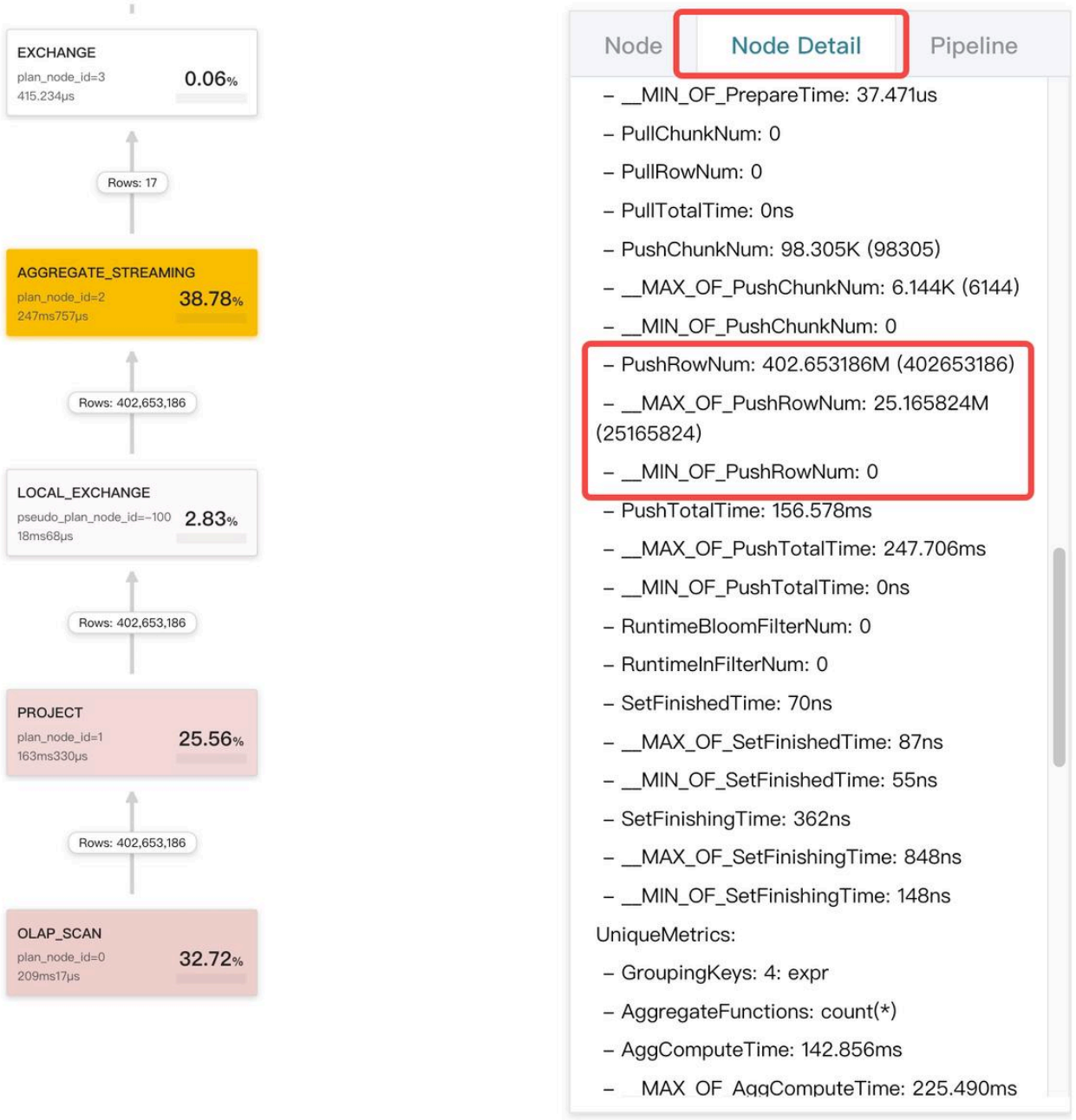


Check whether data is skewed

Click the card of the operator that takes a large proportion of time, and check its `MaxTime` and `MinTime`. A noticeable difference between `MaxTime` and `MinTime` usually indicates data is skewed.



Then, click the **Node Detail** tab, and check if any metric shows an exception. In this example, the metric `PushRowNum` of the Aggregate operator shows data skew.



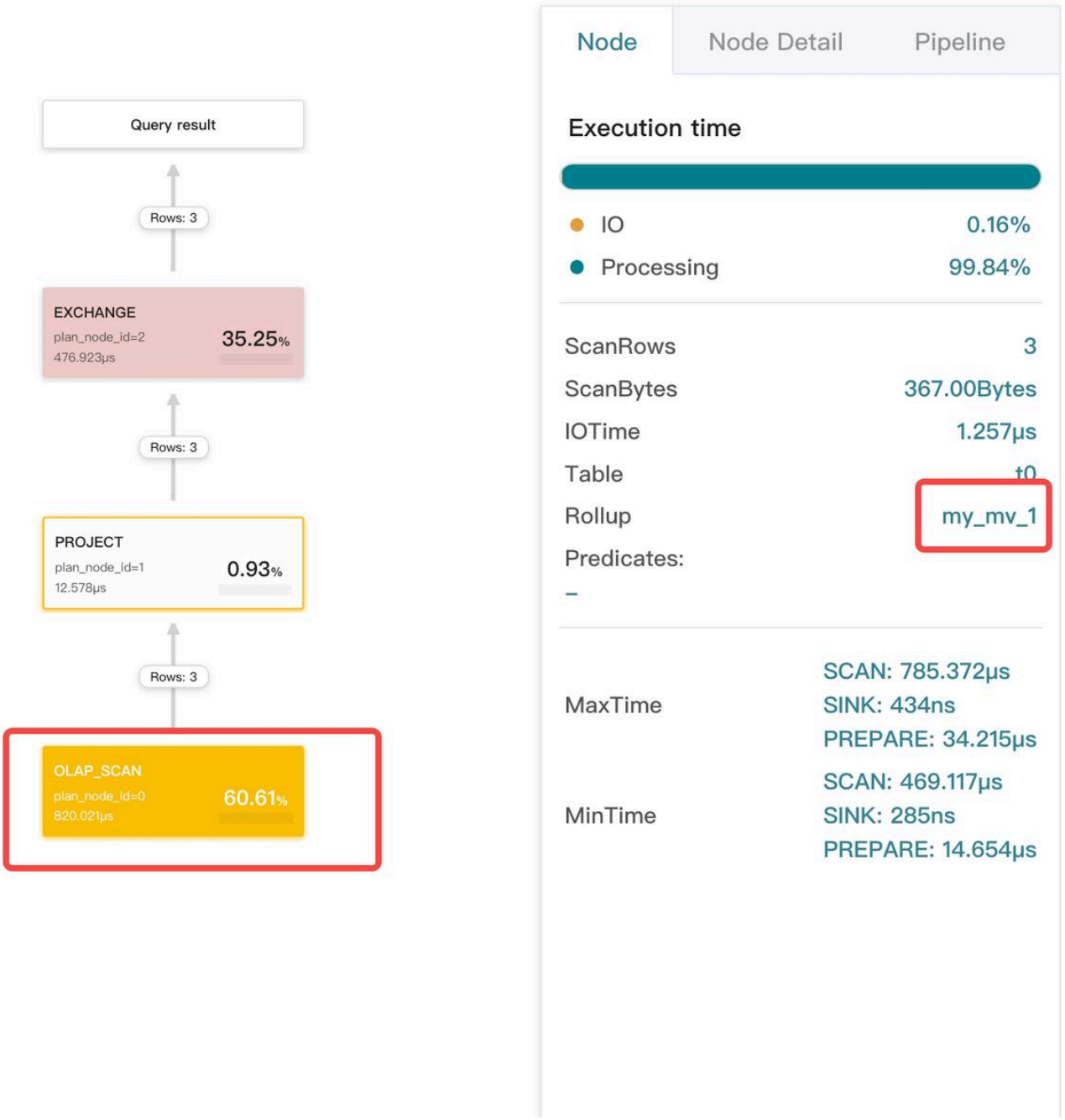
Check whether the partitioning or bucketing strategy takes effect

You can check whether the partitioning or bucketing strategy takes effect by viewing the corresponding query plan using `EXPLAIN <sql_statement>`.

```
| PLAN FRAGMENT 2                                                                                |
|  OUTPUT EXPRS:                                                                                 |
|   PARTITION: RANDOM                                                                            |
|                                                                                                |
|   STREAM DATA SINK                                                                             |
|     EXCHANGE ID: 02                                                                            |
|     UNPARTITIONED                                                                              |
|                                                                                                |
|   1:OlapScanNode                                                                               |
|      TABLE: supplier                                                                           |
|      PREAGGREGATION: ON                                                                        |
|      PREDICATES: 17: s_suppkey = 15                                                            |
|      partitions=1/1                                                                            |
|      rollup: supplier                                                                          |
|      tabletRatio=1/12                                                                          |
|      tabletList=15092                                                                          |
|      cardinality=1                                                                             |
|      avgRowSize=4.0                                                                            |
|      numNodes=0                                                                                |
+------------------------------------------------------------------------------------------------+
```

Check whether the correct materialized view is used

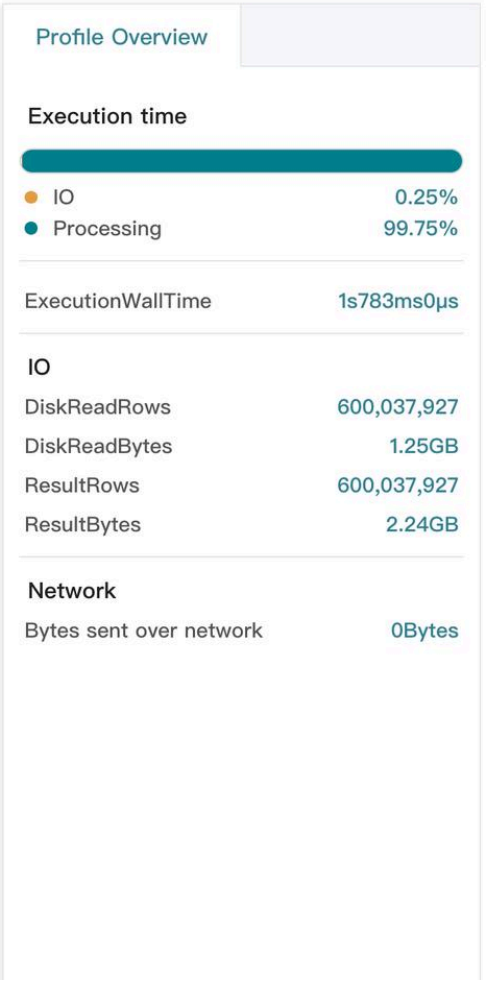
Click the corresponding Scan operator and check the `Rollup` field on the **Node Detail** tab.



Check whether the JOIN plan is proper for left and right tables

Usually, StarRocks selects the smaller table as the right table of Join. An exception occurs if the query profile shows otherwise.





Check whether the distribution type of JOIN is correct

Exchange operators are classified into three types in accordance with the data distribution type:

- `UNPARTITIONED`: Broadcast. Data is made into several copies and distributed to multiple BEs.
- `RANDOM`: Round robin.
- `HASH_PARTITIONED` and `BUCKET_SHUFFLE_HASH_PARTITIONED`: Shuffle. The difference between `HASH_PARTITIONED` and `BUCKET_SHUFFLE_HASH_PARTITIONED` lies in the hash functions used to calculate the hash code.

For Inner Join, the right table can be the `HASH_PARTITIONED` and `BUCKET_SHUFFLE_HASH_PARTITIONED` type or `UNPARTITIONED` type. Usually, `UNPARTITIONED` type is adopted only when there are less than 100K rows in the right table.

In the following example, the type of the Exchange operator is Broadcast, but the size of data transmitted by the operator greatly exceeds the threshold.

Check whether JoinRuntimeFilter takes effect

When the right child of Join is building a hash table, it creates a runtime filter. This runtime filter is sent to the left child tree, and is pushed down to the Scan operator if it is possible. You can check `JoinRuntimeFilter`-related metrics on the **Node Detail** tab of the Scan operator.

FAQ

Why is the time cost of the Exchange operator abnormal?

The time cost of an Exchange operator consists of two parts: CPU time and network time. Network time relies on the system clock. Network time is calculated as follows:

1. The sender records a `send_timestamp` before calling the bRPC interface to send the package.
2. The receiver records a `receive_timestamp` after receiving the package from the bRPC interface (post-receiving processing time excluded).
3. After the processing is complete, the receiver sends a response and calculates the network latency. The package transmission latency is equivalent to `receive_timestamp` - `send_timestamp`.

If system clocks across machines are inconsistent, the time cost of the Exchange operator hits an exception.

Why is the total time cost of all operators significantly less than the query execution time?

Possible cause: In high-concurrency scenarios, some pipeline drivers, despite being schedulable, may not be processed in time because they are queued. The waiting time is not recorded in operators' metrics, but in `PendingTime`, `ScheduleTime`, and `IOTaskWaitTime`.

Example:

From the profile, we can see that `ExecutionWallTime` is about 55 ms. However, the total time cost of all operators is less than 10 ms, which is significantly less than `ExecutionWallTime`.

Performance Optimization

Table Type Selection

StarRocks supports four table types: Duplicate Key table, Aggregate table, Unique Key table, and Primary Key table. All of them are sorted by KEY.

- `AGGREGATE KEY`: When records with the same AGGREGATE KEY is loaded into StarRocks, the old and new records are aggregated. Currently, aggregate tables supports the following aggregate functions: SUM, MIN, MAX, and REPLACE. Aggregate tables support aggregating data in advance, facilitating business statements and multi-dimensional analyses.
- `DUPLICATE KEY`: You only need to specify the sort key for a DUPLICATE KEY table. Records with the same DUPLICATE KEY exist at the same time. It is suitable for analyses that do not involve aggregating data in advance.
- `UNIQUE KEY`: When records with the same UNIQUE KEY is loaded into StarRocks, the new record overwrites the old one. A UNIQUE KEY tables is similar to an aggregate table with REPLACE function. Both are suitable for analyses involving constant updates.
- `PRIMARY KEY`: Primary Key tables guarantee the uniqueness of records, and allow you to perform realtime updating.

```
CREATE TABLE site_visit
(
    siteid      INT,
    city        SMALLINT,
    username    VARCHAR(32),
    pv BIGINT   SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, city, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10;


CREATE TABLE session_data
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
```

```
    city        CHAR(20),
    province    CHAR(20),
    ip          varchar(32),
    brower      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid) BUCKETS 10;

CREATE TABLE sales_order
(
    orderid     BIGINT,
    status      TINYINT,
    username    VARCHAR(32),
    amount      BIGINT DEFAULT '0'
)
UNIQUE KEY(orderid)
DISTRIBUTED BY HASH(orderid) BUCKETS 10;

CREATE TABLE sales_order
(
    orderid     BIGINT,
    status      TINYINT,
    username    VARCHAR(32),
    amount      BIGINT DEFAULT '0'
)
PRIMARY KEY(orderid)
DISTRIBUTED BY HASH(orderid) BUCKETS 10;
```

Colocate Table

To speed up queries, tables with the same distribution can use a common bucketing column. In that case, data can be joined locally without being transferred across the cluster during the `join` operation.

```
CREATE TABLE colocate_table
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
    city        CHAR(20),
    province    CHAR(20),
    ip          varchar(32),
```

```
    brower      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid) BUCKETS 10
PROPERTIES(
    "colocate_with" = "group1"
);
```

For more information about colocate join and replica management, see Colocate join

Flat table and star schema

StarRocks supports star schema, which is more flexible in modelling than flat tables. You can create a view to replace flat tables during modelling and then query data from multiple tables to accelerate queries.

Flat tables have the following drawbacks:

- Costly dimension updates because a flat table usually contains a massive number of dimensions. Each time a dimension is updated, the entire table must be updated. The situation exacerbates as the update frequency increases.
- High maintenance cost because flat tables require additional development workloads, storage space, and data backfilling operations.
- High data ingestion cost because a flat table has many fields and an aggregate table may contain even more key fields. During data loading, more fields need to be sorted, which prolongs data loading.

If you have high requirements on query concurrency or low latency, you can still use flat tables.

Partition and bucket

StarRocks supports two levels of partitioning: the first level is RANGE partition and the second level is HASH bucket.

- RANGE partition: RANGE partition is used to divide data into different intervals (can be understood as dividing the original table into multiple sub-tables). Most users choose

to set partitions by time, which has the following advantages:

    - Easier to distinguish between hot and cold data
    - Be able to leverage StarRocks tiered storage (SSD + SATA)
    - Faster to delete data by partition

- HASH bucket: Divides data into different buckets according to the hash value.

    - It is recommended to use a column with a high degree of discrimination for bucketing to avoid data skew.
    - To facilitate data recovery, it is recommended to keep the size of compressed data in each bucket between 100 MB to 1 GB. We recommend you configure an appropriate number of buckets when you create a table or add a partition.
    - Random bucketing is not recommended. You must explicitly specify the HASH bucketing column when you create a table.

Sparse index and bloomfilter index

StarRocks stores data in an ordered manner and builds sparse indexes at a granularity of 1024 rows.

StarRocks selects a fixed-length prefix (currently 36 bytes) in the schema as the sparse index.

When creating a table, it is recommended to place common filter fields at the beginning of the schema declaration. Fields with the highest differentiation and query frequency must be placed first.

A VARCHAR field must placed at the end of a sparse index because the index gets truncated from the VARCHAR field. If the VARCHAR field appears first, the index may be less than 36 bytes.

Use the above `site_visit` table as an example. The table has four columns: `siteid, city, username, pv`. The sort key contains three columns `siteid, city, username`, which occupy 4, 2, and 32 bytes respectively. So the prefix index (sparse index) can be the first 30 bytes of `siteid + city + username`.

In addition to sparse indexes, StarRocks also provides bloomfilter indexes, which are effective for filtering columns with high discrimination. If you want to place VARCHAR fields

before other fields, you can create bloomfilter indexes.

Inverted Index

StarRocks adopts Bitmap Indexing technology to support inverted indexes that can be applied to all columns of the Duplicate Key table and the key column of the Aggregate table and Unique Key table. Bitmap Index is suitable for columns with a small value range, such as gender, city, and province. As the range expands, the bitmap index expands in parallel.

Materialized view (rollup)

A rollup is essentially a materialized index of the original table (base table). When creating a rollup, only some columns of the base table can be selected as the schema, and the order of the fields in the schema can be different from that of the base table. Below are some use cases of using a rollup:

- Data aggregation in the base table is not high, because the base table has fields with high differentiation. In this case, you may consider selecting some columns to create rollups. Use the above `site_visit` table as an example:

  ```
  site_visit(siteid, city, username, pv)
  ```

  `siteid` may lead to poor data aggregation. If you need to frequently calculate PVs by city, you can create a rollup with only `city` and `pv`.

  ```
  ALTER TABLE site_visit ADD ROLLUP rollup_city(city, pv);
  ```

- The prefix index in the base table cannot be hit, because the way the base table is built cannot cover all the query patterns. In this case, you may consider creating a rollup to adjust the column order. Use the above `session_data` table as an example:

  ```
  session_data(visitorid, sessionid, visittime, city, province, ip,
  brower, url)
  ```

If there are cases where you need to analyze visits by `browser` and `province` in addition to `visitorid`, you can create a separate rollup:

```
ALTER TABLE session_data
ADD ROLLUP rollup_brower(brower,province,ip,url)
DUPLICATE KEY(brower,province);
```

Schema change

There are three ways to change schemas in StarRocks: sorted schema change, direct schema change, and linked schema change.

- Sorted schema change: Change the sorting of a column and reorder the data. For example, deleting a column in a sorted schema leads to data reorder.

  `ALTER TABLE site_visit DROP COLUMN city;`

- Direct schema change: Transform the data instead of reordering it, for example, changing the column type or adding a column to a sparse index.

  `ALTER TABLE site_visit MODIFY COLUMN username varchar(64);`

- Linked schema change: Complete changes without transforming data, for example, adding columns.

  `ALTER TABLE site_visit ADD COLUMN click bigint SUM default '0';`

  It is recommended to choose an appropriate schema when you create tables to accelerate schema changes.

Reference

SQL Reference

7 items

Function Reference

21 items

System variables

StarRocks provides many system variables that can be set and modified to suit your requirements. ...

User-defined variables

This topic describes how to declare and use user-defined variables.

Error Codes

This section describes the common error codes for query requests.

System limits

This topic describes the rules and limits that apply when you use StarRocks.

AWS IAM policies

A policy in AWS IAM declares a set of permissions on a specific AWS resource. After creating a poli...

All statements

This topic lists all the SQL statements supported by StarRocks and categorizes these statements by their functions.

- All statements
  - User account management
  - Cluster management
    - FE, BE, CN, Broker, process
    - Resource group
    - Table, tablet, replica
    - File, index, variable
    - SQL Blacklist
    - Plugin
  - Loading, unloading
    - Routine load
    - Other load
    - Unloading
    - ETL task
  - Catalog, database, resource
    - Catalog
    - Database
    - Resource
  - Create table, partition
  - View, materialized view
    - View
    - Materialized view
  - Function, SELECT
  - CBO statistics
  - Backup and restore
  - Utility commands

User account management

Manages users, roles, and privileges.

- ALTER USER
- CREATE ROLE
- CREATE USER
- DROP ROLE
- DROP USER
- EXECUTE AS
- GRANT
- REVOKE
- SET PASSWORD
- SET PROPERTY
- SHOW AUTHENTICATION
- SHOW GRANTS
- SHOW PROPERTY
- SHOW ROLES

Cluster management

Manages clusters, including FEs, BEs, compute nodes, brokers, resource groups, storage volumes, tables, tablets, replicas, files, indexes, variables, and plugins.

FE, BE, CN, Broker, process

- ADMIN SET CONFIG
- ADMIN SHOW CONFIG
- ALTER SYSTEM
- CANCEL DECOMMISSION
- KILL
- SHOW BACKENDS
- SHOW BROKER
- SHOW COMPUTE NODES

- SHOW FRONTENDS
- SHOW PROC
- SHOW PROCESSLIST

Resource group

- CREATE RESOURCE GROUP
- ALTER RESOURCE GROUP
- DROP RESOURCE GROUP
- SHOW RESOURCE GROUP

Table, tablet, replica

- ADMIN CANCEL REPAIR TABLE
- ADMIN CHECK TABLET
- ADMIN REPAIR TABLE
- ADMIN SET REPLICA STATUS
- ADMIN SHOW REPLICA DISTRIBUTION
- ADMIN SHOW REPLICA STATUS
- RECOVER
- SHOW TABLE STATUS

File, index, variable

- CREATE FILE
- CREATE INDEX
- DROP FILE
- DROP INDEX
- SET (variable)
- SHOW FILE
- SHOW FULL COLUMNS
- SHOW INDEX
- SHOW VARIABLES

SQL Blacklist

- ADD SQLBLACKLIST
- SHOW SQLBLACKLIST
- DELETE SQLBLACKLIST

Plugin

- INSTALL PLUGIN
- SHOW PLUGINS
- UNINSTALL PLUGIN

Loading, unloading

Routine load

- ALTER ROUTINE LOAD
- CREATE ROUTINE LOAD
- PAUSE ROUTINE LOAD
- RESUME ROUTINE LOAD
- SHOW ROUTINE LOAD
- SHOW ROUTINE LOAD TASK
- STOP ROUTINE LOAD

Other load

- ALTER LOAD
- BROKER LOAD
- CANCEL LOAD
- INSERT
- SHOW LOAD
- SHOW TRANSACTION
- SPARK LOAD
- STREAM LOAD

Unloading

- EXPORT
- CANCEL EXPORT
- SHOW EXPORT

ETL task

- SUBMIT TASK
- DROP TASK

Catalog, database, resource

Catalog

- CREATE EXTERNAL CATALOG
- DROP CATALOG
- SHOW CATALOGS
- SHOW CREATE CATALOG

Database

- ALTER DATABASE
- CREATE DATABASE
- DROP DATABASE
- SHOW CREATE DATABASE
- SHOW DATA
- SHOW DATABASES

Resource

- ALTER RESOURCE
- CREATE RESOURCE
- DROP RESOURCE
- SHOW RESOURCES

Create table, partition

- ALTER TABLE
- CANCEL ALTER TABLE
- CREATE TABLE
- CREATE TABLE AS SELECT
- CREATE TABLE LIKE
- DROP TABLE
- REFRESH EXTERNAL TABLE
- TRUNCATE TABLE
- DELETE
- SHOW ALTER TABLE
- SHOW CREATE TABLE
- SHOW DELETE
- SHOW DYNAMIC PARTITION TABLES
- SHOW PARTITIONS
- SHOW TABLES
- SHOW TABLET
- UPDATE

View, materialized view

View

- ALTER VIEW
- CREATE VIEW
- SHOW CREATE VIEW
- DROP VIEW

Materialized view

- ALTER MATERIALIZED VIEW
- CREATE MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

- CANCEL REFRESH MATERIALIZED VIEW
- REFRESH MATERIALIZED VIEW
- SHOW ALTER MATERIALIZED VIEW
- SHOW CREATE MATERIALIZED VIEW
- SHOW MATERIALIZED VIEWS

Function, SELECT

- CREATE FUNCTION
- DROP FUNCTION
- SHOW FUNCTION
- SELECT

CBO statistics

- ANALYZE TABLE
- CREATE ANALYZE
- DROP ANALYZE
- DROP STATS
- KILL ANALYZE
- SHOW ANALYZE JOB
- SHOW ANALYZE STATUS
- SHOW META

Backup and restore

- BACKUP
- CANCEL BACKUP
- CANCEL RESTORE
- CREATE REPOSITORY
- DROP REPOSITORY
- RECOVER
- RESTORE

- SHOW BACKUP
- SHOW REPOSITORIES
- SHOW RESTORE
- SHOW SNAPSHOT

Utility commands

- DESC
- EXPLAIN
- USE

ALTER USER

Description

Changes user information.

Syntax

```
ALTER USER
user_identity [auth_option]

user_identity:
'user_name'@'host'

auth_option: {
IDENTIFIED BY 'auth_string'
IDENTIFIED WITH auth_plugin
IDENTIFIED  WITH auth_plugin BY 'auth_string'
IDENTIFIED WITH auth_plugin AS 'auth_string'
}
```

```
The command is used to change user information.
 "auth_option" specifies authentication method. It currently supports
"mysql_native_password" and "authentication_ldap_simple".
```

Examples

1. Change user's password in mysql.

   ```
   ALTER USER 'jack' IDENTIFIED BY '123456';
   ```

   or

```
ALTER USER 'jack' IDENTIFIED WITH mysql_native_password BY
'123456';
```

2. To avoid transmitting password in plaintext, the following method can also be used to change password.

```
ALTER USER 'jack' IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

or

```
ALTER USER 'jack' IDENTIFIED WITH mysql_native_password AS
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

The encrypted content can be obtained by "PASSWORD()", for example:

```
SELECT PASSWORD('123456');
```

3. Alter user authentication method as "ldap".

```
ALTER USER 'jack' IDENTIFIED WITH authentication_ldap_simple
```

4. Alter user authentication method as "ldap", and specify user's DN (Distinguished Name) in ldap.

```
ALTER USER 'jack' IDENTIFIED WITH authentication_ldap_simple AS
'uid=jack,ou=company,dc=example,dc=com'
```

CREATE ROLE

Description

This statement allows users to create a role.

Syntax

```
CREATE ROLE <role_name>
```

It creates a role with no permission which could be added to the role though "GRANT" command.

Examples

Create a role.

```
CREATE ROLE role1;
```

CREATE USER

Description

Syntax

```
CREATE USER
user_identity [auth_option]
[DEFAULT ROLE 'role_name']

user_identity:
    'user_name'@'host'

auth_option: {
    IDENTIFIED BY 'auth_string'
    IDENTIFIED WITH auth_plugin
    IDENTIFIED WITH auth_plugin BY 'auth_string'
    IDENTIFIED WITH auth_plugin AS 'auth_string'
}
```

1. The command "CREATE USER" is used to create a StarRocks user. In StarRocks, a "user_identity" uniquely identifies a user.
2. "user_identity" consists of two parts, "user_name" and "host". Here, "user_name" is the username and "host" identifies the host address where the client connects. The "host" part can use % for fuzzy matching. If no host is specified, the default is '%', meaning that the user can connect to StarRocks from any host.
3. "auth_option" specifies user authentication method. It currently supports "mysql_native_password" and "authentication_ldap_simple".

If a role is specified, it will automatically grant all permissions the role owns to this newly created user with the precondition that the role already exists. If a role is not specified, the user defaults to having no permission.

Examples

1. Create a passwordless user (without specifying host, it is equivalent to jack@'%')

```
CREATE USER 'jack';
```

2. Create a password user that allows login from '172.10.1.10'.

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY '123456';
```

or

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH
mysql_native_password BY '123456';
```

3. To avoid passing in plaintext, use case 2 can also be created in the following way

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

or

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH
mysql_native_password AS
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

The encrypted content could be obtained though PASSWORD(), for example:

```
SELECT PASSWORD('123456');
```

4. Create a user authenticated by ldap

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH
authentication_ldap_simple
```

5. Create a user authenticated by ldap, and specify user's DN (Distinguished Name) in ldap

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH
authentication_ldap_simple AS
'uid=jack,ou=company,dc=example,dc=com'
```

6. Create a user who is allowed to log in from '192.168' subnet and meanwhile specify its role as example_role

```
CREATE USER 'jack'@'192.168.%' DEFAULT ROLE 'example_role';
```

7. Create a user who is allowed to log in from the domain named 'example_domain'

```
CREATE USER 'jack'@['example_domain'] IDENTIFIED BY '123456';
```

8. Create a user and specify a role

```
CREATE USER 'jack'@'%' IDENTIFIED BY '12345' DEFAULT ROLE
'my_role';
```

DROP ROLE

Description

This statement allows users to delete a role.

Syntax

```
DROP ROLE <role_name>
```

Deleting a role does not affect permissions of users who previously belonged to this role. It only decouples the role from the user without changing permissions that user has already obtained from the role.

Examples

Drop a role.

```
DROP ROLE role1;
```

DROP USER

Description

Deletes a specified user identity.

Syntax

```
 DROP USER '<user_identity>'

`user_identity`:

 user@'host'
user@['domain']
```

Examples

Delete user jack@'192.%'.

```
DROP USER 'jack'@'192.%'
```

EXECUTE AS

Description

After you obtain the privilege to impersonate a user, you can use the EXECUTE AS statement to switch the execution context of the current session to the user.

This command is supported from v2.4.

Syntax

```
EXECUTE AS user WITH NO REVERT
```

Parameters

`user`: The user must already exist.

Usage notes

- The current login user (who calls the EXECUTE AS statement) must be granted the privilege to impersonate another user. For more information, see GRANT.
- The EXECUTE AS statement must contain the WITH NO REVERT clause, which means the execution context of the current session cannot be switched back to the original login user before the current session ends.

Examples

Switch the execution context of the current session to the user `test2`.

```
EXECUTE AS test2 WITH NO REVERT;
```

After the switch succeeds, you can run the `select current_user()` command to obtain the current user.

```
select current_user();
+----------------------------+
| CURRENT_USER()             |
+----------------------------+
| 'default_cluster:test2'@'%' |
+----------------------------+
```

GRANT

Description

You can use the GRANT statement to perform the following operations:

- Grant specific privileges to a user or a role.
- Grant a role to a user. This feature is supported only in StarRock 2.4 and later versions.
- Grant user `a` the privilege to impersonate user `b`. Then user `a` can perform operations as user `b` by using the EXECUTE AS statement. This feature is supported only in StarRock 2.4 and later versions.

Syntax

- Grant specific privileges on a database and a table to a user or a role. The role or user must exist.

```
GRANT privilege_list ON db_name.tbl_name TO {user_identity | ROLE
'role_name'}
```

- Grant specific privileges on a resource to a user or a role. The role or user must exist.

```
GRANT privilege_list ON RESOURCE 'resource_name' TO
{user_identity | ROLE 'role_name'};
```

- Grant user `a` the privilege to impersonate user `b` to perform operations.

```
GRANT IMPERSONATE ON user_identity_b TO user_identity_a;
```

- Grant a role to a user. The role to be granted must exist.

```
GRANT 'role_name' TO user_identity;
```

Parameters

privilege_list

The privileges that can be granted to a user or a role. If you want to grant multiple privileges at a time, separate the privileges with commas (`,`). See Example 3. The following privileges are supported:

- `NODE_PRIV`: the privilege to manage cluster nodes such as enabling nodes and disabling nodes.
- `ADMIN_PRIV`: all privileges except `NODE_PRIV`.
- `GRANT_PRIV`: the privilege of performing operations such as creating users and roles, deleting users and roles, granting privileges, revoking privileges, and setting passwords for accounts.
- `SELECT_PRIV`: the read privilege on databases and tables.
- `LOAD_PRIV`: the privilege to load data into databases and tables.
- `ALTER_PRIV`: the privilege to change schemas of databases and tables.
- `CREATE_PRIV`: the privilege to create databases and tables.
- `DROP_PRIV`: the privilege to delete databases and tables.
- `USAGE_PRIV`: the privilege to use resources.

The preceding privileges can be classified into the following three categories:

- Node privilege: `NODE_PRIV`
- Database and table privilege: `SELECT_PRIV`, `LOAD_PRIV`, `ALTER_PRIV`, `CREATE_PRIV`, and `DROP_PRIV`
- Resource privilege: `USAGE_PRIV`

db_name.tbl_name

The database and table. This parameter supports the following three formats:

- `*.*`: indicates all databases and tables in the cluster. If this format is specified, the global privilege is granted.
- `db.*`: indicates a specific database and all tables in this database.
- `db.tbl`: indicates a specific table in a specific database.

> Note: When you use the `db.*` or `db.tbl` format, you can specify a database or a table that does not exist.

resource_name

The resource name. This parameter supports the following two formats:

- `*`: indicates all the resources.
- `resource`: indicates a specific resource.

> Note: When you use the `resource` format, you can specify a resource that does not exist.

user_identity

This parameter contains two parts: `user_name` and `host`. `user_name` indicates the user name. `host` indicates the IP address of the user. You can leave `host` unspecified or you can specify a domain for `host`. If you leave `host` unspecified, `host` defaults to `%`, which means you can access StarRocks from any host. If you specify a domain for `host`, it may take one minute for the privilege to take effect. The `user_identity` parameter must be created by the CREATE USER statement.

role_name

The role name.

Examples

Example 1: Grant the read privilege on all databases and tables to user `jack`.

```
GRANT SELECT_PRIV ON *.* TO 'jack'@'%';
```

Example 2: Grant the data loading privilege on `db1` and all tables in this database to `my_role`.

```
GRANT LOAD_PRIV ON db1.* TO ROLE 'my_role';
```

Example 3: Grant the read privilege, schema change privilege, and data loading privilege on `tbl1` in `db1` to user `jack`.

```
GRANT SELECT_PRIV,ALTER_PRIV,LOAD_PRIV ON db1.tbl1 TO
'jack'@'192.8.%';
```

Example 4: Grant the privilege to use all the resources to user `jack`.

```
GRANT USAGE_PRIV ON RESOURCE * TO 'jack'@'%';
```

Example 5: Grant the privilege to use `spark_resource` to user `jack`.

```
GRANT USAGE_PRIV ON RESOURCE 'spark_resource' TO 'jack'@'%';
```

Example 6: Grant the privilege to use `spark_resource` to role `my_role`.

```
GRANT USAGE_PRIV ON RESOURCE 'spark_resource' TO ROLE 'my_role';
```

Example 7: Grant `my_role` to user `jack`.

```
GRANT 'my_role' TO 'jack'@'%';
```

Example 8: Grant user `jack` the privilege to impersonate user `rose` to perform operations.

```
GRANT IMPERSONATE ON 'rose'@'%' TO 'jack'@'%';
```

Best practices - Customize roles based on scenarios

We recommend you customize roles to manage privileges and users. The following examples classify a few combinations of privileges for some common scenarios.

Grant global read-only privileges on StarRocks tables

```
-- Create a role.
CREATE ROLE read_only;
-- Grant the USAGE privilege on all catalogs to the role.
GRANT USAGE ON ALL CATALOGS TO ROLE read_only;
-- Grant the privilege to query all tables to the role.
GRANT SELECT ON ALL TABLES IN ALL DATABASES TO ROLE read_only;
-- Grant the privilege to query all views to the role.
GRANT SELECT ON ALL VIEWS IN ALL DATABASES TO ROLE read_only;
-- Grant the privilege to query all materialized views and the
privilege to accelerate queries with them to the role.
GRANT SELECT ON ALL MATERIALIZED VIEWS IN ALL DATABASES TO ROLE
read_only;
```

And you can further grant the privilege to use UDFs in queries:

```
-- Grant the USAGE privilege on all database-level UDF to the role.
GRANT USAGE ON ALL FUNCTIONS IN ALL DATABASES TO ROLE read_only;
-- Grant the USAGE privilege on global UDF to the role.
GRANT USAGE ON ALL GLOBAL FUNCTIONS TO ROLE read_only;
```

Grant global write privileges on StarRocks tables

```
-- Create a role.
CREATE ROLE write_only;
-- Grant the USAGE privilege on all catalogs to the role.
GRANT USAGE ON ALL CATALOGS TO ROLE write_only;
-- Grant the INSERT and UPDATE privileges on all tables to the role.
GRANT INSERT, UPDATE ON ALL TABLES IN ALL DATABASES TO ROLE
write_only;
-- Grant the REFRESH privilege on all materialized views to the role.
GRANT REFRESH ON ALL MATERIALIZED VIEWS IN ALL DATABASES TO ROLE
write_only;
```

Grant read-only privileges on a specific external catalog

```
-- Create a role.
CREATE ROLE read_catalog_only;
-- Switch to the corresponding catalog.
SET CATALOG hive_catalog;
-- Grant the privileges to query all tables and all views in all
```

```
databases.
GRANT SELECT ON ALL TABLES IN ALL DATABASES TO ROLE
read_catalog_only;
GRANT SELECT ON ALL VIEWS IN ALL DATABASES TO ROLE read_catalog_only;
```

Note: You can query only Hive table views (since v3.1).

Grant write-only privileges on a specific external catalog

You can only write data into Iceberg tables (since v3.1).

```
-- Create a role.
CREATE ROLE write_catalog_only;
-- Switch to the corresponding catalog.
SET CATALOG iceberg_catalog;
-- Grant the privilege to write data into Iceberg tables.
GRANT INSERT ON ALL TABLES IN ALL DATABASES TO ROLE
write_catalog_only;
```

Grant privileges to perform backup and restore operations on global, database, table, and partition levels

- Grant privileges to perform global backup and restore operations:

  The privileges to perform global backup and restore operations allow the role to back up and restore any database, table, or partition. It requires the REPOSITORY privilege on the SYSTEM level, the privileges to create databases in the default catalog, to create tables in any database, and to load and export data on any table.

  ```
  -- Create a role.
  CREATE ROLE recover;
  -- Grant the REPOSITORY privilege on the SYSTEM level.
  GRANT REPOSITORY ON SYSTEM TO ROLE recover;
  -- Grant the privilege to create databases in the default
  catalog.
  GRANT CREATE DATABASE ON CATALOG default_catalog TO ROLE recover;
  -- Grant the privilege to create tables in any database.
  GRANT CREATE TABLE ON ALL DATABASE TO ROLE recover;
  -- Grant the privilege to load and export data on any table.
  ```

```
GRANT INSERT, EXPORT ON ALL TABLES IN ALL DATABASES TO ROLE
recover;
```

- Grant the privileges to perform database-level backup and restore operations:

  The privileges to perform database-level backup and restore operations require the REPOSITORY privilege on the SYSTEM level, the privilege to create databases in the default catalog, the privilege to create tables in any database, the privilege to load data into any table, and the privilege export data from any table in the database to be backed up.

```
-- Create a role.
CREATE ROLE recover_db;
-- Grant the REPOSITORY privilege on the SYSTEM level.
GRANT REPOSITORY ON SYSTEM TO ROLE recover_db;
-- Grant the privilege to create databases.
GRANT CREATE DATABASE ON CATALOG default_catalog TO ROLE
recover_db;
-- Grant the privilege to create tables.
GRANT CREATE TABLE ON ALL DATABASE TO ROLE recover_db;
-- Grant the privilege to load data into any table.
GRANT INSERT ON ALL TABLES IN ALL DATABASES TO ROLE recover_db;
-- Grant the privilege to export data from any table in the
database to be backed up.
GRANT EXPORT ON ALL TABLES IN DATABASE <db_name> TO ROLE
recover_db;
```

- Grant the privileges to perform table-level backup and restore operations:

  The privileges to perform table-level backup and restore operations require the REPOSITORY privilege on the SYSTEM level, the privilege to create tables in corresponding databases, the privilege to load data into any table in the database, and the privilege to export data from the table to be backed up.

```
-- Create a role.
CREATE ROLE recover_tbl;
-- Grant the REPOSITORY privilege on the SYSTEM level.
GRANT REPOSITORY ON SYSTEM TO ROLE recover_tbl;
-- Grant the privilege to create tables in corresponding
databases.
GRANT CREATE TABLE ON DATABASE <db_name> TO ROLE recover_tbl;
-- Grant the privilege to load data into any table in a database.
```

```
GRANT INSERT ON ALL TABLES IN DATABASE <db_name> TO ROLE
recover_db;
-- Grant the privilege to export data from the table you want to
back up.
GRANT EXPORT ON TABLE <table_name> TO ROLE recover_tbl;
```

- Grant the privileges to perform partition-level backup and restore operations:

  The privileges to perform partition-level backup and restore operations require the REPOSITORY privilege on the SYSTEM level, and the privilege to load and export data on the corresponding table.

  ```
  -- Create a role.
  CREATE ROLE recover_par;
  -- Grant the REPOSITORY privilege on the SYSTEM level.
  GRANT REPOSITORY ON SYSTEM TO ROLE recover_par;
  -- Grant the privilege to load and export data on the
  corresponding table.
  GRANT INSERT, EXPORT ON TABLE <table_name> TO ROLE recover_par;
  ```

For the best practices of multi-service access control, see Multi-service access control.

REVOKE

Description

You can use the REVOKE statement to perform the following operations:

- Revoke specific privileges from a user or a role.
- Revoke the privilege that allows a user to impersonate another user to perform operations. This feature is supported only in StarRock 2.4 and later versions.
- Revoke a role from a user. This feature is supported only in StarRock 2.4 and later versions.

Syntax

- Revoke specific privileges on a database and a table from a user or a role. The role from which you want to revoke privileges must already exist.

```
REVOKE privilege_list ON db_name.tbl_name FROM {user_identity |
ROLE 'role_name'}
```

- Revoke specific privileges on a resource from a user or a role. The role from which you want to revoke privileges must already exist.

```
REVOKE privilege_list ON RESOURCE 'resource_name' FROM
{user_identity | ROLE 'role_name'};
```

- Revoke the privilege that allows user `a` to impersonate user `b` to perform operations.

```
REVOKE IMPERSONATE ON user_identity_b FROM user_identity_a;
```

- Revoke a role from a user. The role must already exist.

```
REVOKE 'role_name' FROM user_identity;
```

Parameters

privilege_list

The privileges that can be revoked from a user or a role. If you want to revoke multiple privileges at a time, separate the privileges with commas (`,`). You can revoke the following privileges:

- `NODE_PRIV`: the privilege to manage cluster nodes such as enabling nodes and disabling nodes. This privilege can only be granted to the root user.
- `ADMIN_PRIV`: all privileges except `NODE_PRIV`.
- `GRANT_PRIV`: the privilege of performing operations such as creating users and roles, deleting users and roles, granting privileges, revoking privileges, and setting passwords for accounts.
- `SELECT_PRIV`: the read privilege on databases and tables.
- `LOAD_PRIV`: the privilege to load data into databases and tables.
- `ALTER_PRIV`: the privilege to change schemas of databases and tables.
- `CREATE_PRIV`: the privilege to create databases and tables.
- `DROP_PRIV`: the privilege to delete databases and tables.
- `USAGE_PRIV`: the privilege to use resources.

db_name.tbl_name

The database and table. This parameter supports the following three formats:

- `*.*`: indicates all databases and tables in the cluster.
- `db.*`: indicates a specific database and all tables in this database.
- `db.tbl`: indicates a specific table in a specific database.

> Note: When you use the `db.*` or `db.tbl` format, you can specify a database or a table that does not exist.

resource_name

The resource name. This parameter supports the following two formats:

- `*`: indicates all the resources.
- `resource`: indicates a specific resource.

> Note: When you use the `resource` format, you can specify a resource that does not exist.

user_identity

This parameter contains two parts: `user_name` and `host`. `user_name` indicates the user name. `host` indicates the IP address of the user. You can leave `host` unspecified or you can specify a domain for `host`. If you leave `host` unspecified, `host` defaults to `%`, which means you can access StarRocks from any host. If you specify a domain for `host`, it may take one minute for the privilege to take effect. The `user_identity` parameter must be created by the CREATE USER statement.

Examples

Example 1: Revoke the read privilege on all tables in database `db1` from user `jack`.

```
REVOKE SELECT_PRIV ON db1.* FROM 'jack'@'192.%';
```

Example 2: Revoke the privilege to use `spark_resource` from user `jack`.

```
REVOKE USAGE_PRIV ON RESOURCE 'spark_resource' FROM 'jack'@'192.%';
```

Example 3: Revoke `my_role` from user `jack`.

```
REVOKE 'my_role' FROM 'jack'@'%';
```

Example 4: Revoke the privilege that allows user `jack` to impersonate `rose` to perform operations.

```
REVOKE IMPERSONATE ON 'rose'@'%' FROM 'jack'@'%';
```

SET PASSWORD

Description

Syntax

```
SET PASSWORD [FOR user_identity] =
[PASSWORD('plain password')]|['hashed password']
```

SET PASSWORD command can be used to change user's login password. If the field, [FOR user_identity], does not exist, the current user's password shall be modified.

```
Please note that the user_identity must match exactly the
user_identity specified when creating a user by using CREATE USER.
Otherwise, the user will be reported as non-existent. If
user_identity is not specified, the current user is 'username'@'ip,
which may not match any user_identity. The current user can be viewed
through SHOW GRANTS.
```

PASSWORD() inputs plaintext passwords, while the direct usage of strings requires the transmission of encrypted password.

Modifying passwords of other users requires administrator privileges.

Examples

1. Modify the password of the current user

   ```
   SET PASSWORD = PASSWORD('123456')
   SET PASSWORD = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'
   ```

2. Modify the password of the specified user

```
SET PASSWORD FOR 'jack'@'192.%' = PASSWORD('123456')
SET PASSWORD FOR 'jack'@['domain'] =
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'
```

SET PROPERTY

Description

Syntax

```
SET PROPERTY [FOR 'user'] 'key' = 'value' [, 'key' = 'value']
```

Set user attributes, including resources allocated to users and etc. The user properties here means the attributes for users, not for user_identity. That is to say, if two users, 'jack'@'%' and 'jack'@'192.%', are created through CREATE USER statement, then SET PROPERTY statement can only be used for user jack, not 'jack'@'%' or 'jack'@'192.%'.

key:

Super user permission:

```
max_user_connections: Maximum number of connections
resource.cpu_share: cpu resource assignment
```

Ordinary user permission:

```
quota.normal: Resource allocation at the normal level
quota.high: Resource allocation at the high level
quota.low: Resource allocation at the low level
```

Examples

1. Modify the maximum number of connections to 1000 for the user jack

```
SET PROPERTY FOR 'jack' 'max_user_connections' = '1000';
```

2. Modify cpu_share to 1000 for the user jack

```
SET PROPERTY FOR 'jack' 'resource.cpu_share' = '1000';
```

3. Modify the weight of the normal level for the user jack

```
SET PROPERTY FOR 'jack' 'quota.normal' = '400';
```

SHOW AUTHENTICATION

Description

Displays the authentication information of the current user or all users in the current cluster. Each user has permissions to view their authentication information. Only users with the global `GRANT_PRIV` permission can view the authentication information of all users or the authentication information of specified users.

Syntax

```
SHOW [ALL] AUTHENTICATION [FOR USERNAME]
```

Parameters

Parameter	Required	Description
ALL	No	If this keyword is specified, the authentication information of all users in the current cluster is returned. If this keyword is not specified, only the authentication information of the current user is returned.
USERNAME	No	If this parameter is specified, the authentication information of a specified user can be viewed. If this parameter is not specified, only the authentication information of the current user can be viewed.

Output

```
+--------------+----------+------------+------------------+
| UserIdentity  | Password | AuthPlugin  | UserForAuthPlugin |
+--------------+----------+------------+------------------+
```

Field	Description
UserIdentity	The user identity.
Password	Whether a password is used to log in to the StarRocks cluster. • `Yes`: A password is used. • `No`: No password is used.
AuthPlugin	The interface that is used for authentication. Valid values: `MYSQL_NATIVE_PASSWORD` and `AUTHENTICATION_LDAP_SIMPLE`. If no interface is used, `NULL` is returned.
UserForAuthPlugin	The name of the user using the LDAP or Kerberos authentication. If no authentication is used, `NULL` is returned.

Examples

Example 1: Display the authentication information of the current user.

```
SHOW AUTHENTICATION;
+--------------+----------+------------+------------------+
| UserIdentity | Password | AuthPlugin | UserForAuthPlugin |
+--------------+----------+------------+------------------+
| 'root'@'%'   | No       | NULL       | NULL              |
+--------------+----------+------------+------------------+
```

Example 2: Display the authentication information of all users in the current cluster.

```
SHOW ALL AUTHENTICATION;
+--------------+----------+--------------------------+-----------
-------+
```

```
| UserIdentity  | Password | AuthPlugin                 |
UserForAuthPlugin |
+---------------+----------+----------------------------+------------
-------+
| 'root'@'%'    | Yes      | NULL                       | NULL
|
| 'chelsea'@'%' | No       | AUTHENTICATION_LDAP_SIMPLE | NULL
|
+---------------+----------+----------------------------+------------
-------+
```

Example 3: Display the authentication information of a specified user.

```
SHOW AUTHENTICATION FOR root;
+--------------+----------+------------+-------------------+
| UserIdentity | Password | AuthPlugin | UserForAuthPlugin |
+--------------+----------+------------+-------------------+
| 'root'@'%'   | Yes      | NULL       | NULL              |
+--------------+----------+------------+-------------------+
```

SHOW GRANTS

Description

This statement is used to view user permissions.

Syntax:

```
SHOW [ALL] GRANTS [FOR user_identity]
```

Note:

```
1. SHOW ALL GRANTS can view the privileges of all users.
2. When user_identity is specified, the permissions of the user thus
specified will be showed. Meanwhile, this user_identity must be
created through CREATE USER command.
3. When user_identity is not specified, the permissions of the
current user will be showed.
```

Examples

1. View all users' permissions.

   ```
   SHOW ALL GRANTS;
   ```

2. View specified users' permissions.

   ```
   SHOW GRANTS FOR jack@'%';
   ```

3. View the current user's permissions.

   ```
   SHOW GRANTS;
   ```

SHOW PROPERTY

Description

This statement is used to view the user's properties

Syntax:

```
SHOW PROPERTY [FOR user] [LIKE key]
```

Examples

1. View the properties of jack user

   ```
   SHOW PROPERTY FOR 'jack'
   ```

2. View the properties related to the cluster imported by Jack user

   ```
   SHOW PROPERTY FOR 'jack' LIKE '%load_cluster%'
   ```

SHOW ROLES

Description

This statement is used to show all the information about the created roles, containing roles' names, users included and their permissions.

Syntax:

```
SHOW ROLES
```

Examples

View the created roles.

```
SHOW ROLES;
```

ADD SQLBLACKLIST

Description

Adds a regular expression to the SQL blacklist to forbid certain SQL patterns. When the SQL Blacklist feature is enabled, StarRocks compares all SQL statements to be executed against the SQL regular expressions in the blacklist. StarRocks does not execute SQLs that match any regular expression in the blacklist and returns an error. This prevents certain SQLs from triggering cluster crashes or unexpected behavior.

For more about SQL Blacklist, see Manage SQL Blacklist.

> **TIP**
>
> - Only users with the ADMIN privilege can add SQL regular expressions to the SQL blacklist.
> - Currently, StarRocks supports adding SELECT statements to the SQL Blacklist.

Syntax

```
ADD SQLBLACKLIST "<sql_reg_expr>"
```

Parameter

`sql_reg_expr`: the regular expression that is used to specify a certain SQL pattern. To distinguish the special characters in the SQL statement and those in the regular expression, you need to use the escape character `\` as a prefix for the special characters in the SQL statement, such as `(`, `)`, and `+`. Whereas `(` and `)` have been often used in SQL statements, StarRocks can identify `(` and `)` in SQL statements directly. You do not need to use the escape character for `(` and `)`.

Examples

Example 1: Add `count(\*)` to the SQL blacklist.

```
mysql> ADD SQLBLACKLIST "select count(\\*) from .+";
```

Example 2: Add `count(distinct )` to the SQL blacklist.

```
mysql> ADD SQLBLACKLIST "select count(distinct .+) from .+";
```

Example 3: Add `order by limit x, y, 1 <= x <=7, 5 <=y <=7` to the SQL blacklist.

```
mysql> ADD SQLBLACKLIST "select id_int from test_all_type_select1
    order by id_int
    limit [1-7], [5-7]";
```

Example 4: Add a complex SQL regular expression to the SQL blacklist. This example is to demonstrate how to use escape characters for `*` and `-` in SQL statements.

```
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";
```

ADMIN CANCEL REPAIR

Description

This statement is used to cancel repairing specified tables or partitions with high priority.

Syntax

```
ADMIN CANCEL REPAIR TABLE table_name[ PARTITION (p1,...)]
```

Note

> This statement only indicates that the system will no longer repair sharding replicas of specified tables or partitions with high priority. It still repairs these copies by default scheduling.

Examples

1. Cancel high priority repair

```
ADMIN CANCEL REPAIR TABLE tbl PARTITION(p1);
```

ADMIN CHECK TABLET

Description

This statement is used to check a group of tablets.

Syntax:

```
ADMIN CHECK TABLE (tablet_id1, tablet_id2, ...)
PROPERTIES("type" = "...")
```

Note:

1. The "type" property in tablet id and PROPERTIES must be specified.
2. Currently, "type" only supports:

   Consistency: Check the consistency of replicas of the tablet. This command is asynchronous. After sending it, StarRocks will start checking the consistency among corresponding tablets. The final results will be shown in the InconsistentTabletNum column in the result of SHOW PROC "/statistic".

Examples

1. Check the consistency of replicas on a group of specified tablets

   ```
   ADMIN CHECK TABLET (10000, 10001)
   PROPERTIES("type" = "consistency");
   ```

ADMIN REPAIR

Description

This statement is used to try and fix the specified tables or partitions first.

Syntax:

```
ADMIN REPAIR TABLE table_name[ PARTITION (p1,...)]
```

Note:

1. This statement only means that the system attempts to repair sharding replicas of specified tables or partitions with high priority without the guarantee that the repair will be successful. Users can view the repair status through ADMIN SHOW REPLICA STATUS command.
2. The default timeout is 14400 seconds (4 hours). Timeout means the system will not repair sharding replicas of specified tables or partitions with high priority. In case of timeout, the command needs to be used again for the intended settings.

Examples

1. Attempt to repair specified tables

   ```
   ADMIN REPAIR TABLE tbl1;
   ```

2. Attempt to fix specified partition

   ```
   ADMIN REPAIR TABLE tbl1 PARTITION (p1, p2);
   ```

ADMIN SET CONFIG

Description

This statement is used to set configuration items for the cluster (Currently, only FE dynamic configuration items can be set using this command). You can view these configuration items using the ADMIN SHOW FRONTEND CONFIG command.

The configurations will be restored to the default values in the `fe.conf` file after the FE restarts. Therefore, we recommend that you also modify the configuration items in `fe.conf` to prevent the loss of modifications.

Syntax

```
ADMIN SET FRONTEND CONFIG ("key" = "value")
```

Examples

1. Set `disable_balance` to `true`.

```
ADMIN SET FRONTEND CONFIG ("disable_balance" = "true");
```

ADMIN SET REPLICA STATUS

Description

This statement is used to set the status of the specified replicas.

This command currently only used to manually set the status of some replicas to BAD or OK, allowing the system to automatically repair these replicas.

Syntax:

```
ADMIN SET REPLICA STATUS
PROPERTIES ("key" = "value", ...)
```

The following attributes are currently supported:

"table_id': required. Specify a Tablet Id.

"backend_id": required. Specify a Backend Id.

"status": required. Specify the status. Currently only "bad" and "ok" are supported.

If the specified replica does not exist or its status is bad, the replica will be ignored.

Note:

Replicas set to Bad status may be dropped immediately, please proceed with caution.

Examples

1. Set the replica status of tablet 10003 on BE 10001 to bad.

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003",
"backend_id" = "10001", "status" = "bad");
```

2. Set the replica status of tablet 10003 on BE 10001 to ok.

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003",
"backend_id" = "10001", "status" = "ok");
```

ADMIN SHOW CONFIG

Description

Displays the configuration of the current cluster (Currently, only FE configuration items can be displayed). For detailed description of these configuration items, see Configuration.

If you want to set or modify a configuration item, use ADMIN SET CONFIG.

Syntax

```
ADMIN SHOW FRONTEND CONFIG [LIKE "pattern"]
```

Note:

Description of the return parameters:

```
1. Key:        Configuration item name
2. Value:      Configuration item value
3. Type:       Configuration item type
4. IsMutable:  Whether it can be set through the ADMIN SET CONFIG
command
5. MasterOnly: Whether it only applies to leader FE
6. Comment:    Configuration item description
```

Examples

1. View the configuration of the current FE node.

   ```
   ADMIN SHOW FRONTEND CONFIG;
   ```

2. Search for the configuration of the current FE node by using the `like` predicate.

```
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)
```

ADMIN SHOW REPLICA DISTRIBUTION

Description

This statement is used to show the distribution status of a table or a partition replica.

Syntax:

```
ADMIN SHOW REPLICA DISTRIBUTION FROM [db_name.]tbl_name [PARTITION
(p1, ...)]
```

Note:

The Graph column in the result shows the distribution ratio of replicas graphically.

Examples

1. View the replica distribution of tables

   ```
   ADMIN SHOW REPLICA DISTRIBUTION FROM tbl1;
   ```

2. View the replica distribution of partitions in the table

   ```
   ADMIN SHOW REPLICA DISTRIBUTION FROM db1.tbl1 PARTITION(p1, p2);
   ```

ADMIN SHOW REPLICA STATUS

Description

This statement is used to show the status of replicas for a table or a partition.

Syntax:

```
ADMIN SHOW REPLICA STATUS FROM [db_name.]tbl_name [PARTITION (p1,
...)]
[where_clause]
```

```
where_clause:
WHERE STATUS [!]= "replica_status"
```

```
replica_status:
OK:            The replica is healthy
DEAD:          The Backend of replica is not available
VERSION_ERROR: The replica data version is missing
SCHEMA_ERROR:  The schema hash of replica is incorrect
MISSING:       The replica does not exist
```

Examples

1. View the status of all replicas of the table.

   ```
   ADMIN SHOW REPLICA STATUS FROM db1.tbl1;
   ```

2. View the replica of a partition with the status of VERSION_ERROR.

```
ADMIN SHOW REPLICA STATUS FROM tbl1 PARTITION (p1, p2)
WHERE STATUS = "VERSION_ERROR";
```

3. View all unhealthy replicas of the table.

```
ADMIN SHOW REPLICA STATUS FROM tbl1
WHERE STATUS != "OK";
```

ALTER RESOURCE GROUP

Description

Alters the configuration of a resource group.

Syntax

```
ALTER RESOURCE GROUP resource_group_name
{  ADD CLASSIFIER1, CLASSIFIER2, ...
 | DROP (CLASSIFIER_ID_1, CLASSIFIER_ID_2, ...)
 | DROP ALL
 | WITH resource_limit
};
```

Parameters

Parameter	Description
resource_group_name	Name of the resource group to be altered.
ADD	Add classifiers to the resource group. See CREATE RESOURCE GROUP - Parameters for more information on how to define a classifier.
DROP	Drop classifiers from the resource group via classifier IDs. You can check the ID of a classifier via SHOW RESOURCE GROUP statement.
DROP ALL	Drop all classifiers from the resource group.
WITH	Modify the resource limits of the resource group. See CREATE RESOURCE GROUP - Parameters for more information on how

Parameter	Description
	to set resource limits.

Examples

Example 1: Adds a new classifier to the resource group `rg1`.

```
ALTER RESOURCE GROUP rg1 ADD (user='root', query_type in ('select'));
```

Example 2: Drops classifiers with ID `300040`, `300041`, and `300041` from the resource group `rg1`.

```
ALTER RESOURCE GROUP rg1 DROP (300040, 300041, 300041);
```

Example 3: Drops all classifiers from the resource group `rg1`.

```
ALTER RESOURCE GROUP rg1 DROP ALL;
```

Example 4: modifies the resource limits of the resource group `rg1`.

```
ALTER RESOURCE GROUP rg1 WITH (
    'cpu_core_limit' = '20'
);
```

ALTER SYSTEM

Description

Manages FE, BE, CN, Broker nodes, and metadata snapshots in a cluster.

> **NOTE**
>
> Only the `cluster_admin` role has the privilege to execute this SQL statement.

Syntax and parameters

FE

- Add a Follower FE.

```
ALTER SYSTEM ADD FOLLOWER "<fe_host>:<edit_log_port>"[, ...]
```

  You can check the status of the new Follower FE by executing `SHOW PROC '/frontends'\G`.

- Drop a Follower FE.

```
ALTER SYSTEM DROP FOLLOWER "<fe_host>:<edit_log_port>"[, ...]
```

- Add an Observer FE.

```
ALTER SYSTEM ADD OBSERVER "<fe_host>:<edit_log_port>"[, ...]
```

  You can check the status of the new Observer FE by executing `SHOW PROC '/frontends'\G`.

- Drop an Observer FE.

```
ALTER SYSTEM DROP OBSERVER "<fe_host>:<edit_log_port>"[, ...]
```

Parameter	Required	Description
fe_host	Yes	The host name or IP address of the FE instance. Use the value of configuration item `priority_networks` if your instance has multiple IP addresses.
edit_log_port	Yes	BDB JE communication port of the FE node. Default: `9010`.

BE

- Add a BE node.

  ```
  ALTER SYSTEM ADD BACKEND "<be_host>:<heartbeat_service_port>"[, ...]
  ```

  You can check the status of the new BE by executing SHOW BACKENDS.

- Drop a BE node.

  > **NOTE**
  >
  > You cannot drop the BE node that stores the tablets of single-replica tables.

  ```
  ALTER SYSTEM DROP BACKEND "<be_host>:<heartbeat_service_port>"[, ...]
  ```

- Decommission a BE node.

  ```
  ALTER SYSTEM DECOMMISSION BACKEND "<be_host>:
  <heartbeat_service_port>"[, ...]
  ```

  Unlike dropping a BE node, which is removing it forcibly from the cluster, decommissioning a BE means removing it safely. It is an asynchronous operation. When

a BE is decommissioned, the data on the BE is first migrated to other BEs, and then the BE is removed from the cluster. Data loading and query will not be affected during the data migration. You can check whether the operation is successful using SHOW BACKENDS. If the operation is successful, the decommissioned BE will not be returned. If the operation fails, the BE will still be online. You can manually cancel the operation using CANCEL DECOMMISSION.

Parameter	Required	Description
be_host	Yes	The host name or IP address of the BE instance. Use the value of configuration item `priority_networks` if your instance has multiple IP addresses.
heartbeat_service_port	Yes	BE heartbeat service port. BE uses this port to receive heartbeat from FE. Default: `9050`.

CN

- Add a CN node.

```
ALTER SYSTEM ADD COMPUTE NODE "<cn_host>:
<heartbeat_service_port>"[, ...]
```

You can check the status of the new CN by executing SHOW COMPUTE NODES.

- Drop a CN node.

```
ALTER SYSTEM DROP COMPUTE NODE "<cn_host>:
<heartbeat_service_port>"[, ...]
```

> **NOTE**
>
> You cannot decommission a CN node using the `ALTER SYSTEM DECOMMISSION` command.

Parameter	Required	Description
cn_host	Yes	The host name or IP address of the CN instance. Use the value of configuration item `priority_networks` if your instance has multiple IP addresses.
heartbeat_service_port	Yes	CN heartbeat service port. CN uses this port to receive heartbeat from FE. Default: `9050`.

Broker

- Add Broker nodes. You can use Broker nodes to load data from HDFS or cloud storage into StarRocks. For more information, see Load data from HDFS or cloud storage.

```
ALTER SYSTEM ADD BROKER <broker_name> "<broker_host>:
<broker_ipc_port>"[, ...]
```

  You can add multiple Broker nodes with one SQL. Each `<broker_host>:<broker_ipc_port>` pair represents one Broker node. And they share a common `broker_name`. You can check the status of the new Broker node by executing SHOW BROKER.

- Drop Broker nodes.

> **CAUTION**
>
> Dropping a Broker node terminates the tasks currently running on it.

- Drop one or multiple Broker nodes with the same `broker_name`.

```
ALTER SYSTEM DROP BROKER <broker_name> "<broker_host>:
<broker_ipc_port>"[, ...]
```

- Drop all Broker nodes with the same `broker_name`.

```
ALTER SYSTEM DROP ALL BROKER <broker_name>
```

Parameter	Required	Description
broker_name	Yes	The name of the Broker node(s). Multiple Broker nodes can use the same name.
broker_host	Yes	The host name or IP address of the Broker instance. Use the value of configuration item `priority_networks` if your instance has multiple IP addresses.
broker_ipc_port	Yes	The thrift server port on the Broker node. The Broker node uses it to receive requests from FE or BE. Default: `8000`.

Create image

Create an image file. An image file is a snapshot of the FE metadata.

```
ALTER SYSTEM CREATE IMAGE
```

Creating an image is an asynchronous operation on the Leader FE. You can check the start time and end time of the operation in the FE log file **fe.log**. A log like `triggering a new checkpoint manually...` indicates that the image creation has started, and a log like `finished save image...` indicates the image has been created.

Usage notes

- Adding and dropping FE, BE, CN, or Broker nodes are synchronous operations. You cannot cancel the node dropping operations.
- You cannot drop the FE node in a single-FE cluster.
- You cannot directly drop the Leader FE node in a multi-FE cluster. To drop it, you must first restart it. After StarRocks elects a new Leader FE, you can then drop the previous one.
- You cannot drop BE nodes if the number of the remained BE nodes is less than the number of data replicas. For example, if you have three BE nodes in your cluster and

you store your data in three replicas, you cannot drop any of the BE nodes. And if you have four BE nodes and three replicas, you can drop one BE node.

- The difference between dropping and decommissioning a BE node is that, when you drop a BE node, StarRocks removes it forcibly from the cluster and make up the dropped tablets after the removal, and when you decommission a BE node, StarRocks first migrates the tablets on the decommissioned BE node to others, and then removes the node.

Examples

Example 1: Add a Follower FE node.

```
ALTER SYSTEM ADD FOLLOWER "x.x.x.x:9010";
```

Example 2: Drop two Observer FE nodes simultaneously.

```
ALTER SYSTEM DROP OBSERVER "x.x.x.x:9010","x.x.x.x:9010";
```

Example 3: Add a BE node.

```
ALTER SYSTEM ADD BACKEND "x.x.x.x:9050";
```

Example 4: Drop two BE nodes simultaneously.

```
ALTER SYSTEM DROP BACKEND "x.x.x.x:9050", "x.x.x.x:9050";
```

Example 5: Decommission two BE nodes simultaneously.

```
ALTER SYSTEM DECOMMISSION BACKEND "x.x.x.x:9050", "x.x.x.x:9050";
```

Example 6: Add two Broker nodes with the same `broker_name` - `hdfs`.

```
ALTER SYSTEM ADD BROKER hdfs "x.x.x.x:8000", "x.x.x.x:8000";
```

Example 7: Drop two Broker nodes from `amazon_s3`.

```
ALTER SYSTEM DROP BROKER amazon_s3 "x.x.x.x:8000", "x.x.x.x:8000";
```

Example 8: Drop all Broker nodes in `amazon_s3`.

```
ALTER SYSTEM DROP ALL BROKER amazon_s3;
```

CANCEL DECOMMISSION

Description

This statement is used to undo a node decommission. (Administrator only!)

Syntax:

```
CANCEL DECOMMISSION BACKEND "<host>:<heartbeat_port>"[,"<host>:
<heartbeat_port>"...]
```

Examples

1. Cancel decommission of two nodes.

   ```
   CANCEL DECOMMISSION BACKEND "host1:port", "host2:port";
   ```

CREATE FILE

You can execute the CREATE FILE statement to create a file. After a file is created, the file is uploaded and persisted in StarRocks. In a database, only an admin user can create and delete files, and all users who have permission to access the database can use the files that belong to the database.

Basic concepts

File: refers to the file that is created and saved in StarRocks. After a file is created and stored in StarRocks, StarRocks assigns a unique ID to the file. You can find a file based on the database name, catalog, and file name.

Syntax

```
CREATE FILE "file_name" [IN database]
[properties]
```

Parameters

Parameter	Required	Description
file_name	Yes	The name of the file.
database	No	The database to which the file belongs. If you do not specify this parameter, this parameter defaults to the name of the database that you access in the current session.
properties	Yes	The properties of the file. The following table describes the configuration items of properties.

Configuration items of `properties`

Configuration item	Required	Description
url	Yes	The URL from which you can download the file. Only an unauthenticated HTTP URL is supported. After the file is stored in StarRocks, the URL is no longer needed.
catalog	Yes	The category to which the file belongs. You can specify a catalog based on your business requirements. However, in some situations, you must set this parameter to a specific catalog. For example, if you load data from Kafka, StarRocks searches for files in the catalog from the Kafka data source.
MD5	No	The message-digest algorithm that is used to check a file. If you specify this parameter, StarRocks checks the file after the file is downloaded.

Examples

- Create a file named **test.pem** under the category named kafka.

```
CREATE FILE "test.pem"
PROPERTIES
(
    "url" = "https://starrocks-public.oss-cn-
xxxx.aliyuncs.com/key/test.pem",
    "catalog" = "kafka"
);
```

- Create a file named **client.key** under the category named 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"
);
```

CREATE RESOURCE GROUP

Description

Creates a resource group.

For more information, see Resource group.

Syntax

```
CREATE RESOURCE GROUP resource_group_name
TO CLASSIFIER1, CLASSIFIER2, ...
WITH resource_limit
```

Parameters

- `resource_group_name`: The name of the resource group to be created.
- `CLASSIFIER`: Classifiers used to filter the queries on which resource limits are imposed. You must specify a classifier using `"key"="value"` pairs. You can set multiple classifiers for a resource group.

  Parameters for classifiers are listed below:

Parameter	Required	Description
user	No	The name of user.
role	No	The role of user.
query_type	No	The type of query. `SELECT` and `INSERT` (from v2.5) are supported. When INSERT tasks hit a resource group

Parameter	Required	Description
		with `query_type` as `insert`, the BE node reserves the specified CPU resources for the tasks.
source_ip	No	The CIDR block from which the query is initiated.
db	No	The database which the query accesses. It can be specified by strings separated by commas (,).

- `resource_limit`: Resource limits to be imposed on the resource group. You must specify resource limits using `"key"="value"` pairs. You can set multiple resource limits for a resource group.

  Parameters for resource limits are listed below:

Parameter	Required	Description
cpu_core_limit	No	The soft limit for the number of CPU cores that can be allocated to the resource group on the BE. In actual business scenarios, CPU cores that are allocated to the resource group proportionally scale based on the availability of CPU cores on the BE. Valid values: any non-zero positive integer.
mem_limit	No	The percentage of memory that can be used for queries in the total memory that is provided by the BE. Unit: %. Valid values: (0, 1).
concurrency_limit	No	The upper limit of concurrent queries in a resource group. It is used to avoid system overload caused by too many concurrent queries.

Parameter	Required	Description
big_query_cpu_second_limit	No	The upper time limit of CPU occupation for a big query. Concurrent queries add up the time. The unit is second.
big_query_scan_rows_limit	No	The upper limit of row counts that can be scanned by a big query.
big_query_mem_limit	No	The upper limit of memory usage of a big query. The unit is byte.
type	No	The type of resource group. Valid values: `short_query`: When queries from the `short_query` resource group are running, the BE node reserves the CPU cores defined in `short_query.cpu_core_limit`. CPU cores for all `normal` resource groups are limited to "the total CPU cores - `short_query.cpu_core_limit`". `normal`: When no query from the `short_query` resource group is running, the CPU core limit above is not imposed on the `normal` resource groups. Note that you can create only ONE `short_query` resource group in a cluster.

Example

Example 1: Creates the resource group `rg1` based on multiple classifiers.

```
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_core_limit' = '10',
      'mem_limit' = '20%',
      'type' = 'normal',
      'big_query_cpu_second_limit' = '100',
      'big_query_scan_rows_limit' = '100000',
      'big_query_mem_limit' = '1073741824'
);
```

DELETE SQLBLACKLIST

Description

Deletes an SQL regular expression from the SQL blacklist.

For more about SQL Blacklist, see Manage SQL Blacklist.

Syntax

```
DELETE SQLBLACKLIST <sql_index_number>
```

Parameter

`sql_index_number`: the index number of the SQL regular expression in the blacklist. Separate multiple index numbers with a comma (,) and a space. You can obtain the index number using SHOW SQLBLACKLIST.

Examples

```
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 |
+-------+--------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------+
```

DROP FILE

You can execute the DROP FILE statement to delete a file. When you use this statement to delete a file, the file is deleted both in frontend (FE) memory and in Berkeley DB Java Edition (BDBJE).

Syntax

```
DROP FILE "file_name" [FROM database]
[properties]
```

Parameters

Parameter	Required	Description
file_name	Yes	The name of the file.
database	No	The database to which a file belongs.
properties	Yes	The properties of the file. The following table describes the configuration items of properties.

Configuration items of `properties`

Configuration items	Required	Description
catalog	Yes	The category to which a file belongs.

Examples

Delete a file named **ca.pem**.

```
DROP FILE "ca.pem" properties("catalog" = "kafka");
```

DROP RESOURCE GROUP

Description

Drops the specified resource group.

Syntax

```
DROP RESOURCE GROUP <resource_group_name>
```

Parameters

Parameter	Description
resource_group_name	Name of the resource group to be dropped.

Example

Example 1: Drops the resource group `rg1`.

```
DROP RESOURCE GROUP rg1;
```

EXPLAIN

Description

Shows the logical or physical execution plans for a query statement.

Syntax

```
EXPLAIN <query>
```

Parameters

Parameter	Description
query	The query statement whose execution plan you want to view.

Return value

Returns the execution plan of the input query statement.

Return	Description
avgRowSize	The average size of the scanned data rows.
cardinality	The total number of data rows in the scanned table.
colocate	Whether the table is in colocate mode.
numNodes	The number of nodes to scan.

Return	Description
rollup	The Rollup materialized view that is used.
preaggregation	The status of preaggregation.
predicates	Predicates in the query.

Examples

```
mysql> EXPLAIN select count(*)
from store_sales,
    household_demographics,
    time_dim,
    store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 8
    and time_dim.t_minute >= 30
    and household_demographics.hd_dep_count = 5
    and store.s_store_name = 'ese'
order by count(*) limit 100;

+-----------------------------------------------------------------------
----------+
| Explain String
|
+-----------------------------------------------------------------------
----------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:<slot 11>
|
|   PARTITION: UNPARTITIONED
|
|   RESULT SINK
|
|   12:MERGING-EXCHANGE
|
|      limit: 100
```

```
|
|      tuple ids: 5
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 12
|
|     UNPARTITIONED
|
|
|
|   8:TOP-N
|
|   |  order by: <slot 11> ASC
|
|   |  offset: 0
|
|   |  limit: 100
|
|   |  tuple ids: 5
|
|   |
|
|   7:AGGREGATE (update finalize)
|
|   |  output: count(*)
|
|   |  group by:
|
|   |  tuple ids: 4
|
|   |
|
|   6:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  hash predicates:
```

```
|
|   |  colocate: false, reason: left hash join node can not do
colocate        |
|   |  equal join conjunct: `ss_store_sk` = `s_store_sk`
|
|   |  tuple ids: 0 2 1 3
|
|   |
|
|   |----11:EXCHANGE
|
|   |       tuple ids: 3
|
|   |
|
|   4:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  hash predicates:
|
|   |  colocate: false, reason: left hash join node can not do
colocate        |
|   |  equal join conjunct:
`ss_hdemo_sk`=`household_demographics`.`hd_demo_sk`|
|   |  tuple ids: 0 2 1
|
|   |
|
|   |----10:EXCHANGE
|
|   |       tuple ids: 1
|
|   |
|
|   2:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  hash predicates:
|
|   |  colocate: false, reason: table not in same group
|
|   |  equal join conjunct: `ss_sold_time_sk` =
`time_dim`.`t_time_sk`         |
|   |  tuple ids: 0 2
```

```
|
|   |
|
|   |----9:EXCHANGE
|
|   |       tuple ids: 2
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: store_sales
|
|      PREAGGREGATION: OFF. Reason: `ss_sold_time_sk` is value column
|
|      partitions=1/1
|
|      rollup: store_sales
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 0
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 11
|
|     UNPARTITIONED
```

```
|
|
|
|   5:OlapScanNode
|
|      TABLE: store
|
|      PREAGGREGATION: OFF. Reason: null
|
|      PREDICATES: `store`.`s_store_name` = 'ese'
|
|      partitions=1/1
|
|      rollup: store
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 3
|
|
|
| PLAN FRAGMENT 3
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 10
|
|     UNPARTITIONED
|
|
|
|   3:OlapScanNode
|
|      TABLE: household_demographics
```

```
|
|      PREAGGREGATION: OFF. Reason: null
|
|      PREDICATES: `household_demographics`.`hd_dep_count` = 5
|
|      partitions=1/1
|
|      rollup: household_demographics
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 1
|
|
|
| PLAN FRAGMENT 4
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 09
|
|     UNPARTITIONED
|
|
|
|   1:OlapScanNode
|
|      TABLE: time_dim
|
|      PREAGGREGATION: OFF. Reason: null
|
|      PREDICATES: `time_dim`.`t_hour` = 8, `time_dim`.`t_minute` >=
30        |
|      partitions=1/1
```

```
|
|      rollup: time_dim
|
|      tabletRatio=0/0
|
|      tabletList=
|
|      cardinality=-1
|
|      avgRowSize=0.0
|
|      numNodes=0
|
|      tuple ids: 2
|
+-----------------------------------------------------------------------
----------+
128 rows in set (0.02 sec)
```

INSTALL PLUGIN

Description

This statement is used to install a plugin,

Syntax:

```
INSTALL PLUGIN FROM [source] [PROPERTIES ("key"="value", ...)]
```

3 types of sources are supported:

```
1. An absolute path that directs to a zip file
2. An absolute path that directs to a plugin directory
3. A http or https download link that directs to a zip file
```

PROPERTIES supports setting some configurations of plugins, such as setting the md5sum value of the zip file, etc.

Examples

1. Install a plugin from local zip file:

   ```
   INSTALL PLUGIN FROM "/home/users/starrocks/auditdemo.zip";
   ```

2. Install a plugin from local inpath:

   ```
   INSTALL PLUGIN FROM "/home/users/starrocks/auditdemo/";
   ```

3. Download and install a plugin:

```
INSTALL PLUGIN FROM "http://mywebsite.com/plugin.zip";
```

4. Download and install a plugin. Meanwhile, set the md5sum value of the zip file:

```
INSTALL PLUGIN FROM "http://mywebsite.com/plugin.zip"
PROPERTIES("md5sum" = "73877f6029216f4314d712086a146570");
```

KILL

Description

Terminates a connection or a query currently being performed by threads executing within StarRocks.

Syntax

```
KILL [ CONNECTION | QUERY ] <processlist_id>
```

Parameters

Parameter	Description
Modifier: • CONNECTION • QUERY	• With a `CONNECTION` modifier, the KILL statement terminates the connection associated with the given `processlist_id`, after terminating any statement the connection is executing. • With a `QUERY` modifier, the KILL statement terminates the statement the connection is currently executing, but leaves the connection itself intact. • If no modifier is present, the default is `CONNECTION`.
processlist_id	The ID of the thread you want to terminate. You can get the IDs of the threads that are being executed using SHOW PROCESSLIST.

Examples

```
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)
```

SET

Description

Sets the specified system variables or user-defined variables for StarRocks. You can view the system variables of StarRocks using SHOW VARIABLES. For details about system variables, see System Variables. For details about user-defined variables, see User-defined variables.

Syntax

```
SET [ GLOBAL | SESSION ] <variable_name> = <value> [, <variable_name>
= <value>] ...
```

Parameters

Parameter	Description
Modifier: • GLOBAL • SESSION	• With a `GLOBAL` modifier, the statement sets the variables globally. • With a `SESSION` modifier, the statement sets the variables within the session. `LOCAL` is a synonym for `SESSION`. • If no modifier is present, the default is `SESSION`. For details about global and session variables, see System Variables. **NOTE** Only users with the ADMIN privilege can set the variables globally.
variable_name	The name of the variable.
value	The value of the variable.

Examples

Example 1: Set the `time_zone` to `Asia/Shanghai` within the session.

```
mysql> SET time_zone = "Asia/Shanghai";
Query OK, 0 rows affected (0.00 sec)
```

Example 2: Set the `exec_mem_limit` to `2147483648` globally.

```
mysql> SET GLOBAL exec_mem_limit = 2147483648;
Query OK, 0 rows affected (0.00 sec)
```

Example 3: Set multiple global variables. The `GLOBAL` keyword needs to be preposed for each variable.

```
mysql> SET
       GLOBAL exec_mem_limit = 2147483648,
       GLOBAL time_zone = "Asia/Shanghai";
Query OK, 0 rows affected (0.00 sec)
```

SHOW BACKENDS

Description

Shows the information of all BE nodes in the cluster.

Syntax

```
SHOW BACKENDS
```

Return

```
+-----------+-----+---------------+--------+---------+----------+---
------------+---------------+-------+--------------------+---------
-------------+-----------+-----------------+--------------+-------
--------+---------+---------------+--------+---------+--------+----
---------------+-------------+-----------+
| BackendId | IP  | HeartbeatPort | BePort | HttpPort | BrpcPort |
LastStartTime | LastHeartbeat | Alive | SystemDecommissioned |
ClusterDecommissioned | TabletNum | DataUsedCapacity | AvailCapacity
| TotalCapacity | UsedPct | MaxDiskUsedPct | ErrMsg | Version  |
Status | DataTotalCapacity | DataUsedPct | CpuCores |
+-----------+-----+---------------+--------+---------+----------+---
------------+---------------+-------+--------------------+---------
-------------+-----------+-----------------+--------------+-------
--------+---------+---------------+--------+---------+--------+----
---------------+-------------+-----------+
```

Return	Description
BackendId	The ID of the BE node.
IP	The IP address of the BE node.

Return	Description
HeartbeatPort	The heartbeat port of the BE node. It is used to receive heartbeats from the FE node.
BePort	Thrift server port of the BE node. It is used to receive requests from the FE node.
HttpPort	HTTP server port of the BE node. It is used to access the BE node via web page.
BrpcPort	bRPC port of the BE node. It is used for communication across BE nodes.
LastStartTime	The last time when the BE node is started.
LastHeartbeat	The last time when the FE node sends a heartbeat and the BE responds.
Alive	Whether the BE node is alive or not. • `true`: the BE node is alive. • `false`: the BE node is not alive.
SystemDecommissioned	Whether the BE node is decommissioned or not. Before it is decommissioned, the BE node migrates the data to other BE nodes. Data loading and queries are not affected during migration. • `true`: the cluster has marked the BE node as decommissioned. The data could be successfully migrated or migrating. • `false`: the BE node is running.
ClusterDecommissioned	This parameter is used for system compatibility.
TabletNum	The number of tablets on the BE node.

Return	Description
DataUsedCapacity	The storage capacity occupied by the data file.
AvailCapacity	Available storage capacity in the BE node.
TotalCapacity	The total storage capacity. It is equivalent to `DataUsedCapacity` + `AvailCapacity` + Storage capacity occupied by the non-data files.
UsedPct	The percentage of the used storage capacity.
MaxDiskUsedPct	If a BE node has multiple directories to store data, this parameter shows the maximum percentage of the used storage capacity among all directories.
ErrMsg	The error message returned when the BE node fails to receive the heartbeat from the FE node.
Version	The StarRocks version of cluster.
Status	The last time when the BE node reports the tablet number to the FE node. Displayed in JSON format.
DataTotalCapacity	It is equivalent to `DataUsedCapacity` + `AvailCapacity`. It indicates the sum of the storage capacity occupied by the data file and the available storage capacity in the BE node.
DataUsedPct	It is equivalent to `DataUsedCapacity`/`DataTotalCapacity`. It indicates the proportion of the storage capacity occupied by the data file to the sum of the data-occupied storage capacity and the available storage capacity.
CpuCores	The number of CPU cores in the BE node.

Example

View the information of all BE nodes in the 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 BROKER

Description

This statement is used to view broker that currently exists.

Syntax:

```
SHOW BROKER
```

Note:

1. LastStartTime represents the latest BE start-up time.
2. LastHeartbeat represents the latest heartbeat.
3. Alive indicates whether the node survives.
4. ErrMsg is used to display error messages when the heartbeat fails.

SHOW COMPUTE NODES

Description

View all compute nodes in your StarRocks cluster.

Syntax

```
SHOW COMPUTE NODES
```

Output

The following table describes the parameters returned by this statement.

Parameter	Description
LastStartTime	The last time at which the compute node starts.
LastHeartbeat	The last time at which the compute node sends a heartbeat.
Alive	Whether the compute node is available or not.
SystemDecommissioned	If the value of the parameter is `true`, the compute node is removed from your StarRocks cluster. Before removal, the data in the compute node is cloned.
ErrMsg	The error message if the compute node fails to send a heartbeat.
Status	The state of the compute node, which displays in the JSON format. Currently, you can only see the last time at which the compute node sends its state.

SHOW FILE

You can execute the SHOW FILE statement to view the information about a file stored in a database.

Syntax

```
SHOW FILE [FROM database]
```

The file information returned by this statement is as follows:

- `FileId`: the globally unique ID of the file.
- `DbName`: the database to which the file belongs.
- `Catalog`: the category to which the file belongs.
- `FileName`: the name of the file.
- `FileSize`: the size of the file. The unit is bytes.
- `MD5`: the message-digest algorithm that is used to check the file.

Examples

View the file stored in `my_database`.

```
SHOW FILE FROM my_database;
```

SHOW FRONTENDS

Description

This statement is used to view FE nodes.

Syntax:

```
SHOW FRONTENDS
```

Note:

1. Name represents the name of FE node in BDBJE.
2. Join being true means this node has already joined the cluster. But it does not mean that the node is still in the cluster, as it may be missing.
3. Alive indicates whether the node survives.
4. ReplayedJournalId represents the maximum metadata log ID that the node has currently replayed.
5. LastHeartbeat is the latest heartbeat.
6. IsHelper indicates whether the node is a helper node from BDBJE.
7. ErrMsg is used to display error messages when the heartbeat fails.

SHOW FULL COLUMNS

Description

This statement is used to show content in columns from specified tables.

Syntax:

```
SHOW FULL COLUMNS FROM <tbl_name>
```

Examples

1. View column content from specified tables.

   ```
   SHOW FULL COLUMNS FROM tbl;
   ```

SHOW INDEX

Description

This statement is used to show information related to index in a table. It currently only supports bitmap index.

Syntax:

```
SHOW INDEX[ES] FROM [db_name.]table_name [FROM database]
Or
SHOW KEY[S] FROM [db_name.]table_name [FROM database]
```

Examples

1. Show all indexes under the specified table_name:

```
SHOW INDEX FROM example_db.table_name;
```

SHOW PLUGINS

Description

This statement is used to view the installed plugins.

Syntax:

```
SHOW PLUGINS
```

This command will display all built-in plugins and custom plugins.

Examples

1. Show the installed plugins:

   ```
   SHOW PLUGINS;
   ```

SHOW PROC

Description

Shows certain indicators of the StarRocks cluster.

Syntax

```
SHOW PROC { '/auth' | '/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' }
```

Parameters

Parameter	Description
'/auth'	Shows the user privilege and authentication information in the cluster.
'/backends'	Shows the information of BE nodes in the cluster.
'/compute_nodes'	Shows the information of CN nodes in the cluster.
'/dbs'	Shows the information of databases in the cluster.
'/jobs'	Shows the information of jobs in the cluster.
'/statistic'	Shows the statistics of each database in the cluster.

Parameter	Description
'/tasks'	Shows the total number of all generic tasks and the failed tasks in the cluster.
'/frontends'	Shows the information of FE nodes in the cluster.
'/brokers'	Shows the information of Broker nodes in the cluster.
'/resources'	Shows the information of resources in the cluster.
'/load_error_hub'	Shows the configuration of the cluster's Load Error Hub, which is used to manage error messages of loading jobs.
'/transactions'	Shows the information of transactions in the cluster.
'/monitor'	Shows the monitoring information in the cluster.
'/current_queries'	Shows the information of running queries on the current FE node.
'/current_backend_instances'	Shows the BE nodes that are processing requests in the cluster.
'/cluster_balance'	Shows the load balance information in the cluster.
'/routine_loads'	Shows the information of Routine Load in the cluster.
'/colocation_group'	Shows the information of Colocate Join groups in the cluster.
'/catalog'	Shows the information of catalogs in the cluster.

Examples

Example 1: Shows the user privilege and authentication information in the cluster.

```
mysql> SHOW PROC '/auth';
+-----------------------------------+----------+-----------+------
------------+-------------------------------------------------------------
--------------------+-----------------------------------------------------
---------------------------------------------------------------------------
-----------------------------------------------------------------------+-
-----------+--------------+
| UserIdentity                          | Password | AuthPlugin |
UserForAuthPlugin | GlobalPrivs
| DatabasePrivs
| TablePrivs | ResourcePrivs |
+-----------------------------------+----------+-----------+------
------------+-------------------------------------------------------------
--------------------+-----------------------------------------------------
---------------------------------------------------------------------------
-----------------------------------------------------------------------+-
-----------+--------------+
| 'root'@'%'                            | Yes      | NULL       | NULL
| Node_priv Admin_priv  (false)
| NULL
| NULL       | NULL          |
| 'default_cluster:johndoe'@'%'       | Yes      | NULL       | NULL
| Admin_priv Select_priv Load_priv Alter_priv Create_priv Drop_priv
(false) | information_schema: Select_priv  (false)
| NULL       | NULL          |
| 'default_cluster:katherine'@'%'     | Yes      | NULL       | NULL
|  (false)
| information_schema: Select_priv  (false)
| NULL       | NULL          |
+-----------------------------------+----------+-----------+------
------------+-------------------------------------------------------------
--------------------+-----------------------------------------------------
---------------------------------------------------------------------------
-----------------------------------------------------------------------+-
-----------+--------------+
```

Return	Description
UserIdentity	User identity, shown as `username@'userhost'` or `username@['domain']`.
Password	If the user connects to the cluster using password.

Return	Description
AuthPlugin	Authentication plugin the user adopted, including `mysql_native_password` and `authentication_ldap_simple`.
UserForAuthPlugin	Distinguished Name (DN) of the user if they connect to the cluster via LDAP.
GlobalPrivs	Global privileges owned by the user.
DatabasePrivs	Database-level privileges owned by the user.
TablePrivs	Table-level privileges owned by the user.
ResourcePrivs	Resource-level privileges owned by the user.

Example 2: Shows the information of BE nodes in the cluster.

```
mysql> SHOW PROC '/backends';
+-----------+---------------+---------------+--------+----------+----
-----+-------------------+-------------------+-------+----------
-----------+----------------------+-----------+-----------------+-
-------------+---------------+---------+---------------+--------+--
----------------+-----------------------------------------------------
-----+------------------+------------+----------+
| BackendId | IP            | HeartbeatPort | BePort | HttpPort |
BrpcPort | LastStartTime       | LastHeartbeat       | Alive |
SystemDecommissioned | ClusterDecommissioned | TabletNum |
DataUsedCapacity | AvailCapacity | TotalCapacity | UsedPct |
MaxDiskUsedPct | ErrMsg | Version           | Status
| DataTotalCapacity | DataUsedPct | CpuCores |
+-----------+---------------+---------------+--------+----------+----
-----+-------------------+-------------------+-------+----------
-----------+----------------------+-----------+-----------------+-
-------------+---------------+---------+---------------+--------+--
----------------+-----------------------------------------------------
-----+------------------+------------+----------+
| 56038515  | xxx.xx.xx.xxx | 9052          | 9060   | 8097     |
8059     | 2022-10-09 19:57:57 | 2022-10-10 16:23:24 | true  | false
| false                | 47015     | 67.952 GB        | 1.089 TB
```

```
| 1.968 TB      | 44.69 % | 44.69 %        |        | UNKNOWN-
de75d4fbb | {"lastSuccessReportTabletsTime":"2022-10-10 16:22:44"} |
1.155 TB          | 5.74 %      | 104      |
+-----------+---------------+---------------+--------+----------+----
------+-------------------+-------------------+-------+----------
-----------+---------------------+----------+-----------------+-
-------------+--------------+---------+--------------+--------+--
---------------+-----------------------------------------------------
-----+------------------+------------+----------+
```

Return	Description
BackendId	ID of the BE node.
IP	IP address of the BE node.
HeartbeatPort	Heartbeat service port of the BE node.
BePort	Thrift Server port of the BE node.
HttpPort	HTTP Server port of the BE node.
BrpcPort	bRPC port of the BE node.
LastStartTime	The last time when the BE node was started.
LastHeartbeat	The last time when the BE node received a heartbeat.
Alive	If the BE node is alive.
SystemDecommissioned	If the BE node is decommissioned.
ClusterDecommissioned	If the BE node is decommissioned within the cluster.
TabletNum	Number of tablets in the BE node.
DataUsedCapacity	Storage capacity that is used for data in the BE node.
AvailCapacity	Available storage capacity in the BE node.

Return	Description
TotalCapacity	Total storage capacity in the BE node.
UsedPct	Percentage at which the storage capacity is used in the BE node.
MaxDiskUsedPct	The maximum percentage at which the storage capacity is used in the BE node.
ErrMsg	Error messages in the BE node.
Version	StarRocks version of the BE node.
Status	Status information of the BE node, including the last time when the BE node reported tablets.
DataTotalCapacity	Total of used and available data storage capacity. The sum of `DataUsedCapacity` and `AvailCapacity`.
DataUsedPct	Percentage at which the data storage takes up the total data capacity (DataUsedCapacity/DataTotalCapacity).
CpuCores	Number of CPU cores in the BE node.

Example 3: Shows the information of databases in the cluster.

```
mysql> SHOW PROC '/dbs';
+---------+----------------------+----------+---------------+-----
--------------------+--------------------+
| DbId    | DbName               | TableNum | Quota         |
LastConsistencyCheckTime | ReplicaQuota        |
+---------+----------------------+----------+---------------+-----
--------------------+--------------------+
| 1       | information_schema   | 22       | 8388608.000 TB | NULL
| 9223372036854775807 |
| 840997  | tpcds_100g           | 25       | 1024.000 GB    | NULL
| 1073741824          |
| 1275196 | _statistics_         | 3        | 8388608.000 TB |
2022-09-06 23:00:58      | 9223372036854775807 |
```

```
| 1286207 | tpcds_n                | 24       | 8388608.000 TB | NULL
| 9223372036854775807 |
| 1381289 | test                   | 6        | 8388608.000 TB |
2022-01-14 23:10:18      | 9223372036854775807 |
| 6186781 | test_stddev            | 1        | 8388608.000 TB |
2022-09-06 23:00:58      | 9223372036854775807 |
+---------+------------------------+----------+----------------+-----
--------------------+---------------------+
```

Return	Description
DbId	Database ID.
DbName	Database name.
TableNum	Number of tables in the database.
Quota	Storage quota of the database.
LastConsistencyCheckTime	The last time when consistency check is executed.
ReplicaQuota	Data replica quota of the database.

Example 4: Shows the information of jobs in the cluster.

```
mysql> SHOW PROC '/jobs';
+-------+---------------------------------------+
| DbId  | DbName                                |
+-------+---------------------------------------+
| 10005 | default_cluster:_statistics_          |
| 0     | default_cluster:information_schema    |
| 12711 | default_cluster:starrocks_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)
```

Return	Description
DbId	Database ID.
DbName	Database name.
JobType	Job type.
Pending	Number of jobs that are pending.
Running	Number of jobs that are running.
Finished	Number of jobs that are finished.
Cancelled	Number of jobs that are cancelled.
Total	Total number of jobs.

Example 5: Shows the statistics of each database in the 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]  |
+------------------+--------------------+---------------+----------
---------+
```

Return	Description
DbId	Database ID.
DbName	Database name.
TableNum	Number of tables in the database.

Return	Description
PartitionNum	Number of partitions in the database.
IndexNum	Number of indexes in the database.
TabletNum	Number of tablets in the database.
ReplicaNum	Number of replicas in the database.
UnhealthyTabletNum	Number of unfinished (unhealthy) tablets in the database during data redistribution.
InconsistentTabletNum	Number of inconsistent tablets in the database.
CloningTabletNum	Number of tablets that are being cloned in the database.
ErrorStateTabletNum	In a Primary Key type table, the number of tablets in Error state.
ErrorStateTablets	In a Primary Key type table, the IDs of the tablets in Error state.

Example 6: Shows the total number of all generic tasks and the failed tasks in the 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         |
+-------------------------+-----------+-----------+
```

Return	Description
TaskType	Task type.
FailedNum	Number of failed tasks.
TotalNum	Total number of tasks.

Example 7: Shows the information of FE nodes in the 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     |
+--------------------------------+--------------+-------------+---
-------+----------+---------+---------+-----------+-------+-------
+------------------+--------------+----------+--------------+-----
------+---------+
```

Return	Description
Name	FE node name.
IP	IP address of the FE node.
EditLogPort	Port for communication between FE nodes.
HttpPort	HTTP Server port of the FE node.
QueryPort	MySQL Server port of the FE node.
RpcPort	RPC port of the FE node.
Role	Role of the FE node (Leader, Follower, or Observer).
ClusterId	Cluster ID.
Join	If the FE node has joined a cluster.
Alive	If the FE node is alive.
ReplayedJournalId	The largest metadata ID that the FE node has replayed.
LastHeartbeat	The last time when the FE node sent a heartbeat.
IsHelper	If the FE node is the BDBJE helper node.
ErrMsg	Error messages in the FE node.
StartTime	Time when the FE node is started.
Version	StarRocks version of the FE node.

Example 8: Shows the information of Broker nodes in the 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 |        |
+-------------+--------------+------+-------+--------------+-------
--------------+--------+
```

Return	Description
Name	Broker node name.
IP	IP address of the broker node.
Port	Thrift Server port of the broker node. The port is used to receive requests.
Alive	If the broker node is alive.
LastStartTime	The last time when the broker node was started.
LastUpdateTime	The last time when the broker node was updated.
ErrMsg	Error message in the broker node.

Example 9: Shows the information of resources in the 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  |
+---------------------------+--------------+----------------------+-----
------------------------+
```

Return	Description
Name	Resource name.
ResourceType	Resource type.
Key	Resource key.
Value	Resource value.

Example 10: Shows the information of transactions in the cluster.

```
mysql> SHOW PROC '/transactions';
+-------+---------------------------------------+
| DbId  | DbName                                |
+-------+---------------------------------------+
| 10005 | default_cluster:_statistics_          |
| 12711 | default_cluster:starrocks_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)
```

Return	Description
DbId	Database ID.
DbName	Database name.
State	The state of the transaction.
Number	Number of transactions.

Example 11: Shows the monitoring information in the cluster.

```
mysql> SHOW PROC '/monitor';
+------+------+
| Name | Info |
+------+------+
| jvm  |      |
+------+------+
```

Return	Description
Name	JVM name.
Info	JVM information.

Example 12: Shows the load balance information in the cluster.

```
mysql> SHOW PROC '/cluster_balance';
+------------------+--------+
| Item             | Number |
+------------------+--------+
| cluster_load_stat | 1      |
| working_slots     | 3      |
| sched_stat        | 1      |
| priority_repair   | 0      |
| pending_tablets   | 2001   |
| running_tablets   | 0      |
| history_tablets   | 1000   |
+------------------+--------+
```

Return	Description
Item	Sub-command item in `cluster_balance`. • cluster_load_stat: The current load status of the cluster. • working_slots: The number of currently available working slots. • sched_stat: The current status of the scheduler. • priority_repair: The number of Tablet repair tasks that are prioritized. • pending_tablets: The number of Tablets waiting to be processed. • running_tablets: The number of Tablets currently being repaired. • history_tablets: The total number of Tablets repaired historically.
Number	Number of each sub-command in `cluster_balance`.

Example 13: Shows the information of Colocate Join groups in the 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     |
+----------------+----------------------------+--------------------
--------------------------------------------------------------------
-----------------------------------------------------+----------
-+---------------+------------+----------+
```

Return	Description
GroupId	Colocate Join Group ID.
GroupName	Colocate Join Group name.
TableIds	IDs of tables in the Colocate Join Group.
BucketsNum	Buckets in the Colocate Join Group.
ReplicationNum	Replications in the Colocate Join Group.
DistCols	Distribution column of the Colocate Join Group.
IsStable	If the Colocate Join Group is stable.

Example 14: Shows the information of catalogs in the 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     |
```

```
+-------------------------------------------------------------------+-------+--------------------+
```

Return	Description
Catalog	Catalog name.
Type	Catalog type.
Comment	Comments for the catalog.

SHOW PROCESSLIST

Description

Lists the operations currently being performed by threads executing within the server. The current version of StarRocks only supports listing queries.

Syntax

```
SHOW [FULL] PROCESSLIST
```

Parameter

Parameter	Required	Description
FULL	No	If you specify this parameter, the complete SQL statement will be displayed. Otherwise, only the first 100 characters of the statement are displayed.

Return

Return	Description
Id	Connection ID.
User	The name of the user who runs the operation.
Host	The hostname of the client which runs the operation.

Return	Description
Db	The name of the database where the operation is executed.
Command	The type of the command.
ConnectionStartTime	Time when the connection starts.
Time	The time (in second) since the operation has entered the current state.
State	The state of the operation.
Info	The command that the operation is executing.

Usage note

If the current user is `root`, this statement lists the operations of all users in the cluster. Otherwise, only operations of the current user are listed.

Example

Example 1: lists the operations state via the user `root`.

```
SHOW PROCESSLIST;
+------+------+--------------------+-------+---------+-------------
-------+------+-------+-----------------+
| Id   | User | Host               | Db    | Command |
ConnectionStartTime | Time | State | Info            |
+------+------+--------------------+-------+---------+-------------
-------+------+-------+-----------------+
|  0   | root | x.x.x.x:xxxx       | tpcds | Query   | 2022-10-09
19:58:25 |    0 | OK    | SHOW PROCESSLIST |
+------+------+--------------------+-------+---------+-------------
-------+------+-------+-----------------+
```

SHOW RESOURCE GROUP

Description

Shows the information of resource groups.

Syntax

```
SHOW { RESOURCE GROUPS [ALL] | RESOURCE GROUP <resource_group_name> }
```

Parameters

Parameter	Description
RESOURCE GROUPS	Show the resource groups that match the current user.
ALL	Specify this parameter to show all resource groups in the cluster.
RESOURCE GROUP	Show the specified resource group.
resource_group_name	Name of the resource group to be shown.

Return

Return	Description
name	Name of the resource group.
id	ID of the resource group.

Return	Description
cpu_core_limit	CPU core limit of the resource group.
mem_limit	Memory limit of the resource group.
big_query_cpu_second_limit	Big query upper time limit of the resource group.
big_query_scan_rows_limit	Big query scan row limit of the resource group.
big_query_mem_limit	Big query memory limit of the resource group.
concurrency_limit	Concurrency limit of the resource group.
type	Type of resource group.
classifiers	Classifiers that are associated with the resource group. `id` is the ID of the classifier, and `weight` is the degree of matching.

Example

Example 1: Shows all resource groups in the cluster.

```
mysql> SHOW RESOURCE GROUPS ALL;
+-------+--------+---------------+----------+--------------------
------+-------------------------+--------------------+-----------
-------+--------+-----------------------------------------------------
-----------------------------------------------------------+
| name  | id     | cpu_core_limit | mem_limit |
big_query_cpu_second_limit | big_query_scan_rows_limit |
big_query_mem_limit | concurrency_limit | type   | classifiers
|
+-------+--------+---------------+----------+--------------------
------+-------------------------+--------------------+-----------
-------+--------+-----------------------------------------------------
-----------------------------------------------------------+
| rg1   | 625126 | 10             | 20.0%     | 100
| 100000                   | 1073741824          | null
```

```
| NORMAL | (id=625127, weight=4.459375, user=rg1_user1,
role=rg1_role1, query_type in (SELECT), source_ip=172.26.xxx.xx/24) |
| rg1   | 625126 | 10             | 20.0%     | 100
| 100000                    | 1073741824          | null
| NORMAL | (id=625128, weight=3.459375, user=rg1_user2, query_type in
(SELECT), source_ip=172.26.xxx.xx/24)                 |
| rg1   | 625126 | 10             | 20.0%     | 100
| 100000                    | 1073741824          | null
| NORMAL | (id=625129, weight=2.359375, user=rg1_user3,
source_ip=172.26.xxx.xx/24)                                       |
| rg1   | 625126 | 10             | 20.0%     | 100
| 100000                    | 1073741824          | null
| NORMAL | (id=625130, weight=1.0, user=rg1_user4)
|
| rg1   | 625126 | 10             | 20.0%     | 100
| 100000                    | 1073741824          | null
| NORMAL | (id=625131, weight=10.0, db='db1')
|
+-------+--------+----------------+-----------+---------------------
------+--------------------------+--------------------+------------
-------+--------+-----------------------------------------------------
-------------------------------------------------------------+
```

SHOW SQLBLACKLIST

Description

Shows the SQL regular expressions in the SQL blacklist.

For more about SQL Blacklist, see Manage SQL Blacklist.

Syntax

```
SHOW SQLBLACKLIST
```

Return value

Return	Description
Index	The index number of the SQL regular expression that has been added to the blacklist.
Forbidden SQL	The SQL regular expression that has been added to the blacklist.

Examples

```
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 .+
|
+-------+-------------------------------------------------------------
-------------------------------------------------------------------
-------------------------------------------------------------------
-------------------------------------------------------------------
----------------------------+
```

SHOW TABLE STATUS

Description

This statement is used to view some of the information in Table.

Syntax:

```
SHOW TABLE STATUS
[FROM db] [LIKE "pattern"]
```

> Note
>
> This statement is mainly compatible with MySQL syntax. At present, it only shows a few information, such as Comment.

Examples

1. View all the information of tables under the current database.

   ```
   SHOW TABLE STATUS;
   ```

2. View all the information of tables whose names contain example and who are under specified databases.

   ```
   SHOW TABLE STATUS FROM db LIKE "%example%";
   ```

SHOW VARIABLES

Description

Shows the system variables of StarRocks. For details about system variables, see System Variables.

Syntax

```
SHOW [ GLOBAL | SESSION ] VARIABLES
    [ LIKE <pattern> | WHERE <expr> ]
```

Parameters

Parameter	Description
Modifier: • GLOBAL • SESSION	• With a `GLOBAL` modifier, the statement displays global system variable values. These are the values used to initialize the corresponding session variables for new connections to StarRocks. If a variable has no global value, no value is displayed. • With a `SESSION` modifier, the statement displays the system variable values that are in effect for the current connection. If a variable has no session value, the global value is displayed. `LOCAL` is a synonym for `SESSION`. • If no modifier is present, the default is `SESSION`.
pattern	The pattern used to match the variable by the variable name with the LIKE clause. You can use the % wildcard in this parameter.

Parameter	Description
expr	The expression used to match the variable by the variable name `variable_name` or the variable value `value` with the WHERE clause.

Return value

Return	Description
Variable_name	The name of the variable.
Value	The value of the variable.

Examples

Example 1: Show a variable by exactly matching the variable name with the LIKE clause.

```
mysql> SHOW VARIABLES LIKE 'wait_timeout';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.01 sec)
```

Example 2: Show variables by approximately matching the variable name with the LIKE clause and the wildcard (%).

```
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)
```

Example 3: Show a variable by exactly matching the variable name with the WHERE clause.

```
mysql> show variables where variable_name = 'wait_timeout';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.17 sec)
```

Example 4: Show a variable by exactly matching the value of the variable with the WHERE clause.

```
mysql> show variables where value = '28800';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.70 sec)
```

UNINSTALL PLUGIN

Description

This statement is used to uninstall a plugin.

Syntax:

```
UNINSTALL PLUGIN <plugin_name>
```

plugin_name can be viewed through SHOW PLUGINS command

Only non-builtin plugins can be uninstalled.

Examples

1. Uninstall a plugin:

   ```
   UNINSTALL PLUGIN auditdemo;
   ```

ALTER DATABASE

Description

Configures the properties of the specified database. (Administrator only)

Syntax

1. Set database data quota in B/K/KB/M/MB/G/GB/T/TB/P/PB.

```
ALTER DATABASE <db_name> SET DATA QUOTA <quota>;
```

2. Rename a database.

```
ALTER DATABASE <db_name> RENAME <new_db_name>;
```

3. Set database replica quota.

```
ALTER DATABASE <db_name> SET REPLICA QUOTA <quota>;
```

Note:

```
- After renaming the database, use REVOKE and GRANT commands to
modify the corresponding user permission if necessary.
- The database's default data quota and the default replica quota are
2^63-1.
```

Examples

1. Set data quota for a database.

```
ALTER DATABASE example_db SET DATA QUOTA 109951162777760B;
-- The above unit is bytes, equivalent to the following
statement.
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. Rename the database `example_db` as `example_db2`.

```
ALTER DATABASE example_db RENAME example_db2;
```

3. Set database replica quota.

```
ALTER DATABASE example_db SET REPLICA QUOTA 102400;
```

References

- CREATE DATABASE
- USE
- SHOW DATABASES
- DESC
- DROP DATABASE

ALTER MATERIALIZED VIEW

Description

This SQL statement can：

- Alter the name of an asynchronous materialized view.
- Alter the refresh strategy of an asynchronous materialized view.
- Alter the status of an asynchronous materialized view to active or inactive.
- Alter the properties of an asynchronous materialized view.

  You can use this SQL statement to alter the following properties:

  - `partition_ttl_number`
  - `partition_refresh_number`
  - `resource_group`
  - `auto_refresh_partitions_limit`
  - `excluded_trigger_tables`
  - `mv_rewrite_staleness_second`
  - `unique_constraints`
  - `foreign_key_constraints`
  - `colocate_with`
  - All session variable-related properties. For information on session variables, see System variables。

Syntax

```
ALTER MATERIALIZED VIEW [db_name.]<mv_name>
    { RENAME [db_name.]<new_mv_name>
    | REFRESH <new_refresh_scheme_desc>
```

```
    | ACTIVE | INACTIVE
    | SET ( "<key>" = "<value>"[,...]) }
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
mv_name	yes	The name of the materialized view to alter.
new_refresh_scheme_desc	no	New async refresh strategy, see SQL Reference - CREATE MATERIALIZED VIEW - Parameters for details.
new_mv_name	no	New name for the materialized view.
ACTIVE	no	Set the status of the materialized view to active. StarRocks automatically sets a materialized view to inactive if any of its base tables is changed, for example, dropped and re-created, to prevent the situation that original metadata mismatches the changed base table. Inactive materialized views cannot be used for query acceleration or query rewrite. You can use this SQL to activate the materialized view after changing the base tables.
INACTIVE	no	Set the status of the materialized view to inactive. An inactive asynchronous materialized view cannot be refreshed. But you can still query it as a table.
key	no	The name of the property to alter, see SQL Reference - CREATE MATERIALIZED VIEW - Parameters for details.

Parameter	Required	Description
		NOTE If you want to alter a session variable-related property of the materialized view, you must add a `session.` prefix to the property, for example, `session.query_timeout`. You do not need to specify the prefix for non-session properties, for example, `mv_rewrite_staleness_second`.
value	no	The value of the property to alter.

Example

Example 1: Alter the name of the materialized view

```
ALTER MATERIALIZED VIEW lo_mv1 RENAME lo_mv1_new_name;
```

Example 2: Alter the refresh interval of the materialized view

```
ALTER MATERIALIZED VIEW lo_mv2 REFRESH ASYNC EVERY(INTERVAL 1 DAY);
```

Example 3: Alter the materialized view's properties.

```
-- Change mv1's query_timeout to 40000 seconds.
ALTER MATERIALIZED VIEW mv1 SET ("session.query_timeout" = "40000");
-- Change mv1's mv_rewrite_staleness_second to 600 seconds.
ALTER MATERIALIZED VIEW mv1 SET ("mv_rewrite_staleness_second" =
"600");
```

Example 4: Alter the materialized view's status to active.

```
ALTER MATERIALIZED VIEW order_mv ACTIVE;
```

ALTER RESOURCE

Description

You can use the ALTER RESOURCE statement to modify the properties of a resource.

Syntax

```
ALTER RESOURCE 'resource_name' SET PROPERTIES ("key"="value", ...)
```

Parameters

- `resource_name`: the name of the resource to be modified.
- `PROPERTIES ("key"="value", ...)`: the properties of the resource. You can modify different properties based on resource types. Currently, StarRocks supports modifying the URI of the Hive metastore of the following resources.
    - Apache Iceberg resources support modifying the following properties:
        - `iceberg.catalog-impl`: the fully qualified class name of custom catalog.
        - `iceberg.catalog.hive.metastore.uris`: the URI of the Hive metastore.
    - Apache Hive™ resources and Apache Hudi resources support modifying `hive.metastore.uris`, which indicates the URI of the Hive metastore.

Usage notes

After you reference a resource to create an external table, if you modify the URI of the Hive metastore of this resource, the external table becomes unavailable. If you still want to use the external table to query data, make sure that the new metastore contains a table whose name and schema are the same as that in the original metastore.

Examples

Modify the URI of the Hive metastore of Hive resource `hive0`.

```
ALTER RESOURCE 'hive0' SET PROPERTIES ("hive.metastore.uris" =
"thrift://xx.xx.xx.xx:9083")
```

ALTER TABLE

Description

Modifies an existing table, including:

- Rename table, partition, index
- Atomic swap
- Add/delete partitions and modify partition attributes
- Schema change
- Create/delete rollup index
- Modify bitmap index

Syntax

```
ALTER TABLE [database.]table
alter_clause1[, alter_clause2, ...]
```

`alter_clause` is classified into six operations: partition, rollup, schema change, rename, index, and swap.

- rename: renames a table, rollup index, or partition.
- comment: modifies the table comment (supported from **v3.1 onwards**).
- partition: modifies partition properties, drops a partition, or adds a partition.
- bucket: modifies the bucketing method and number of buckets.
- column: adds, drops, or reorders columns, or modifies column type.
- rollup index: creates or drops a rollup index.
- bitmap index: modifies index (only Bitmap index can be modified).
- swap: atomic exchange of two tables.

Limits and usage notes

- Operations on partition, column, and rollup index cannot be performed in one ALTER TABLE statement.
- Column names cannot be modified.
- Column comments cannot be modified.
- One table can have only one ongoing schema change operation at a time. You cannot run two schema change commands on a table at the same time.
- Operations on bucket, column and rollup index are asynchronous operations. A success message is return immediately after the task is submitted. You can run the SHOW ALTER TABLE command to check the progress, and run the CANCEL ALTER TABLE command to cancel the operation.
- Operations on rename, comment, partition, bitmap index and swap are synchronous operations, and a command return indicates that the execution is finished.

Rename

Rename supports modification of table name, rollup index, and partition name.

Rename a table

```
ALTER TABLE <tbl_name> RENAME <new_tbl_name>
```

Rename a rollup index

```
ALTER TABLE [<db_name>.]<tbl_name>
RENAME ROLLUP <old_rollup_name> <new_rollup_name>
```

Rename a partition

```
ALTER TABLE [<db_name>.]<tbl_name>
RENAME PARTITION <old_partition_name> <new_partition_name>
```

Modify partition

Add a partition

Syntax:

```
ALTER TABLE [database.]table
ADD PARTITION [IF NOT EXISTS] <partition_name>
partition_desc ["key"="value"]
[DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]];
```

Note:

1. Partition_desc supports the following two expressions:

   ```
   VALUES LESS THAN [MAXVALUE|("value1", ...)]
   VALUES ("value1", ...), ("value1", ...)
   ```

2. partition is the left-closed-right-open interval. If the user only specifies the right boundary, the system will automatically determine the left boundary.

3. If the bucket mode is not specified, the bucket method used by the built-in table is automatically used.

4. If the bucket mode is specified, only the bucket number can be modified, and the bucket mode or bucket column cannot be modified.

5. User can set some properties of the partition in ["key"="value"]. See CREATE TABLE for details.

Drop a partition

Syntax:

```
-- Before 2.0
ALTER TABLE [database.]table
DROP PARTITION [IF EXISTS | FORCE] <partition_name>
-- 2.0 or later
ALTER TABLE [database.]table
DROP PARTITION [IF EXISTS] <partition_name> [FORCE]
```

Note:

1. Keep at least one partition for partitioned tables.
2. After executing DROP PARTITION for a while, the dropped partition can be recovered by the RECOVER statement. See the RECOVER statement for details.

3. If DROP PARTITION FORCE is executed, the partition will be deleted directly and cannot be recovered without checking whether there are any unfinished activities on the partition. Thus, generally this operation is not recommended.

Add a temporary partition

Syntax:

```
ALTER TABLE [database.]table
ADD TEMPORARY PARTITION [IF NOT EXISTS] <partition_name>
partition_desc ["key"="value"]
[DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]]
```

Use a temporary partition to replace current partition

Syntax:

```
ALTER TABLE [database.]table
REPLACE PARTITION <partition_name>
partition_desc ["key"="value"]
WITH TEMPORARY PARTITION
partition_desc ["key"="value"]
[PROPERTIES ("key"="value", ...)]
```

Drop a temporary partition

Syntax:

```
ALTER TABLE [database.]table
DROP TEMPORARY PARTITION <partition_name>
```

Modify partition properties

Syntax:

```
ALTER TABLE [database.]table
MODIFY PARTITION p1|(p1[, p2, ...]) SET ("key" = "value", ...)
```

Note:

1. The following properties of a partition can be modified:
    - storage_medium
    - storage_cooldown_time
    - replication_num
    - in_memory
2. For single-partition tables, partition name is the same as the table name.

Schema change

Schema change supports the following modifications.

Add a column to specified location of specified index

Syntax:

```
ALTER TABLE [database.]table
ADD COLUMN column_name column_type [KEY | agg_type] [DEFAULT
"default_value"]
[AFTER column_name|FIRST]
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

```
1. If you add a value column to an Aggregate table, you need to
specify agg_type.
2. If you add a key column to a non-Aggregate table (such as a
Duplicate Key table), you need to specify the KEY keyword.
3. You cannot add a column that already exists in the base index to
the rollup index. (You can recreate a rollup index if needed.)
```

Add multiple columns to specified index

Syntax:

```
ALTER TABLE [database.]table
ADD COLUMN (column_name1 column_type [KEY | agg_type] DEFAULT
"default_value", ...)
```

```
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

1. If you add a value column to an aggregate table, you need to specify `agg_type`.
2. If you add a key column to a non-aggregate table, you need to specify the KEY keyword.
3. You cannot add a column that already exists in the base index to the rollup index. (You can create another rollup index if needed.)

Drop a column from specified index

Syntax:

```
ALTER TABLE [database.]table
DROP COLUMN column_name
[FROM rollup_index_name];
```

Note:

1. You cannot drop partition column.
2. If the column is dropped from the base index, it will also be dropped if it is included in the rollup index.

Modify the column type and column position of specified index

Syntax:

```
ALTER TABLE [database.]table
MODIFY COLUMN column_name column_type [KEY | agg_type] [NULL | NOT
NULL] [DEFAULT "default_value"]
[AFTER column_name|FIRST]
[FROM rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

1. If you modify the value column in aggregation models, you need to specify agg_type.

2. If you modify the key column in non-aggregation models, you need to specify the KEY keyword.

3. Only the type of column can be modified. The other properties of the column remain as they are currently. (i.e. other properties need to be explicitly written in the statement according to the original property, see example 8).

4. The partition column cannot be modified.

5. The following types of conversions are currently supported (accuracy loss is guaranteed by the user).

    - Convert TINYINT/SMALLINT/INT/BIGINT to TINYINT/SMALLINT/INT/BIGINT/DOUBLE.
    - Convert TINTINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE/DECIMAL to VARCHAR. VARCHAR supports modification of maximum length.
    - Convert VARCHAR to TINTINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE.
    - Convert VARCHAR to DATE (currently support six formats: "%Y-%m-%d", "%y-%m-%d", "%Y%m%d", "%y%m%d", "%Y/%m/%d, "%y/%m/%d")
    - Convert DATETIME to DATE(only year-month-day information is retained, i.e. `2019-12-09 21:47:05` `<-->` `2019-12-09`)
    - Convert DATE to DATETIME (set hour, minute, second to zero, For example: `2019-12-09` `<-->` `2019-12-09 00:00:00`)
    - Convert FLOAT to DOUBLE
    - Convert INT to DATE (If the INT data fails to convert, the original data remains the same)

6. Conversion from NULL to NOT NULL is not supported.

Reorder the columns of specified index

Syntax:

```
ALTER TABLE [database.]table
ORDER BY (column_name1, column_name2, ...)
[FROM rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

1. All columns in the index must be written.
2. The value column is listed after the key column.

Modify table properties

Currently, StarRocks supports modifying the following table properties:

- `replication_num`
- `default.replication_num`
- `storage_cooldown_ttl`
- `storage_cooldown_time`
- Dynamic partitioning related properties
- `enable_persistent_index`
- `bloom_filter_columns`
- `colocate_with`

Syntax:

```
ALTER TABLE [<db_name>.]<tbl_name>
SET ("key" = "value",...)
```

Note: You can also modify the properties by merging into the above schema change operation. See the following examples.

Modify rollup index

Create a rollup index

Syntax:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)]
```

PROPERTIES: Support setting timeout time and the default timeout time is one day.

Example:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP r1(col1,col2) from r0;
```

Create rollup indexes in batches

Syntax:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP [rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)],...];
```

Example:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP r1(col1,col2) from r0, r2(col3,col4) from r0;
```

Note:

1. If from_index_name is not specified, then create from base index by default.
2. The columns in the rollup table must be existing columns in from_index.
3. In properties, user can specify the storage format. See CREATE TABLE for details.

Drop a rollup index

Syntax:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP ROLLUP rollup_name [PROPERTIES ("key"="value", ...)];
```

Example:

```
ALTER TABLE [<db_name>.]<tbl_name> DROP ROLLUP r1;
```

Batch drop rollup indexes

Syntax:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP ROLLUP [rollup_name [PROPERTIES ("key"="value", ...)],...];
```

Example:

```
ALTER TABLE [<db_name>.]<tbl_name> DROP ROLLUP r1, r2;
```

Note: You cannot drop the base index.

Modify bitmap indexes

Bitmap index supports the following modifications:

Create a bitmap index

Syntax:

```
 ALTER TABLE [database.]table
ADD INDEX index_name (column [, ...],) [USING BITMAP] [COMMENT
'balabala'];
```

Note:

```
1. Bitmap index is only supported for the current version.
2. A BITMAP index is created only in a single column.
```

Drop an index

Syntax:

```
DROP INDEX index_name;
```

Swap

Swap supports atomic exchange of two tables.

Syntax:

```
ALTER TABLE [database.]table
SWAP WITH table_name;
```

Examples

Table

1. Modify the default number of replicas of the table, which is used as the default number of replicas for newly added partitions.

   ```
   ALTER TABLE example_db.my_table
   SET ("default.replication_num" = "2");
   ```

2. Modify the actual number of replicas of a single-partition table.

   ```
   ALTER TABLE example_db.my_table
   SET ("replication_num" = "3");
   ```

3. Modify data writing and replication mode among replicas.

   ```
   ALTER TABLE example_db.my_table
   SET ("replicated_storage" = "false");
   ```

   This example sets the data writing and replication mode among replicas to "leaderless replication", which means data is written to multiple replicas at the same time without differentiating primary and secondary replicas. For more information, see the `replicated_storage` parameter in CREATE TABLE.

Partition

1. Add a partition and use the default bucketing mode. The existing partition is [MIN, 2013-01-01). The added partition is [2013-01-01, 2014-01-01).

   ```
   ALTER TABLE example_db.my_table
   ADD PARTITION p1 VALUES LESS THAN ("2014-01-01");
   ```

2. Add a partition and use the new number of buckets.

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
DISTRIBUTED BY HASH(k1) BUCKETS 20;
```

3. Add a partition and use the new number of replicas.

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
("replication_num"="1");
```

4. Alter the number of replicas for a partition.

```
ALTER TABLE example_db.my_table
MODIFY PARTITION p1 SET("replication_num"="1");
```

5. Batch alter the `in_memory` property of specified partitions.

```
ALTER TABLE example_db.my_table
MODIFY PARTITION (p1, p2, p4) SET("in_memory"="true");
```

6. Batch alter the storage medium of all partitions.

```
ALTER TABLE example_db.my_table
MODIFY PARTITION (*) SET("storage_medium"="HDD");
```

7. Drop a partition.

```
ALTER TABLE example_db.my_table
DROP PARTITION p1;
```

8. Add a partition that has upper and lower boundaries.

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES [("2014-01-01"), ("2014-02-01"));
```

Rollup

1. Create an index `example_rollup_index` based on the base index (k1,k2,k3,v1,v2). Column-based storage is used.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index(k1, k3, v1, v2)
PROPERTIES("storage_type"="column");
```

2. Create an index `example_rollup_index2` based on `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. Create an index `example_rollup_index3` based on the base index (k1, k2, k3, v1). The rollup timeout time is set to one hour.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index3(k1, k3, v1)
PROPERTIES("storage_type"="column", "timeout" = "3600");
```

4. Drop an index `example_rollup_index2`.

```
ALTER TABLE example_db.my_table
DROP ROLLUP example_rollup_index2;
```

Schema Change

1. Add a key column `new_col` (non-aggregate column) after the `col1` column of `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT KEY DEFAULT "0" AFTER col1
TO example_rollup_index;
```

2. Add a value column `new_col` (non-aggregate column) after the `col1` column of `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT DEFAULT "0" AFTER col1
TO example_rollup_index;
```

3. Add a key column `new_col` (aggregate column) after the `col1` column of `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT DEFAULT "0" AFTER col1
TO example_rollup_index;
```

4. Add a value column `new_col SUM` (aggregate column) after the `col1` column of `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT SUM DEFAULT "0" AFTER col1
TO example_rollup_index;
```

5. Add multiple columns to `example_rollup_index` (aggregate).

```
ALTER TABLE example_db.my_table
ADD COLUMN (col1 INT DEFAULT "1", col2 FLOAT SUM DEFAULT "2.3")
TO example_rollup_index;
```

6. Drop a column from `example_rollup_index`.

```
ALTER TABLE example_db.my_table
DROP COLUMN col2
FROM example_rollup_index;
```

7. Modify the column type of col1 of the base index to BIGINT and put it after `col2`.

```
ALTER TABLE example_db.my_table
```

```
MODIFY COLUMN col1 BIGINT DEFAULT "1" AFTER col2;
```

8. Modify the maximum length of the `val1` column of the base index to 64. The original length is 32.

```
ALTER TABLE example_db.my_table
MODIFY COLUMN val1 VARCHAR(64) REPLACE DEFAULT "abc";
```

9. Reorder the columns in `example_rollup_index`. The original column order is k1, k2, k3, v1, v2.

```
ALTER TABLE example_db.my_table
ORDER BY (k3,k1,k2,v2,v1)
FROM example_rollup_index;
```

10. Perform two operations (ADD COLUMN and ORDER BY) at one time.

```
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;
```

11. Alter the bloomfilter columns of the table.

```
ALTER TABLE example_db.my_table
SET ("bloom_filter_columns"="k1,k2,k3");
```

    This operation can also be merged into the above schema change operation (note that the syntax of multiple clauses is slightly different).

```
ALTER TABLE example_db.my_table
DROP COLUMN col2
PROPERTIES ("bloom_filter_columns"="k1,k2,k3");
```

12. Alter the Colocate property of the table.

```
```

```
ALTER TABLE example_db.my_table
SET ("colocate_with" = "t1");
```

13. Alter the bucketing mode of the table from Random Distribution to Hash Distribution.

```
ALTER TABLE example_db.my_table
SET ("distribution_type" = "hash");
```

14. Alter the dynamic partition property of the table.

```
ALTER TABLE example_db.my_table
SET ("dynamic_partition.enable" = "false");
```

If you need to add dynamic partition properties to a table with no dynamic partition properties configured, you need to specify all the dynamic partition properties.

```
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"
    );
```

15. Alter the `in_memory` property of the table.

```
ALTER TABLE example_db.my_table
SET ("in_memory" = "true");
```

Rename

1. Rename `table1` to `table2`.

```
ALTER TABLE table1 RENAME table2;
```

2. Rename rollup index `rollup1` of `example_table` to `rollup2`.

```
ALTER TABLE example_table RENAME ROLLUP rollup1 rollup2;
```

3. Rename partition `p1` of `example_table` to `p2`.

```
ALTER TABLE example_table RENAME PARTITION p1 p2;
```

Index

1. Create a bitmap index for column `siteid` in `table1`.

```
ALTER TABLE table1
ADD INDEX index_1 (siteid) [USING BITMAP] COMMENT 'balabala';
```

2. Drop the bitmap index of column `siteid` in `table1`.

```
ALTER TABLE table1
DROP INDEX index_1;
```

Swap

1. Atomic swap between `table1` and `table2`.

```
ALTER TABLE table1 SWAP WITH table2
```

References

- CREATE TABLE
- SHOW CREATE TABLE
- SHOW ALTER TABLE

ALTER VIEW

Description

Modifies the definition of a view.

Syntax

```
ALTER VIEW
[db_name.]view_name
(column1[ COMMENT "col comment"][, column2, ...])
AS query_stmt
```

Note:

1. View is logical, where the data isn't stored in the physical medium. The view will be used as a subquery in the statement when queried. Therefore, modifying the definition of views is equivalent to modifying query_stmt.
2. query_stmt is arbitrarily supported SQL.

Examples

Alter `example_view` on `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
```

ANALYZE TABLE

Description

Creates a manual collection task for collecting CBO statistics. By default, manual collection is a synchronous operation. You can also set it to an asynchronous operation. In asynchronous mode, after you run ANALYZE TABLE, the system immediately returns whether this statement is successful. However, the collection task will be running in the background and you do not have to wait for the result. You can check the status of the task by running SHOW ANALYZE STATUS. Asynchronous collection is suitable for tables with large data volume, whereas synchronous collection is suitable for tables with small data volume.

Manual collection tasks are run only once after creation. You do not need to delete manual collection tasks.

This statement is supported from v2.4.

Manually collect basic statistics

For more information about basic statistics, see Gather statistics for CBO.

Syntax

```
ANALYZE [FULL|SAMPLE] TABLE tbl_name (col_name [,col_name])
[WITH SYNC | ASYNC MODE]
PROPERTIES (property [,property])
```

Parameter description

- Collection type
  - FULL: indicates full collection.
  - SAMPLE: indicates sampled collection.
  - If no collection type is specified, full collection is used by default.

- `col_name`: columns from which to collect statistics. Separate multiple columns with commas (`,`). If this parameter is not specified, the entire table is collected.
- [WITH SYNC | ASYNC MODE]: whether to run the manual collection task in synchronous or asynchronous mode. Synchronous collection is used by default if you do not specify this parameter.
- `PROPERTIES`: custom parameters. If `PROPERTIES` is not specified, the default settings in the `fe.conf` file are used. The properties that are actually used can be viewed via the `Properties` column in the output of SHOW ANALYZE STATUS.

PROPERTIES	Type	Default value	Description
statistic_sample_collect_rows	INT	200000	The minimum number of rows to collect for sampled collection.If the parameter value exceeds the actual number of rows in your table, full collection is performed.

Examples

Example 1: Manual full collection

```
-- Manually collect full stats of a table using default settings.
ANALYZE TABLE tbl_name;

-- Manually collect full stats of a table using default settings.
ANALYZE FULL TABLE tbl_name;

-- Manually collect stats of specified columns in a table using
default settings.
ANALYZE TABLE tbl_name(c1, c2, c3);
```

Example 2: Manual sampled collection

```
-- Manually collect partial stats of a table using default settings.
ANALYZE SAMPLE TABLE tbl_name;

-- Manually collect stats of specified columns in a table, with the
```

```
number of rows to collect specified.
ANALYZE SAMPLE TABLE tbl_name (v1, v2, v3) PROPERTIES(
    "statistic_sample_collect_rows" = "1000000"
);
```

Manually collect histograms

For more information about histograms, see Gather statistics for CBO.

Syntax

```
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON col_name [, col_name]
[WITH SYNC | ASYNC MODE]
[WITH N BUCKETS]
PROPERTIES (property [,property]);
```

Parameter description

- `col_name`: columns from which to collect statistics. Separate multiple columns with commas (`,`). If this parameter is not specified, the entire table is collected. This parameter is required for histograms.
- [WITH SYNC | ASYNC MODE]: whether to run the manual collection task in synchronous or asynchronous mode. Synchronous collection is used by default if you do not specify this parameter.
- `WITH N BUCKETS`: `N` is the number of buckets for histogram collection. If not specified, the default value in `fe.conf` is used.
- PROPERTIES: custom parameters. If `PROPERTIES` is not specified, the default settings in `fe.conf` are used. The properties that are actually used can be viewed via the `Properties` column in the output of SHOW ANALYZE STATUS.

PROPERTIES	Type	Default value	Description
statistic_sample_collect_rows	INT	200000	The minimum number of rows to collect. If the parameter value exceeds

PROPERTIES	Type	Default value	Description
			the actual number of rows in your table, full collection is performed.
histogram_buckets_size	LONG	64	The default bucket number for a histogram.
histogram_mcv_size	INT	100	The number of most common values (MCV) for a histogram.
histogram_sample_ratio	FLOAT	0.1	The sampling ratio for a histogram.
histogram_max_sample_row_count	LONG	10000000	The maximum number of rows to collect for a histogram.

The number of rows to collect for a histogram is controlled by multiple parameters. It is the larger value between `statistic_sample_collect_rows` and table row count * `histogram_sample_ratio`. The number cannot exceed the value specified by `histogram_max_sample_row_count`. If the value is exceeded, `histogram_max_sample_row_count` takes precedence.

Examples

```
-- Manually collect histograms on v1 using the default settings.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1;

-- Manually collect histograms on v1 and v2, with 32 buckets, 32
MCVs, and 50% sampling ratio.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1,v2 WITH 32 BUCKETS
PROPERTIES(
   "histogram_mcv_size" = "32",
   "histogram_sample_ratio" = "0.5"
);
```

References

SHOW ANALYZE STATUS: view the status of a manual collection task.

KILL ANALYZE: cancel a manual collection task that is running.

For more information about collecting statistics for CBO, see Gather statistics for CBO.

BACKUP

Description

Backs up data in a specified database, table, or partition. Currently, StarRocks only supports backing up data in OLAP tables. For more information, see data backup and restoration.

BACKUP is an asynchronous operation. You can check the status of a BACKUP job status using SHOW BACKUP, or cancel a BACKUP job using CANCEL BACKUP. You can view the snapshot information using SHOW SNAPSHOT.

> **CAUTION**
>
> - Only users with the ADMIN privilege can back up data.
> - In each database, only one running BACKUP or RESTORE job is allowed each time. Otherwise, StarRocks returns an error.
> - StarRocks does not support specifying data compression algorithm for data backup.

Syntax

```
BACKUP SNAPSHOT <db_name>.<snapshot_name>
TO <repository_name>
[ ON ( <table_name> [ PARTITION ( <partition_name> [, ...] ) ]
       [, ...] ) ]
[ PROPERTIES ("key"="value" [, ...] ) ]
```

Parameters

Parameter	Description
db_name	Name of the database that stores the data to be backed up.

Parameter	Description
snapshot_name	Specify a name for the data snapshot. Globally unique.
repository_name	Repository name. You can create a repository using CREATE REPOSITORY.
ON	Name of the tables to be backed up. The whole database is backed up if this parameter is not specified.
PARTITION	Name of the partitions to be backed up. The whole table is backed up if this parameter is not specified.
PROPERTIES	Properties of the data snapshot. Valid keys: `type`: Backup type. Currently, only full backup `FULL` is supported. Default: `FULL`. `timeout`: Task timeout. Unit: second. Default: `86400`.

Examples

Example 1: Backs up the database `example_db` to the repository `example_repo`.

```
BACKUP SNAPSHOT example_db.snapshot_label1
TO example_repo
PROPERTIES ("type" = "full");
```

Example 2: Backs up the table `example_tbl` in `example_db` to `example_repo`.

```
BACKUP SNAPSHOT example_db.snapshot_label2
TO example_repo
ON (example_tbl);
```

Example 2: Backs up the partitions `p1` and `p2` of `example_tbl` and the table `example_tbl2` in `example_db` to `example_repo`.

```
BACKUP SNAPSHOT example_db.snapshot_label3
TO example_repo
```

```
ON(
    example_tbl PARTITION (p1, p2),
    example_tbl2
);
```

Usage notes

- Performing backup and restore operations on global, database, table, and partition levels requires different privileges.
- In each database, only one running BACKUP or RESTORE job is allowed each time. Otherwise, StarRocks returns an error.
- Because BACKUP and RESTORE jobs occupy many resources of your StarRocks cluster, you can back up and restore your data while your StarRocks cluster is not heavily loaded.
- StarRocks does not support specifying data compression algorithms for data backup.
- Because data is backed up as snapshots, the data loaded upon snapshot generation is not included in the snapshot. Therefore, if you load data into the old cluster after the snapshot is generated and before the RESTORE job is completed, you also need to load the data into the cluster that data is restored into. It is recommended that you load data into both clusters in parallel for a period of time after the data migration is complete, and then migrate your application to the new cluster after verifying the correctness of the data and services.
- Before the RESTORE job is completed, you cannot operate the table to be restored.
- Primary Key tables cannot be restored to a StarRocks cluster earlier than v2.5.
- You do not need to create the table to be restored in the new cluster before restoring it. The RESTORE job automatically creates it.
- If there is an existing table that has a duplicated name with the table to be restored, StarRocks first checks whether or not the schema of the existing table matches that of the table to be restored. If the schemas match, StarRocks overwrites the existing table with the data in the snapshot. If the schema does not match, the RESTORE job fails. You can either rename the table to be restored using the keyword `AS`, or delete the existing table before restoring data.
- If the RESTORE job overwrites an existing database, table, or partition, the overwritten data cannot be restored after the job enters the COMMIT phase. If the RESTORE job fails or is canceled at this point, the data may be corrupted and inaccessible. In this case, you can only perform the RESTORE operation again and wait for the job to

complete. Therefore, we recommend that you do not restore data by overwriting unless you are sure that the current data is no longer used. The overwrite operation first checks metadata consistency between the snapshot and the existing database, table, or partition. If an inconsistency is detected, the RESTORE operation cannot be performed.

- Currently, StarRocks does not support backing up and restoring logical views.
- Currently, StarRocks does not support backing up and restoring the configuration data related to user accounts, privileges, and resource groups.
- Currently, StarRocks does not support backing up and restoring the Colocate Join relationship among tables.

CANCEL ALTER TABLE

Description

Cancels the following operations performed with the ALTER TABLE statement on a given table:

- Table schema: add and delete columns, reorder columns, and modify data types of columns.
- Rollup index: create and delete rollup indexes.

This statement is a synchronous operation and requires you to have the `ALTER_PRIV` privilege on the table.

Syntax

- Cancel schema changes.

```
CANCEL ALTER TABLE COLUMN FROM [db_name.]table_name
```

- Cancel changes to rollup indexes.

```
CANCEL ALTER TABLE ROLLUP FROM [db_name.]table_name
```

Parameters

Parameter	Required	Description
db_name	No	The name of the database to which the table belongs. If this parameter is not specified, your current database is used by default.

Parameter	Required	Description
table_name	Yes	The table name.

Examples

Example 1: Cancel the schema changes to `example_table` in the `example_db` database.

```
CANCEL ALTER TABLE COLUMN FROM example_db.example_table;
```

Example 2: Cancel rollup index changes to `example_table` in your current database.

```
CANCEL ALTER TABLE ROLLUP FROM example_table;
```

CANCEL BACKUP

Description

Cancels an on-going BACKUP task in a specified database. For more information, see data backup and restoration.

Syntax

```
CANCEL BACKUP FROM <db_name>
```

Parameters

Parameter	Description
db_name	Name of the database that the BACKUP task belongs to.

Examples

Example 1: Cancels the BACKUP task under the database `example_db`.

```
CANCEL BACKUP FROM example_db;
```

CANCEL RESTORE

Description

Cancels an on-going RESTORE task in a specified database. For more information, see data backup and restoration.

> **CAUTION**
>
> If a RESTORE task is canceled during the COMMIT phase, the restored data will be corrupted and inaccessible. In this case, you can only perform the RESTORE operation again and wait for the job to complete.

Syntax

```
CANCEL RESTORE FROM <db_name>
```

Parameters

Parameter	Description
db_name	Name of the database that the RESTORE task belongs to.

Examples

Example 1: Cancels the RESTORE task under the database `example_db`.

```
CANCEL RESTORE FROM example_db;
```

CREATE ANALYZE

Description

Customizes an automatic collection task for collecting CBO statistics.

By default, StarRocks automatically collects full statistics of a table. It checks for any data updates every 5 minutes. If data change is detected, data collection will be automatically triggered. If you do not want to use automatic full collection, you can set the FE configuration item `enable_collect_full_statistic` to `false` and customize a collection task.

Before creating a custom automatic collection task, you must disable automatic full collection (`enable_collect_full_statistic = false`). Otherwise, custom tasks cannot take effect.

This statement is supported from v2.4.

Syntax

```
-- Automatically collect stats of all databases.
CREATE ANALYZE [FULL|SAMPLE] ALL PROPERTIES (property [,property])

-- Automatically collect stats of all tables in a database.
CREATE ANALYZE [FULL|SAMPLE] DATABASE db_name
PROPERTIES (property [,property])

-- Automatically collect stats of specified columns in a table.
CREATE ANALYZE [FULL|SAMPLE] TABLE tbl_name (col_name [,col_name])
PROPERTIES (property [,property])
```

Parameter description

- Collection type

    - FULL: indicates full collection.
    - SAMPLE: indicates sampled collection.
    - If no collection type is specified, full collection is used by default.

- `col_name`: columns from which to collect statistics. Separate multiple columns with commas (`,`). If this parameter is not specified, the entire table is collected.

- `PROPERTIES`: custom parameters. If `PROPERTIES` is not specified, the default settings in `fe.conf` are used. The properties that are actually used can be viewed via the `Properties` column in the output of SHOW ANALYZE JOB.

PROPERTIES	Type	Default value	Description
statistic_auto_collect_ratio	FLOAT	0.8	The threshold for determining whether the statistics for automatic collection are healthy. If the statistics health is below this threshold, automatic collection is triggered.
statistics_max_full_collect_data_size	INT	100	The size of the largest partition for automatic collection to collect data. Unit: GB.If a partition exceeds this value, full collection is discarded and sampled collection is performed instead.
statistic_sample_collect_rows	INT	200000	The minimum number of rows to collect.If the parameter value exceeds the actual number of rows in your table, full collection is performed.

Examples

Example 1: Automatic full collection

```
-- Automatically collect full stats of all databases.
CREATE ANALYZE ALL;

-- Automatically collect full stats of a database.
CREATE ANALYZE DATABASE db_name;

-- Automatically collect full stats of all tables in a database.
CREATE ANALYZE FULL DATABASE db_name;

-- Automatically collect full stats of specified columns in a table.
CREATE ANALYZE TABLE tbl_name(c1, c2, c3);
```

Example 2: Automatic sampled collection

```
-- Automatically collect stats of all tables in a database using
default settings.
CREATE ANALYZE SAMPLE DATABASE db_name;

-- Automatically collect stats of specified columns in a table, with
statistics health and the number of rows to collect specified.
CREATE ANALYZE SAMPLE TABLE tbl_name(c1, c2, c3) PROPERTIES(
   "statistic_auto_collect_ratio" = "0.5",
   "statistic_sample_collect_rows" = "1000000"
);
```

References

SHOW ANALYZE JOB: view the status of a custom collection task.

DROP ANALYZE: delete a custom collection task.

KILL ANALYZE: cancel a custom collection task that is running.

For more information about collecting statistics for CBO, see Gather statistics for CBO.

CREATE DATABASE

Description

This statement is used to create databases.

Syntax

```
CREATE DATABASE [IF NOT EXISTS] <db_name>
```

Examples

1. Create database `db_test`.

   ```
   CREATE DATABASE db_test;
   ```

References

- SHOW CREATE DATABASE
- USE
- SHOW DATABASES
- DESC
- DROP DATABASE

CREATE EXTERNAL CATALOG

Description

Creates an external catalog. You can use external catalogs to query data in external data sources without loading data into StarRocks or creating external tables. Currently, you can create the following types of external catalogs:

- Hive catalog: used for querying data from Apache Hive™.
- Iceberg catalog: used for querying data from Apache Iceberg.
- Hudi catalog: used for querying data from Apache Hudi.
- Delta Lake catalog: used to query data from Delta Lake.

Before you create external catalogs, configure your StarRocks cluster to meet the requirements of the data storage system (such as Amazon S3), metadata service (such as Hive metastore), and authenticating service (such as Kerberos) of external data sources. For more information, see the "Before you begin" section in each external catalog topic.

Syntax

```
CREATE EXTERNAL CATALOG <catalog_name>
PROPERTIES ("key"="value", ...)
```

Parameters

Parameter	Required	Description
catalog_name	Yes	The name of the external catalog. The naming conventions are as follows: • The name can contain letters, digits (0-9), and underscores (_). It must start with a letter.

Parameter	Required	Description
		• The name is case-sensitive and cannot exceed 1023 characters in length.
comment	No	The description of the external catalog.
PROPERTIES	Yes	The properties of an external catalog. Configure properties based on the types of external catalogs. For more information, see Hive catalog, Iceberg catalog, Hudi catalog, and Delta Lake catalog.

Examples

Example 1: Create a Hive catalog named `hive_metastore_catalog`. The corresponding Hive cluster uses Hive metastore as its metadata service.

```
CREATE EXTERNAL CATALOG hive_metastore_catalog
PROPERTIES(
   "type"="hive",
   "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Example 2: Create a Hive catalog named `hive_glue_catalog`. The corresponding Hive cluster uses AWS Glue as its metadata service.

```
CREATE EXTERNAL CATALOG hive_glue_catalog
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"
);
```

Example 3: Create an Iceberg catalog named `iceberg_metastore_catalog`. The corresponding Iceberg cluster uses Hive metastore as its metadata service.

```
CREATE EXTERNAL CATALOG iceberg_metastore_catalog
PROPERTIES(
    "type"="iceberg",
    "iceberg.catalog.type"="hive",
    "iceberg.catalog.hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Example 4: Create an Iceberg catalog named `iceberg_glue_catalog`. The corresponding Iceberg cluster uses AWS Glue as its metadata service.

```
CREATE EXTERNAL CATALOG iceberg_glue_catalog
PROPERTIES(
    "type"="iceberg",
    "iceberg.catalog.type"="glue",
    "aws.hive.metastore.glue.aws-access-key"="xxxxx",
    "aws.hive.metastore.glue.aws-secret-key"="xxxxxxxxxxx",
    "aws.hive.metastore.glue.endpoint"="https://glue.x-x-
x.amazonaws.com"
);
```

Example 5: Create a Hudi catalog named `hudi_metastore_catalog`. The corresponding Hudi cluster uses Hive metastore as its metadata service.

```
CREATE EXTERNAL CATALOG hudi_metastore_catalog
PROPERTIES(
    "type"="hudi",
    "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Example 6: Create a Hudi catalog named `hudi_glue_catalog`. The corresponding Hudi cluster uses AWS Glue as its metadata service.

```
CREATE EXTERNAL CATALOG hudi_glue_catalog
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"
);
```

Example 7: Create a Delta Lake catalog named `delta_metastore_catalog`. The corresponding Delta Lake service uses Hive metastore as its metadata service.

```
CREATE EXTERNAL CATALOG delta_metastore_catalog
PROPERTIES(
    "type"="deltalake",
    "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Example 8: Create a Delta Lake catalog named `delta_glue_catalog`. The corresponding Delta Lake service uses AWS Glue as its metadata service.

```
CREATE EXTERNAL CATALOG delta_glue_catalog
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"
);
```

References

- To view all catalogs in your StarRocks cluster, see SHOW CATALOGS.
- To delete an external catalog from your StarRocks cluster, see DROP CATALOG.

CREATE FUNCTION

Description

Creates a user-defined function (UDF). Currently, you can only create Java UDFs, including Scalar functions, user-defined aggregate functions (UDAFs), user-defined window functions (UDWFs), and user-defined table functions (UDTFs).

For details about how to compile, create, and use a Java UDF, see Java UDF.

> **NOTE**
>
> To create a global UDF, you must have the SYSTEM-level CREATE GLOBAL FUNCTION privilege. To create a database-wide UDF, you must have the DATABASE-level CREATE FUNCTION privilege.

Syntax

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

Parameters

Parameter	Required	Description
GLOBAL	No	Whether to create a global UDF, supported from v3.0.
AGGREGATE	No	Whether to create a UDAF or UDWF.

Parameter	Required	Description
TABLE	No	Whether to create a UDTF. If both `AGGREGATE` and `TABLE` are not specified, a Scalar function is created.
function_name	Yes	The name of the function you want to create. You can include the name of the database in this parameter, for example, `db1.my_func`. If `function_name` includes the database name, the UDF is created in that database. Otherwise, the UDF is created in the current database. The name of the new function and its parameters cannot be the same as an existing name in the destination database. Otherwise, the function cannot be created. The creation succeeds if the function name is the same but the parameters are different.
arg_type	Yes	Argument type of the function. The added argument can be represented by `, ...`. For the supported data types, see Java UDF.
return_type	Yes	The return type of the function. For the supported data types, see Java UDF.
PROPERTIES	Yes	Properties of the function, which vary depending on the type of the UDF to create. For details, see Java UDF。

CREATE INDEX

Description

This statement is used to create indexes.

Syntax:

```
CREATE INDEX index_name ON table_name (column [, ...],) [USING
BITMAP] [COMMENT'balabala']
```

Note:

1. Only support bitmap index in the current version.
2. Create BITMAP index only in a single column.

Examples

1. Create bitmap index for `siteid` on `table1`.

   ```
   CREATE INDEX index_name ON table1 (siteid) USING BITMAP COMMENT
   'balabala';
   ```

CREATE MATERIALIZED VIEW

Description

Creates a materialized view. For usage information about materialized views, see Synchronous materialized view and Asynchronous materialized view.

> **CAUTION**
>
> Only users with the CREATE MATERIALIZED VIEW privilege in the database where the base table resides can create a materialized view.

Creating a materialized view is an asynchronous operation. Running this command successfully indicates that the task of creating the materialized view is submitted successfully. You can view the building status of a synchronous materialized view in a database via SHOW ALTER MATERIALIZED VIEW command, and view that of an asynchronous materialized view by querying the metadata tables `tasks` and `task_runs` in Information Schema.

StarRocks supports asynchronous materialized views from v2.4. The major differences between asynchronous materialized views and synchronous materialized views in previous versions are as follows:

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
ASYNC MV	Yes	Yes	Yes	• Asynchronous refresh • Manual refresh	Multiple tables from: • Default catalog • External catalogs (v2.5)

	Single-table aggregation	Multi-table join	Query rewrite	Refresh strategy	Base table
					• Existing materialized views (v2.5)
SYNC MV (Rollup)	Limited choices of aggregate functions	No	Yes	Synchronous refresh during data loading	Single table in the default catalog

Synchronous materialized view

Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [database.]<mv_name>
[COMMENT ""]
[PROPERTIES ("key"="value", ...)]
AS
<query_statement>
```

Parameters in brackets [] are optional.

Parameters

mv_name (required)

The name of the materialized view. The naming requirements are as follows:

- The name must consist of letters (a-z or A-Z), digits (0-9), or underscores (_), and it can only start with a letter.
- The length of the name cannot exceed 64 characters.
- The name is case-sensitive.

COMMENT (optional)

Comment on the materialized view. Note that `COMMENT` must be placed after `mv_name`. Otherwise, the materialized view cannot be created.

query_statement (required)

The query statement to create the materialized view. Its result is the data in the materialized view. The syntax is as follows:

```
SELECT select_expr[, select_expr ...]
[GROUP BY column_name[, column_name ...]]
[ORDER BY column_name[, column_name ...]]
```

- select_expr (required)

  All columns in the query statement, that is, all columns in the materialized view schema. This parameter supports the following values:

  - Simple columns or aggregate columns such as `SELECT a, abs(b), min(c) FROM table_a`, where `a`, `b`, and `c` are the names of columns in the base table. If you do not specify column names for the materialized view, StarRocks automatically assigns names to the columns.
  - Expressions such as `SELECT a+1 AS x, b+2 AS y, c*c AS z FROM table_a`, where `a+1`, `b+2` and `c*c` are the expressions that reference the columns in the base tables, and `x`, `y` and `z` are the aliases assigned to the columns in the materialized view.

> **NOTE**
>
> - You must specify at least one column in `select_expr`.
> - Synchronous materialized views only support aggregate functions on a single column. Query statements in the form of `sum(a+b)` are not supported.
> - When creating a synchronous materialized view with an aggregate function, you must specify the GROUP BY clause, and specify at least one GROUP BY column in `select_expr`.
> - Synchronous materialized views do not support clauses such as JOIN, WHERE, and the HAVING clause of GROUP BY.
> - When using ALTER TABLE DROP COLUMN to drop a specific column in a base table, you must ensure that all synchronous materialized views of the base

table do not contain the dropped column, otherwise the drop operation will fail. Before you drop the column, you must first drop all synchronous materialized views that contain the column.

- Creating too many synchronous materialized views for a table will affect the data load efficiency. When data is being loaded to the base table, the data in synchronous materialized view and base table will be updated synchronously. If a base table contains `n` synchronous materialized views, the efficiency of loading data into the base table is about the same as the efficiency of loading data into `n` tables.

- GROUP BY (optional)

  The GROUP BY column of the query. If this parameter is not specified, the data will not be grouped by default.

- ORDER BY (optional)

  The ORDER BY column of the query.

  - Columns in the ORDER BY clause must be declared in the same order as the columns in `select_expr`.
  - If the query statement contains a GROUP BY clause, the ORDER BY columns must be identical to the GROUP BY columns.
  - If this parameter is not specified, the system will automatically supplement the ORDER BY column according to the following rules:
    - If the materialized view is the AGGREGATE type, all GROUP BY columns are automatically used as sort keys.
    - If the materialized view is not the AGGREGATE type, StarRocks automatically selects sort keys based on the prefix columns.

Automatic query rewrite with synchronous materialized view

When a query that follows the pattern of a synchronous materialized view is executed, the original query statement is automatically rewritten and the intermediate results stored in the materialized view are used.

The following table shows the correspondence between the aggregate function in the original query and the aggregate function used to construct the materialized view. You can

select the corresponding aggregate function to build a materialized view according to your business scenario.

aggregate function in the original query	aggregate function of the materialized view
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

Asynchronous materialized view

Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [database.]<mv_name>
[COMMENT ""]
-- You must specify either `distribution_desc` or `refresh_scheme`,
or both.
-- distribution_desc
DISTRIBUTED BY HASH(<bucket_key>[,<bucket_key2> ...]) [BUCKETS
<bucket_number>]
-- refresh_scheme
[REFRESH
    [ASYNC | ASYNC [START (<start_time>)] EVERY (INTERVAL
<refresh_interval>) | MANUAL]
]
```

```
-- partition_expression
[PARTITION BY
    {<date_column> | date_trunc(fmt, <date_column>)}
]
-- order_by_expression
[ORDER BY (<sort_key>)]
[PROPERTIES ("key"="value", ...)]
AS
<query_statement>
```

Parameters in brackets [] are optional.

Parameters

mv_name (required)

The name of the materialized view. The naming requirements are as follows:

- The name must consist of letters (a-z or A-Z), digits (0-9), or underscores (_), and it can only start with a letter.
- The length of the name cannot exceed 64 characters.
- The name is case-sensitive.

> **CAUTION**
>
> Multiple materialized views can be created on the same base table, but the names of the materialized views in the same database cannot be duplicated.

COMMENT (optional)

Comment on the materialized view. Note that `COMMENT` must be placed after `mv_name`. Otherwise, the materialized view cannot be created.

distribution_desc (reuqired)

The bucketing strategy of the asynchronous materialized view.

Syntax

```
DISTRIBUTED BY HASH (<bucket_key1>[,<bucket_key2> ...]) [BUCKETS
<bucket_number>]
```

For more information, see Data distribution.

> **NOTE**
>
> Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see determine the number of buckets.

refresh_scheme (optional)

> **NOTE**
>
> While creating an asynchronous materialized view, you must specify either `distribution_desc` or `refresh_scheme`, or both.

The refresh strategy of the asynchronous materialized view. Valid values:

- `ASYNC`: Automatic refresh mode. Each time the base table data changes, the materialized view is automatically refreshed.
- `ASYNC [START (<start_time>)] EVERY(INTERVAL <interval>)`: Regular refresh mode. The materialized view is refreshed regularly at the interval defined. You can specify the interval as `EVERY (interval n day/hour/minute/second)` using the following units: `DAY`, `HOUR`, `MINUTE`, and `SECOND`. The default value is `10 MINUTE`. You can further specify the refresh start time as `START('yyyy-MM-dd hh:mm:ss')`. If the start time is not specified, the current time is used. Example: `ASYNC START ('2023-09-12 16:30:25') EVERY (INTERVAL 5 MINUTE)`.
- `MANUAL`: Manual refresh mode. The materialized view will not be refreshed unless you trigger a refresh task manually.

If this parameter is not specified, the default value `MANUAL` is used.

partition_expression (optional)

The partitioning strategy of the asynchronous materialized view. As for the current version of StarRocks, only one partition expression is supported when creating an asynchronous materialized view.

> **CAUTION**
>
> Currently, asynchronous materialized views do not support the list partitioning strategy.

Valid values:

- `column_name`: The name of the column used for partitioning. The expression `PARTITION BY dt` means to partition the materialized view according to the `dt` column.
- date_trunc function: The function used to truncate the time unit. `PARTITION BY date_trunc("MONTH", dt)` means that the `dt` column is truncated to month as the unit for partitioning. The date_trunc function supports truncating time to units including `YEAR`, `MONTH`, `DAY`, `HOUR`, and `MINUTE`.

If this parameter is not specified, no partitioning strategy is adopted by default.

PROPERTIES (optional)

Properties of the asynchronous materialized view. You can modify the properties of an existing materialized view using ALTER MATERIALIZED VIEW.

- `replication_num`: The number of materialized view replicas to create.
- `storage_medium`: Storage medium type. Valid values: `HDD` and `SSD`.
- `storage_cooldown_time`: the storage cooldown time for a partition. If both HDD and SSD storage mediums are used, data in the SSD storage is moved to the HDD storage after the time specified by this property. Format: "yyyy-MM-dd HH:mm:ss". The specified time must be later than the current time. If this property is not explicitly specified, the storage cooldown is not performed by default.
- `partition_ttl_number`: The number of most recent materialized view partitions to keep. For the partitions with a start time earlier than the current time, after the number of these partitions exceeds this value, less recent partitions will be deleted. StarRocks will periodically check materialized view partitions according to the time interval specified in the FE configuration item `dynamic_partition_check_interval_seconds`, and automatically delete expired partitions. If you enabled the dynamic partitioning strategy, the partitions created in advance are not counted in. When the value is `-1`, all partitions of the materialized view will be preserved. Default: `-1`.
- `partition_refresh_number`: In a single refresh, the maximum number of partitions to refresh. If the number of partitions to be refreshed exceeds this value, StarRocks will split the refresh task and complete it in batches. Only when the previous batch of partitions is refreshed successfully, StarRocks will continue to refresh the next batch of partitions until all partitions are refreshed. If any of the partitions fail to be refreshed, no

subsequent refresh tasks will be generated. When the value is `-1`, the refresh task will not be split. Default: `-1`.

- `excluded_trigger_tables`: If a base table of the materialized view is listed here, the automatic refresh task will not be triggered when the data in the base table is changed. This parameter only applies to load-triggered refresh strategy, and is usually used together with the property `auto_refresh_partitions_limit`. Format: `[db_name.]table_name`. When the value is an empty string, any data change in all base tables triggers the refresh of the corresponding materialized view. The default value is an empty string.
- `auto_refresh_partitions_limit`: The number of most recent materialized view partitions that need to be refreshed when a materialized view refresh is triggered. You can use this property to limit the refresh range and reduce the refresh cost. However, because not all the partitions are refreshed, the data in the materialized view may not be consistent with the base table. Default: `-1`. When the value is `-1`, all partitions will be refreshed. When the value is a positive integer N, StarRocks sorts the existing partitions in chronological order, and refreshes the current partition and N-1 most recent partitions. If the number of partitions is less than N, StarRocks refreshes all existing partitions. If there are dynamic partitions created in advance in your materialized view, StarRocks refreshes all pre-created partitions.

> **CAUTION**
>
> The Unique Key constraints and Foreign Key constraints are only used for query rewrite. The Foreign Key constraint checks are not guaranteed when data is loaded into the table. You must ensure the data loaded into the table meets the constraints.

query_statement (required)

The query statement to create the asynchronous materialized view.

> **CAUTION**
>
> Currently, StarRocks does not support creating asynchronous materialized views with base tables created with the list partitioning strategy.

Query an asynchronous materialized view

An asynchronous materialized view is a physical table. You can operate it as any regular table **except that you cannot directly load data into an asynchronous materialized**

view.

Automatic query rewrite with asynchronous materialized view

StarRocks v2.5 supports automatic and transparent query rewrite based on the SPJG-type asynchronous materialized views. The SPJG-type materialized views refer to materialized views whose plan only includes Scan, Filter, Project, and Aggregate types of operators. The SPJG-type materialized views query rewrite includes single table query rewrite, Join query rewrite, aggregation query rewrite, Union query rewrite and query rewrite based on nested materialized views.

See Asynchronous materialized view - Rewrite queries with the asynchronous materialized view for further information.

Supported data types

- Asynchronous materialized views created based on the StarRocks default catalog support the following data types:
    - **Date**: DATE, DATETIME
    - **String**: CHAR, VARCHAR
    - **Numeric**: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, PERCENTILE
    - **Semi-structured**: ARRAY, JSON
    - **Other**: BITMAP, HLL

> **NOTE**
>
> BITMAP, HLL, and PERCENTILE have been supported since v2.4.5.

- Asynchronous materialized views created based on the StarRocks external catalogs support the following data types:
    - 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

Usage notes

- The current version of StarRocks does not support creating multiple materialized views at the same time. A new materialized view can only be created when the one before is completed.
- About synchronous materialized views:
    - Synchronous materialized views only support aggregate functions on a single column. Query statements in the form of `sum(a+b)` are not supported.
    - Synchronous materialized views support only one aggregate function for each column of the base table. Query statements such as `select sum(a), min(a) from table` are not supported.
    - When creating a synchronous materialized view with an aggregate function, you must specify the GROUP BY clause, and specify at least one GROUP BY column in SELECT.
    - Synchronous materialized views do not support clauses such as JOIN, WHERE, and the HAVING clause of GROUP BY.
    - When using ALTER TABLE DROP COLUMN to drop a specific column in a base table, you must ensure that all synchronous materialized views of the base table do not contain the dropped column, otherwise, the drop operation will fail. Before you

drop the column, you must first drop all synchronous materialized views that contain the column.
    - Creating too many synchronous materialized views for a table will affect the data load efficiency. When data is being loaded to the base table, the data in the synchronous materialized view and the base table will be updated synchronously. If a base table contains `n` synchronous materialized views, the efficiency of loading data into the base table is about the same as the efficiency of loading data into `n` tables.
- About nested asynchronous materialized views:
    - The refresh strategy for each materialized view only applies to the corresponding materialized view.
    - Currently, StarRocks does not limit the number of nesting levels. In a production environment, we recommend that the number of nesting layers not exceed THREE.
- About external catalog asynchronous materialized views:
    - External catalog materialized view only support async fixed-interval refresh and manual refresh.
    - Strict consistency is not guaranteed between the materialized view and the base tables in the external catalog.
    - Currently, building materialized views based on external resources is not supported.
    - Currently, StarRocks cannot perceive if the base table data in the external catalog has changed, so all partitions will be refreshed by default every time the base table is refreshed. You can manually refresh only some of partitions using REFRESH MATERIALIZED VIEW.

Examples

Examples of synchronous materialized views

The schema of the base table is as follows:

```
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     |       |
+-------+--------+------+------+---------+-------+
```

Example 1: Create a synchronous materialized view that only contains the columns of the original table (k1, k2).

```
create materialized view k1_k2 as
select k1, k2 from duplicate_table;
```

The materialized view contains only two columns k1 and k2 without any aggregation.

```
+-----------------+-------+--------+------+------+---------+-------+
| IndexName       | Field | Type   | Null | Key  | Default | Extra |
+-----------------+-------+--------+------+------+---------+-------+
| k1_k2           | k1    | INT    | Yes  | true | N/A     |       |
|                 | k2    | INT    | Yes  | true | N/A     |       |
+-----------------+-------+--------+------+------+---------+-------+
```

Example 2: Create a synchronous materialized view sorted by k2.

```
create materialized view k2_order as
select k2, k1 from duplicate_table order by k2;
```

The materialized view's schema is shown below. The materialized view contains only two columns k2 and k1, where column k2 is a sort column without any aggregation.

```
+-----------------+-------+--------+------+-------+---------+-------+
| IndexName       | Field | Type   | Null | Key   | Default | Extra |
+-----------------+-------+--------+------+-------+---------+-------+
| k2_order        | k2    | INT    | Yes  | true  | N/A     |       |
|                 | k1    | INT    | Yes  | false | N/A     | NONE  |
+-----------------+-------+--------+------+-------+---------+-------+
```

Example 3: Create a synchronous materialized view grouped by k1 and k2, and a SUM aggregation on k3.

```
create materialized view k1_k2_sumk3 as
select k1, k2, sum(k3) from duplicate_table group by k1, k2;
```

The materialized view's schema is shown below. The materialized view contains three columns k1, k2 and sum (k3), where k1, k2 are grouped columns, and sum (k3) is the sum of the k3 columns grouped according to k1 and 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   |
+----------------+-------+--------+------+-------+---------+-------+
```

Because the materialized view does not declare a sort column, and it adopts an aggregation function, StarRocks supplements the grouped columns k1 and k2 by default.

Example 4: Create a synchronous materialized view to remove duplicate rows.

```
create materialized view deduplicate as
select k1, k2, k3, k4 from duplicate_table group by k1, k2, k3, k4;
```

The materialized view's schema is shown below. The materialized view contains k1, k2, k3, and k4 columns, and there are no duplicate rows.

```
+----------------+-------+--------+------+-------+---------+-------+
| 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     |       |
+----------------+-------+--------+------+-------+---------+-------+
```

Example 5: Create a non-aggregated synchronous materialized view that does not declare a sort column.

The schema of the base table is shown below:

```
+-------+-------------+------+-------+---------+-------+
| 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  |
+-------+-------------+------+-------+---------+-------+
```

The materialized view contains k3, k4, k5, k6, and k7 columns, and no sort column is declared. Create the materialized view with the following statement:

```
create materialized view mv_1 as
select k3, k4, k5, k6, k7 from all_type_table;
```

StarRocks automatically uses k3, k4, and k5 as the sort columns by default. The sum of the bytes occupied by these three column types is 4 (INT) + 8 (BIGINT) + 16 (DECIMAL) = 28 < 36. So these three columns are added as sort columns.

The materialized view's schema is as follows.

```
+---------------+-------+-------------+------+-------+---------+---
----+
| 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  |
```

```
+---------------+-------+-------------+------+-------+---------+---
----+
```

It can be observed that the `key` field of the k3, k4, and k5 columns is `true`, which indicates that they are the sort keys. The key field of the k6, and k7 columns is `false`, which indicates that they are not the sort keys.

Examples of asynchronous materialized views

The following examples are based on the base tables below:

```
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"
);
```

Example 1: Create a non-partitioned materialized view.

```
CREATE MATERIALIZED VIEW lo_mv1
DISTRIBUTED BY HASH(`lo_orderkey`)
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
order by lo_orderkey;
```

Example 2: Create a partitioned materialized view.

```
CREATE MATERIALIZED VIEW lo_mv2
PARTITION BY `lo_orderdate`
DISTRIBUTED BY HASH(`lo_orderkey`)
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
order by lo_orderkey;

-- Use the date_trunc() function to partition the materialized view
by month.
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;
```

Example 3: Create an asynchronous materialized view.

```
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;
```

CREATE REPOSITORY

Description

Creates a repository in a remote storage system that is used to store data snapshots for backing up and restoring data.

> **CAUTION**
>
> Only users with the ADMIN privilege can create a repository.

For detailed instructions on deleting a repository, see DROP REPOSITORY.

Syntax

```
CREATE [READ ONLY] REPOSITORY <repository_name>
WITH BROKER
ON LOCATION "<repository_location>"
PROPERTIES ("key"="value", ...)
```

Parameters

Parameter	Description
READ ONLY	Create a read-only repository. Note that you can only restore data from a read-only repository. When creating the same repository for two clusters to migrate data, you can create a read-only repository for the new cluster and only grant it RESTORE permissions.
repository_name	Repository name.
repository_location	Location of the repository in the remote storage system.

Parameter	Description
PROPERTIES	The credential method for accessing the remote storage system.

PROPERTIES:

StarRocks supports creating repositories in HDFS, AWS S3, and Google GCS.

- For HDFS:
  - "username": The username of the account that you want to use to access the NameNode of the HDFS cluster.
  - "password": The password of the account that you want to use to access the NameNode of the HDFS cluster.
- For AWS S3:
  - "aws.s3.use_instance_profile": Whether or not to allow instance profile and assumed role as credential methods for accessing AWS S3. Default: `false`.
    - If you use IAM user-based credential (Access Key and Secret Key) to access AWS S3, you don't need to specify this parameter, and you need to specify "aws.s3.access_key", "aws.s3.secret_key", and "aws.s3.endpoint".
    - If you use Instance Profile to access AWS S3, you need to set this parameter to `true` and specify "aws.s3.region".
    - If you use Assumed Role to access AWS S3, you need to set this parameter to `true` and specify "aws.s3.iam_role_arn" and "aws.s3.region".
  - "aws.s3.access_key": The Access Key ID that you can use to access the Amazon S3 bucket.
  - "aws.s3.secret_key": The Secret Access Key that you can use to access the Amazon S3 bucket.
  - "aws.s3.endpoint": The endpoint that you can use to access the Amazon S3 bucket.
  - "aws.s3.iam_role_arn": The ARN of the IAM role that has privileges on the AWS S3 bucket in which your data files are stored. If you want to use assumed role as the credential method for accessing AWS S3, you must specify this parameter. Then,

StarRocks will assume this role when it analyzes your Hive data by using a Hive catalog.

- "aws.s3.region": The region in which your AWS S3 bucket resides. Example: `us-west-1`.

> **NOTE**
>
> StarRocks supports creating repositories in AWS S3 only according to the S3A protocol. Therefore, when you create repositories in AWS S3, you must replace `s3://` in the S3 URI you pass as a repository location in `ON LOCATION` with `s3a://`.

- For Google GCS:
  - "fs.s3a.access.key": The Access Key that you can use to access the Google GCS bucket.
  - "fs.s3a.secret.key": The Secret Key that you can use to access the Google GCS bucket.
  - "fs.s3a.endpoint": The endpoint that you can use to access the Google GCS bucket. Do not specify `https` in the endpoint address.

> **NOTE**
>
> StarRocks supports creating repositories in Google GCS only according to the S3A protocol. Therefore, when you create repositories in Google GCS, you must replace the prefix in the GCS URI you pass as a repository location in `ON LOCATION` with `s3a://`.

Examples

Example 1: Create a repository named `hdfs_repo` in an 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"
);
```

Example 2: Create a read-only repository named `s3_repo` in the 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.endpoint" = "s3.us-east-1.amazonaws.com"
);
```

Example 3: Create a repository named `gcs_repo` in the 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"
);
```

CREATE RESOURCE

Description

This statement is used to create resources. Only user root or admin can create resources. Currently, only Spark and Hive resources are supported. Other external resources may be added to StarRocks in the future, such as Spark/GPU for query, HDFS/S3 for external storage, and MapReduce for ETL.

Syntax:

```
CREATE [EXTERNAL] RESOURCE "resource_name"
PROPERTIES ("key"="value", ...)
```

Note:

1. PROPERTIES specifies the resource type. Currently, only Spark and Hive are supported.
2. The PROPERTIES vary depending on the resource type. See examples for details.

Examples

1. Create a Spark resource named spark0 in yarn Cluster mode.

```
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/starrocks",
    "broker" = "broker0",
    "broker.username" = "user0",
    "broker.password" = "password0"
);
```

Parameters related to Spark are as follows:

```
1. spark.master: required. Currently, yarn and spark
://host:port. are supported.
2. spark.submit.deployMode: deployment mode of the Spark program
is required. Support cluster and client.
3. spark.hadoop.yarn.resourcemanager.address: required when
master is yarn.
4. spark.hadoop.fs.defaultFS: required when master is yarn.
5. Other parameters are optional. Please refer to
http://spark.apache.org/docs/latest/configuration.html
```

If Spark is used for ETL, working_DIR and broker need to be specified. The instructions are as follows:

```
working_dir: Directory used by ETL. It is required when spark is
used as ETL resource. For example:
hdfs://host:port/tmp/starrocks.
broker: Name of broker. It is required when spark is used as ETL
resource and needs to be configured beforehand by using `ALTER
SYSTEM ADD BROKER` command.
broker.property_key: It is the property information needed to be
specified when broker reads the intermediate files created by
ETL.
```

2. Create a Hive resource named hive0.

```
CREATE EXTERNAL RESOURCE "hive0"
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.uris" = "thrift://10.10.44.98:9083"
);
```

CREATE TABLE

Description

Creates a new table in StarRocks.

Syntax

```
CREATE [EXTERNAL] 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]
[PROPERTIES ("key"="value", ...)]
[BROKER PROPERTIES ("key"="value", ...)]
```

Parameters

column_definition

Syntax:

```
col_name col_type [agg_type] [NULL | NOT NULL] [DEFAULT
"default_value"]
```

col_name: Column name.

Note that normally you cannot create a column whose name is initiated with `__op` or `__row` because these name formats are reserved for special purposes in StarRocks and creating

such columns may result in undefined behavior. If you do need to create such column, set the FE dynamic parameter `allow_system_reserved_names` to `TRUE`.

col_type: Column type. Specific column information, such as types and ranges:

- TINYINT (1 byte): Ranges from -2^7 + 1 to 2^7 - 1.
- SMALLINT (2 bytes): Ranges from -2^15 + 1 to 2^15 - 1.
- INT (4 bytes): Ranges from -2^31 + 1 to 2^31 - 1.
- BIGINT (8 bytes): Ranges from -2^63 + 1 to 2^63 - 1.
- LARGEINT (16 bytes): Ranges from -2^127 + 1 to 2^127 - 1.
- FLOAT (4 bytes): Supports scientific notation.
- DOUBLE (8 bytes): Supports scientific notation.
- DECIMAL[(precision, scale)] (16 bytes)
    - Default value: DECIMAL(10, 0)
    - precision: 1 ~ 38
    - scale: 0 ~ precision
    - Integer part: precision - scale

      Scientific notation is not supported.
- DATE (3 bytes): Ranges from 0000-01-01 to 9999-12-31.
- DATETIME (8 bytes): Ranges from 0000-01-01 00:00:00 to 9999-12-31 23:59:59.
- CHAR[(length)]: Fixed length string. Range: 1 ~ 255. Default value: 1.
- VARCHAR[(length)]: A variable-length string. The default value is 1. Unit: bytes. In versions earlier than StarRocks 2.1, the value range of `length` is 1–65533. [Preview] In StarRocks 2.1 and later versions, the value range of `length` is 1–1048576.
- HLL (1~16385 bytes): For HLL type, there's no need to specify length or default value. The length will be controlled within the system according to data aggregation. HLL

column can only be queried or used by hll_union_agg, Hll_cardinality, and hll_hash.

- BITMAP: Bitmap type does not require specified length or default value. It represents a set of unsigned bigint numbers. The largest element could be up to 2^64 - 1.

agg_type: aggregation type. If not specified, this column is key column. If specified, it is value column. The aggregation types supported are as follows:

- SUM, MAX, MIN, REPLACE
- HLL_UNION (only for HLL type)
- BITMAP_UNION(only for BITMAP)
- REPLACE_IF_NOT_NULL: This means the imported data will only be replaced when it is of non-null value. If it is of null value, StarRocks will retain the original value.

> NOTE
>
> - When the column of aggregation type BITMAP_UNION is imported, its original data types must be TINYINT, SMALLINT, INT, and BIGINT.
> - If NOT NULL is specified by REPLACE_IF_NOT_NULL column when the table was created, StarRocks will still convert the data to NULL without sending an error report to the user. With this, the user can import selected columns.

This aggregation type applies ONLY to the Aggregate table whose key_desc type is AGGREGATE KEY.

NULL | NOT NULL: Whether the column is allowed to be `NULL`. By default, `NULL` is specified for all columns in a table that uses the Duplicate Key, Aggregate, or Unique Key table. In a table that uses the Primary Key table, by default, value columns are specified with `NULL`, whereas key columns are specified with `NOT NULL`. If `NULL` values are included in the raw data, present them with `\N`. StarRocks treats `\N` as `NULL` during data loading.

DEFAULT "default_value": the default value of a column. When you load data into StarRocks, if the source field mapped onto the column is empty, StarRocks automatically fills the default value in the column. You can specify a default value in one of the following ways:

- **DEFAULT current_timestamp**: Use the current time as the default value. For more information, see current_timestamp().
- **DEFAULT `<default_value>`**: Use a given value of the column data type as the default value. For example, if the data type of the column is VARCHAR, you can specify a

VARCHAR string, such as beijing, as the default value, as presented in `DEFAULT "beijing"`. Note that default values cannot be any of the following types: ARRAY, BITMAP, JSON, HLL, and BOOLEAN.

- **DEFAULT (<expr>)**: Use the result returned by a given function as the default value. Only the uuid() and uuid_numeric() expressions are supported.

index_definition

You can only create bitmap indexes when you create tables. For more information about parameter descriptions and usage notes, see Bitmap indexing.

```
INDEX index_name (col_name[, col_name, ...]) [USING BITMAP] COMMENT
'xxxxxx'
```

ENGINE type

Default value: olap. If this parameter is not specified, an OLAP table (StarRocks native table) is created by default.

Optional value: `mysql`, `elasticsearch`, `hive`, `jdbc` (2.3 and later), `iceberg`, and `hudi` (2.2 and later). If you want to create an external table to query external data sources, specify `CREATE EXTERNAL TABLE` and set `ENGINE` to any of these values. You can refer to External table for more information.

- For MySQL, specify the following properties:

```
PROPERTIES (
    "host" = "mysql_server_host",
    "port" = "mysql_server_port",
    "user" = "your_user_name",
    "password" = "your_password",
    "database" = "database_name",
    "table" = "table_name"
)
```

Note:

"table_name" in MySQL should indicate the real table name. In contrast, "table_name" in CREATE TABLE statement indicates the name of this mysql table on StarRocks. They

can either be different or the same.

The aim of creating MySQL tables in StarRocks is to access MySQL database. StarRocks itself does not maintain or store any MySQL data.

- For Elasticsearch, specify the following properties:

```
PROPERTIES (

"hosts" = "http://192.168.0.1:8200,http://192.168.0.2:8200",
"user" = "root",
"password" = "root",
"index" = "tindex",
"type" = "doc"
)
```

  - `hosts`: the URL that is used to connect your Elasticsearch cluster. You can specify one or more URLs.
  - `user`: the account of the root user that is used to log in to your Elasticsearch cluster for which basic authentication is enabled.
  - `password`: the password of the preceding root account.
  - `index`: the index of the StarRocks table in your Elasticsearch cluster. The index name is the same as the StarRocks table name. You can set this parameter to the alias of the StarRocks table.
  - `type`: the type of index. The default value is `doc`.

- For Hive, specify the following properties:

```
PROPERTIES (

    "database" = "hive_db_name",
    "table" = "hive_table_name",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
)
```

  Here, database is the name of the corresponding database in Hive table. Table is the name of Hive table. `hive.metastore.uris` is the server address.

- For JDBC, specify the following properties:

```
PROPERTIES (
"resource"="jdbc0",
"table"="dest_tbl"
)
```

`resource` is the JDBC resource name and `table` is the destination table.

- For Iceberg, specify the following properties:

```
PROPERTIES (
"resource" = "iceberg0",
"database" = "iceberg",
"table" = "iceberg_table"
)
```

`resource` is the Iceberg resource name. `database` is the Iceberg database. `table` is the Iceberg table.

- For Hudi, specify the following properties:

```
PROPERTIES (
"resource" = "hudi0",
"database" = "hudi",
"table" = "hudi_table"
)
```

key_desc

Syntax:

```
key_type(k1[,k2 ...])
```

Data is sequenced in specified key columns and has different attributes for different key types:

- AGGREGATE KEY: Identical content in key columns will be aggregated into value columns according to the specified aggregation type. It usually applies to business scenarios such as financial statements and multi-dimensional analysis.

- UNIQUE KEY/PRIMARY KEY: Identical content in key columns will be replaced in value columns according to the import sequence. It can be applied to make addition, deletion, modification and query on key columns.
- DUPLICATE KEY: Identical content in key columns, which also exists in StarRocks at the same time. It can be used to store detailed data or data with no aggregation attributes. **DUPLICATE KEY is the default type. Data will be sequenced according to key columns.**

> **NOTE**
>
> Value columns do not need to specify aggregation types when other key_type is used to create tables with the exception of AGGREGATE KEY.

COMMENT

You can add a table comment when you create a table, optional. Note that COMMENT must be placed after `key_desc`. Otherwise, the table cannot be created.

partition_desc

Partition description can be used in the following ways:

Create partitions dynamically

Dynamic partitioning provides a time-to-live (TTL) management for partitions. StarRocks automatically creates new partitions in advance and removes expired partitions to ensure data freshness. To enable this feature, you can configure Dynamic partitioning related properties at table creation.

Create partitions one by one

Specify only the upper bound for a partition

Syntax:

```
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>", ... ] )
  , ... ]
)
```

Note:

Please use specified key columns and specified value ranges for partitioning.

- Partition name only supports [A-z0-9_]
- Columns in Range partition only support the following types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DATE, and DATETIME.
- Partitions are left closed and right open. The left boundary of the first partition is of minimum value.
- NULL value is stored only in partitions that contain minimum values. When the partition containing the minimum value is deleted, NULL values can no longer be imported.
- Partition columns can either be single columns or multiple columns. The partition values are the default minimum values.
- When only one column is specified as the partitioning column, you can set `MAXVALUE` as the upper bound for the partitioning column of the most recent partition.

```
PARTITION BY RANGE (pay_dt) (
  PARTITION p1 VALUES LESS THAN ("20210102"),
  PARTITION p2 VALUES LESS THAN ("20210103"),
  PARTITION p3 VALUES LESS THAN MAXVALUE
)
```

Please note:

- Partitions are often used for managing data related to time.
- When data backtracking is needed, you may want to consider emptying the first partition for adding partitions later when necessary.

Specify both the lower and upper bounds for a partition

Syntax:

```
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>", ... ] ) )
    , ...]
)
```

Note:

- Fixed Range is more flexible than LESS THAN. You can customize the left and right partitions.
- Fixed Range is the same as LESS THAN in the other aspects.
- When only one column is specified as the partitioning column, you can set `MAXVALUE` as the upper bound for the partitioning column of the most recent partition.

```
PARTITION BY RANGE (pay_dt) (
  PARTITION p202101 VALUES [("20210101"), ("20210201")),
  PARTITION p202102 VALUES [("20210201"), ("20210301")),
  PARTITION p202103 VALUES [("20210301"), (MAXVALUE))
)
```

Create multiple partitions in a batch

Syntax

- If the partitioning column is of a date type.

```
PARTITION BY RANGE (<partitioning_column>) (
    START ("<start_date>") END ("<end_date>") EVERY (INTERVAL <N>
<time_unit>)
)
```

- If the partitioning column is of an integer type.

```
PARTITION BY RANGE (<partitioning_column>) (
    START ("<start_integer>") END ("<end_integer>") EVERY
(<partitioning_granularity>)
)
```

Description

You can specify the start and end values in `START()` and `END()` and the time unit or partitioning granularity in `EVERY()` to create multiple partitions in a batch.

- The partitioning column can be of a date or integer type.
- If the partitioning column is of a date type, you need to use the `INTERVAL` keyword to specify the time interval. You can specify the time unit as hour (since v3.0), day, week, month, or year. The naming conventions of partitions are the same as those for dynamic partitions.

For more information, see Data distribution.

distribution_desc

Syntax:

```
DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]
```

Data in partitions can be subdivided into tablets based on the hash values of the bucketing columns and the number of buckets. We recommend that you choose the column that meets the following two requirements as the bucketing column.

- High cardinality column such as ID
- Column that is often used as a filter in queries

If such a column does not exist, you can determine the bucketing column according to the complexity of queries.

- If the query is complex, we recommend that you select a high cardinality column as the bucketing column to ensure balanced data distribution among buckets and improve cluster resource utilization.
- If the query is relatively simple, we recommend that you select the column that is often used as the query condition as the bucketing column to improve query efficiency.

If partition data cannot be evenly distributed into each tablet by using one bucketing column, you can choose multiple bucketing columns (at most three). For more information, see Choose bucketing columns.

Precautions:

- **When a table is created, you must specify the bucketing columns**.
- The values of bucketing columns cannot be updated.
- Bucketing columns cannot be modified after they are specified.
- Since StarRocks 2.5, you do not need to set the number of buckets when you create a table. StarRocks automatically sets the number of buckets. If you want to set this parameter, see Determine the number of tablets.

PROPERTIES

Specify storage medium, storage cooldown time, replica number

If the engine type is `olap`, you can specify storage medium, storage cooldown time, and replica number when you create a table.

> **NOTE**
>
> `storage_cooldown_time` can be configured only when `storage_medium` is set to `SSD`. If you want to set `storage_medium` to SSD, make sure that your cluster uses SSD disks, that is, `storage_root_path` reported by BEs includes SSD. For more information about `storage_root_path`, see Configuration.

```
PROPERTIES (
    "storage_medium" = "[SSD|HDD]",
    [ "storage_cooldown_time" = "yyyy-MM-dd HH:mm:ss", ]
```

```
    [ "replication_num" = "3" ]
)
```

storage_medium: the initial storage medium, which can be set to SSD or HDD.

> **NOTE**
>
> - From 2.5.1, the system automatically infers storage medium based on BE disk type if `storage_medium` is not explicitly specified. Inference mechanism: If `storage_root_path` reported by BEs contain only SSD, the system automatically sets this parameter to SSD. If `storage_root_path` reported by BEs contain only HDD, the system automatically sets this parameter to HDD. If `storage_root_path` reported by BEs contain both SSD and HDD, the system automatically sets this parameter to SSD. From 2.5.4 onwards, if `storage_root_path` reported by BEs contain both SSD and HDD and the property `storage_cooldown_time` is specified, `storage_medium` is set to SSD; if the property `storage_cooldown_time` is not specified, `storage_medium` is set to HDD.
> - If the FE configuration item `enable_strict_storage_medium_check` is set to `true`, the system strictly checks BE disk type when you create a table. If the storage medium you specified in CREATE TABLE is inconsistent with BE disk type, an error "Failed to find enough host in all backends with storage medium is SSD|HDD." is returned and table creation fails. If `enable_strict_storage_medium_check` is set to `false`, the system ignores this error and forcibly creates the table. However, cluster disk space may be unevenly distributed after data is loaded.

storage_cooldown_time: the storage cooldown time for a partition. If the storage medium is SSD, SSD is switched to HDD after the time specified by this parameter. Format: "yyyy-MM-dd HH:mm:ss". The specified time must be later than the current time. If this parameter is not explicitly specified, storage cooldown is not performed by default.

replication_num: number of replicas in the specified partition. Default number: 3.

If the table has only one partition, the properties belong to the table. If the table has two levels of partitions, the properties belong to each partition. You can also specify different properties for different partitions by using ALTER TABLE ADD PARTITION or ALTER TABLE MODIFY PARTITION.

Add bloom filter index for a column

If the Engine type is olap, you can specify a column to adopt bloom filter indexes.

The following limits apply when you use bloom filter index:

- You can create bloom filter indexes for all columns of a Duplicate Key or Primary Key table. For an Aggregate table or Unique Key table, you can only create bloom filter indexes for key columns.
- TINYINT, FLOAT, DOUBLE, and DECIMAL columns do not support creating bloom filter indexes.
- Bloom filter indexes can only improve the performance of queries that contain the `in` and `=` operators, such as `Select xxx from table where x in {}` and `Select xxx from table where column = xxx`. More discrete values in this column will result in more precise queries.

For more information, see Bloom filter indexing

```
PROPERTIES (
    "bloom_filter_columns"="k1,k2,k3"
)
```

Use Colocate Join

If you want to use Colocate Join attributes, specify it in `properties`.

```
PROPERTIES (
    "colocate_with"="table1"
)
```

Configure dynamic partitions

If you want to use dynamic partition attributes, please specify it in 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

Parameter	Required	Description
dynamic_partition.enable	No	Whether to enable dynamic partitioning. Valid values: `TRUE` and `FALSE`. Default value: `TRUE`.
dynamic_partition.time_unit	Yes	The time granularity for dynamically created partitions. It is a required parameter. Valid values: `DAY`, `WEEK`, and `MONTH`. The time granularity determines the suffix format for dynamically created partitions. - If the value is `DAY`, the suffix format for dynamically created partitions is `yyyyMMdd`. An example partition name suffix is `20200321`. - If the value is `WEEK`, the suffix format for dynamically created partitions is `yyyy_ww`, for example `2020_13` for the 13th week of 2020. - If the value is `MONTH`, the suffix format for dynamically created partitions is `yyyyMM`, for example `202003`.
dynamic_partition.start	No	The starting offset of dynamic partitioning. The value of this parameter must be a negative integer. The partitions before this offset will be deleted based on the current day, week, or month which is determined by `dynamic_partition.time_unit`. The default value is `Integer.MIN_VALUE`, namely, -2147483648, which means that historical partitions will not be deleted.

Parameter	Required	Description
dynamic_partition.end	Yes	The end offset of dynamic partitioning. The value of this parameter must be a positive integer. The partitions from the current day, week, or month to the end offset will be created in advance.
dynamic_partition.prefix	No	The prefix added to the names of dynamic partitions. Default value: `p`.
dynamic_partition.buckets	No	The number of buckets per dynamic partition. The default value is the same as the number of buckets determined by the reserved word `BUCKETS` or automatically set by StarRocks.

Set data compression algorithm

You can specify a data compression algorithm for a table by adding property `compression` when you create a table.

The valid values of `compression` are:

- `LZ4`: the LZ4 algorithm.
- `ZSTD`: the Zstandard algorithm.
- `ZLIB`: the zlib algorithm.
- `SNAPPY`: the Snappy algorithm.

For more information about how to choose a suitable data compression algorithm, see Data compression.

Set write quorum for data loading

If your StarRocks cluster has multiple data replicas, you can set different write quorum for tables, that is, how many replicas are required to return loading success before StarRocks can determine the loading task is successful. You can specify write quorum by adding the property `write_quorum` when you create a table. This property is supported from v2.5.

The valid values of `write_quorum` are:

- `MAJORITY`: Default value. When the **majority** of data replicas return loading success, StarRocks returns loading task success. Otherwise, StarRocks returns loading task failed.
- `ONE`: When **one** of the data replicas returns loading success, StarRocks returns loading task success. Otherwise, StarRocks returns loading task failed.
- `ALL`: When **all** of the data replicas return loading success, StarRocks returns loading task success. Otherwise, StarRocks returns loading task failed.

> **CAUTION**
>
> - Setting a low write quorum for loading increases the risk of data inaccessibility and even loss. For example, you load data into a table with one write quorum in a StarRocks cluster of two replicas, and the data was successfully loaded into only one replica. Despite that StarRocks determines the loading task succeeded, there is only one surviving replica of the data. If the server which stores the tablets of loaded data goes down, the data in these tablets becomes inaccessible. And if the disk of the server is damaged, the data is lost.
> - StarRocks returns the loading task status only after all data replicas have returned the status. StarRocks will not return the loading task status when there are replicas whose loading status is unknown. In a replica, loading timeout is also considered as loading failed.

Specify data writing and replication mode among replicas

If your StarRocks cluster has multiple data replicas, you can specify the `replicated_storage` parameter in `PROPERTIES` to configure the data writing and replication mode among replicas.

- `true` indicates "single leader replication", which means data is written only to the primary replica. Other replicas synchronize data from the primary replica. This mode significantly reduces CPU cost caused by data writing to multiple replicas. It is supported from v2.5.
- `false` (**default in v2.5**) indicates "leaderless replication", which means data is directly written to multiple replicas, without differentiating primary and secondary replicas. The CPU cost is multiplied by the number of replicas.

In most cases, using the default value gains better data writing performance. If you want to change the data writing and replication mode among replicas, run the ALTER TABLE command. Example:

```
    ALTER TABLE example_db.my_table
    SET ("replicated_storage" = "true");
```

Create rollup in bulk

You can create rollup in bulk when you create a table.

Syntax:

```
ROLLUP (rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)],...)
```

Examples

Create an Aggregate table that uses Hash bucketing and columnar storage

```
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 starrocks table"
DISTRIBUTED BY HASH(k1) BUCKETS 10
PROPERTIES ("storage_type"="column");
```

Create an Aggregate table and set the storage medium and cooldown time

```
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) BUCKETS 10
PROPERTIES(
    "storage_type"="column",
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

Or

```
CREATE TABLE example_db.table_hash
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT SUM DEFAULT "10"
)
ENGINE=olap
PRIMARY KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2) BUCKETS 10
PROPERTIES(
    "storage_type"="column",
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

Create a Duplicate Key table that uses Range partition, Hash bucketing, and column-based storage, and set the storage medium and cooldown time

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) BUCKETS 10
PROPERTIES(
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

Note:

This statement will create three data partitions:

```
( {    MIN     },   {"2014-01-01"} )
[ {"2014-01-01"},   {"2014-06-01"} )
[ {"2014-06-01"},   {"2014-12-01"} )
```

Data outside these ranges will be not be loaded.

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) BUCKETS 10
PROPERTIES(
    "storage_medium" = "SSD"
);
```

Create a MySQL external table

```
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"
)
```

Create a table that contains HLL columns

```
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) BUCKETS 10
PROPERTIES ("storage_type"="column");
```

Create a table containing BITMAP_UNION aggregation type

The original data type of `v1` and `v2` columns must be TINYINT, SMALLINT, or 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) BUCKETS 10
PROPERTIES ("storage_type"="column");
```

Create two tables that support Colocate Join

```
CREATE TABLE `t1`
(
     `id` int(11) COMMENT "",
    `value` varchar(8) COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES
(
    "colocate_with" = "t1"
);

CREATE TABLE `t2`
(
```

```
    `id` int(11) COMMENT "",
    `value` varchar(8) COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES
(
    "colocate_with" = "t1"
);
```

Create a table with bitmap index

```
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 starrocks table"
DISTRIBUTED BY HASH(k1) BUCKETS 10
PROPERTIES ("storage_type"="column");
```

Create a dynamic partition table

The dynamic partitioning function must be enabled ("dynamic_partition.enable" = "true") in FE configuration. For more information, see Configure dynamic partitions.

This example creates partitions for the next three days and deletes partitions created three days ago. For example, if today is 2020-01-08, partitions with the following names will be created: p20200108, p20200109, p20200110, p20200111, and their ranges are:

```
[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) BUCKETS 10
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"
);
```

Create a table where multiple partitions are created in a batch, and an integer type column is specified as partitioning column

In the following example, the partitioning column `datekey` is of the INT type. All the partitions are created by only one simple partition clause `START ("1") END ("5") EVERY (1)`. The range of all the partitions starts from `1` and ends at `5`, with a partition granularity of `1`:

> **NOTE**

> The partitioning column values in **START()** and **END()** need to be wrapped in quotation marks, while the partition granularity in the **EVERY()** does not need to be wrapped in quotation marks.

```
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");
```

Create a Hive external table

Before you create a Hive external table, you must have created a Hive resource and database. For more information, see External table.

```
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"
);
```

References

- SHOW CREATE TABLE

- SHOW TABLES
- USE
- ALTER TABLE
- DROP TABLE

CREATE TABLE AS SELECT

Description

You can use the CREATE TABLE AS SELECT (CTAS) statement to synchronously or asynchronously query a table and create a new table based on the query result, and then insert the query result into the new table.

Syntax

- Synchronously query a table and create a new table based on the query result, and then insert the query result into the new table.

```
CREATE TABLE [IF NOT EXISTS] [database.]table_name
[(column_name [, column_name2, ...]]
[key_desc]
[COMMENT "table comment"]
[partition_desc]
[distribution_desc]
[PROPERTIES ("key"="value", ...)]
AS SELECT query
[ ... ]
```

- Asynchronously query a table and create a new table based on the query result, and then insert the query result into the new table.

```
SUBMIT [/*+ SET_VAR(key=value) */] TASK [[database.]
<task_name>]AS
CREATE TABLE [IF NOT EXISTS] [database.]table_name
[(column_name [, column_name2, ...]]
[key_desc]
[COMMENT "table comment"]
[partition_desc]
[distribution_desc]
```

```
[PROPERTIES ("key"="value", ...)]AS SELECT query
[ ... ]
```

Parameters

Parameter	Required	Description
column_name	Yes	The name of a column in the new table. You do not need to specify the data type for the column. StarRocks automatically specifies an appropriate data type for the column . StarRocks converts FLOAT and DOUBLE data into DECIMAL(38,9) data. StarRocks also converts CHAR, VARCHAR, and STRING data into VARCHAR(65533) data.
key_desc	No	The syntax is `key_type ( <col_name1> [, <col_name2> , ...])`. **Parameters**: • `key_type`: the key type of the new table. Valid values: `DUPLICATE KEY` and `PRIMARY KEY`. Default value: `DUPLICATE KEY`. • `col_name`: the column to form the key.
COMMENT	No	The comment of the new table.
partition_desc	No	The partitioning method of the new table. If you do not specify this parameter, by default, the new table has no partition. For more information about partitioning, see CREATE TABLE.
distribution_desc	No	The bucketing method of the new table. If you do not specify this parameter, the bucket column defaults to the column with the highest cardinality collected by the cost-based optimizer (CBO). The number of buckets defaults to 10. If the CBO does not collect information about the cardinality, the bucket column

Parameter	Required	Description
		defaults to the first column in the new table. For more information about bucketing, see CREATE TABLE.
Properties	No	The properties of the new table.
AS SELECT query	Yes	The query result. You can specify columns in `... AS SELECT query`, for example, `... AS SELECT a, b, c FROM table_a;`. In this example, `a`, `b`, and `c` indicates the column names of the table that is queried. If you do not specify the column names of the new table, the column names of the new table are also `a`, `b`, and `c`. You can specify expressions in `... AS SELECT query`, for example, `... AS SELECT a+1 AS x, b+2 AS y, c*c AS z FROM table_a;`. In this example, `a+1`, `b+2`, and `c*c` indicates the column names of the table that is queried, and `x`, `y`, and `z` indicates the column names of the new table. Note: The number of columns in the new table need to be the same as the number of the columns specified in the SELECT statement . We recommend that you use column names that are easy to identify.

Usage notes

- The CTAS statement can only create a new table that meets the following requirements:
  - `ENGINE` is `OLAP`.
  - The table is a Duplicate Key table by default. You can also specify it as a Primary Key table in `key_desc`.
  - The sort keys are the first three columns, and the storage space of the data types of these three columns does not exceed 36 bytes.
- The CTAS statement does not support setting indexes for a newly created table.

- If the CTAS statement fails to execute due to reasons, such as an FE restart, one of the following issues may occur:
    - A new table is created successfully but does not contain data.
    - A new table fails to be created.
- After a new table is created, if multiple methods (such as INSERT INTO) are used to insert data into the new table, the method that completes the INSERT operation first will insert its data into the new table.
- After a new table is created, you need to manually grant permissions on this table to users.
- If you do not specify a name for a Task when you asynchronously query a table and create a new table based on the query result, StarRocks automatically generates a name for the Task.

Examples

Example 1: Synchronously query a table `order` and create a new table `order_new` based on the query result, and then insert the query result into the new table.

```
CREATE TABLE order_new
AS SELECT * FROM order;
```

Example 2: Synchronously query the `k1`, `k2`, and `k3` columns in the table `order` and create a new table `order_new` based on the query result, and then insert the query result into the new table. Additionally, set the column names of the new table to `a`, `b`, and `c`.

```
CREATE TABLE order_new (a, b, c)
AS SELECT k1, k2, k3 FROM order;
```

or

```
CREATE TABLE order_new
AS SELECT k1 AS a, k2 AS b, k3 AS c FROM order;
```

Example 3: Synchronously query the largest value of the `salary` column in the table `employee` and create a new table `employee_new` based on the query result, and then insert the query result into the new table. Additionally, set the column name of the new table to `salary_max`.

```
CREATE TABLE employee_new
AS SELECT MAX(salary) AS salary_max FROM employee;
```

After data is inserted, query the new table.

```
SELECT * FROM employee_new;

+------------+
| salary_max |
+------------+
|   10000    |
+------------+
```

Example 4: Use CTAS to create a Primary Key table. Note that the number of data rows in the Primary Key table may be less than that in the query result. It is because the Primary Key table only stores the most recent data row among a group of rows that have the same 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;
```

Example 5: Synchronously query four tables, including `lineorder`, `customer`, `supplier`, and `part` and create a new table `lineorder_flat` based on the query result, and then insert the query result to the new table. Additionally, specify the partitioning method and bucketing method for the new table.

```
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`) BUCKETS 120 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;
```

Example 6: Asynchronously query the table `order_detail` and create a new table `order_statistics` based on the query result, and then insert the query result into the new table.

```
SUBMIT TASK AS CREATE TABLE order_statistics AS SELECT COUNT(*) as
count FROM order_detail;

+----------------------------------------+-----------+
| TaskName                               | Status    |
+----------------------------------------+-----------+
| ctas-df6f7930-e7c9-11ec-abac-bab8ee315bf2 | SUBMITTED |
+----------------------------------------+-----------+
```

Check information of the Task.

```
SELECT * FROM INFORMATION_SCHEMA.tasks;

-- Information of the 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
```

Check the state of the TaskRun.

```
SELECT * FROM INFORMATION_SCHEMA.task_runs;

-- State of the 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
```

Query the new table when the state of the TaskRun is `SUCCESS`.

```
SELECT * FROM order_statistics;
```

CREATE TABLE LIKE

Description

Creates an identical empty table based on the definition of another table. The definition includes column definition, partitions, and table properties.

Syntax

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [database.]table_name LIKE
[database.]table_name
```

> **NOTE**

1. You must have the `SELECT` privilege on the original table.
2. You can copy an external table such as MySQL.

Example

1. Under the test1 Database, create an empty table with the same table structure as table1, named table2.

   ```
   CREATE TABLE test1.table2 LIKE test1.table1
   ```

2. Under the test2 Database, create an empty table with the same table structure as test1.table1, named table2.

   ```
   CREATE TABLE test2.table2 LIKE test1.table1
   ```

3. Under the test1 Database, create an empty table with the same table structure as MySQL external table, named table2.

```
CREATE TABLE test1.table2 LIKE test1.table1
```

CREATE VIEW

Description

Creates a view.

A view, or a logical view, is a virtual table whose data is derived from a query against other existing physical tables. Therefore, a view uses no physical storage, and all queries against the view are equivalent to sub-queries of the query statement used to build the view.

For information about materialized views supported by StarRocks, see Synchronous materialized views and Asynchronous materialized views.

> **CAUTION**
>
> Only users with the CREATE VIEW privilege on a specific database can perform this operation.

Syntax

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS]
[<database>.]<view_name>
(
    <column_name>[ COMMENT 'column comment']
    [, <column_name>[ COMMENT 'column comment'], ...]
)
[COMMENT 'view comment']
AS <query_statement>
```

Parameters

Parameter	Description
OR REPLACE	Replace an existing view.
database	The name of the database where the view resides.
view_name	The name of the view.
column_name	The name of the column(s) in the view. Note that the columns in the view and the columns queried in the `query_statement` must agree in number.
COMMENT	The comment on the column in the view or the view itself.
query_statement	The query statement used to create the view. It can be any query statement supported by StarRocks.

Usage notes

- Querying a view requires the SELECT privilege on the view and on its corresponding base tables.
- If the query statement used to build a view cannot be executed due to the Schema Change on the base tables, StarRocks returns an error when you query the view.

Examples

Example 1: Create a view named `example_view` in `example_db` with an aggregate query against `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;
```

Example 2: Create a view named `example_view` in the database `example_db` with an aggregate query against the table `example_table`, and specify comments for the view and

each column in it.

```
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;
```

Relevant SQLs

- SHOW CREATE VIEW
- ALTER VIEW
- DROP VIEW

DROP ANALYZE

Description

Deletes a custom collection task.

By default, StarRocks automatically collects full statistics of a table. It checks for any data updates every 5 minutes. If data change is detected, data collection will be automatically triggered. If you do not want to use automatic full collection, you can set the FE configuration item `enable_collect_full_statistic` to `false` and customize a collection task.

This statement is supported from v2.4.

Syntax

```
DROP ANALYZE <ID>
```

The task ID can be obtained by using the SHOW ANALYZE JOB statement.

Examples

```
DROP ANALYZE 266030;
```

References

CREATE ANALYZE: customize an automatic collection task.

SHOW ANALYZE JOB: view the status of an automatic collection task.

KILL ANALYZE: cancel a custom collection task that is running.

For more information about collecting statistics for CBO, see Gather statistics for CBO.

DROP CATALOG

Description

Deletes an external catalog. The internal catalog cannot be deleted. A StarRocks cluster has only one internal catalog named `default_catalog`.

Syntax

```
DROP CATALOG <catalog_name>
```

Parameters

`catalog_name`: The name of an external catalog.

Examples

Create a Hive catalog named `hive1`.

```
CREATE EXTERNAL CATALOG hive1
PROPERTIES(
  "type"="hive",
  "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Delete the Hive catalog.

```
DROP CATALOG hive1;
```

DROP DATABASE

Description

Drops a database in StarRocks.

Syntax

```
DROP DATABASE [IF EXISTS] <db_name> [FORCE]
```

Note:

1. After executing DROP DATABASE for a while, you can restore the dropped database through RECOVER statement. See RECOVER statement for more detail.
2. If DROP DATABASE FORCE is executed, the database will be deleted directly and cannot be recovered without checking whether there are unfinished activities in the database. Generally this operation is not recommended.

Examples

1. Drop database db_text.

   ```
   DROP DATABASE db_test;
   ```

References

- CREATE DATABASE
- SHOW CREATE DATABASE
- USE
- DESC

- SHOW DATABASES

DROP FUNCTION

Description

Deletes a custom function. The function can only be deleted when its name and parameter type are consistent.

Only the owner of the custom function have the permissions to delete the function.

Syntax

```
DROP FUNCTION function_name(arg_type [, ...])
```

Parameters

`function_name`: the name of function to be dropped.

`arg_type`: the argument type of function to be dropped.

Examples

1. Drop a function.

```
DROP FUNCTION my_add(INT, INT)
```

DROP INDEX

Description

This statement is used to drop a specified index on a table. Currently, only bitmap index is supported in this version.

Syntax:

```
DROP INDEX index_name ON [db_name.]table_name
```

DROP MATERIALIZED VIEW

Description

Drops a materialized view.

You cannot drop a synchronous materialized view that is being created in process with this command. To drop a synchronous materialized view that is being created in process, see Synchronous materialized View - Drop an unfinished materialized view for further instructions.

> **CAUTION**
>
> Only users with the `DROP_PRIV` privilege in the database where the base table resides can drop a materialized view.

Syntax

```
DROP MATERIALIZED VIEW [IF EXISTS] [database.]mv_name
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
IF EXISTS	no	If this parameter is specified, StarRocks will not throw an exception when deleting a materialized view that does not exist. If this parameter is not specified, the system will throw an exception when deleting a materialized view that does not exist.

Parameter	Required	Description
mv_name	yes	The name of the materialized view to delete.

Examples

Example 1: Drop an existing materialized view

1. View all existing materialized views in the database.

```
MySQL > SHOW MATERIALIZED VIEW\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. Drop the materialized view `order_mv1`.

```
DROP MATERIALIZED VIEW order_mv1;
```

3. Check if the dropped materialized view exists.

```
MySQL > SHOW MATERIALIZED VIEW;
Empty set (0.01 sec)
```

Example 2: Drop a non-existing materialized view

- If the parameter `IF EXISTS` is specified, StarRocks will not throw an exception when deleting a materialized view that does not exist.

```
MySQL > DROP MATERIALIZED VIEW IF EXISTS k1_k2;
Query OK, 0 rows affected (0.00 sec)
```

- If the parameter `IF EXISTS` is not specified, the system will throw an exception when deleting a materialized view that does not exist.

```
MySQL > DROP MATERIALIZED VIEW k1_k2;
ERROR 1064 (HY000): Materialized view k1_k2 is not find
```

DROP REPOSITORY

Description

Deletes a repository. Repositories are used to store data snapshots for data backup and restoration.

> **CAUTION**
>
> - Only root user and superuser can delete a repository.
> - This operation only deletes the mapping of the repository in StarRocks, and does not delete the actual data. You need to delete it manually in the remote storage system. After deletion, you can map to that repository again by specifying the same remote storage system path.

Syntax

```
DROP REPOSITORY <repository_name>
```

Parameters

Parameter	Description
repository_name	Name of the repository to be deleted.

Example

Example 1: deletes a repository named `oss_repo`.

```
DROP REPOSITORY `oss_repo`;
```

DROP RESOURCE

Description

This statement is used to drop an existing resource. Only root or superuser can drop resources.

Syntax:

```
DROP RESOURCE 'resource_name'
```

Examples

1. Drop Spark resource named spark0.

   ```
   DROP RESOURCE 'spark0';
   ```

2. Drop Hive resource named hive0.

   ```
   DROP RESOURCE 'hive0';
   ```

DROP STATS

Description

Deletes CBO statistics, which include basic statistics and histograms. For more information, see Gather statistics for CBO.

You can delete statistical information you do not need. When you delete statistics, both the data and metadata of the statistics are deleted, as well as the statistics in expired cache. Note that if an automatic collection task is ongoing, previously deleted statistics may be collected again. You can use `SHOW ANALYZE STATUS` to view the history of collection tasks.

This statement is supported from v2.4.

Syntax

Delete basic statistics

```
DROP STATS tbl_name
```

Delete histograms

```
ANALYZE TABLE tbl_name DROP HISTOGRAM ON col_name [, col_name];
```

References

For more information about collecting statistics for CBO, see Gather statistics for CBO.

DROP TABLE

Description

This statement is used to delete a table.

Syntax

```
DROP TABLE [IF EXISTS] [db_name.]table_name [FORCE]
```

Note:

- If a table was deleted within 24 hours by using the DROP TABLE statement, you can use the RECOVER statement to restore the table.
- If DROP Table FORCE is executed, the table will be deleted directly and cannot be recovered without checking whether there are unfinished activities in the database. Generally this operation is not recommended.

Examples

1. Drop a table.

   ```
   DROP TABLE my_table;
   ```

2. If it exists, then drop the table on the specified database.

   ```
   DROP TABLE IF EXISTS example_db.my_table;
   ```

3. Force to drop the table and clear its data on disk.

```
DROP TABLE my_table FORCE;
```

References

- CREATE TABLE
- SHOW TABLES
- SHOW CREATE TABLE
- ALTER TABLE
- SHOW ALTER TABLE

DROP VIEW

Description

This statement is used to drop a logical view VIEW

Syntax

```
DROP VIEW [IF EXISTS]
[db_name.]view_name
```

Examples

1. If it exists, then drop the view example_view on example_db.

```
DROP VIEW IF EXISTS example_db.example_view;
```

HLL

Description

HLL enables the development of programmes based on the HyperLogLog algorithm. It is used to store intermediate results of the HyperLogLog calculation process. It can only be used as the value column type of the table. HLL reduces the amount of data through aggregation to speed up the query process. There may be a 1% deviation in the estimated results.

HLL column is generated based on the imported data or data from other columns. When data is imported, the hll_hash function specifies which column will be used to generated the HLL column. HLL is often used to replace COUNT DISTINCT and quickly calculate unique views (UVs) with rollup.

Related functions:

HLL_UNION_AGG(hll): This function is an aggregate function used to estimate the cardinality of all data that meet the conditions. This can also be used to analyze functions. It only supports default window and does not support window clause.

HLL_RAW_AGG(hll): This function is an aggregate function used to aggregate fields of hll type and returns with hll type.

HLL_CARDINALITY(hll): This function is used to estimate the cardinality of a single hll column.

HLL_HASH(column_name): This generates HLL column type and is used for inserts or imports. See the instructions for the use of imports.

HLL_EMPTY: This generates empty HLL column and is used to fill in default values during inserts or imports. See the instructions for the use of imports.

Examples

1. First, create a table with hll column.

```
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) buckets 32;
```

2. Import data. Please refer to Stream Load for the import method.

```
a. Use a table column to generate hll column
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. Use a data column to generate hll column
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. Aggregate data in the following three ways: (Without aggregation, direct query on base table may be as slow as using approx_count_distinct)

```
a. Create a rollup to aggregate hll column
alter table test add rollup test_rollup(dt, set1);

b. Create another table to calculate uv and insert data into it

create table test_uv(
dt date,
id int
uv_set hll hll_union)
distributed by hash(id) buckets 32;

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

```
c. Create another table to calculate uv. Insert data and generate
hll column by testing other columns through hll_hash

create table test_uv(
dt date,
id int,
id_set hll hll_union)
distributed by hash(id) buckets 32;

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

4. Query. HLL column does not support direct query into its original values. It can be queried by matching functions.

```
a. Calculate the total nv
select HLL_UNION_AGG(uv_set) from test_uv;

b. Calculate uv for each day
select dt, HLL_CARDINALITY(uv_set) from test_uv;

c. Calculate the aggregation value of set 1 in the test table
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;
```

KILL ANALYZE

Description

Cancels a **running** collection task, including manual and custom automatic tasks.

This statement is supported from v2.4.

Syntax

```
KILL ANALYZE <ID>
```

The task ID for a manual collection task can be obtained from SHOW ANALYZE STATUS. The task ID for a custom collection task can be obtained from SHOW ANALYZE SHOW ANALYZE JOB.

References

SHOW ANALYZE STATUS

SHOW ANALYZE JOB

For more information about collecting statistics for CBO, see Gather statistics for CBO.

RECOVER

Description

This statement is used to recover deleted database, table or partition.

Syntax:

1. Recover database

```
RECOVER DATABASE <db_name>
```

2. Recover database

```
RECOVER TABLE [<db_name>.]<table_name>
```

3. Recover partition

```
RECOVER PARTITION partition_name FROM [<db_name>.]<table_name>
```

Note:

1. It can only recover meta-information deleted some time ago. The default time: one day. (You can change it through parameter configuration catalog_trash_expire_second in fe.conf.)
2. If the meta-information is deleted with an identical meta-information created, the previous one will not be recovered.

Examples

1. Recover database named example_db

```
```

```
RECOVER DATABASE example_db;
```

2. Recover table name example_tbl

```
RECOVER TABLE example_db.example_tbl;
```

3. Recover partition named p1 in the example_tbl table

```
RECOVER PARTITION p1 FROM example_tbl;
```

REFRESH EXTERNAL TABLE

Description

Updates Hive and Hudi metadata cached in StarRocks. This statement is used in one of the following scenarios:

- **External table**: When using a Hive external table or Hudi external table to query data in Apache Hive™ or Apache Hudi, you can execute this statement to update the metadata of a Hive table or Hudi table cached in StarRocks.
- **External catalog**: When using a Hive catalog or Hudi catalog to query data in Hive or Hudi, you can execute this statement to update the metadata of a Hive table or Hudi table cached in StarRocks.

Basic concepts

- **Hive external table**: is created and stored in StarRocks. You can use it to query Hive data.
- **Hudi external table**: is created and stored in StarRocks. You can use it to query Hudi data.
- **Hive table**: is created and stored in Hive.
- **Hudi table**: is created and stored in Hudi.

Syntaxes and parameters

The following describes the syntaxes and parameters based on different cases:

- External table

```
REFRESH EXTERNAL TABLE table_name
[PARTITION ('partition_name', ...)]
```

Parameter	Required	Description
table_name	Yes	The name of a Hive external table or Hudi external table.
partition_name	No	The names of the partitions of a Hive table or Hudi table. Specifying this parameter updates the metadata of the partitions of the Hive table and Hudi table cached in StarRocks.

- External catalog

```
REFRESH EXTERNAL TABLE [external_catalog.][db_name.]table_name
[PARTITION ('partition_name', ...)]
```

Parameter	Required	Description
external_catalog	No	The name of a Hive catalog or Hudi catalog.
db_name	No	The name of the database where a Hive table or Hudi table resides.
table_name	Yes	The name of a Hive table or a Hudi table.
partition_name	No	The names of the partitions of a Hive table or Hudi table. Specifying this parameter updates the metadata of the partitions of the Hive table and Hudi table cached in StarRocks.

Usage notes

Only users who have the `ALTER_PRIV` privilege can execute this statement to update the metadata of Hive tables and Hudi tables cached in StarRocks.

Examples

Usage examples in different cases are as follows:

External table

Example 1: Update the cached metadata of the corresponding Hive table in StarRocks by specifying the external table `hive1`.

```
REFRESH EXTERNAL TABLE hive1;
```

Example 2: Update the cached metadata of the partitions of the corresponding Hudi table in StarRocks by specifying the external table `hudi1` and the partitions of the corresponding Hudi table.

```
REFRESH EXTERNAL TABLE hudi1
PARTITION ('date=2022-12-20', 'date=2022-12-21');
```

External catalog

Example 1: Update the cached metadata of `hive_table` in StarRocks.

```
REFRESH EXTERNAL TABLE hive_catalog.hive_db.hive_table;
```

Or

```
USE hive_catalog.hive_db;
REFRESH EXTERNAL TABLE hive_table;
```

Example 2: Update the cached metadata of the partitions of `hudi_table` in StarRocks.

```
REFRESH EXTERNAL TABLE hudi_catalog.hudi_db.hudi_table
PARTITION ('date=2022-12-20', 'date=2022-12-21');
```

RESTORE

Description

Restores data to a specified database, table, or partition. Currently, StarRocks only supports restoring data to OLAP tables. For more information, see data backup and restoration.

RESTORE is an asynchronous operation. You can check the status of a RESTORE job using SHOW RESTORE, or cancel a RESTORE job using CANCEL RESTORE.

> **CAUTION**
>
> - Only users with the ADMIN privilege can restore data.
> - In each database, only one running BACKUP or RESTORE job is allowed each time. Otherwise, StarRocks returns an error.

Syntax

```
RESTORE SNAPSHOT <db_name>.<snapshot_name>
FROM <repository_name>
[ ON ( <table_name> [ PARTITION ( <partition_name> [, ...] ) ]
    [ AS <table_alias>] [, ...] ) ]
PROPERTIES ("key"="value", ...)
```

Parameters

Parameter	Description
db_name	Name of the database that the data is restored to.
snapshot_name	Name for the data snapshot.

Parameter	Description
repository_name	Repository name.
ON	Name of the tables to restored. The whole database is restored if this parameter is not specified.
PARTITION	Name of the partitions to be restored. The whole table is restored if this parameter is not specified. You can view the partition name using SHOW PARTITIONS.
PROPERTIES	Properties of the RESTORE operation. Valid keys: • `backup_timestamp`: Backup timestamp. **Required**. You can view backup timestamps using SHOW SNAPSHOT. • `replication_num`: Specify the number of replicas to be restored. Default: `3`. • `meta_version`: This parameter is only used as a temporary solution to restore the data backed up by the earlier version of StarRocks. The latest version of the backed up data already contains `meta version`, and you do not need to specify it. • `timeout`: Task timeout. Unit: second. Default: `86400`.

Examples

Example 1: Restores the table `backup_tbl` in the snapshot `snapshot_label1` from `example_repo` repository to the database `example_db`, and the backup timestamp is `2018-05-04-16-45-08`. Restores one 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"
);
```

Example 2: Restores partitions `p1` and `p2` of table `backup_tbl` in `snapshot_label2` and table `backup_tbl2` from `example_repo` to database `example_db`, and rename `backup_tbl2` to `new_tbl`. The backup timestamp is `2018-05-04-17-11-01`. Restores three replicas by default.

```
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"
);
```

Usage notes

- Performing backup and restore operations on global, database, table, and partition levels requires different privileges.
- In each database, only one running BACKUP or RESTORE job is allowed each time. Otherwise, StarRocks returns an error.
- Because BACKUP and RESTORE jobs occupy many resources of your StarRocks cluster, you can back up and restore your data while your StarRocks cluster is not heavily loaded.
- StarRocks does not support specifying data compression algorithms for data backup.
- Because data is backed up as snapshots, the data loaded upon snapshot generation is not included in the snapshot. Therefore, if you load data into the old cluster after the snapshot is generated and before the RESTORE job is completed, you also need to load the data into the cluster that data is restored into. It is recommended that you load data into both clusters in parallel for a period of time after the data migration is complete, and then migrate your application to the new cluster after verifying the correctness of the data and services.
- Before the RESTORE job is completed, you cannot operate the table to be restored.
- Primary Key tables cannot be restored to a StarRocks cluster earlier than v2.5.
- You do not need to create the table to be restored in the new cluster before restoring it. The RESTORE job automatically creates it.

- If there is an existing table that has a duplicated name with the table to be restored, StarRocks first checks whether or not the schema of the existing table matches that of the table to be restored. If the schemas match, StarRocks overwrites the existing table with the data in the snapshot. If the schema does not match, the RESTORE job fails. You can either rename the table to be restored using the keyword `AS`, or delete the existing table before restoring data.
- If the RESTORE job overwrites an existing database, table, or partition, the overwritten data cannot be restored after the job enters the COMMIT phase. If the RESTORE job fails or is canceled at this point, the data may be corrupted and inaccessible. In this case, you can only perform the RESTORE operation again and wait for the job to complete. Therefore, we recommend that you do not restore data by overwriting unless you are sure that the current data is no longer used. The overwrite operation first checks metadata consistency between the snapshot and the existing database, table, or partition. If an inconsistency is detected, the RESTORE operation cannot be performed.
- Currently, StarRocks does not support backing up and restoring logical views.
- Currently, StarRocks does not support backing up and restoring the configuration data related to user accounts, privileges, and resource groups.
- Currently, StarRocks does not support backing up and restoring the Colocate Join relationship among tables.

SHOW ANALYZE JOB

Description

Views the information and status of custom collection tasks.

By default, StarRocks automatically collects full statistics of a table. It checks for any data updates every 5 minutes. If data change is detected, data collection will be automatically triggered. If you do not want to use automatic full collection, you can set the FE configuration item `enable_collect_full_statistic` to `false` and customize a collection task.

This statement is supported from v2.4.

Syntax

```
SHOW ANALYZE JOB [WHERE]
```

You can filter results by using the WHERE clause. The statement returns the following columns.

Column	Description
Id	The ID of the collection task.
Database	The database name.
Table	The table name.
Columns	The column names.
Type	The type of statistics, including `FULL` and `SAMPLE`.

Column	Description
Schedule	The type of scheduling. The type is `SCHEDULE` for an automatic task.
Properties	Custom parameters.
Status	The task status, including PENDING, RUNNING, SUCCESS, and FAILED.
LastWorkTime	The time of the last collection.
Reason	The reason why the task failed. NULL is returned if task execution was successful.

Examples

```
-- View all the custom collection tasks.
SHOW ANALYZE JOB

-- View custom collection tasks of database `test`.
SHOW ANALYZE JOB where `database` = 'test';
```

References

CREATE ANALYZE: customize an automatic collection task.

DROP ANALYZE: delete a custom collection task.

KILL ANALYZE: cancel a custom collection task that is running.

For more information about collecting statistics for CBO, see Gather statistics for CBO.

SHOW ANALYZE STATUS

Description

Views the status of collection tasks.

This statement cannot be used to view the status of custom collection tasks. To view the status of custom collection tasks, use SHOW ANALYZE JOB.

This statement is supported from v2.4.

Syntax

```
SHOW ANALYZE STATUS [WHERE]
```

You can use `LILE or WHERE` to filter the information to return.

This statement returns the following columns.

List name	Description
Id	The ID of the collection task.
Database	The database name.
Table	The table name.
Columns	The column names.
Type	The type of statistics, including FULL, SAMPLE, and HISTOGRAM.

List name	Description
Schedule	The type of scheduling. `ONCE` means manual and `SCHEDULE` means automatic.
Status	The status of the task.
StartTime	The time when the task starts executing.
EndTime	The time when the task execution ends.
Properties	Custom parameters.
Reason	The reason why the task failed. NULL is returned if the execution was successful.

References

ANALYZE TABLE: create a manual collection task.

KILL ANALYZE: cancel a custom collection task that is running.

For more information about collecting statistics for CBO, see Gather statistics for CBO.

SHOW FUNCTIONS

Description

Queries all the custom (or built-in) functions under a database. If no database is specified, the current database is used by default.

Syntax

```
SHOW [FULL] [BUILTIN] FUNCTIONS [IN|FROM db] [LIKE
'function_pattern']
```

Parameters

```
full: indicates showing all the functions.

builtin: indicates showing functions provided by the system.

db: the name of the database to query.

function_pattern: the pattern used to filter function name.
```

Examples

```
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":"_ZN9starrocks_udf6AddUdfEPNS_15FunctionCo
*************************** 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":"_ZN9st

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)
```

SHOW META

Description

Views metadata of CBO statistics, including basic statistics and histograms.

This statement is supported from v2.4.

View metadata of basic statistics

Syntax

```
SHOW STATS META [WHERE]
```

This statement returns the following columns.

Column	Description
Database	The database name.
Table	The table name.
Columns	The column names.
Type	The type of statistics. `FULL` means full collection and `SAMPLE` means sampled collection.
UpdateTime	The latest statistics update time for the current table.
Properties	Custom parameters.
Healthy	The health of statistical information.

View metadata of histograms

Syntax

```
SHOW HISTOGRAM META [WHERE];
```

This statement returns the following columns.

Column	Description
Database	The database name.
Table	The table name.
Column	The columns.
Type	Type of statistics. The value is `HISTOGRAM` for histograms.
UpdateTime	The latest statistics update time for the current table.
Properties	Custom parameters.

References

For more information about collecting statistics for CBO, see Gather statistics for CBO.

SHOW RESOURCES

Description

This statement is used to show the resources that the users have permissions to use. Ordinary users can only use the resources they have permission to use, while root or admin users can show all resources.

Syntax:

```
SHOW RESOURCES
```

TRUNCATE TABLE

Description

This statement is used to truncate the specified table and partition data.

Syntax:

```
TRUNCATE TABLE [db.]tbl[ PARTITION(p1, p2, ...)]
```

Note:

1. This statement is used to truncate data while retaining tables or partitions.
2. Unlike DELETE, this statement can only empty the specified tables or partitions as a whole, and filtering conditions cannot be added.
3. Unlike DELETE, using this method to clear data will not affect query performance.
4. This statement directly deletes data. The deleted data cannot be recovered.
5. The table on which you perform this operation must be in the NORMAL state. For example, you cannot perform TRUNCATE TABLE on a table with SCHEMA CHANGE going on.

Examples

1. Truncate table `tbl` under `example_db`.

   ```
   TRUNCATE TABLE example_db.tbl;
   ```

2. Truncate partitions `p1` and `p2` in table `tbl`.

   ```
   TRUNCATE TABLE tbl PARTITION(p1, p2);
   ```

USE

Description

Specifies the active database for your session. You can then perform operations, such as creating tables and executing queries.

Syntax

```
USE [<catalog_name>.]<db_name>
```

Parameters

Parameter	Required	Description
catalog_name	No	The catalog name. • If this parameter is not specified, a database in `default_catalog` is used by default. • You must specify this parameter when you use a database from an external catalog. For more information, see Example 2. • You must specify this parameter when you switch databases between different catalogs. For more information, see Example 3. For more information about catalogs, see Overview.
db_name	Yes	The database name. The database must exist.

Examples

Example 1: Use `example_db` from `default_catalog` as the active database of your session.

```
USE default_catalog.example_db;
```

Or

```
USE example_db;
```

Example 2: Use `example_db` from `hive_catalog` as the active database of your session.

```
USE hive_catalog.example_db;
```

Example 3: Switch the active database of your session from `hive_catalog.example_table1` to `iceberg_catalog.example_table2`.

```
USE iceberg_catalog.example_table2;
```

ALTER LOAD

Description

Changes the priority of a Broker Load job that is in the **QUEUEING** or **LOADING** state. This statement is supported since v2.5.

> **NOTE**
>
> Changing the priority of a Broker Load job that is in the **LOADING** state does not affect the execution of the job.

Syntax

```
ALTER LOAD FOR <label_name>
properties
(
    'priority'='{LOWEST | LOW | NORMAL | HIGH | HIGHEST}'
)
```

Parameters

Parameter	Required	Description
label_name	Yes	The label of the load job. Format: `[<database_name>.]<label_name>`. See BROKER LOAD.
priority	Yes	The new priority that you want to specify for the load job. Valid values: `LOWEST`, `LOW`, `NORMAL`, `HIGH`, and `HIGHEST`. See BROKER LOAD.

Examples

Suppose that you have a Broker Load job whose label is `test_db.label1` and the job is in the **QUEUEING** state. If you want to run the job at the soonest, you can run the following command to change the priority of the job to `HIGHEST`:

```
ALTER LOAD FOR test_db.label1
properties
(
    'priority'='HIGHEST'
);
```

ALTER ROUTINE LOAD

Description

Alters a Routine Load job that is in the `PAUSED` state. You can execute PAUSE ROUTINE LOAD to pause a Routine Load job.

After successfully altering a Routine Load job, you can:

- Check the modifications made to the Routine Load job by using SHOW ROUTINE LOAD.
- Resume the Routine Load job by using RESUME ROUTINE LOAD.

Syntax

```
ALTER ROUTINE LOAD FOR [<db_name>.]<job_name>
[load_properties]
[job_properties]
FROM data_source
[data_source_properties]
```

Parameters

- **`[<db_name>.]<job_name>`**
  - **`db_name`**: Optional. The name of the StarRocks database.
  - **`job_name`**: Required. The name of the Routine Load job to be altered.
- **`load_properties`**

  The properties of the source data to be loaded. The syntax is as follows:

```
[COLUMNS TERMINATED BY '<column_separator>'],
[ROWS TERMINATED BY '<row_separator>'],
[COLUMNS ([<column_name> [, ...] ] [, column_assignment [, ...] ]
```

```
)],
[WHERE <expr>],
[PARTITION ([ <partition_name> [, ...] ])]
[TEMPORARY PARTITION (<temporary_partition1_name>[,
<temporary_partition2_name>,...])]
```

For detailed parameter descriptions, see CREATE ROUTINE LOAD.

- **`job_properties`**

  The properties of the load job. The syntax is as follows:

```
PROPERTIES ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

  Only the following parameters can be altered:

  - `desired_concurrent_number`
  - `max_error_number`
  - `max_batch_interval`
  - `max_batch_rows`
  - `max_batch_size`
  - `jsonpaths`
  - `json_root`
  - `strip_outer_array`
  - `strict_mode`
  - `timezone`

  For detailed parameter descriptions, see CREATE ROUTINE LOAD.

- **`data_source`** and **`data_source_properties`**

  - **`data_source`**

Required. The source of the data you want to load. Valid value: `KAFKA`.

- **`data_source_properties`**

  The properties of the data source. Currently, only the following properties can be altered:

  - `kafka_partitions` and `kafka_offsets`: Note that StarRocks only supports modifying the offset of Kafka partitions that have already been consumed but does not support adding new Kafka partitions.
  - `property.*`: Custom parameters for the data source Kafka, such as `property.kafka_default_offsets`.

Examples

1. The following example increases the value of the property `desired_concurrent_number` of the load job to `5` in order to increase the parallelism of load tasks. For details on task parallelism, see how to improve load performance.

```
ALTER ROUTINE LOAD FOR example_tbl_ordertest
PROPERTIES
(
"desired_concurrent_number" = "5"
);
```

2. The following example alters the load job's properties and data source information at the same time.

```
ALTER ROUTINE LOAD FOR example_tbl_ordertest
PROPERTIES
(
"desired_concurrent_number" = "5"
)
FROM KAFKA
(
"kafka_partitions" = "0, 1, 2",
"kafka_offsets" = "100, 200, 100",
"property.group.id" = "new_group"
);
```

3. The following example alters the filtering condition and the StarRocks partitions that data is loaded into at the same time.

```
ALTER ROUTINE LOAD FOR example_tbl_ordertest
WHERE pay_dt < 2023-06-31
PARTITION (p202306);
```

BROKER LOAD

Description

StarRocks provides the MySQL-based loading method Broker Load. After you submit a load job, StarRocks asynchronously runs the job. You need to use SHOW LOAD or `curl` to check the job result. For more information about the background information, prerequisites, principles, supported data file formats, supported external storage systems, and how to perform single-table loads and multi-table loads, see Load data from HDFS or cloud storage.

Syntax

```
LOAD LABEL [<database_name>.]<label_name>
(
    data_desc[, data_desc ...]
)
WITH BROKER
(
    StorageCredentialParams
)
[PROPERTIES
(
    opt_properties
)
]
```

Note that in StarRocks some literals are used as reserved keywords by the SQL language. Do not directly use these keywords in SQL statements. If you want to use such a keyword in an SQL statement, enclose it in a pair of backticks (`). See Keywords.

Parameters

database_name and label_name

`label_name` specifies the label of the load job.

`database_name` optionally specifies the name of the database to which the destination table belongs.

Each load job has a label that is unique across the entire database. You can use the label of a load job to view the execution status of the load job and prevent repeatedly loading the same data. When a load job enters the **FINISHED** state, its label cannot be reused. Only the label of a load job that has entered the **CANCELLED** state can be reused. In most cases, the label of a load job is reused to retry that load job and load the same data, thereby implementing Exactly-Once semantics.

For label naming conventions, see System limits.

data_desc

The description of a batch of data to be loaded. Each `data_desc` descriptor declares information such as the data source, ETL functions, destination StarRocks table, and destination partitions.

Broker Load supports loading multiple data files at a time. In one load job, you can use multiple `data_desc` descriptors to declare multiple data files you want to load, or use one `data_desc` descriptor to declare one file path from which you want to load all data files in it. Broker Load can also ensure the transactional atomicity of each load job that is run to load multiple data files. Atomicity means that the loading of multiple data files in one load job must all succeed or fail. It never happens that the loading of some data files succeeds while the loading of the other files fails.

`data_desc` supports the following syntax:

```
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"]
[(column_list)]
[COLUMNS FROM PATH AS (<partition_field_name>[,
```

```
<partition_field_name> ...])]
[SET <k1=f1(v1)>[, <k2=f2(v2)> ...]]
[WHERE predicate]
```

`data_desc` must include the following parameters:

- `file_path`

  Specifies the save path of one or more data files you want to load.

  You can specify this parameter as the save path of one data file. For example, you can specify this parameter as `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/20210411"` to load a data file named `20210411` from the path `/user/data/tablename` on the HDFS server.

  You can also specify this parameter as the save path of multiple data files by using wildcards `?`, `*`, `[]`, `{}`, or `^`. See Wildcard reference. For example, you can specify this parameter as `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/*/*"` or `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/dt=202104*/*"` to load the data files from all partitions or only `202104` partitions in the path `/user/data/tablename` on the HDFS server.

  > **NOTE**
  >
  > Wildcards can also be used to specify intermediate paths.

  In the preceding examples, the `hdfs_host` and `hdfs_port` parameters are described as follows:

  - `hdfs_host`: the IP address of the NameNode host in the HDFS cluster.
  - `hdfs_host`: the FS port of the NameNode host in the HDFS cluster. The default port number is `9000`.

  > **NOTICE**
  >
  > Broker Load supports accessing AWS S3 according to the S3 or S3A protocol. Therefore, when you load data from AWS S3, you can include `s3://` or `s3a://` as the prefix in the S3 URI that you pass as the file path.

- `INTO TABLE`

  Specifies the name of the destination StarRocks table.

`data_desc` can also optionally include the following parameters:

- `NEGATIVE`

  Revokes the loading of a specific batch of data. To achieve this, you need to load the same batch of data with the `NEGATIVE` keyword specified.

  > **NOTE**
  >
  > This parameter is valid only when the StarRocks table is an Aggregate table and all its value columns are computed by the `sum` function.

- `PARTITION`

  Specifies the partitions into which you want to load data. By default, if you do not specify this parameter, the source data will be loaded into all partitions of the StarRocks table.

- `TEMPORARY_PARTITION`

  Specifies the name of the temporary partition into which you want to load data. You can specify multiple temporary partitions, which must be separated by commas (,).

- `COLUMNS TERMINATED BY`

  Specifies the column separator used in the data file. By default, if you do not specify this parameter, this parameter defaults to `\t`, indicating tab. The column separator you specify using this parameter must be the same as the column separator that is actually used in the data file. Otherwise, the load job will fail due to inadequate data quality, and its `State` will be `CANCELLED`.

  Broker Load jobs are submitted according to the MySQL protocol. StarRocks and MySQL both escape characters in the load requests. Therefore, if the column separator is an invisible character such as tab, you must add a backslash () preceding the column separator. For example, you must input `\\t` if the column separator is `\t`, and you must input `\\n` if the column separator is `\n`. Apache Hive™ files use `\x01` as their column separator, so you must input `\\x01` if the data file is from Hive.

> **NOTE**
>
> - For CSV data, you can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
> - Null values are denoted by using `\N`. For example, a data file consists of three columns, and a record from that data file holds data in the first and third columns but no data in the second column. In this situation, you need to use `\N` in the second column to denote a null value. This means the record must be compiled as `a,\N,b` instead of `a,,b`. `a,,b` denotes that the second column of the record holds an empty string.

- `ROWS TERMINATED BY`

  Specifies the row separator used in the data file. By default, if you do not specify this parameter, this parameter defaults to `\n`, indicating line break. The row separator you specify using this parameter must be the same as the row separator that is actually used in the data file. Otherwise, the load job will fail due to inadequate data quality, and its `State` will be `CANCELLED`. This parameter is supported from v2.5.4 onwards.

  For the usage notes about the row separator, see the usage notes for the preceding `COLUMNS TERMINATED BY` parameter.

- `FORMAT AS`

  Specifies the format of the data file. Valid values: `CSV`, `Parquet`, and `ORC`. By default, if you do not specify this parameter, StarRocks determines the data file format based on the filename extension **.csv**, **.parquet**, or **.orc** specified in the `file_path` parameter.

- `column_list`

  Specifies the column mapping between the data file and the StarRocks table. Syntax: `(<column_name>[, <column_name> ...])`. The columns declared in `column_list` are mapped by name onto the StarRocks table columns.

  > **NOTE**
  >
  > If the columns of the data file are mapped in sequence onto the columns of the StarRocks table, you do not need to specify `column_list`.

If you want to skip a specific column of the data file, you only need to temporarily name that column as different from any of the StarRocks table columns. For more information, see Transform data at loading.

- `COLUMNS FROM PATH AS`

  Extracts the information about one or more partition fields from the file path you specify. This parameter is valid only when the file path contains partition fields.

  For example, if the data file is stored in the path `/path/col_name=col_value/file1` in which `col_name` is a partition field and can be mapped onto a column of the StarRocks table, you can specify this parameter as `col_name`. As such, StarRocks extracts `col_value` values from the path and loads them into the StarRocks table column onto which `col_name` is mapped.

  > **NOTE**
  >
  > This parameter is available only when you load data from HDFS.

- `SET`

  Specifies one or more functions that you want to use to convert a column of the data file. Examples:

  - The StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence. The data file consists of four columns, among which the first two columns are mapped in sequence onto `col1` and `col2` of the StarRocks table and the sum of the last two columns is mapped onto `col3` of the StarRocks table. In this case, you need to specify `column_list` as `(col1,col2,tmp_col3,tmp_col4)` and specify `(col3=tmp_col3+tmp_col4)` in the SET clause to implement data conversion.
  - The StarRocks table consists of three columns, which are `year`, `month`, and `day` in sequence. The data file consists of only one column that accommodates date and time values in `yyyy-mm-dd hh:mm:ss` format. In this case, you need to specify `column_list` as `(tmp_time)` and specify `(year = year(tmp_time), month=month(tmp_time), day=day(tmp_time))` in the SET clause to implement data conversion.

- `WHERE`

Specifies the conditions based on which you want to filter the source data. StarRocks loads only the source data that meets the filter conditions specified in the WHERE clause.

WITH BROKER

In v2.4 and earlier, input `WITH BROKER "<broker_name>"` to specify the broker you want to use. From v2.5 onwards, you no longer need to specify a broker, but you still need to retain the `WITH BROKER` keyword. For more information, see Load data from HDFS or cloud storage > Background information.

StorageCredentialParams

The authentication information used by StarRocks to access your storage system.

HDFS

Open-source HDFS supports two authentication methods: simple authentication and Kerberos authentication. Broker Load uses simple authentication by default. Open-source HDFS also supports configuring an HA mechanism for the NameNode. If you choose open-source HDFS as your storage system, you can specify the authentication configuration and HA configuration as follows:

- Authentication configuration
  - If you use simple authentication, configure `StorageCredentialParams` as follows:

```
"hadoop.security.authentication" = "simple",
"username" = "<hdfs_username>",
"password" = "<hdfs_password>"
```

The following table describes the parameters in `StorageCredentialParams`.

Parameter	Description
hadoop.security.authentication	The authentication method. Valid values: `simple` and `kerberos`. Default value: `simple`. `simple` represents simple

Parameter	Description
	authentication, meaning no authentication, and `kerberos` represents Kerberos authentication.
username	The username of the account that you want to use to access the NameNode of the HDFS cluster.
password	The password of the account that you want to use to access the NameNode of the HDFS cluster.

- If you use Kerberos authentication, configure `StorageCredentialParams` as follows:

```
"hadoop.security.authentication" = "kerberos",
"kerberos_principal" = "nn/zelda1@ZELDA.COM",
"kerberos_keytab" = "/keytab/hive.keytab",
"kerberos_keytab_content" = "YWFhYWFh"
```

The following table describes the parameters in `StorageCredentialParams`.

Parameter	Description
hadoop.security.authentication	The authentication method. Valid values: `simple` and `kerberos`. Default value: `simple`. `simple` represents simple authentication, meaning no authentication, and `kerberos` represents Kerberos authentication.
kerberos_principal	The Kerberos principal to be authenticated. Each principal consists of the following three parts to ensure that it is unique across the HDFS cluster:

Parameter	Description
	`username` or `servicename`: The name of the principal. `instance`: the name of the server that hosts the node to be authenticated in the HDFS cluster. The server name helps ensure that the principal is unique, for example, when the HDFS cluster consists of multiple DataNodes that each are independently authenticated. `realm`: The name of the realm. The realm name must be capitalized. Example: `nn/zelda1@ZELDA.COM`.
kerberos_keytab	The save path of the Kerberos keytab file.
kerberos_keytab_content	The Base64-encoded content of the the Kerberos keytab file. You can choose to specify either `kerberos_keytab` or `kerberos_keytab_content`.

If you have configured multiple Kerberos users, make sure that at least one independent broker group is deployed, and in the load statement you must input `WITH BROKER "<broker_name>"` to specify the broker group you want to use. Additionally, you must open the broker startup script file **start_broker.sh** and modify line 42 of the file to enable the brokers to read the **krb5.conf** file. Example:

```
export JAVA_OPTS="-Dlog4j2.formatMsgNoLookups=true -Xmx1024m -
Dfile.encoding=UTF-8 -Djava.security.krb5.conf=/etc/krb5.conf"
```

> **NOTE**
>
> - In the preceding example, `/etc/krb5.conf` can be replaced with your actual save path of the **krb5.conf** file. Make sure that the broker has read permissions on that file. If the broker group consists of multiple brokers, you must modify the **start_broker.sh** file on each broker node and then restart the broker nodes to make the modifications take effect.

> - You can use the SHOW BROKER statement to check for brokers that are deployed in your StarRocks cluster.

- HA configuration

  You can configure an HA mechanism for the NameNode of the HDFS cluster. This way, if the NameNode is switched over to another node, StarRocks can automatically identify the new node that serves as the NameNode. This includes the following scenarios:

  - If you load data from a single HDFS cluster that has one Kerberos user configured, both broker-based loading and broker-free loading are supported.
    - To perform broker-based loading, make sure that at least one independent broker group is deployed, and place the `hdfs-site.xml` file to the `{deploy}/conf` path on the broker node that serves the HDFS cluster. StarRocks will add the `{deploy}/conf` path to the environment variable `CLASSPATH` upon broker startup, allowing the brokers to read information about the HDFS cluster nodes.
    - To perform broker-free loading, place the `hdfs-site.xml` file to the `{deploy}/conf` paths of each FE node and each BE node.
  - If you load data from a single HDFS cluster that has multiple Kerberos users configured, only broker-based loading is supported. Make sure that at least one independent broker group is deployed, and place the `hdfs-site.xml` file to the `{deploy}/conf` path on the broker node that serves the HDFS cluster. StarRocks will add the `{deploy}/conf` path to the environment variable `CLASSPATH` upon broker startup, allowing the brokers to read information about the HDFS cluster nodes.
  - If you load data from multiple HDFS clusters (regardless of whether one or multiple Kerberos users are configured), only broker-based loading is supported. Make sure that at least one independent broker group is deployed for each of these HDFS clusters, and take one of the following actions to enable the brokers to read information about the HDFS cluster nodes:
    - Place the `hdfs-site.xml` file to the `{deploy}/conf` path on the broker node that serves each HDFS cluster. StarRocks will add the `{deploy}/conf` path to the environment variable `CLASSPATH` upon broker startup, allowing the brokers to read information about the nodes in that HDFS cluster.

- Add the following HA configuration at job creation:

```
"dfs.nameservices" = "ha_cluster",
"dfs.ha.namenodes.ha_cluster" = "ha_n1,ha_n2",
"dfs.namenode.rpc-address.ha_cluster.ha_n1" = "<hdfs_host>:<h
"dfs.namenode.rpc-address.ha_cluster.ha_n2" = "<hdfs_host>:<h
"dfs.client.failover.proxy.provider" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailover
```

The following table describes the parameters in the HA configuration.

Parameter	Description
dfs.nameservices	The name of the HDFS cluster.
dfs.ha.namenodes.XXX	The name of the NameNode in the HDFS c separate them with commas (`,`). `xxx` is t `dfs.nameservices`.
dfs.namenode.rpc-address.XXX.NN	The RPC address of the NameNode in the have specified in `dfs.ha.namenodes.XX`
dfs.client.failover.proxy.provider	The provider of the NameNode to which t `org.apache.hadoop.hdfs.server.nam`

> **NOTE**
>
> You can use the SHOW BROKER statement to check for brokers that are deployed in your StarRocks cluster.

AWS S3

If you choose AWS S3 as your storage system, take one of the following actions:

- To choose the instance profile-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the assumed role-based authentication method, configure `StorageCredentialParams` as follows:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- To choose the IAM user-based authentication method, configure `StorageCredentialParams` as follows:

```
"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>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.use_instance_profile	Yes	Specifies whether to enable the credential methods instance profile and assumed role. Valid values: `true` and `false`. Default value: `false`.
aws.s3.iam_role_arn	No	The ARN of the IAM role that has privileges on your AWS S3 bucket. If you choose assumed role as the credential method for accessing AWS S3, you must specify this parameter. Then, StarRocks will assume this role when it accesses your Hive data by using a Hive catalog.
aws.s3.region	Yes	The region in which your AWS S3 bucket resides. Example: `us-west-1`.
aws.s3.access_key	No	The access key of your AWS IAM user. If you choose IAM user as the credential

Parameter	Required	Description
		method for accessing AWS S3, you must specify this parameter. Then, StarRocks will assume this role when it accesses your Hive data by using a Hive catalog.
aws.s3.secret_key	No	The secret key of your AWS IAM user. If you choose IAM user as the credential method for accessing AWS S3, you must specify this parameter. Then, StarRocks will assume this user when it accesses your Hive data by using a Hive catalog.

For information about how to choose an authentication method for accessing AWS S3 and how to configure an access control policy in AWS IAM Console, see Authentication parameters for accessing AWS S3.

Google GCS

If you choose Google GCS as your storage system, configure `StorageCredentialParams` as follows:

```
"fs.s3a.access.key" = "<gcs_access_key>",
"fs.s3a.secret.key" = "<gcs_secret_key>",
"fs.s3a.endpoint" = "<gcs_endpoint>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

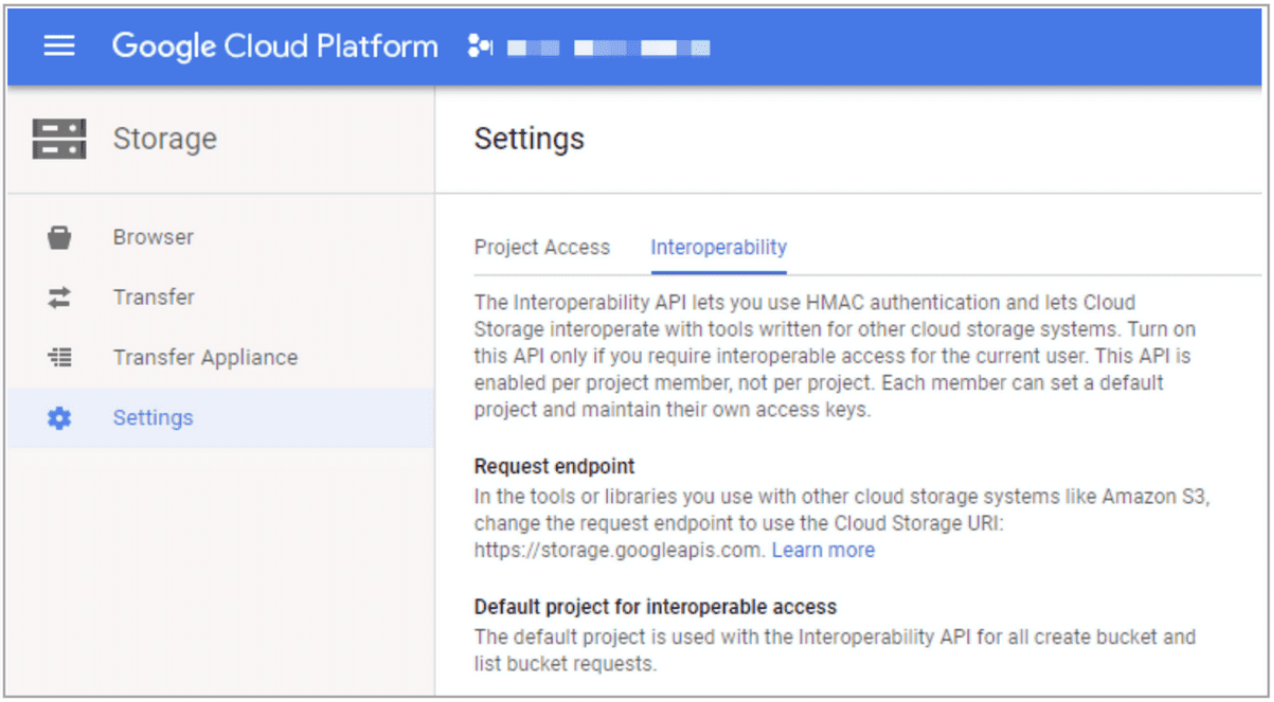
Parameter	Description
fs.s3a.access.key	The Access Key that you can use to access your Google GCS bucket.
fs.s3a.secret.key	The Secret Key that you can use to access your Google GCS bucket.

Parameter	Description
fs.s3a.endpoint	The endpoint that you can use to access your Google GCS bucket.

> **NOTE**
>
> Broker Load supports accessing Google GCS only according to the S3A protocol. Therefore, when you load data from Google GCS, you must replace the prefix in the GCS URI you pass as a file path into `DATA INFILE` with `s3a://`.

To create an Access/Secret key pair to access your Google GCS bucket, follow these steps:

1. Log in to Google GCP.
2. In the left-side navigation pane, choose **Google Cloud Storage** and then **Settings**.
3. Click the **Interoperability** tab.
4. (Optional.) If you have not enabled the Interoperability feature, click **Interoperable Access**.



5. Click the **Create new Key** button to create an Access/Secret key pair.

Other S3-compatible storage system

If you choose other S3-compatible storage system, such as MinIO, configure `StorageCredentialParams` as follows:

```
"aws.s3.enable_ssl" = "{true | false}",
"aws.s3.enable_path_style_access" = "{true | false}",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

The following table describes the parameters you need to configure in `StorageCredentialParams`.

Parameter	Required	Description
aws.s3.enable_ssl	Yes	Specifies whether to enable SSL connection. Valid values: `true` and `false`. Default value: `true`.
aws.s3.enable_path_style_access	Yes	Specifies whether to enable path-style URL access. Valid values: `true` and `false`. Default value: `false`.
aws.s3.endpoint	Yes	The endpoint that is used to connect to your AWS S3 bucket.
aws.s3.access_key	Yes	The access key of your AWS IAM user.
aws.s3.secret_key	Yes	The secret key of your AWS IAM user.

opt_properties

Specifies some optional parameters whose settings are applied to the entire load job. Syntax:

```
PROPERTIES ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

The following parameters are supported:

- `timeout`

  Specifies the timeout period of the load job. Unit: second. The default timeout period is 4 hours. We recommend that you specify a timeout period shorter than 6 hours. If the load job does not finish within the timeout period, StarRocks cancels the load job and the status of the load job becomes **CANCELLED**.

  > **NOTE**
  >
  > In most cases, you do not need to set the timeout period. We recommend that you set the timeout period only when the load job cannot finish within the default timeout period.

  Use the following formula to infer the timeout period:

  Timeout period > (Total size of the data files to be loaded x Total number of the data files to be loaded and the materialized views created on the data files)/(Average load speed x Maximum number of concurrent instances allowed per task)

  > **NOTE**
  >
  > - "Average load speed" is the average load speed for your entire StarRocks cluster. The average load speed varies for each cluster depending on the server configuration and the maximum number of concurrent query tasks allowed for the cluster. You can infer the average load speed based on the load speeds of historical load jobs.
  > - "Maximum number of concurrent instances allowed per task" is specified by the `max_broker_concurrency` parameter. For more information, see Job splitting and concurrent running.

  Suppose that you want to load a 1-GB data file on which two materialized views are created into a StarRocks cluster whose average load speed is 10 MB/s and maximum number of concurrent instances allowed per task is 3. The amount of time required for the data load is approximately 102 seconds.

  (1 x 1024 x 3)/(10 x 3) = 102 (second)

For this example, we recommend that you set the timeout period to a value greater than 102 seconds.

- `max_filter_ratio`

  Specifies the maximum error tolerance of the load job. The maximum error tolerance is the maximum percentage of rows that can be filtered out as a result of inadequate data quality. Valid values: `0`~`1`. Default value: `0`.

  - If you set this parameter to `0`, StarRocks does not ignore unqualified rows during loading. As such, if the source data contains unqualified rows, the load job fails. This helps ensure the correctness of the data loaded into StarRocks.
  - If you set this parameter to a value greater than `0`, StarRocks can ignore unqualified rows during loading. As such, the load job can succeed even if the source data contains unqualified rows.

    > **NOTE**
    >
    > Rows that are filtered out due to inadequate data quality do not include rows that are filtered out by the WHERE clause.

  If the load job fails because the maximum error tolerance is set to `0`, you can use SHOW LOAD to view the job result. Then, determine whether unqualified rows can be filtered out. If unqualified rows can be filtered out, calculate the maximum error tolerance based on the values returned for `dpp.abnorm.ALL` and `dpp.norm.ALL` in the job result, adjust the maximum error tolerance, and submit the load job again. The formula for calculating the maximum error tolerance is as follows:

  `max_filter_ratio` = [`dpp.abnorm.ALL`/(`dpp.abnorm.ALL` + `dpp.norm.ALL`)]

  The sum of the values returned for `dpp.abnorm.ALL` and `dpp.norm.ALL` is the total number of rows to be loaded.

- `load_mem_limit`

  Specifies the maximum amount of memory that can be provided to the load job. Unit: bytes: The default memory limit is 2 GB.

- `strict_mode`

Specifies whether to enable the strict mode. Valid values: `true` and `false`. Default value: `false`. `true` specifies to enable the strict mode, and `false` specifies to disable the strict mode.

- `timezone`

  Specifies the time zone of the load job. Default value: `Asia/Shanghai`. The time zone setting affects the results returned by functions such as strftime, alignment_timestamp, and from_unixtime. For more information, see Configure a time zone. The time zone specified in the `timezone` parameter is a session-level time zone.

- `priority`

  Specifies the priority of the load job. Valid values: `LOWEST`, `LOW`, `NORMAL`, `HIGH`, and `HIGHEST`. Default value: `NORMAL`. Broker Load provides the FE parameter `max_broker_load_job_concurrency`, determines the maximum number of Broker Load jobs that can be concurrently run within your StarRocks cluster. If the number of Broker Load jobs that are submitted within the specified time period exceeds the maximum number, excessive jobs in the task pool will be waiting to be scheduled based on their priorities.

  You can use the ALTER LOAD statement to change the priority of an existing load job that is in the `QUEUEING` or `LOADING` state.

  StarRocks allows setting the `priority` parameter for a Broker Load job since v2.5.

Column mapping

If the columns of the data file can be mapped one on one in sequence to the columns of the StarRocks table, you do not need to configure the column mapping between the data file and the StarRocks table.

If the columns of the data file cannot be mapped one on one in sequence to the columns of the StarRocks table, you need to use the `columns` parameter to configure the column mapping between the data file and the StarRocks table. This includes the following two use cases:

- **Same number of columns but different column sequence. Also, the data from the data file does not need to be computed by functions before it is loaded into the**

matching StarRocks table columns.

In the `columns` parameter, you need to specify the names of the StarRocks table columns in the same sequence as how the data file columns are arranged.

For example, the StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence, and the data file also consists of three columns, which can be mapped to the StarRocks table columns `col3`, `col2`, and `col1` in sequence. In this case, you need to specify `"columns: col3, col2, col1"`.

- **Different number of columns and different column sequence. Also, the data from the data file needs to be computed by functions before it is loaded into the matching StarRocks table columns.**

  In the `columns` parameter, you need to specify the names of the StarRocks table columns in the same sequence as how the data file columns are arranged and specify the functions you want to use to compute the data. Two examples are as follows:

  - The StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence. The data file consists of four columns, among which the first three columns can be mapped in sequence to the StarRocks table columns `col1`, `col2`, and `col3` and the fourth column cannot be mapped to any of the StarRocks table columns. In this case, you need to temporarily specify a name for the fourth column of the data file, and the temporary name must be different from any of the StarRocks table column names. For example, you can specify `"columns: col1, col2, col3, temp"`, in which the fourth column of the data file is temporarily named `temp`.
  - The StarRocks table consists of three columns, which are `year`, `month`, and `day` in sequence. The data file consists of only one column that accommodates date and time values in `yyyy-mm-dd hh:mm:ss` format. In this case, you can specify `"columns: col, year = year(col), month=month(col), day=day(col)"`, in which `col` is the temporary name of the data file column and the functions `year = year(col)`, `month=month(col)`, and `day=day(col)` are used to extract data from the data file column `col` and loads the data into the mapping StarRocks table columns. For example, `year = year(col)` is used to extract the `yyyy` data from the data file column `col` and loads the data into the StarRocks table column `year`.

For detailed examples, see Configure column mapping.

Examples

This section uses HDFS as an example to describe various load configurations.

Load CSV data

This section uses CSV as an example to explain the various parameter configurations that you can use to meet your diverse load requirements.

Set timeout period

Your StarRocks database `test_db` contains a table named `table1`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example1.csv` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table1`.

If you want to load all data from `example1.csv` into `table1` within up to 3600 seconds, run the following command:

```
LOAD LABEL test_db.label1
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example1.csv")
    INTO TABLE table1
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Set maximum error tolerance

Your StarRocks database `test_db` contains a table named `table2`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example2.csv` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table2`.

If you want to load all data from `example2.csv` into `table2` with a maximum error tolerance of `0.1`, run the following command:

```
LOAD LABEL test_db.label2
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example2.csv")
    INTO TABLE table2

)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "max_filter_ratio" = "0.1"
);
```

Load all data files from a file path

Your StarRocks database `test_db` contains a table named `table3`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

All data files stored in the `/user/starrocks/data/input/` path of your HDFS cluster also each consist of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table3`. The column separator used in these data files is `\x01`.

If you want to load data from all these data files stored in the `hdfs://<hdfs_host>:<hdfs_port>/user/starrocks/data/input/` path into `table3`, run the following command:

```
LOAD LABEL test_db.label3
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/*")
    INTO TABLE table3
    COLUMNS TERMINATED BY "\\x01"
```

```
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

Set NameNode HA mechanism

Your StarRocks database `test_db` contains a table named `table4`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example4.csv` also consists of three columns, which are mapped onto `col1`, `col2`, and `col3` of `table4`.

If you want to load all data from `example4.csv` into `table4` with an HA mechanism configured for the NameNode, run the following command:

```
LOAD LABEL test_db.label4
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/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.ConfiguredFailoverProxyProvi
);
```

Set Kerberos authentication

Your StarRocks database `test_db` contains a table named `table5`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example5.csv` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table5`.

If you want to load all data from `example5.csv` into `table5` with Kerberos authentication configured and the keytab file path specified, run the following command:

```
LOAD LABEL test_db.label5
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example5.csv")
    INTO TABLE table5
    COLUMNS TERMINATED BY "\t"
)
WITH BROKER
(
    "hadoop.security.authentication" = "kerberos",
    "kerberos_principal" = "starrocks@YOUR.COM",
    "kerberos_keytab" = "/home/starRocks/starRocks.keytab"
);
```

Revoke data loading

Your StarRocks database `test_db` contains a table named `table6`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example6.csv` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table6`.

You have loaded all data from `example6.csv` into `table6` by running a Broker Load job.

If you want to revoke the data you have loaded, run the following command:

```
LOAD LABEL test_db.label6
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example6.csv")
    NEGATIVE
    INTO TABLE table6
    COLUMNS TERMINATED BY "\t"
)
WITH BROKER
(
```

```
    "hadoop.security.authentication" = "kerberos",
    "kerberos_principal" = "starrocks@YOUR.COM",
    "kerberos_keytab" = "/home/starRocks/starRocks.keytab"
);
```

Specify destination partitions

Your StarRocks database `test_db` contains a table named `table7`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example7.csv` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table7`.

If you want to load all data from `example7.csv` into two partitions, `p1` and `p2`, of `table7`, run the following command:

```
LOAD LABEL test_db.label7
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example7.csv")
    INTO TABLE table7
    PARTITION (p1, p2)
    COLUMNS TERMINATED BY ","
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

Configure column mapping

Your StarRocks database `test_db` contains a table named `table8`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example8.csv` also consists of three columns, which are mapped in sequence onto `col2`, `col1`, and `col3` of `table8`.

If you want to load all data from `example8.csv` into `table8`, run the following command:

```
LOAD LABEL test_db.label8
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example8.csv")
    INTO TABLE table8
    COLUMNS TERMINATED BY ","
    (col2, col1, col3)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> In the preceding example, the columns of `example8.csv` cannot be mapped onto the columns of `table8` in the same sequence as how these columns are arranged in `table8`. Therefore, you need to use `column_list` to configure the column mapping between `example8.csv` and `table8`.

Set filter conditions

Your StarRocks database `test_db` contains a table named `table9`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example9.csv` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table9`.

If you want to load only the data records whose values in the first column are greater than `20180601` from `example9.csv` into `table9`, run the following command:

```
LOAD LABEL test_db.label9
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example9.csv")
    INTO TABLE table9
    (col1, col2, col3)
    where col1 > 20180601
)
WITH BROKER
(
```

```
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> In the preceding example, the columns of `example9.csv` can be mapped onto the columns of `table9` in the same sequence as how these columns are arranged in `table9`, but you need to use the WHERE clause to specify column-based filter conditions. Therefore, you need to use `column_list` to configure the column mapping between `example9.csv` and `table9`.

Load data into tables containing HLL-type columns

Your StarRocks database `test_db` contains a table named `table10`. The table consists of four columns, which are `id`, `col1`, `col2`, and `col3` in sequence. `col1` and `col2` are defined as HLL-type columns.

Your data file `example10.csv` consists of three columns, among which the first column is mapped onto `id` of `table10` and the second and third columns are mapped in sequence onto `col1` and `col2` of `table10`. The values in the second and third columns of `example10.csv` can be converted into HLL-type data by using functions before they are loaded into `col1` and `col2` of `table10`.

If you want to load all data from `example10.csv` into `table10`, run the following command:

```
LOAD LABEL test_db.label10
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/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**
>
> In the preceding example, the three columns of `example10.csv` are named `id`, `temp1`, and `temp2` in sequence by using `column_list`. Then, functions are used to convert data as follows:
>
> - The `hll_hash` function is used to convert the values in `temp1` and `temp2` of `example10.csv` into HLL-type data and map `temp1` and `temp2` of `example10.csv` onto `col1` and `col2` of `table10`.
> - The `hll_empty` function is used to fill the specified default value into `col3` of `table10`.

For usage of the functions `hll_hash` and `hll_empty`, see hll_hash and hll_empty.

Extract partition field values from file path

Broker Load supports parsing the values of specific partition fields contained in a file path based on the column definitions of the destination StarRocks table. This feature of StarRocks is similar to the Partition Discovery feature of Apache Spark™.

Your StarRocks database `test_db` contains a table named `table11`. The table consists of five columns, which are `col1`, `col2`, `col3`, `city`, and `utc_date` in sequence.

The file path `/user/starrocks/data/input/dir/city=beijing` of your HDFS cluster contains the following data files:

- `/user/starrocks/data/input/dir/city=beijing/utc_date=2019-06-26/0000.csv`
- `/user/starrocks/data/input/dir/city=beijing/utc_date=2019-06-26/0001.csv`

These data files each consist of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table11`.

If you want to load data from all data files from the file path `/user/starrocks/data/input/dir/city=beijing/utc_date=*/*` into `table11` and, at the same time, you want to extract the values of the partition fields `city` and `utc_date` contained in the file path and load the extracted values into `city` and `utc_date` of `table11`, run the following command:

```
LOAD LABEL test_db.label11
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/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>"
);
```

Extract partition field values from `%3A`-included file path

In HDFS, file paths cannot contain colons (:). All colons (:) will be converted into `%3A`.

Your StarRocks database `test_db` contains a table named `table12`. The table consists of three columns, which are `data_time`, `col1`, and `col2` in sequence. The table schema is as follows:

```
data_time DATETIME,
col1        INT,
col2        INT
```

The file path `/user/starrocks/data` of your HDFS cluster contains the following data files:

- `/user/starrocks/data/data_time=2020-02-17 00%3A00%3A00/example12.csv`
- `/user/starrocks/data/data_time=2020-02-18 00%3A00%3A00/example12.csv`

If you want to load all data from `example12.csv` into `table12` and, at the same time, you want to extract the values of the partition field `data_time` from the file path and load the extracted values into `data_time` of `table12`, run the following command:

```
LOAD LABEL test_db.label12
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/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**
>
> In the preceding example, the values extracted from the partition field `data_time` are strings that contain `%3A`, such as `2020-02-17 00%3A00%3A00`. Therefore, you need to use the `str_to_date` function to convert the strings into DATETIME-type data before they are loaded into `data_time` of `table8`.

Load Parquet data

This section describes some parameters settings you need to take note of when you load Parquet data.

Your StarRocks database `test_db` contains a table named `table13`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example13.parquet` also consists of three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table13`.

If you want to load all data from `example13.parquet` into `table13`, run the following command:

```
LOAD LABEL test_db.label13
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/starrocks/data/input/example13.parquet")
    INTO TABLE table13
    FORMAT AS "parquet"
    (col1, col2, col3)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> By default, when you load Parquet data, StarRocks determines the data file format based on whether the filename contains the extension **.parquet**. If the filename does not contain the extension **.parquet**, you must use `FORMAT AS` to specify the data file format as `Parquet`.

Load ORC data

This section describes some parameters settings you need to take note of when you load ORC data.

Your StarRocks database `test_db` contains a table named `table14`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example14.orc` also contains three columns, which are mapped in sequence onto `col1`, `col2`, and `col3` of `table14`.

If you want to load all data from `example14.orc` into `table14`, run the following command:

```
LOAD LABEL test_db.label14
(
    DATA INFILE("hdfs://<hdfs_host>:
```

```
<hdfs_port>/user/starrocks/data/input/example14.orc")
    INTO TABLE table14
    FORMAT AS "orc"
    (col1, col2, col3)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

NOTE

- By default, when you load ORC data, StarRocks determines the data file format based on whether the filename contains the extension **.orc**. If the filename does not contain the extension **.orc**, you must use `FORMAT AS` to specify the data file format as `ORC`.
- In StarRocks v2.3 and earlier, if the data file contains ARRAY-type columns, you must make sure that the columns of the ORC data file have the same names as their mapping columns in the StarRocks table and the columns cannot be specified in the SET clause.

CANCEL EXPORT

Description

Cancels a given data unloading job. Unloading jobs with the state `CANCELLED` or `FINISHED` cannot be canceled. Canceling an unloading job is an asynchronous process. You can use the SHOW EXPORT statement to check whether an unloading job is successfully canceled. The unloading job is successfully canceled if the value of `State` is `CANCELLED`.

The CANCEL EXPORT statement requires that you have at least one of the following privileges on the database to which the given unloading job belongs: `SELECT_PRIV`, `LOAD_PRIV`, `ALTER_PRIV`, `CREATE_PRIV`, `DROP_PRIV`, and `USAGE_PRIV`. For more information about privilege descriptions, see GRANT.

Syntax

```
CANCEL EXPORT
[FROM db_name]
WHERE QUERYID = "query_id"
```

Parameters

Parameter	Required	Description
db_name	No	The name of the databases to which the unloading job belongs. If this parameter is not specified, an unloading job in your current database is canceled.
query_id	Yes	The query ID of the unloading job. You can obtain the ID by using the LAST_QUERY_ID() function. Note that this function only returns the latest query ID.

Examples

Example 1: Cancel an unloading job whose query ID is `921d8f80-7c9d-11eb-9342-acde48001121` in the current database.

```
CANCEL EXPORT
WHERE queryid = "921d8f80-7c9d-11eb-9342-acde48001121";
```

Example 2: Cancel an unloading job whose query ID is `921d8f80-7c9d-11eb-9342-acde48001121` in the `example_db` database.

```
CANCEL EXPORT
FROM example_db
WHERE queryid = "921d8f80-7c9d-11eb-9342-acde48001122";
```

CANCEL LOAD

Description

Cancels a given load job: Broker Load, Spark Load, or INSERT. A load job in the `PREPARED`, `CANCELLED` or `FINISHED` state cannot be canceled.

Canceling a load job is an asynchronous process. You can use the SHOW LOAD statement to check whether a load job is successfully canceled. The load job is successfully canceled if the value of `State` is `CANCELLED` and the value of `type` (displayed in `ErrorMsg`) is `USER_CANCEL`.

Syntax

```
CANCEL LOAD
[FROM db_name]
WHERE LABEL = "label_name"
```

Parameters

Parameter	Required	Description
db_name	No	The name of the database to which the load job belongs. If this parameter is not specified, a load job in your current database is canceled by default.
label_name	Yes	The label of the load job.

Examples

Example 1: Cancel the load job whose label is `example_label` in the current database.

```
CANCEL LOAD
WHERE LABEL = "example_label";
```

Example 2: Cancel the load job whose label is `example_label` in the `example_db` database.

```
CANCEL LOAD
FROM example_db
WHERE LABEL = "example_label";
```

CANCEL REFRESH MATERIALIZED VIEW

Description

Cancels a refresh task for an asynchronous materialized view.

Syntax

```
CANCEL REFRESH MATERIALIZED VIEW [<database_name>.]
<materialized_view_name>
```

Parameters

Parameter	Required	Description
database_name	No	Name of the database where the materialized view resides. If this parameter is not specified, the current database is used.
materialized_view_name	Yes	Name of the materialized view.

Examples

Example 1: Cancel the refresh task for the ASYNC refresh materialized view `lo_mv1`.

```
CANCEL REFRESH MATERIALIZED VIEW lo_mv1;
```

CREATE ROUTINE LOAD

Description

Routine Load can continuously consumes messages from Apache Kafka® and loads data into StarRocks. Routine Load can consume CSV and JSON data from a Kafka cluster and access Kafka via multiple security protocols, including `plaintext`, `ssl`, `sasl_plaintext`, and `sasl_ssl`.

This topic describes the syntax, parameters, and examples of the CREATE ROUTINE LOAD statement.

> **NOTE**
>
> For information about the application scenarios, principles, and basic operations of Routine Load, see Continuously load data from Apache Kafka®.

Syntax

```
CREATE ROUTINE LOAD <database_name>.<job_name> ON <table_name>
[load_properties]
[job_properties]
FROM data_source
[data_source_properties]
```

Parameters

`database_name`, `job_name`, `table_name`

`database_name`

Optional. The name of the StarRocks database.

`job_name`

Required. The name of the Routine Load job. A table may receive data from multiple Routine Load jobs. We recommend that you set a meaningful Routine Load job name by using identifiable information, for example, Kafka topic name and the approximate job creation time, to distinguish multiple Routine Load jobs. The name of the Routine Load job must be unique within the same database.

`table_name`

Required. The name of the StarRocks table to which data is loaded.

`load_properties`

Optional. The properties of the data. Syntax:

```
[COLUMNS TERMINATED BY '<column_separator>'],
[ROWS TERMINATED BY '<row_separator>'],
[COLUMNS (<column1_name>[, <column2_name>, <column_assignment>, ...
])],
[WHERE <expr>],
[PARTITION (<partition1_name>[, <partition2_name>, ...])]
[TEMPORARY PARTITION (<temporary_partition1_name>[,
<temporary_partition2_name>, ...])]
```

`COLUMNS TERMINATED BY`

The column separator for CSV-formatted data. The default column separator is `\t` (Tab). For example, you can use `COLUMNS TERMINATED BY ","` to specify the column separator as a comma.

> **Note**
>
> - Ensure that the column separator specified here is the same as the column separator in the data to be ingested.
> - You can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
> - Null values are denoted by using `\N`. For example, a data record consists of three columns, and the data record holds data in the first and third columns but does not hold data in the second column. In this situation, you need to use `\N` in the second

> column to denote a null value. This means the record must be compiled as `a,\N,b` instead of `a,,b`. `a,,b` denotes that the second column of the record holds an empty string.

`ROWS TERMINATED BY`

The row separator for CSV-formatted data. The default row separator is `\n`.

`COLUMNS`

The mapping between the columns in the source data and the columns in the StarRocks table. For more information, see Column mapping in this topic.

- `column_name`: If a column of the source data can be mapped to a column of the StarRocks table without any computation, you only need to specify the column name. These columns can be referred to as mapped columns.
- `column_assignment`: If a column of the source data cannot be directly mapped to a column of the StarRocks table, and the column's values must be computed by using functions before data loading, you must specify the computation function in `expr`. These columns can be referred to as derived columns. It is recommended to place derived columns after mapped columns because StarRocks first parses mapped columns.

`WHERE`

The filter condition. Only data that meets the filter condition can be loaded into StarRocks. For example, if you only want to ingest rows whose `col1` value is greater than `100` and `col2` value is equal to `1000`, you can use `WHERE col1 > 100 and col2 = 1000`.

> **NOTE**
>
> The columns specified in the filter condition can be source columns or derived columns.

`PARTITION`

If a StarRocks table is distributed on partitions p0, p1, p2 and p3, and you want to load the data only to p1, p2, and p3 in StarRocks and filter out the data that will be stored in p0, then you can specify `PARTITION(p1, p2, p3)` as a filter condition. By default, if you do not specify this parameter, the data will be loaded into all the partitions. Example:

```
PARTITION (p1, p2, p3)
```

`TEMPORARY PARTITION`

The name of the temporary partition into which you want to load data. You can specify multiple temporary partitions, which must be separated by commas (,).

`job_properties`

Required. The properties of the load job. Syntax:

```
PROPERTIES ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

Property	Required	Description
desired_concurrent_number	No	The expected task parallelism of a single Rou Load job. Default value: `3`. The actual task parallelism is determined by the minimum valu the multiple parameters: `min(alive_be_num partition_number, desired_concurrent_number, max_routine_load_task_concurrent_num` • `alive_be_number`: the number of alive nodes. • `partition_number`: the number of part to be consumed. • `desired_concurrent_number`: the exp task parallelism of a single Routine Load j Default value: `3`. • `max_routine_load_task_concurrent` the default maximum task parallelism of a Routine Load job, which is `5`. See FE dyn parameter. The maximum actual task parallelism is deter by either the number of alive BE nodes or the

Property	Required	Description
		number of partitions to be consumed.
max_batch_interval	No	The scheduling interval for a task, that is, how often a task is executed. Unit: seconds. Value range: `5` ~ `60`. Default value: `10`. It is recommended to set a value larger than `10`. I scheduling is shorter than 10 seconds, too ma tablet versions are generated due to an exces high loading frequency.
max_batch_rows	No	This property is only used to define the windc error detection. The window is the number of of data consumed by a single Routine Load ta The value is `10 * max_batch_rows`. The de value is `10 * 200000 = 2000000`. The Rout Load task detects error data in the error dete window. Error data refers to data that StarRoc cannot parse, such as invalid JSON-formatte data.
max_error_number	No	The maximum number of error data rows allo within an error detection window. If the numb error data rows exceeds this value, the load jc pause. You can execute SHOW ROUTINE LOA and view the error logs by using `ErrorLogUr` After that, you can correct the error in Kafka according to the error logs. The default value which means error rows are not allowed. **NOTE** • Error data rows do not include data rows are filtered out by the WHERE clause. • This parameter, along with the next parar `max_filter_ratio`, controls the maxim number of error data records. When `max_filter_ratio` is not set, the value this parameter takes effect. When

Property	Required	Description
		`max_filter_ratio` is set, the load job pauses as soon as the number of error da record reaches either the threshold set b parameter or the `max_filter_ratio` parameter.
max_filter_ratio	No	The maximum error tolerance of the load job. error tolerance is the maximum percentage of records that can be filtered out due to inadeq data quality in all data records requested by t load job. Valid values: `0` to `1`. Default value: (We recommend that you retain the default val This way, if unqualified data records are detec the load job pauses, thereby ensuring data correctness. If you want to ignore unqualified data records can set this parameter to a value greater than This way, the load job can succeed even if the file contains unqualified data records. **NOTE** • Unqualified data records do not include c records that are filtered out by the WHER clause. • This parameter, along with the last param `max_error_number`, controls the maxim number of error data records. When this parameter is not set, the value of the `max_error_number` parameter takes eff When this parameter is set, the load job pauses as soon as the number of error da records reaches either the threshold set t this parameter or the `max_error_numbe` parameter.

Property	Required	Description
strict_mode	No	Specifies whether to enable the strict mode. V values: `true` and `false`. Default value: `fals` When the strict mode is enabled, if the value column in the loaded data is `NULL` but the tar table does not allow a `NULL` value for this col the data row will be filtered out.
timezone	No	The time zone used by the load job. Default va `Asia/Shanghai`. The value of this parameter affects the results returned by functions such strftime(), alignment_timestamp(), and from_unixtime(). The time zone specified by t parameter is a session-level time zone. For m information, see Configure a time zone.
merge_condition	No	Specifies the name of the column you want to as the condition to determine whether to upd data. Data will be updated only when the valu the data to be loaded into this column is great than or equal to the current value of this colur For more information, see Change data throug loading. **NOTE** Only tables that use the Primary Key tables support conditional updates. The column that specify cannot be a primary key column.
format	No	The format of the data to be loaded. Valid valu `CSV` and `JSON`. Default value: `CSV`.
merge_condition	No	Specifies the name of the column you want to as the condition to determine whether to upd data. Data will be updated only when the valu the data to be loaded into this column is great than or equal to the current value of this colur For more information, see Change data throug

Property	Required	Description
		loading. **NOTE** Only Primary Key tables support conditional updates. The column that you specify cannot primary key column.
strip_outer_array	No	Specifies whether to strip the outermost array structure of the JSON-formatted data. Valid values: `true` and `false`. Default value: `fals` real-world business scenarios, JSON-formatt data may have an outermost array structure a indicated by a pair of square brackets `[]`. In t situation, we recommend that you set this parameter to `true`, so StarRocks removes th outermost square brackets `[]` and loads eac inner array as a separate data record. If you s this parameter to `false`, StarRocks parses th entire JSON-formatted data into one array an loads the array as a single data record. Use th JSON-formatted data `[{"category" : 1, "author" : 2}, {"category" : 3, "aut : 4} ]` as an example. If you set this parame `true`, `{"category" : 1, "author" : 2}` `{"category" : 3, "author" : 4}` are pa as two separate data records and are loaded i two StarRocks data rows.
jsonpaths	No	The names of the fields that you want to load JSON-formatted data. The value of this paran is a valid JsonPath expression. For more information, see Configure column mapping f loading JSON-formatted data in this topic.
json_root	No	The root element of the JSON-formatted data load. StarRocks extracts the elements of the r node through `json_root` for parsing. By def

Property	Required	Description
		the value of this parameter is empty, indicatin all JSON-formatted data will be loaded. For m information, see Specify the root element of t JSON-formatted data to be loaded in this top

`data_source`, `data_source_properties`

Required. The data source and relevant properties.

```
FROM <data_source>
 ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

`data_source`

Required. The source of the data you want to load. Valid value: `KAFKA`.

`data_source_properties`

The properties of the data source.

Property	Required	Description
kafka_broker_list	Yes	Kafka's broker connection information. The is `<kafka_broker_ip>:<broker_ port>` Multiple brokers are separated by commas default port used by Kafka brokers is `9092` Example:`"kafka_broker_list" =` `""xxx.xx.xxx.xx:9092,xxx.xx.xxx.xx`
kafka_topic	Yes	The Kafka topic to be consumed. A Routine job can only consume messages from one t

Property	Required	Description
kafka_partitions	No	The Kafka partitions to be consumed, for ex `"kafka_partitions" = "0, 1, 2, 3"`. property is not specified, all partitions are consumed by default.
kafka_offsets	No	The starting offset from which to consume a Kafka partition as specified in `kafka_partitions`. If this property is not specified, the Routine Load job consumes c starting from the latest offsets in `kafka_partitions`. Valid values: • A specific offset: consumes data starti a specific offset. • `OFFSET_BEGINNING`: consumes data s from the earliest offset possible. • `OFFSET_END`: consumes data starting the latest offset. Multiple starting offsets are separated by c (,), for example, `"kafka_offsets" = "10 OFFSET_BEGINNING, OFFSET_END, 2000`
property.kafka_default_offsets	No	The default starting offset for all consumer partitions. The supported values for this pro are same as those for the `kafka_offsets` property.

More data source-related properties

You can specify additional data source (Kafka) related properties, which are equivalent to using the Kafka command line `--property`. For more supported properties, see the properties for a Kafka consumer client in librdkafka configuration properties.

> **NOTE**

> If the value of a property is a file name, add the keyword `FILE:` preceding the file name. For information about how to create a file, see CREATE FILE.

- **Specify the default initial offset for all the partitions to be consumed**

```
"property.kafka_default_offsets" = "OFFSET_BEGINNING"
```

- **Specify the ID of the consumer group used by the Routine Load job**

```
"property.group.id" = "group_id_0"
```

If `property.group.id` is not specified, StarRocks generates a random value based on the name of the Routine Load job, in the format of `{job_name}_{random uuid}`, such as `simple_job_0a64fe25-3983-44b2-a4d8-f52d3af4c3e8`.

- **Specify the security protocol and relevant parameters used by BE to access Kafka**

  The security protocol can be specified as `plaintext` (default), `ssl`, `sasl_plaintext`, or `sasl_ssl`. And you need to configure the related parameters according to the specified security protocol.

  When the security protocol is set to `sasl_plaintext` or `sasl_ssl`, the following SASL authentication mechanisms are supported:

  - PLAIN
  - SCRAM-SHA-256 and SCRAM-SHA-512
  - OAUTHBEARER

  Examples:

  - Access Kafka using the SSL security protocol:

```
"property.security.protocol" = "ssl", -- Specify the security protocol as SSL.
"property.ssl.ca.location" = "FILE:ca-cert", -- File or directory path to CA certificate(s) for verifying the kafka broker's key.
-- If the Kafka server enables client authentication, the following three parameters are also required:
"property.ssl.certificate.location" = "FILE:client.pem", --
```

```
Path to the client's public key used for authentication.
"property.ssl.key.location" = "FILE:client.key", -- Path to
the client's private key used for authentication.
"property.ssl.key.password" = "xxxxxx" -- Password for the
client's private key.
```

- Access Kafka using the SASL_PLAINTEXT security protocol and SASL/PLAIN authentication mechanism:

```
"property.security.protocol" = "SASL_PLAINTEXT", -- Specify
the security protocol as SASL_PLAINTEXT.
"property.sasl.mechanism" = "PLAIN", -- Specify the SASL
mechanism as PLAIN which is a simple username/password
authentication mechanism.
"property.sasl.username" = "admin",  -- SASL username.
"property.sasl.password" = "xxxxxx"  -- SASL password.
```

FE and BE configuration items

For FE and BE configuration items related to Routine Load, see configuration items.

Column mapping

Configure column mapping for loading CSV-formatted data

If the columns of the CSV-formatted data can be mapped one on one in sequence to the columns of the StarRocks table, you do not need to configure the column mapping between the data and the StarRocks table.

If the columns of the CSV-formatted data cannot be mapped one on one in sequence to the columns of the StarRocks table, you need to use the `columns` parameter to configure the column mapping between the data file and the StarRocks table. This includes the following two use cases:

- **Same number of columns but different column sequence. Also, the data from the data file does not need to be computed by functions before it is loaded into the matching StarRocks table columns.**

    - In the `columns` parameter, you need to specify the names of the StarRocks table columns in the same sequence as how the data file columns are arranged.

    - For example, the StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence, and the data file also consists of three columns, which can be mapped to the StarRocks table columns `col3`, `col2`, and `col1` in sequence. In this case, you need to specify `"columns: col3, col2, col1"`.

- **Different number of columns and different column sequence. Also, the data from the data file needs to be computed by functions before it is loaded into the matching StarRocks table columns.**

  In the `columns` parameter, you need to specify the names of the StarRocks table columns in the same sequence as how the data file columns are arranged and specify the functions you want to use to compute the data. Two examples are as follows:

    - The StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence. The data file consists of four columns, among which the first three columns can be mapped in sequence to the StarRocks table columns `col1`, `col2`, and `col3` and the fourth column cannot be mapped to any of the StarRocks table columns. In this case, you need to temporarily specify a name for the fourth column of the data file, and the temporary name must be different from any of the StarRocks table column names. For example, you can specify `"columns: col1, col2, col3, temp"`, in which the fourth column of the data file is temporarily named `temp`.
    - The StarRocks table consists of three columns, which are `year`, `month`, and `day` in sequence. The data file consists of only one column that accommodates date and time values in `yyyy-mm-dd hh:mm:ss` format. In this case, you can specify `"columns: col, year = year(col), month=month(col), day=day(col)"`, in which `col` is the temporary name of the data file column and the functions `year = year(col)`, `month=month(col)`, and `day=day(col)` are used to extract data from the data file column `col` and loads the data into the mapping StarRocks table columns. For example, `year = year(col)` is used to extract the `yyyy` data from the data file column `col` and loads the data into the StarRocks table column `year`.

For more examples, see Configure column mapping.

Configure column mapping for loading JSON-formatted data

If the keys of the JSON-formatted data have the same names as the columns of the StarRocks table, you can load the JSON-formatted data by using the simple mode. In simple mode, you do not need to specify the `jsonpaths` parameter. This mode requires that the JSON-formatted data must be an object as indicated by curly brackets `{}`, such as `{"category": 1, "author": 2, "price": "3"}`. In this example, `category`, `author`, and `price` are key names, and these keys can be mapped one on one by name to the columns `category`, `author`, and `price` of the StarRocks table. For examples, please see [simple mode](#Column names of the target table are consistent with JSON keys).

If the keys of the JSON-formatted data have different names than the columns of the StarRocks table, you can load the JSON-formatted data by using the matched mode. In matched mode, you need to use the `jsonpaths` and `COLUMNS` parameters to specify the column mapping between the JSON-formatted data and the StarRocks table:

- In the `jsonpaths` parameter, specify the JSON keys in the sequence as how they are arranged in the JSON-formatted data.
- In the `COLUMNS` parameter, specify the mapping between the JSON keys and the StarRocks table columns:
  - The column names specified in the `COLUMNS` parameter are mapped one on one in sequence to the JSON-formatted data.
  - The column names specified in the `COLUMNS` parameter are mapped one on one by name to the StarRocks table columns.

For examples, see matched mode.

Examples

Load CSV-formatted data

This section uses CSV-formatted data as an example to describe how you can employ various parameter settings and combinations to meet your diverse loading requirements.

Prepare a dataset

Suppose you want to load CSV-formatted data from a Kafka topic named `ordertest1`.Every message in the dataset includes six columns: order ID, payment date, customer name, nationality, gender, and price.

```
2020050802,2020-05-08,Johann Georg Faust,Deutschland,male,895
2020050802,2020-05-08,Julien Sorel,France,male,893
2020050803,2020-05-08,Dorian Grey,UK,male,1262
2020050901,2020-05-09,Anna Karenina,Russia,female,175
2020051001,2020-05-10,Tess Durbeyfield,US,female,986
2020051101,2020-05-11,Edogawa Conan,japan,male,8924
```

Create a table

According to the columns of CSV-formatted data, create a table named `example_tbl1` in the database `example_db`.

```
CREATE TABLE example_db.example_tbl1 (
    `order_id` bigint NOT NULL COMMENT "Order ID",
    `pay_dt` date NOT NULL COMMENT "Payment date",
    `customer_name` varchar(26) NULL COMMENT "Customer name",
    `nationality` varchar(26) NULL COMMENT "Nationality",
    `gender` varchar(26) NULL COMMENT "Gender",
    `price` double NULL COMMENT "Price")
DUPLICATE KEY (order_id,pay_dt)
DISTRIBUTED BY HASH(`order_id`) BUCKETS 5;
```

Consume data starting from specified offsets for specified partitions

If the Routine Load job needs to consume data starting from specified partitions and offsets, you need to configure the parameters `kafka_partitions` and `kafka_offsets`.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON
example_tbl1
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, gender, price)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1",
    "kafka_partitions" ="0,1,2,3,4", -- partitions to be consumed
```

```
    "kafka_offsets" = "1000, OFFSET_BEGINNING, OFFSET_END, 2000" --
corresponding initial offsets
);
```

Improve loading performance by increasing task parallelism

To improve loading performance and avoid accumulative consumption, you can increase task parallelism by increasing the `desired_concurrent_number` value when you create the Routine Load job. Task parallelism allows splitting one Routine Load job into as many parallel tasks as possible.

> **Note**
>
> For more ways to improve loading performance, see Routine Load FAQ.

Note that the actual task parallelism is determined by the minimum value among the following multiple parameters:

```
min(alive_be_number, partition_number, desired_concurrent_number,
max_routine_load_task_concurrent_num)
```

> **Note**
>
> The maximum actual task parallelism is either the number of alive BE nodes or the number of partitions to be consumed.

Therefore, when the number of alive BE nodes and the number of partitions to be consumed are greater than the values of the other two parameters `max_routine_load_task_concurrent_num` and `desired_concurrent_number`, you can increase the values of the other two parameters to increase the actual task parallelism.

Assume that the number of partitions to be consumed is 7, the number of alive BE nodes is 5, and `max_routine_load_task_concurrent_num` is the default value `5`. If you want to increase the actual task parallelism, you can set `desired_concurrent_number` to `5` (the default value is `3`). In this case, the actual task parallelism `min(5,7,5,5)` is configured at `5`.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON
example_tbl1
COLUMNS TERMINATED BY ",",
```

```
COLUMNS (order_id, pay_dt, customer_name, nationality, gender, price)
PROPERTIES
(
"desired_concurrent_number" = "5" -- set the value of
desired_concurrent_number to 5
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1"
);
```

Configure column mapping

If the sequence of columns in the CSV-formatted data is inconsistent with the columns in the target table, assuming that the fifth column in the CSV-formatted data does not need to be imported to the target table, you need to specify the column mapping between the CSV-formatted data and the target table through the `COLUMNS` parameter.

Target database and table

Create the target table `example_tbl2` in the target database `example_db` according to the columns in the CSV-formatted data. In this scenario, you need to create five columns corresponding to the five columns in the CSV-formatted data, except for the fifth column that stores gender.

```
CREATE TABLE example_db.example_tbl2 (
    `order_id` bigint NOT NULL COMMENT "Order ID",
    `pay_dt` date NOT NULL COMMENT "Payment date",
    `customer_name` varchar(26) NULL COMMENT "Customer name",
    `nationality` varchar(26) NULL COMMENT "Nationality",
    `price` double NULL COMMENT "Price"
)
DUPLICATE KEY (order_id,pay_dt)
DISTRIBUTED BY HASH(order_id) BUCKETS 5;
```

Routine Load job

In this example, since the fifth column in the CSV-formatted data does not need to be loaded to the target table, the fifth column is temporarily named `temp_gender` in `COLUMNS`, and the other columns are directly mapped to the table `example_tbl2`.

```
CREATE ROUTINE LOAD example_db.example_tbl2_ordertest1 ON
example_tbl2
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, temp_gender,
price)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1"
);
```

Set filter conditions

If you want to load only data that meets certain conditions, you can set filter conditions in the `WHERE` clause, for example, `price > 100.`

```
CREATE ROUTINE LOAD example_db.example_tbl2_ordertest1 ON
example_tbl2
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, gender,
price),
WHERE price > 100 -- set the filter condition
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1"
);
```

Enable strict mode to filter out rows with NULL values

In `PROPERTIES`, you can set `"strict_mode" = "true"`, which means that the Routine Load job is in strict mode. If there is a `NULL` value in a source column, but the destination StarRocks table column does not allow NULL values, the row that holds a NULL value in the source column is filtered out.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON
example_tbl1
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, gender, price)
```

```
PROPERTIES
(
"strict_mode" = "true" -- enable the strict mode
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1"
);
```

Set error tolerance

If your business scenario has low tolerance for unqualified data, you need to set the error detection window and the maximum number of error data rows by configuring the parameters `max_batch_rows` and `max_error_number`. When the number of error data rows within an error detection window exceeds the value of `max_error_number`, the Routine Load job pauses.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON
example_tbl1
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, gender, price)
PROPERTIES
(
"max_batch_rows" = "100000",-- The value of max_batch_rows multiplied
by 10 equals the error detection window.
"max_error_number" = "100" -- The maximum number of error data rows
allowed within an error detection window.
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1"
);
```

Specify the security protocol as SSL and configure the related parameters

If you need to specify the security protocol as SSL used by BE to access Kafka, you need to configure `"property.security.protocol" = "ssl"` and related parameters.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON
example_tbl1
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, gender, price)
PROPERTIES
(
-- Specify the security protocol as SSL.
"property.security.protocol" = "ssl",
-- The location of the CA certificate.
"property.ssl.ca.location" = "FILE:ca-cert",
-- If authentication is enabled for Kafka clients, you need to
configure the following properties:
-- The location of the Kafka client's public key.
"property.ssl.certificate.location" = "FILE:client.pem",
-- The location of the Kafka client's private key.
"property.ssl.key.location" = "FILE:client.key",
-- The password to the Kafka client's private key.
"property.ssl.key.password" = "abcdefg"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1"
);
```

Load JSON-formatted data

StarRocks table column names consistent with JSON key names

Prepare a dataset

For example, the following JSON-formatted data exists in the Kafka topic `ordertest2`.

```
{"commodity_id": "1", "customer_name": "Mark Twain", "country":
"US","pay_time": 1589191487,"price": 875}
{"commodity_id": "2", "customer_name": "Oscar Wilde", "country":
"UK","pay_time": 1589191487,"price": 895}
{"commodity_id": "3", "customer_name": "Antoine de Saint-
Exupéry","country": "France","pay_time": 1589191487,"price": 895}
```

> **Note** Each JSON object must be in one Kafka message. Otherwise, an error that indicates a failure in parsing JSON-formatted data occurs.

Target database and table

Create table `example_tbl3` in the target database `example_db` in the StarRocks cluster. The column names are consistent with the keys names in the JSON-formatted data.

```
CREATE TABLE example_db.example_tbl3 (
    commodity_id varchar(26) NULL,
    customer_name varchar(26) NULL,
    country varchar(26) NULL,
    pay_time bigint(20) NULL,
    price double SUM NULL COMMENT "Price")
AGGREGATE KEY(commodity_id,customer_name,country,pay_time)
DISTRIBUTED BY HASH(commodity_id) BUCKETS 5;
```

Routine Load job

You can use the simple mode for the Routine Load job. That is, you do not need to specify `jsonpaths` and `COLUMNS` parameters when creating the Routine Load job. StarRocks extracts the keys of JSON-formatted data in the topic `ordertest2` of the Kafka cluster according to the column names of the target table `example_tbl3` and loads the JSON-formatted data into the target table.

```
CREATE ROUTINE LOAD example_db.example_tbl3_ordertest2 ON
example_tbl3
PROPERTIES
(
    "format" ="json"
 )
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest2"
);
```

> **Note**

- If the outermost layer of the JSON-formatted data is an array structure, you need to set `"strip_outer_array"="true"` in `PROPERTIES` to strip the outermost array structure. Additionally, when you need to specify `jsonpaths`, the root element of the entire JSON-formatted data is the flattened JSON object because the outermost array structure of the JSON-formatted data is stripped.
- You can use `json_root` to specify the root element of the JSON-formatted data.

StarRocks table contains derived columns generated using expressions

Prepare a dataset

For example, the following JSON-formatted data exists in the topic `ordertest2` of the Kafka cluster.

```
{"commodity_id": "1", "customer_name": "Mark Twain", "country":
"US","pay_time": 1589191487,"price": 875}
{"commodity_id": "2", "customer_name": "Oscar Wilde", "country":
"UK","pay_time": 1589191487,"price": 895}
{"commodity_id": "3", "customer_name": "Antoine de Saint-
Exupéry","country": "France","pay_time": 1589191487,"price": 895}
```

Target database and table

Create a table named `example_tbl4` in the database `example_db` in the StarRocks cluster. The column `pay_dt` is a derived column whose values are generated by computing values of the key `pay_time` in the JSON-formatted data.

```
CREATE TABLE example_db.example_tbl4 (
    `commodity_id` varchar(26) NULL,
    `customer_name` varchar(26) NULL,
    `country` varchar(26) NULL,
    `pay_time` bigint(20) NULL,
    `pay_dt` date NULL,
    `price` double SUM NULL)
AGGREGATE
KEY(`commodity_id`,`customer_name`,`country`,`pay_time`,`pay_dt`)
DISTRIBUTED BY HASH(`commodity_id`) BUCKETS 5;
```

Routine Load job

You can use the matched mode for the Routine Load job. That is, you need to specify `jsonpaths` and `COLUMNS` parameters when creating the Routine Load job.

You need to specify the keys of the JSON-formatted data and arrange them in sequence in the `jsonpaths` parameter.

And since the values in the key `pay_time` of the JSON-formatted data need to be converted to the DATE type before the values are stored in the `pay_dt` column of the `example_tbl4` table, you need to specify the computation by using `pay_dt=from_unixtime(pay_time,'%Y%m%d')` in `COLUMNS`. The values of other keys in the JSON-formatted data can be directly mapped to the `example_tbl4` table.

```
CREATE ROUTINE LOAD example_db.example_tbl4_ordertest2 ON example_tbl4
COLUMNS(commodity_id, customer_name, country, pay_time,
pay_dt=from_unixtime(pay_time, '%Y%m%d'), price)
PROPERTIES
(
    "format" ="json",
    "jsonpaths" ="
[\"$.commodity_id\",\"$.customer_name\",\"$.country\",\"$.pay_time\",\"
 )
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest2"
);
```

> **Note**
>
> - If the outermost layer of the JSON data is an array structure, you need to set `"strip_outer_array"="true"` in the `PROPERTIES` to strip the outermost array structure. Additionally, when you need to specify `jsonpaths`, the root element of the entire JSON data is the flattened JSON object because the outermost array structure of the JSON data is stripped.
> - You can use `json_root` to specify the root element of the JSON-formatted data.

StarRocks table contains derived column generated using CASE expression

Prepare a dataset

For example, the following JSON-formatted data exists in the Kafka topic `topic-expr-test`.

```
{"key1":1, "key2": 21}
{"key1":12, "key2": 22}
{"key1":13, "key2": 23}
{"key1":14, "key2": 24}
```

Target database and table

Create a table named `tbl_expr_test` in the database `example_db` in the StarRocks cluster. The target table `tbl_expr_test` contains two columns, where the values of the `col2` column need to be computed by using case expression on JSON data.

```
CREATE TABLE tbl_expr_test (
    col1 string, col2 string)
DISTRIBUTED BY HASH (col1);
```

Routine Load job

Because the values in the `col2` column in the target table are generated using a CASE expression, you need to specify the corresponding expression in the `COLUMNS` parameter for the Routine load job.

```
CREATE ROUTINE LOAD rl_expr_test ON tbl_expr_test
COLUMNS (
      key1,
      key2,
      col1 = key1,
      col2 = CASE WHEN key1 = "1" THEN "key1=1"
                  WHEN key1 = "12" THEN "key1=12"
                  ELSE "nothing" END)
PROPERTIES ("format" = "json")
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "topic-expr-test"
);
```

Query StarRocks table

Query the StarRocks table. The result shows that the values in the `col2` column are the output of the CASE expression.

```
MySQL [example_db]> SELECT * FROM tbl_expr_test;
+------+---------+
| col1 | col2    |
+------+---------+
| 1    | key1=1  |
| 12   | key1=12 |
| 13   | nothing |
| 14   | nothing |
+------+---------+
4 rows in set (0.015 sec)
```

Specify the root element of the JSON-formatted data to be loaded

You need to use `json_root` to specify the root element of the JSON-formatted data to be loaded and the value must be a valid JsonPath expression.

Prepare a dataset

For example, the following JSON-formatted data exists in the topic `ordertest3` of the Kafka cluster. And the root element of the JSON-formatted data to be loaded is `$.RECORDS`.

```
{"RECORDS":[{"commodity_id": "1", "customer_name": "Mark Twain",
"country": "US","pay_time": 1589191487,"price": 875},{"commodity_id":
"2", "customer_name": "Oscar Wilde", "country": "UK","pay_time":
1589191487,"price": 895},{"commodity_id": "3", "customer_name":
"Antoine de Saint-Exupéry","country": "France","pay_time":
1589191487,"price": 895}]}
```

Target database and table

Create a table named `example_tbl3` in the database `example_db` in the StarRocks cluster.

```
CREATE TABLE example_db.example_tbl3 (
    commodity_id varchar(26) NULL,
```

```
    customer_name varchar(26) NULL,
    country varchar(26) NULL,
    pay_time bigint(20) NULL,
    price double SUM NULL)
AGGREGATE KEY(commodity_id,customer_name,country,pay_time)
ENGINE=OLAP
DISTRIBUTED BY HASH(commodity_id) BUCKETS 5;
```

Routine Load job

You can set `"json_root" = "$.RECORDS"` in `PROPERTIES` to specify the root element of the JSON-formatted data to be loaded. Also, since the JSON-formatted data to be loaded is in an array structure, you must also set `"strip_outer_array" = "true"` to strip the outermost array structure.

```
CREATE ROUTINE LOAD example_db.example_tbl3_ordertest3 ON
example_tbl3
PROPERTIES
(
    "format" ="json",
    "json_root" = "$.RECORDS",
    "strip_outer_array" = "true"
 )
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest2"
);
```

DELETE

Deletes data rows from a table based on the specified conditions. The table can be a partitioned or non-partitioned table.

For Duplicate Key tables, Aggregate tables, and Unique Key tables, you can delete data from specified partitions. From v2.3, Primary Key tables support complete `DELETE...WHERE` semantics, which allows you to delete data rows based on the primary key, any column, or the results of a subquery.

Usage notes

- You must have privileges on the table and database you want to perform DELETE.
- Frequent DELETE operations are not recommended. If needed, perform such operations during off-peak hours.
- The DELETE operation only deletes data in the table. The table remains. To drop the table, run DROP TABLE.
- To prevent misoperations from deleting data in the entire table, you must specify the WHERE clause in the DELETE statement.
- The deleted rows are not immediately cleaned. They are marked as "deleted" and will be temporarily saved in Segment. Physically, the rows are removed only after data version merge (compaction) is completed.
- This operation also deletes data of the materialized views that reference this table.

Duplicate Key tables, Aggregate tables, and Unique Key tables

Syntax

```
DELETE FROM  [<db_name>.]<table_name> [PARTITION <partition_name>]
WHERE
```

```
column_name1 op { value | value_list } [ AND column_name2 op { value
| value_list } ...]
```

Parameters

Parameter	Required	Description
`db_name`	No	The database to which the destination table belongs. If this parameter is not specified, the current database is used by default.
`table_name`	Yes	The table from which you want to delete data.
`partition_name`	No	The partition from which you want to delete data.
`column_name`	Yes	The column you want to use as the DELETE condition. You can specify one or more columns.
`op`	Yes	The operator used in the DELETE condition. The supported operators are `=`, `>`, `<`, `>=`, `<=`, `!=`, `IN`, and `NOT IN`.

Limits and usage notes

- For Duplicate Key tables, you can use **any column** as the DELETE condition. For Aggregate tables and Unique Key tables, only **key columns** can be used as the DELETE condition.
- The conditions that you specify must be in the AND relation. If you want to specify conditions in OR relation, you must specify the conditions in separate DELETE statements.
- For Duplicate Key tables, Aggregate tables, and Unique Key tables, the DELETE statement does not support using subquery results as conditions.

Impacts

After you execute the DELETE statement, the query performance of your cluster may deteriorate for a period of time (before compaction is completed). The degree of deterioration varies based on the number of conditions that you specify. A larger number of conditions indicates a higher degree of deterioration.

Examples

Create a table and insert data

The following example creates a partitioned Duplicate Key table.

```
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`) BUCKETS 1
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');
```

Query data

```
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  |
+------------+------+------+
```

Delete data

Delete data from a specified partition

Delete rows whose `k1` values are `3` from the `p1` partition.

```
DELETE FROM my_table PARTITION p1
WHERE k1 = 3;

-- The query result shows rows whose `k1` values are `3` are deleted.

select * from my_table partition (p1);
+------------+------+------+
| date       | k1   | k2   |
+------------+------+------+
| 2022-03-11 |    2 | acb  |
| 2022-03-11 |    4 | abc  |
+------------+------+------+
```

Delete data from a specified partition using AND

Delete rows whose `k1` values are greater than or equal to `3` and whose `k2` values are `"abc"` from the `p1` partition.

```
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  |
+------------+------+------+
```

Delete data from all partitions

Delete rows whose `k2` values are `"abc"` or `"cba"` from all partitions.

```
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 tables

From v2.3, Primary Key tables support complete `DELETE...WHERE` semantics, which allows you to delete data rows based on the primary key, any column, or a subquery.

Syntax

```
DELETE FROM <table_name> WHERE <condition>;
```

Parameters

Parameter	Required	Description
`table_name`	Yes	The table from which you want to delete data.
`condition`	Yes	The conditions based on which to delete data. You can specify one or more conditions. This parameter is required to prevent misoperations from deleting the entire table.

Limits and usage notes

- Primary Key tables do not support deleting data from a specified partition, for example, `DELETE FROM <table_name> PARTITION <partition_id> WHERE <where_condition>`.

- The following comparison operators are supported: `=`, `>`, `<`, `>=`, `<=`, `!=`, `IN`, `NOT IN`.
- The following logical operators are supported: `AND` and `OR`.
- You cannot use the DELETE statement to run concurrent DELETE operations or to delete data at data loading. If you perform such operations, the atomicity, consistency, isolation, and durability (ACID) of transactions may not be ensured.

Examples

Create a table and insert data

Create a Primary Key table named `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`) BUCKETS 1
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT",
"enable_persistent_index" = "false"
);

INSERT INTO score_board VALUES
(0, 'Jack', 21),
(1, 'Bob', 21),
(2, 'Stan', 21),
(3, 'Sam', 22);
```

Query data

Execute the following statement to insert data into the `score_board` table:

```
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)
```

Delete data

Delete data by primary key

You can specify the primary key in the DELETE statement, so StarRocks does not need to scan the entire table.

Delete rows whose `id` values are `0` from the `score_board` table.

```
DELETE FROM score_board WHERE id = 0;

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    1 | Bob  |   21 |
|    2 | Stan |   21 |
|    3 | Sam  |   22 |
+------+------+------+
```

Delete data by condition

Example 1: Delete rows whose `score` values are `22` from the `score_board` table.

```
DELETE FROM score_board WHERE score = 22;

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    0 | Jack |   21 |
|    1 | Bob  |   21 |
|    2 | Stan |   21 |
+------+------+------+
```

Example 2: Delete rows whose `score` values are less than `22` from the `score_board` table.

```
DELETE FROM score_board WHERE score < 22;

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    3 | Sam  |   22 |
+------+------+------+
```

Example 3: Delete rows whose `score` values are less than `22` and whose `name` values are not `Bob` from the `score_board` table.

```
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)
```

Delete data by subquery result

You can nest one or more subqueries in the `DELETE` statement and use the subquery results as conditions.

Before you delete data, execute the following statement to create another table named `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`) BUCKETS 1
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT",
"enable_persistent_index" = "false"
);
```

Insert data into the `users` table:

```
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)
```

Nest a subquery to find the rows whose `country` values are `China` from the `users` table, and delete the rows, which have the same `name` values as the rows returned from the subquery, from the `score_board` table. You can use one of the following methods to fulfill the purpose:

- Method 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 |
+------+------+------+
```

- Method 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)
```

References

SHOW DELETE: queries historical DELETE operations that were successfully performed on Duplicate Key tables, Aggregate tables, and Unique Key tables.

DROP TASK

Description

Drops an asynchronous ETL task submitted using SUBMIT TASK. This feature has been supported since StarRocks v2.5.7.

> **NOTE**
>
> Dropping a task with DROP TASK simultaneously cancels the corresponding TaskRun.

Syntax

```
DROP TASK `<task_name>`
```

Parameters

Parameter	Description
task_name	The name of the task to drop. Please wrap the task name with backticks (`) to prevent any parse failure.

Usage notes

You can check the information of asynchronous tasks by querying the metadata tables `tasks` and `task_runs` in Information Schema.

```
SELECT * FROM INFORMATION_SCHEMA.tasks;
SELECT * FROM information_schema.tasks WHERE task_name =
'<task_name>';
SELECT * FROM information_schema.task_runs;
```

```
SELECT * FROM information_schema.task_runs WHERE task_name =
'<task_name>';
```

Examples

```
MySQL > SUBMIT /*+set_var(query_timeout=100000)*/ TASK `ctas` AS
    -> CREATE TABLE insert_wiki_edit_new
    -> AS SELECT * FROM source_wiki_edit;
+----------+-----------+
| TaskName | Status    |
+----------+-----------+
| ctas     | SUBMITTED |
+----------+-----------+
1 row in set (1.19 sec)

MySQL > DROP TASK `ctas`;
Query OK, 0 rows affected (0.35 sec)
```

EXPORT

Description

Exports the data of a table to a specified location.

This is an asynchronous operation. The export result is returned after you submit the export task. You can use SHOW EXPORT to view the progress of the export task.

Syntax

```
EXPORT TABLE <table_name>
[PARTITION (<partition_name>[, ...])]
[(<column_name>[, ...])]
TO <export_path>
[opt_properties]
WITH BROKER
[broker_properties]
```

Parameters

- `table_name`

  The name of the table. StarRocks supports exporting the data of tables whose `engine` is `olap` or `mysql`.

- `partition_name`

  The partitions from which you want to export data. By default, if you do not specify this parameter, StarRocks exports the data from all partitions of the table.

- `column_name`

The columns from which you want to export data. The sequence of columns that you specify by using this parameter can differ from the schema of the table. By default, if you do not specify this parameter, StarRocks exports the data from all columns of the table.

- `export_path`

  The location to which you want to export the data of the table. If the location contains a path, make sure that the path ends with a slash (/). Otherwise, the part following the last slash (/) in the path will be used as the prefix to the name of the exported file. By default, `data_` is used as the file name prefix if no file name prefix is specified.

- `opt_properties`

  Optional properties that you can configure for the export task.

  Syntax:

```
[PROPERTIES ("<key>"="<value>", ...)]
```

Property	Description
column_separator	The column separator that you want to use in the exported file. Default value: `\t`.
line_delimiter	The row separator that you want to use in the exported file. Default value: `\n`.
load_mem_limit	The maximum memory that is allowed for the export task on each individual BE. Unit: bytes. The default maximum memory is 2 GB.
timeout	The amount of time after which the export task times out. Unit: second. Default value: `86400`, meaning 1 day.
include_query_id	Specifies whether the name of the exported file contains `query_id`. Valid values: `true` and `false`. The value `true` specifies that the file name contains `query_id`, and the value

Property	Description
	`false` specifies that the file name does not contain `query_id`.

- `WITH BROKER`

  In v2.4 and earlier, input `WITH BROKER "<broker_name>"` to specify the broker you want to use. From v2.5 onwards, you no longer need to specify a broker, but you still need to retain the `WITH BROKER` keyword. For more information, see Export data using EXPORT > Background information.

- `broker_properties`

  The information that is used to authenticate the source data. The authentication information varies depending on the data source. For more information, see BROKER LOAD.

Examples

Export all data of a table to HDFS

The following example exports all data of the `testTbl` table to the `hdfs://<hdfs_host>:<hdfs_port>/a/b/c/` path of an HDFS cluster:

```
EXPORT TABLE testTbl
TO "hdfs://<hdfs_host>:<hdfs_port>/a/b/c/"
WITH BROKER
(
        "username"="xxx",
        "password"="yyy"
);
```

Export the data of specified partitions of a table to HDFS

The following example exports the data of two partitions, `p1` and `p2`, of the `testTbl` table to the `hdfs://<hdfs_host>:<hdfs_port>/a/b/c/` path of an HDFS cluster:

```
EXPORT TABLE testTbl
PARTITION (p1,p2)
TO "hdfs://<hdfs_host>:<hdfs_port>/a/b/c/"
WITH BROKER
(
        "username"="xxx",
        "password"="yyy"
);
```

Export all data of a table to HDFS with column separator specified

The following example exports all data of the `testTbl` table to the `hdfs://<hdfs_host>:<hdfs_port>/a/b/c/` path of an HDFS cluster and specifies that commas (`,`) is used as the column separator:

```
EXPORT TABLE testTbl
TO "hdfs://<hdfs_host>:<hdfs_port>/a/b/c/"
PROPERTIES
(
    "column_separator"=","
)
WITH BROKER
(
    "username"="xxx",
    "password"="yyy"
);
```

The following example exports all data of the `testTbl` table to the `hdfs://<hdfs_host>:<hdfs_port>/a/b/c/` path of an HDFS cluster and specifies that `\x01` (the default column separator supported by Hive) is used as the column separator:

```
EXPORT TABLE testTbl
TO "hdfs://<hdfs_host>:<hdfs_port>/a/b/c/"
PROPERTIES
(
    "column_separator"="\\x01"
)
WITH BROKER;
```

Export all data of a table to HDFS with file name prefix specified

The following example exports all data of the `testTbl` table to the `hdfs://<hdfs_host>:<hdfs_port>/a/b/c/` path of an HDFS cluster and specifies that `testTbl_` is used as the prefix to the name of the exported file:

```
EXPORT TABLE testTbl
TO "hdfs://<hdfs_host>:<hdfs_port>/a/b/c/testTbl_"
WITH BROKER;
```

Export data to AWS S3

The following example exports all data of the `testTbl` table to the `s3-package/export/` path of an AWS S3 bucket:

```
EXPORT TABLE testTbl
TO "s3a://s3-package/export/"
WITH BROKER
(
    "aws.s3.access_key" = "xxx",
    "aws.s3.secret_key" = "yyy",
    "aws.s3.region" = "zzz"
);
```

INSERT

Description

Inserts data into a specific table or overwrites a specific table with data. For detailed information about the application scenarios, see Load data with INSERT.

Syntax

```
INSERT {INTO | OVERWRITE} table_name
[ PARTITION (<partition1_name>[, <partition2_name>, ...]) ]
[ TEMPORARY_PARTITION (<temporary_partition1_name>[,
<temporary_partition2_name>, ...]) ]
[ WITH LABEL label]
[ (column_name [, ...]) ]
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

Parameters

Parameter	Description
INTO	To append data to the table.
OVERWRITE	To overwrite the table with data.
table_name	The name of the table into which you want to load data. It can be specified with the database the table resides as `db_name.table_name`.
PARTITION	The partitions into which you want to load the data. You can specify multiple partitions, which must be separated by commas (,). It must be set to partitions that exist in the destination table. If you specify this parameter, the data will

Parameter	Description
	be inserted only into the specified partitions. If you do not specify this parameter, the data will be inserted into all partitions.
TEMPORARY_PARTITION	The name of the temporary partition into which you want to load data. You can specify multiple temporary partitions, which must be separated by commas (,).
label	The unique identification label for each data load transaction within the database. If it is not specified, the system automatically generates one for the transaction. We recommend you specify the label for the transaction. Otherwise, you cannot check the transaction status if a connection error occurs and no result is returned. You can check the transaction status via `SHOW LOAD WHERE label="label"` statement. For limitations on naming a label, see System Limits.
column_name	The name of the destination column(s) to load data in. It must be set as columns that exist in the destination table. The destination columns you specify are mapped one on one in sequence to the columns of the source table, regardless of what the destination column names are. If no destination column is specified, the default value is all columns in the destination table. If the specified column in the source table does not exist in the destination column, the default value will be written to this column, and the transaction will fail if the specified column does not have a default value. If the column type of the source table is inconsistent with that of the destination table, the system will perform an implicit conversion on the mismatched column. If the conversion fails, a syntax parsing error will be returned.
expression	Expression that assigns values to the column.

Parameter	Description
DEFAULT	Assigns default value to the column.
query	Query statement whose result will be loaded into the destination table. It can be any SQL statement supported by StarRocks.

Return

```
Query OK, 5 rows affected, 2 warnings (0.05 sec)
{'label':'insert_load_test', 'status':'VISIBLE', 'txnId':'1008'}
```

Return	Description
rows affected	Indicates how many rows are loaded. `warnings` indicates the rows that are filtered out.
label	The unique identification label for each data load transaction within the database. It can be assigned by user or automatically by the system.
status	Indicates if the loaded data is visible. `VISIBLE`: the data is successfully loaded and visible. `COMMITTED`: the data is successfully loaded but invisible for now.
txnId	The ID number corresponding to each INSERT transaction.

Usage notes

- As for the current version, when StarRocks executes the INSERT INTO statement, if any row of data mismatches the destination table format (for example, the string is too long), the INSERT transaction fails by default. You can set the session variable `enable_insert_strict` to `false` so that the system filters out the data that mismatches the destination table format and continues to execute the transaction.

- After INSERT OVERWRITE statement is executed, StarRocks creates temporary partitions for the partitions that store the original data, inserts data into the temporary partitions, and swaps the original partitions with the temporary partitions. All these operations are executed in the Leader FE node. Therefore, if the Leader FE node crashes while executing INSERT OVERWRITE statement, the whole load transaction fails, and the temporary partitions are deleted.

Example

The following examples are based on table `test`, which contains two columns `c1` and `c2`. The `c2` column has a default value of DEFAULT.

- Import a row of data into the `test` table.

```
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);
```

When no destination column is specified, the columns are loaded in sequence into the destination table by default. Therefore, in the above example, the outcomes of the first and second SQL statements are the same.

If a destination column (with or without data inserted) uses DEFAULT as the value, the column will use the default value as the loaded data. Therefore, in the above example, the outputs of the third and fourth statements are the same.

- Load multiple rows of data into the `test` table at one time.

```
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);
```

Because the results of expressions are equivalent, the outcomes of the first and second statements are the same. The outcomes of the third and fourth statements are the same because they both use default value.

- Import a query statement result into the `test` table.

```
INSERT INTO test SELECT * FROM test2;
INSERT INTO test (c1, c2) SELECT * from test2;
```

- Import a query result into the `test` table, and specify partition and label.

```
INSERT INTO test PARTITION(p1, p2) WITH LABEL `label1` SELECT * FROM
test2;
INSERT INTO test WITH LABEL `label1` (c1, c2) SELECT * from test2;
```

- Overwrite the `test` table with a query result, and specify partition and label.

```
INSERT OVERWRITE test PARTITION(p1, p2) WITH LABEL `label1` SELECT *
FROM test3;
INSERT OVERWRITE test WITH LABEL `label1` (c1, c2) SELECT * from
test3;
```

PAUSE ROUTINE LOAD

Description

Pauses a Routine Load job but does not terminate this job. You can execute RESUME ROUTINE LOAD to resume it. After the load job is paused, you can execute SHOW ROUTINE LOAD and ALTER ROUTINE LOAD to view and modify information about the paused load job.

You can manage Routine Load jobs that run on StarRocks tables only as a user who has the INSERT privilege on those StarRocks tables. If you do not have the INSERT privilege, follow the instructions provided in GRANT to grant the INSERT privilege to the user that you use to connect to your StarRocks cluster.

Syntax

```
PAUSE ROUTINE LOAD FOR [db_name.]<job_name>;
```

Parameters

Parameter	Required	Description
db_name		The name of the database to which the Routine Load job belongs.
job_name	✅	The name of the Routine Load job. A table may have multiple Routine Load jobs, it is recommended to set a meaningful Routine Load job name by using identifiable information, for example, Kafka topic name or time when you create the load job, to distinguish multiple routine load jobs. The name of the Routine Load job must be unique within the same database

Examples

Pause the Routine Load job `example_tbl1_ordertest1` in the database `example_db`.

```
PAUSE ROUTINE LOAD FOR example_db.example_tbl1_ordertest1;
```

REFRESH MATERIALIZED VIEW

Description

Manually refresh a specific asynchronous materialized view or partitions within.

> **CAUTION**
>
> You can only manually refresh materialized views that adopt ASYNC or MANUAL refresh strategy. You can check the refresh strategy of an asynchronous materialized view using SHOW MATERIALIZED VIEW.

Syntax

```
REFRESH MATERIALIZED VIEW [database.]mv_name
[PARTITION START ("<partition_start_date>") END ("
<partition_end_date>")]
[FORCE]
[WITH { SYNC | ASYNC } MODE]
```

Parameters

Parameter	Required	Description
mv_name	yes	The name of the materialized view to refresh manually.
PARTITION START () END ()	no	Manually refresh partitions within a certain time interval.

Parameter	Required	Description
partition_start_date	no	The start date of the partitions to refresh manually.
partition_end_date	no	The end date of the partitions to refresh manually.
FORCE	no	If you specify this parameter, StarRocks forcibly refreshes the corresponding materialized view or partitions. If you do not specify this parameter, StarRocks automatically judges if the data is updated and refreshes the partition only when needed.
WITH ... MODE	no	Make a synchronous or asynchronous call of the refresh task. `SYNC` indicates making a synchronous call of the refresh task, and StarRocks returns the task result only when the task succeeds or fails. `ASYNC` indicates making an asynchronous call of the refresh task, and StarRocks returns success right after the task is submitted, leaving the task to be executed asynchronously in the background. You can check the refresh task status of an asynchronous materialized view by querying the `tasks` and `task_runs` metadata tables in StarRocks' Information Schema. For more information, see Check the execution status of asynchronous materialized view. Default: `ASYNC`. Supported from v2.5.8, v3.0.4, and v3.1.0 onwards.

> **CAUTION**
>
> When refreshing materialized views created based on the external catalogs, StarRocks refreshes all partitions in the materialized views.

Examples

Example 1: Manually refresh a specific materialized view via an asynchronous call.

```
REFRESH MATERIALIZED VIEW lo_mv1;

REFRESH MATERIALIZED VIEW lo_mv1 WITH ASYNC MODE;
```

Example 2: Manually refresh certain partitions of a specific materialized view.

```
REFRESH MATERIALIZED VIEW lo_mv1
PARTITION START ("2020-02-01") END ("2020-03-01");
```

Example 3: Forcibly refresh certain partitions of a specific materialized view.

```
REFRESH MATERIALIZED VIEW lo_mv1
PARTITION START ("2020-02-01") END ("2020-03-01") FORCE;
```

Example 4: Manually refresh a materialized view via a synchronous call.

```
REFRESH MATERIALIZED VIEW lo_mv1 WITH SYNC MODE;
```

RESUME ROUTINE LOAD

Description

Resumes a Routine load job. The job will temporarily enter **NEED_SCHEDULE** state because the job is being re-scheduled. And after some time, the job will be resumed to **RUNNING** state, continuing consuming messages from the data source and loading data. You can check the job's information using the SHOW ROUTINE LOAD statement.

You can manage Routine Load jobs that run on StarRocks tables only as a user who has the INSERT privilege on those StarRocks tables. If you do not have the INSERT privilege, follow the instructions provided in GRANT to grant the INSERT privilege to the user that you use to connect to your StarRocks cluster.

Syntax

```
RESUME ROUTINE LOAD FOR [db_name.]<job_name>
```

Parameters

Parameter	Required	Description
db_name		The name of the database to which the Routine Load job belongs.
job_name	✅	The name of the Routine Load job.

Examples

Resume the Routine Load job `example_tbl1_ordertest1` in the database `example_db`.

```
RESUME ROUTINE LOAD FOR example_db.example_tbl1_ordertest1;
```

SELECT

Description

Queries data from one or more tables, views, or materialized views. The SELECT statement generally consists of the following clauses:

- WITH
- WHERE and operators
- GROUP BY
- HAVING
- UNION
- INTERSECT
- EXCEPT/MINUS
- ORDER BY
- LIMIT
- OFFSET
- Joins
- Subqueries
- DISTINCT
- Alias

SELECT can work as an independent statement or a clause nested in other statements. The output of the SELECT clause can be used as the input of other statements.

StarRocks' query statement basically conforms to the SQL92 standard. Here is a brief description of the supported select usage.

WITH

A clause that can be added before a SELECT statement to define an alias for a complex expression that is referenced multiple times inside SELECT.

Similar to CRATE VIEW, but the table and column names defined in the clause do not persist after the query ends and do not conflict with names in the actual table or VIEW.

The benefits of using a WITH clause are:

Convenient and easy to maintain, reducing duplication within queries.

It is easier to read and understand SQL code by abstracting the most complex parts of a query into separate blocks.

Examples:

```
-- 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 operations combine data from two or more tables and then return a result set of some columns from some of them.

StarRocks supports self joins, cross joins, inner joins, outer joins, semi joins, and anti joins. Outer joins include left joins, right joins, and full joins.

Syntax:

```
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

StarRocks supports self-joins, which are self-joins and self-joins. For example, different columns of the same table are joined.

There is actually no special syntax to identify self-join. The conditions on both sides of a join in a self-join come from the same table.

We need to assign them different aliases.

Examples:

```
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 can produce a lot of results, so cross join should be used with caution.

Even if you need to use cross join, you need to use filter conditions and ensure that fewer results are returned. Example:

```
SELECT * FROM t1, t2;

SELECT * FROM t1 CROSS JOIN t2;
```

Inner Join

Inner join is the most well-known and commonly used join. Returns results from columns requested by two similar tables, joined if the columns of both tables contain the same value.

If a column name of both tables is the same, we need to use the full name (in the form of table_name.column_name) or alias the column name.

Examples:

The following three queries are equivalent.

```
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 returns the left or right table or all rows of both. If there is no matching data in another table, set it to NULL. Example:

```
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;
```

Equivalent and unequal join

Usually, users use the most equal join, which requires the operator of the join condition to be an equal sign.

Unequal join can be used on join conditions!=, Equal sign. Unequal joins produce a large number of results and may exceed the memory limit during calculation.

Use with caution. Unequal join only supports inner join. Example:

```
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 returns only the rows in the left table that match the data in the right table, regardless of how many rows match the data in the right table.

This row of the left table is returned at most once. Right semi join works similarly, except that the data returned is a right table.

Examples:

```
SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT SEMI JOIN t2 ON t1.id =
t2.id;
```

Anti Join

Left anti join only returns rows from the left table that do not match the right table.

Right anti join reverses this comparison, returning only rows from the right table that do not match the left table. Example:

```
SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT ANTI JOIN t2 ON t1.id =
t2.id;
```

Equi-join and Non-equi-join

The various joins supported by StarRocks can be classified as equi-joins and non-equi-joins depending on the join conditions specified in the joins.

Equi-joins****	**Self joins, cross joins, inner joins, outer joins, semi joins, and anti joins**
Non-equi-joins	cross joins, inner joins, left semi joins, left anti joins, and outer joins

- Equi-joins

  An equi-join uses a join condition in which two join items are combined by the `=` operator. Example: `a JOIN b ON a.id = b.id`.

- Non-equi-joins

A non-equi-join uses a join condition in which two join items are combined by a comparison operator such as `<`, `<=`, `>`, `>=`, or `<>`. Example: `a JOIN b ON a.id < b.id`. Non-equi-joins run slower than equi-joins. We recommend that you exercise caution when you use non-equi-joins.

The following two examples show how to run 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;
```

ORDER BY

The ORDER BY clause of a SELECT statement sorts the result set by comparing the values from one or more columns.

ORDER BY is a time- and resource-consuming operation because all the results must be sent to one node for merging before the results can be sorted. Sorting consumes more memory resources than a query without ORDER BY.

Therefore, if you only need the first `N` results from the sorted result set, you can use the LIMIT clause, which reduces memory usage and network overhead. If the LIMIT clause is not specified, the first 65535 results are returned by default.

Syntax:

```
ORDER BY <column_name>
    [ASC | DESC]
    [NULLS FIRST | NULLS LAST]
```

`ASC` specifies that the results should be returned in ascending order. `DESC` specifies that the results should be returned in descending order. If the order is not specified, ASC (ascending) is the default. Example:

```
select * from big_table order by tiny_column, short_column desc;
```

Sort order for NULL values: `NULLS FIRST` indicates that NULL values should be returned before non-NULL values. `NULLS LAST` indicates that NULL values should be returned after non-NULL values.

Examples:

```
select  *  from  sales_record  order by  employee_id  nulls first;
```

GROUP BY

The GROUP BY clause is often used with aggregate functions such as COUNT(), SUM(), AVG(), MIN(), and MAX().

The column specified by GROUP BY will not participate in the aggregation operation. The GROUP BY clause can be added with the Having clause to filter the results produced by the aggregate function.

Examples:

```
select tiny_column, sum(short_column)
from small_table
group by tiny_column;
```

```
+-------------+--------------------+
| tiny_column |  sum('short_column')|
+-------------+--------------------+
|      1      |        2           |
|      2      |        1           |
+-------------+--------------------+
```

Syntax

```
SELECT ...
FROM ...
[ ... ]
GROUP BY [
```

```
    , ... |
    GROUPING SETS [, ...] (  groupSet [ , groupSet [ , ... ] ] ) |
    ROLLUP(expr  [ , expr [ , ... ] ]) |
    CUBE(expr  [ , expr [ , ... ] ])
    ]
[ ... ]
```

Parameters

`groupSet` represents a set composed of columns, aliases or expressions in the select list.
`groupSet ::= { ( expr [ , expr [ , ... ] ] )}`

`expr` indicates the column, alias or expression in the select list.

Note

StarRocks supports syntax like PostgreSQL. The syntax examples are as follows:

```
SELECT a, b, SUM( c ) FROM tab1 GROUP BY GROUPING SETS ( (a, b), (a),
(b), ( ) );
SELECT a, b,c, SUM( d ) FROM tab1 GROUP BY ROLLUP(a,b,c)
SELECT a, b,c, SUM( d ) FROM tab1 GROUP BY CUBE(a,b,c)
```

`ROLLUP(a,b,c)` is equivalent to the following `GROUPING SETS` statement:

```
GROUPING SETS (
(a,b,c),
( a, b ),
( a),
( )
)
```

`CUBE ( a, b, c )` is equivalent to the following `GROUPING SETS` statement:

```
GROUPING SETS (
( a, b, c ),
( a, b ),
( a,    c ),
( a       ),
(    b, c ),
(    b    ),
```

```
(       c ),
(         )
)
```

Examples

The following is an example of actual data:

```
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)
```

GROUP BY `GROUPING SETS` | `CUBE` | `ROLLUP` is an extension of the GROUP BY clause. It can realize the aggregation of groups of multiple sets in a GROUP BY clause. The result is equivalent to the UNION operation of multiple corresponding GROUP BY clauses.

GROUP BY clause is a special case of GROUP BY GROUPING SETS containing only one element. For example, the GROUPING SETS statement:

```
SELECT a, b, SUM( c ) FROM tab1 GROUP BY GROUPING SETS ( (a, b), (a),
(b), ( ) );
```

The query result is equivalent to:

```
SELECT a, b, SUM( c ) FROM tab1 GROUP BY a, b
UNION
SELECT a, null, SUM( c ) FROM tab1 GROUP BY a
UNION
SELECT null, b, SUM( c ) FROM tab1 GROUP BY b
UNION
SELECT null, null, SUM( c ) FROM tab1
```

`GROUPING(expr)` indicates whether a column is an aggregate column. If it is an aggregate column, it is 0, otherwise it is 1.

`GROUPING_ID(expr [ , expr [ , ... ] ])` is similar to GROUPING. GROUPING_ ID calculates the bitmap value of a column list according to the specified column order, and each bit is the value of GROUPING.

GROUPING_ID() function returns the decimal value of the bit vector.

HAVING

The HAVING clause does not filter row data in a table, but filters the results of aggregate functions.

Generally speaking, HAVING is used with aggregate functions (such as COUNT(), SUM(), AVG(), MIN(), MAX()) and GROUP BY clauses.

Examples:

```
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

LIMIT clauses are used to limit the maximum number of rows returned. Setting the maximum number of rows returned can help StarRocks optimize memory usage.

This clause is mainly used in the following scenarios:

Returns the result of the top-N query.

Think about what's included in the table below.

The size of the query result set needs to be limited because of the large amount of data in the table or because the where clause does not filter too much data.

Instructions for use: The value of the LIMIT clause must be a numeric literal constant.

Examples:

```
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

The OFFSET clause causes the result set to skip the first few rows and return the following results directly.

The result set defaults to start at line 0, so offset 0 and no offset return the same results.

Generally speaking, OFFSET clauses need to be used with ORDER BY and LIMIT clauses to be valid.

Examples:

```
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)
```

Note: It is allowed to use offset syntax without order by, but offset does not make sense at this time.

In this case, only the limit value is taken, and the offset value is ignored. So without order by.

Offset exceeds the maximum number of rows in the result set and is still a result. It is recommended that users use offset with order by.

UNION

Combines the result of multiple queries.

Syntax:

```
query_1 UNION [DISTINCT | ALL] query_2
```

- DISTINCT (default) returns only unique rows. UNION is equivalent to UNION DISTINCT.
- ALL combines all rows, including duplicates. Since de-duplication is memory intensive, queries using UNION ALL are faster and less memory-consuming. For better performance, use UNION ALL.

> **NOTE**
>
> Each query statement must return the same number of columns and the columns must have compatible data types.

Examples:

Create tables `select1` and `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);
```

Example 1: Return all IDs in the two tables, including duplicates.

```
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)
```

Example 2: Return all the unique IDs in the two tables. The following two statements are equivalent.

```
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)
```

Example 3: Return the first three IDs among all the unique IDs in the two tables. The following two statements are equivalent.

```
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

Calculates the intersection of the results of multiple queries, that is, the results that appear in all the result sets. This clause returns only unique rows among the result sets. The ALL keyword is not supported.

Syntax:

```
query_1 INTERSECT [DISTINCT] query_2
```

> **NOTE**
>
> - INTERSECT is equivalent to INTERSECT DISTINCT.
> - Each query statement must return the same number of columns and the columns must have compatible data types.

Examples:

The two tables in UNION are used.

Return distinct `(id, price)` combinations that are common to both tables. The following two statements are equivalent.

```
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

Returns distinct results of the left-hand query that do not exist in the right-hand query. EXCEPT is equivalent to MINUS.

Syntax:

```
query_1 {EXCEPT | MINUS} [DISTINCT] query_2
```

> **NOTE**
>
> - EXCEPT is equivalent to EXCEPT DISTINCT. The ALL keyword is not supported.
> - Each query statement must return the same number of columns and the columns must have compatible data types.

Examples:

The two tables in UNION are used.

Return distinct `(id, price)` combinations in `select1` that cannot be found in `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

The DISTINCT keyword deduplicates the result set. Example:

```
-- 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 can be used with aggregate functions (usually count functions), and count (distinct) is used to calculate how many different combinations are contained on one or more columns.

```
-- 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;
```

StarRocks supports multiple aggregate functions using distinct at the same time.

```
-- Count the unique value from multiple aggregation function
separately.
select count(distinct tiny_column, int_column), count(distinct
varchar_column) from big_table;
```

Subquery

Subqueries are categorized into two types in terms of relevance:

- A noncorrelated subquery obtains its results independently of its outer query.
- A correlated subquery requires values from its outer query.

Noncorrelated subquery

Noncorrelated subqueries support [NOT] IN and EXISTS.

Example:

```
SELECT x FROM t1 WHERE x [NOT] IN (SELECT y FROM t2);
```

```
SELECT x FROM t1 WHERE EXISTS (SELECT y FROM t2 WHERE y = 1);
```

Correlated subquery

Related subqueries support [NOT] IN and [NOT] EXISTS.

Example:

```
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);
```

Subqueries also support scalar quantum queries. It can be divided into irrelevant scalar quantum query, related scalar quantum query and scalar quantum query as parameters of the general function.

Examples:

1. Uncorrelated scalar quantum query with predicate = sign. For example, output information about the person with the highest wage.

   ```
   SELECT name FROM table WHERE salary = (SELECT MAX(salary) FROM
   table);
   ```

2. Uncorrelated scalar quantum queries with predicates `>`, `<` etc. For example, output information about people who are paid more than average.

   ```
   SELECT name FROM table WHERE salary > (SELECT AVG(salary) FROM
   table);
   ```

3. Related scalar quantum queries. For example, output the highest salary information for each department.

   ```
   SELECT name FROM table a WHERE salary = (SELECT MAX(salary) FROM
   table b WHERE b.Department= a.Department);
   ```

4. Scalar quantum queries are used as parameters of ordinary functions.

   ```
   SELECT name FROM table WHERE salary = abs((SELECT MAX(salary)
   FROM table));
   ```

Where and Operators

SQL operators are a series of functions used for comparison and are widely used in where clauses of select statements.

Arithmetic operator

Arithmetic operators usually appear in expressions that contain left, right, and most often left operands

+and-: can be used as a unit or as a 2-ary operator. When used as a unit operator, such as +1, -2.5 or -col_ name, which means the value is multiplied by +1 or -1.

So the cell operator + returns an unchanged value, and the cell operator - changes the symbol bits of that value.

Users can overlay two cell operators, such as +5 (returning a positive value), -+2 or +2 (returning a negative value), but users cannot use two consecutive - signs.

Because--is interpreted as a comment in the following statement (when a user can use two-signs, a space or parenthesis is required between the two-signs, such as-(-2) or - -2, which actually results in + 2).

When + or - is a binary operator, such as 2+2, 3+1.5, or col1+col2, it means that the left value is added or subtracted from the right value. Both left and right values must be numeric types.

and/: represent multiplication and division, respectively. The operands on both sides must be data types. When two numbers are multiplied.

Smaller operands may be promoted if needed (e.g., SMALLINT to INT or BIGINT), and the result of the expression will be promoted to the next larger type.

For example, TINYINT multiplied by INT will produce a BIGINT type of result. When two numbers are multiplied, both operands and expression results are interpreted as DOUBLE types to avoid loss of precision.

If the user wants to convert the result of the expression to another type, it needs to be converted using the CAST function.

%: Modulation operator. Returns the remainder of the left operand divided by the right operand. Both left and right operands must be integers.

&, |and ^: The bitwise operator returns the result of bitwise AND, bitwise OR, bitwise XOR operations on two operands. Both operands require an integer type.

If the types of the two operands of a bitwise operator are inconsistent, the operands of a smaller type are promoted to the operands of a larger type, and the corresponding bitwise operations are performed.

Multiple arithmetic operators can appear in an expression, and the user can enclose the corresponding arithmetic expression in parentheses. Arithmetic operators usually do not have corresponding mathematical functions to express the same functions as arithmetic operators.

For example, we don't have the MOD() function to represent the% operator. Conversely, mathematical functions do not have corresponding arithmetic operators. For example, the power function POW() does not have a corresponding ** exponentiation operator.

Users can find out which arithmetic functions we support through the Mathematical Functions section.

Between Operator

In a where clause, expressions may be compared with both upper and lower bounds. If the expression is greater than or equal to the lower bound and less than or equal to the upper bound, the result of the comparison is true.

Syntax:

```
expression BETWEEN lower_bound AND upper_bound
```

Data type: Usually an expression evaluates to a numeric type, which also supports other data types. If you must ensure that both the lower and upper bounds are comparable characters, you can use the cast() function.

Instructions for use: If the operand is of type string, note that a long string starting with an upper bound will not match the upper bound, which is larger than the upper bound. For example, "between'A'and'M' will not match 'MJ'.

If you need to make sure the expression works correctly, you can use functions such as upper(), lower(), substr(), trim().

Example:

```
select c1 from t1 where month between 1 and 6;
```

Comparison operators

Comparison operators are used to compare two values. `=`, `!=`, `>=` apply to all data types.

The `<>` and `!=` operators are equivalent, which indicate that two values are not equal.

In Operator

The In operator compares to the VALUE collection and returns TRUE if it can match any of the elements in the collection.

Parameters and VALUE collections must be comparable. All expressions using the IN operator can be written as equivalent comparisons connected with OR, but the syntax of IN is simpler, more precise, and easier for StarRocks to optimize.

Examples:

```
select * from small_table where tiny_column in (1,2);
```

Like Operator

This operator is used to compare to a string. ''matches a single character,'%' matches multiple characters. The parameter must match the complete string. Typically, placing'%'at the end of a string is more practical.

Examples:

```
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)
```

Logical Operator

Logical operators return a BOOL value, including unit and multiple operators, each of which handles parameters that are expressions that return BOOL values. Supported operators are:

AND: 2-ary operator, the AND operator returns TRUE if the parameters on the left and right are both calculated as TRUE.

OR: 2-ary operator that returns TRUE if one of the parameters on the left and right is calculated as TRUE. If both parameters are FALSE, the OR operator returns FALSE.

NOT: Unit operator, the result of inverting an expression. If the parameter is TRUE, the operator returns FALSE; If the parameter is FALSE, the operator returns TRUE.

Examples:

```
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)
```

Regular Expression Operator

Determines whether the regular expression is matched. Using POSIX standard regular expressions,'^'matches the first part of the string,'$' matches the end of the string.

"." matches any single character, "*" matches zero or more options, "+" matches one or more options, "?" means greedy representation, and so on. Regular expressions need to match complete values, not just parts of strings.

If you want to match the middle part, the front part of the regular expression can be written as'^. 'or'.'. '^'and'$' are usually omitted. The RLIKE operator and the REGEXP operator are

synonyms.

The'|'operator is an optional operator. Regular expressions on either side of'|' only need to satisfy one side condition. The'|'operator and regular expressions on both sides usually need to be enclosed in ().

Examples:

```
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

When you write the names of tables, columns, or expressions that contain columns in a query, you can assign them an alias. Aliases are usually shorter and better to remember than original names.

When an alias is needed, you can simply add an AS clause after the table, column, and expression names in the select list or from list. The AS keyword is optional. You can also specify aliases directly after the original name without using AS.

If an alias or other identifier has the same name as an internal StarRocks keyword, you need to enclose the name in a pair of backticks, for example, `rank`.

Aliases are case-sensitive, but column aliases and expression aliases are not case-sensitive.

Examples:

```
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;
```

SHOW ALTER TABLE

Description

Shows the execution of ongoing ALTER TABLE tasks.

Syntax

```
SHOW ALTER TABLE {COLUMN | ROLLUP} [FROM <db_name>]
```

Parameters

- COLUMN | ROLLUP
  - If COLUMN is specified, this statement shows tasks for modifying columns. If you need to nest a WHERE clause, the supported syntax is `[WHERE TableName|CreateTime|FinishTime|State] [ORDER BY] [LIMIT]`.
  - If ROLLUP is specified, this statement shows tasks for creating or deleting ROLLUP indexes.
- `db_name`: optional. If `db_name` is not specified, the current database is used by default.

Examples

Example 1: Show column modification tasks in the current database.

```
SHOW ALTER TABLE COLUMN;
```

Example 2: Show the latest column modification task of a table.

```
SHOW ALTER TABLE COLUMN WHERE TableName = "table1"
ORDER BY CreateTime DESC LIMIT 1;
```

Example 3: Show tasks for creating or deleting ROLLUP indexes in a specified database.

```
SHOW ALTER TABLE ROLLUP FROM example_db;
```

References

- CREATE TABLE
- ALTER TABLE
- SHOW TABLES
- SHOW CREATE TABLE

SHOW ALTER MATERIALIZED VIEW

Description

Shows the building status of synchronous materialized views.

Syntax

```
SHOW ALTER MATERIALIZED VIEW [ { FROM | IN } db_name]
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
db_name	no	The name of the database to which the materialized view resides. If this parameter is not specified, the current database is used by default.

Returns

Return	Description
JobId	The ID of refresh job.
TableName	The name of the table.

Return	Description
CreateTime	The time when refresh job is created.
FinishedTime	The time when refresh job is finished.
BaseIndexName	The name of the base table.
RollupIndexName	The name of the materialized view.
RollupId	The ID of the materialized view rollup.
TransactionId	The ID of transaction that waits to be executed.
State	The state of the job.
Msg	Error message.
Progress	Progress of the refresh job.
Timeout	Timeout for the refresh job.

Examples

Example 1: building status of synchronous materialized views

```
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 BACKUP

Description

Views the last BACKUP task in a specified database. For more information, see data backup and restoration.

> **NOTE**
>
> Only the information of the last BACKUP task is saved in StarRocks.

Syntax

```
SHOW BACKUP [FROM <db_name>]
```

Parameters

Parameter	Description
db_name	Name of the database that the BACKUP task belongs to.

Return

Return	Description
JobId	Unique job ID.
SnapshotName	Name of the data snapshot.
DbName	Name of the database that the BACKUP task belongs to.

Return	Description
State	Current state of the BACKUP task: • PENDING: Initial state after submitting a job. • SNAPSHOTING: Creating snapshot. • UPLOAD_SNAPSHOT: Snapshot complete, ready for upload. • UPLOADING: Uploading snapshot. • SAVE_META: Creating local metadata files. • UPLOAD_INFO: Uploading metadata files and information of the BACKUP task. • FINISHED: BACKUP task finished. • CANCELLED: BACKUP task failed or cancelled.
BackupObjs	Backed up objects.
CreateTime	Task submission time.
SnapshotFinishedTime	Snapshot completion time.
UploadFinishedTime	Snapshot upload completion time.
FinishedTime	Task completion Time.
UnfinishedTasks	Unfinished subtask IDs in the SNAPSHOTING and UPLOADING phases.
Progress	The progress of snapshot uploading tasks.
TaskErrMsg	Error messages.
Status	Status information.
Timeout	Task timeout. Unit: second.

Examples

Example 1: Views the last BACKUP task in the database `example_db`.

```
SHOW BACKUP FROM example_db;
```

SHOW CATALOGS

Description

Query all catalogs in the current StarRocks cluster, including the internal catalog and external catalogs.

Syntax

```
SHOW CATALOGS
```

Output

```
+----------+--------+----------+
| Catalog  | Type   | Comment  |
+----------+--------+----------+
```

The following table describes the fields returned by this statement.

Field	Description
Catalog	The catalog name.
Type	The catalog type. `Internal` is returned if the catalog is `default_catalog`. The corresponding catalog type is returned if the catalog is an external catalog, such as `Hive`, `Hudi`, or `Iceberg`.
Comment	The comments of a catalog. StarRocks does not support adding comments to an external catalog. Therefore, the value is `NULL` for an external catalog. If the catalog is `default_catalog`, the comment is `An internal catalog contains this cluster's self-managed tables.` by

Field	Description
	default. `default_catalog` is the only internal catalog in a StarRocks cluster.

Examples

Query all catalogs in the current 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

Description

Queries the creation statement of an external catalog, such as a Hive, Iceberg, Hudi, or Delta Lake catalog. See Hive Catalog, Iceberg Catalog, Hudi Catalog, and Delta Lake Catalog. Note that authentication-related information in the return result will be anonymized.

This command is supported from v2.5.4 onwards.

Syntax

```
SHOW CREATE CATALOG <catalog_name>;
```

Parameters

Parameter	Required	Description
catalog_name	Yes	The name of the catalog whose creation statement you want to view.

Return result

```
+-----------+-----------------+
| Catalog    | Create Catalog  |
+-----------+-----------------+
```

Field	Description
Catalog	The name of the catalog.
Create Catalog	The statement that was executed to create the catalog.

Examples

The following example queries the creation statement of a Hive catalog named `hive_catalog_hms`:

```
SHOW CREATE CATALOG hive_catalog_hms;
```

The return result is as follows:

```
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"
)
```

SHOW CREATE DATABASE

Shows the SQL command used to create a database.

Syntax

```
SHOW CREATE DATABASE <db_name>
```

Parameters

`db_name`: the database name, required.

Returns

- `Database`: the database name
- `Create Database`: the SQL command used to create the database

Examples

```
mysql > show create database zj_test;
+----------+--------------------------+
| Database | Create Database          |
+----------+--------------------------+
| zj_test  | CREATE DATABASE `zj_test` |
+----------+--------------------------+
```

References

- CREATE DATABASE

- SHOW DATABASES
- USE
- DESC
- DROP DATABASE

SHOW CREATE MATERIALIZED VIEW

Description

Shows the definition of a specific asynchronous materialized view.

Syntax

```
SHOW CREATE MATERIALIZED VIEW [database.]<mv_name>
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
mv_name	yes	The name of the materialized view to show.

Returns

Return	Description
Materialized View	The name of the materialized view.
Create Materialized View	The definition of the materialized view.

Examples

Example 1: Show the definition of a specific materialized view

```
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`) BUCKETS 10
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 CREATE TABLE

Returns the CREATE TABLE statement that was used to create a given table.

> **NOTE**
>
> The SHOW CREATE TABLE statement requires you to have the `SELECT_PRIV` privilege on the table.

Since v2.5.7, StarRocks can automatically set the number of buckets (BUCKETS) when you create a table or add a partition. You no longer need to manually set the number of buckets. For detailed information, see Determine the number of buckets.

- If you specified the number of buckets when creating a table, the output of SHOW CREATE TABLE will display the number of buckets.
- If you did not specify the number of buckets when creating a table, the output of SHOW CREATE TABLE will not display the number of buckets. You can run SHOW PARTITIONS to view the number of buckets for each partition.

In versions earlier than v2.5.7, you are required to set the number of buckets when creating a table. Therefore, SHOW CREATE TABLE displays the number of buckets by default.

Syntax

```
SHOW CREATE TABLE [db_name.]table_name
```

Parameters

Parameter	Required	Description
db_name	No	The database name. If this parameter is not specified, the CREATE TABLE statement of a given table in your current database is returned by default.

Parameter	Required	Description
table_name	Yes	The table name.

Output

```
+-----------+----------------+
| Table     | Create Table   |
+-----------+----------------+
```

The following table describes the parameters returned by this statement.

Parameter	Description
Table	The table name.
Create Table	The CREATE TABLE statement of the table.

Examples

Bucket number is not specified

Create a table named `example_table` with no bucket number specified in 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 starrocks table"
DISTRIBUTED BY HASH(k1) BUCKETS 10;
```

Run SHOW CREATE TABLE to display the CREATE TABLE statement of `example_table`. No bucket number is displayed in DISTRIBUTED BY. Note that if you did not specify PROPERTIES when you create the table, the default properties are displayed in the output of SHOW CREATE TABLE.

```
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 starrocks table"
DISTRIBUTED BY HASH(`k1`)
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"enable_persistent_index" = "false",
"replicated_storage" = "true",
"compression" = "LZ4"
);
```

Bucket number is specified

Create a table named `example_table1` with bucket number set to 10 in 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 starrocks table"
DISTRIBUTED BY HASH(k1) BUCKETS 10;
```

Run SHOW CREATE TABLE to display the CREATE TABLE statement of `example_table`. The bucket number (`BUCKETS 10`) is displayed in DISTRIBUTED BY. Note that if you did not specify PROPERTIES when you create the table, the default properties are displayed in the output of SHOW CREATE TABLE.

```
SHOW CREATE TABLE example_table1\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 starrocks table"
DISTRIBUTED BY HASH(`k1`) BUCKETS 10
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"enable_persistent_index" = "false",
"replicated_storage" = "true",
"compression" = "LZ4"
);
```

References

- CREATE TABLE
- SHOW TABLES
- ALTER TABLE
- DROP TABLE

SHOW CREATE VIEW

Returns the CREATE statement that was used to create a given view. The CREATE VIEW statement helps you understand how the view is defined and provides a reference for you to modify or reconstruct the view. Note that the SHOW CREATE VIEW statement requires you to have the `SELECT_PRIV` privilege on the view and the table based on which the view is created.

From v2.5.4 onwards, you can use SHOW CREATE VIEW to query the statement that is used to create a **materialized view**.

Syntax

```
SHOW CREATE VIEW [db_name.]view_name
```

Parameters

Parameter	Required	Description
db_name	No	The database name. If this parameter is not specified, the CREATE VIEW statement of a given view in your current database is returned by default.
view_name	Yes	The view name.

Output

```
+--------+-------------+----------------------+----------------------
--+
| View   | Create View  | character_set_client |
collation_connection |
```

```
+--------+-------------+----------------------+-----------------------
--+
```

The following table describes the parameters returned by this statement.

Parameter	Description
View	The view name.
Create View	The CREATE VIEW statement of the view.
character_set_client	The character set the client uses to send statements to StarRocks.
collation_connection	The rules for comparing characters in a character set.

Examples

Create a table named `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) BUCKETS 10;
```

Create a view named `example_view` based on `example_table`.

```
CREATE VIEW example_view (k1, k2, k3, v1)
AS SELECT k1, k2, k3, v1 FROM example_table;
```

Display the CREATE VIEW statement of `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      |
+--------------+-----------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------+-------
--------------+----------------------+
```

SHOW DATA

Description

This statement is used to display the amount of data, the number of copies, and the number of statistical rows.

Syntax:

```
SHOW DATA [FROM <db_name>[.<table_name>]]
```

Note:

1. If the FROM clause is not specified, the amount of data and copies subdivided into each table in the current db will be displayed. Where the data volume is the total data volume of all replicas. The number of copies is the number of copies of all partitions of the table and all materialized views.
2. If the FROM clause is specified, the amount of data, the number of copies and the number of statistical rows subdivided into each materialized view under the table are displayed. Where the data volume is the total data volume of all replicas. The number of copies is the number of copies of all partitions corresponding to the materialized view. The number of statistical rows is the number of statistical rows of all partitions corresponding to the materialized view.
3. When counting the number of rows, the copy with the largest number of rows among multiple copies shall prevail.
4. The Total row in the result set represents the summary row. The Quota row represents the quota set by the current database. The Left line represents the remaining quota.

Examples

1. Displays the data volume, copy quantity, summary data volume and summary copy quantity of each table in the default db.

```
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   |
+-----------+-------------+--------------+
```

2. Displays the amount of breakdown data, the number of copies, and the number of statistical rows in the specified db table

```
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

Description

Views databases in your current StarRocks cluster or an external data source. StarRocks supports viewing databases of an external data source from v2.3 onwards.

Syntax

```
SHOW DATABASES [FROM <catalog_name>]
```

Parameters

Parameter	Required	Description
catalog_name	No	The name of the internal catalog or an external catalog. • If you do not specify the parameter or specify the name of the internal catalog, which is `default_catalog`, you can view databases in your current StarRocks cluster. • If you set the value of the parameter to the name of an external catalog, you can view databases in the corresponding external data source. You can run SHOW CATALOGS to view internal and external catalogs.

Examples

Example 1: View databases in your current StarRocks cluster.

```
SHOW DATABASES;
```

Or

```
SHOW DATABASES FROM default_catalog;
```

The output of the preceding statements is as follows.

```
+----------+
| Database |
+----------+
| db1      |
| db2      |
| db3      |
+----------+
```

Example 2: View databases in a Hive cluster by using the `Hive1` external catalog.

```
SHOW DATABASES FROM hive1;

+-----------+
| Database  |
+-----------+
| hive_db1  |
| hive_db2  |
| hive_db3  |
+-----------+
```

References

- CREATE DATABASE
- SHOW CREATE DATABASE
- USE
- DESC
- DROP DATABASE

SHOW DELETE

Description

Queries historical DELETE operations that were successfully performed on Duplicate Key, Unique Key, and Aggregate tables in a specified database. For more information about data deletion, see DELETE.

Note that this command cannot be used to query DELETE operations that were performed on Primary Key tables.

Syntax

```
SHOW DELETE [FROM <db_name>]
```

`db_name`: the database name, optional. If this parameter is not specified, the current database is used by default.

Return fields:

- `TableName`: the table from which data is deleted.
- `PartitionName`: the partition from which data is deleted. If the table is a non-partitioned table, `*` is displayed.
- `CreateTime`: the time when the DELETE task was created.
- `DeleteCondition`: the specified DELETE condition.
- `State`: the status of the DELETE task.

Examples

Show all historical DELETE operations of `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

Description

This statement is used to display the status of all the partitioned tables for which dynamic partitioning properties are configured in a database.

Syntax

```
SHOW DYNAMIC PARTITION TABLES FROM <db_name>
```

This statement returns the following fields:

- TableName: the name of the table.
- Enable: whether dynamic partitioning is enabled.
- TimeUnit: the time granularity for the partitions.
- Start: the starting offset of dynamic partitioning.
- End: the end offset of dynamic partitioning.
- Prefix: the prefix of the partition name.
- Buckets: the number of buckets per partition.
- ReplicationNum: the number of replicas for the table.
- StartOf
- LastUpdateTime: the time when the table was last updated.
- LastSchedulerTime: the time when data in the table was last scheduled.
- State: the status of the table.
- LastCreatePartitionMsg: the message for the latest partition creation operation.
- LastDropPartitionMsg: the message for the latest partition dropping operation.

Examples

Display the status of all the partitioned tables for which dynamic partitioning properties are configured in `db_test`.

```
SHOW DYNAMIC PARTITION TABLES FROM db_test;
```

SHOW EXPORT

Description

Queries the execution information of export jobs that meet the specified conditions.

Syntax

```
SHOW EXPORT
[ FROM <db_name> ]
[
WHERE
    [ QUERYID = <query_id> ]
    [ STATE = { "PENDING" | "EXPORTING" | "FINISHED" | "CANCELLED" }
]
]
[ ORDER BY <field_name> [ ASC | DESC ] [, ... ] ]
[ LIMIT <count> ]
```

Parameters

This statement can contain the following optional clauses:

- FROM

  Specifies the name of the database that you want to query. If you do not specify a FROM clause, StarRocks queries the current database.

- WHERE

  Specifies the conditions based on which you want to filter export jobs. Only the export jobs that meet the specified conditions are returned in the result set of the query.

Parameter	Required	Description
QUERYID	No	The ID of the export job that you want to query. This parameter is used to query the execution information of a single export job.
STATE	No	The status of the export jobs that you want to query. Valid values: ◦ `PENDING`: specifies to query export jobs that are waiting to be scheduled. ◦ `EXPORING`: specifies to query export jobs that are being executed. ◦ `FINISHED`: specifies to query export jobs that have been successfully completed. ◦ `CANCELLED`: specifies to query export jobs that have failed.

- ORDER BY

  Specifies the name of the field based on which you want to sort the export job records in the result set of the query. You can specify multiple fields, which must be separated with commas (`,`). Additionally, you can use the `ASC` or `DESC` keyword to specify that the export job records are sorted in ascending or descending order based on the specified field.

- LIMIT

  Restricts the result set of the query to a specified maximum number of rows. Valid values: positive integer. If you do not specify a LIMIT clause, StarRocks returns all export jobs that meet the specified conditions.

Return result

For example, query the execution information of an export job whose ID is `edee47f0-abe1-11ec-b9d1-00163e1e238f`:

```
SHOW EXPORT
WHERE QUERYID = "edee47f0-abe1-11ec-b9d1-00163e1e238f";
```

The following execution information is returned:

```
     JobId: 14008
   QueryId: edee47f0-abe1-11ec-b9d1-00163e1e238f
     State: FINISHED
  Progress: 100%
  TaskInfo: {"partitions":["*"],"column separator":"\t","columns":
["*"],"tablet num":10,"broker":"","coord
num":1,"db":"db0","tbl":"tbl_simple","row delimiter":"\n","mem
limit":2147483648}
      Path: hdfs://127.0.0.1:9000/users/230320/
CreateTime: 2023-03-20 11:16:14
 StartTime: 2023-03-20 11:16:17
FinishTime: 2023-03-20 11:16:26
   Timeout: 7200
```

The parameters in the return result are described as follows:

- `JobId`: the ID of the export job.
- `QueryId`: the ID of the query.
- `State`: the status of the export job.

  Valid values:

  - `PENDING`: The export job is waiting to be scheduled.
  - `EXPORTING`: The export job is being executed.
  - `FINISHED`: The export job has been successfully completed.
  - `CANCELLED`: The export job has failed.
- `Progress`: the progress of the export job. The progress is measured in the unit of query plans. Suppose that the export job is divided into 10 query plans and three of them have finished. In this case, the progress of the export job is 30%. For more information, see "Export data using EXPORT > Workflow".
- `TaskInfo`: the information of the export job.

The information is a JSON object that consists of the following keys:

  - `partitions`: the partitions on which the exported data resides. If a wildcard (`*`) is returned as a value for this key, the export job is run to export data from all partitions.
  - `column separator`: the column separator used in the exported data file.
  - `columns`: the names of the columns whose data is exported.
  - `tablet num`: the total number of tablets that are exported.
  - `broker`: In v2.4 and earlier, this field is used to return the name of the broker that is used by the export job. From v2.5 onwards, this field returns an empty string. For more information, see "Export data using EXPORT > Background information".
  - `coord num`: the number of query plans into which the export job is divided.
  - `db`: the name of the database to which the exported data belongs.
  - `tbl`: the name of the table to which the exported data belongs
  - `row delimiter`: the row separator used in the exported data file.
  - `mem limit`: the maximum amount of memory allowed for the export job. Unit: bytes.
- `Path`: the path to which the exported data is stored on the remote storage.
- `CreateTime`: the time when the export job was created.
- `StartTime`: the time when the export job started to be scheduled.
- `FinishTime`: the time when the export job was finished.
- `Timeout`: the amount of time that the export job took than expected. Unit: seconds. The time is counted from `CreateTime`.
- `ErrorMsg`: the reason why the export job throws an error. This field is returned only when the export job encounters an error.

Examples

- Query all export jobs in the current database:

```
SHOW EXPORT;
```

- Query the export job whose ID is `921d8f80-7c9d-11eb-9342-acde48001122` in the database `example_db`:

```
SHOW EXPORT FROM example_db
WHERE queryid = "921d8f80-7c9d-11eb-9342-acde48001122";
```

- Query export jobs that are in the `EXPORTING` state in the database `example_db` and specify to sort the export job records in the result set by `StartTime` in ascending order:

```
SHOW EXPORT FROM example_db
WHERE STATE = "exporting"
ORDER BY StartTime ASC;
```

- Query all export jobs in the database `example_db` and specify to sort the export job records in the result set by `StartTime` in descending order:

```
SHOW EXPORT FROM example_db
ORDER BY StartTime DESC;
```

SHOW LOAD

Description

Displays information of all load jobs or given load jobs in a database. This statement can only display load jobs that are created by using Broker Load, Spark Load, or INSERT. You can also view information of load jobs via the `curl` command. For more information, see Load data from HDFS or cloud storage, Bulk load using Apache Spark™, and Load data using INSERT.

In addition to the preceding loading methods, StarRocks supports using Stream Load and Routine Load to load data. Stream Load is a synchronous operation and will directly return information of Stream Load jobs. Routine Load is an asynchronous operation where you can use the SHOW ROUTINE LOAD statement to display information of Routine Load jobs.

Syntax

```
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 } ]
```

> **Note**
>
> You can add the `\G` option to the statement (such as `SHOW LOAD WHERE LABEL = "label1"\G;`) to vertically display output rather than in the usual horizontal table format. For more information, see Example 1.

Parameters

Parameter	Required	Description
db_name	No	The database name. If this parameter is not specified, your current database is used by default.
LABEL = "label_name"	No	The labels of load jobs.
LABEL LIKE "label_matcher"	No	If this parameter is specified, the information of load jobs whose labels contain `label_matcher` is returned.
AND	No	• If you specify only one filter condition in the WHERE clause, do not specify this keyword. Example: `WHERE STATE = "PENDING"`. • If you specify two or three filter conditions in the WHERE clause, you must specify this keyword. Example: `WHERE LABEL = "label_name" AND STATE = "PENDING"`.
STATE	No	The states of load jobs. The states vary based on loading methods. • Broker Load ◦ `PENDING`: The load job is created. ◦ `QUEUEING`: The load job is in the queue waiting to be scheduled. ◦ `LOADING`: The load job is running. ◦ `PREPARED`: The transaction has been committed. ◦ `FINISHED`: The load job succeeded. ◦ `CANCELLED`: The load job failed. • Spark Load ◦ `PENDING`: Your StarRocks cluster is preparing configurations related to ETL and then submits an ETL job to your Apache Spark™ cluster.

Parameter	Required	Description
		<code>ETL</code>: Your Spark cluster is executing the ETL job and then writes the data into the corresponding HDFS cluster. <code>LOADING</code>: The data in the HDFS cluster is being loaded to your StarRocks cluster, which means the load job is running. <code>PREPARED</code>: The transaction has been committed. <code>FINISHED</code>: The load job succeeded. <code>CANCELLED</code>: The load job failed. - INSERT <code>FINISHED</code>: The load job succeeded. <code>CANCELLED</code>: The load job failed. If the <code>STATE</code> parameter is not specified, the information of load jobs in all states is returned by default. If the <code>STATE</code> parameter is specified, only the information of load jobs in the given state is returned. For example, <code>STATE = "PENDING"</code> returns the information of load jobs in the <code>PENDING</code> state.

Parameter	Required	Description
ORDER BY field_name [ASC | DESC]	No	If this parameter is specified, the output is sorted in ascending or descending order based on a field. The following fields are supported: `JobId`, `Label`, `State`, `Progress`, `Type`, `EtlInfo`, `TaskInfo`, `ErrorMsg`, `CreateTime`, `EtlStartTime`, `EtlFinishTime`, `LoadStartTime`, `LoadFinishTime`, `URL`, and `JobDetails`. • To sort the output in ascending order, specify `ORDER BY field_name ASC`. • To sort the output in descending order, specify `ORDER BY field_name DESC`. If you do not specify the field and the sort order, the output is sorted in ascending order of `JobId` by default.
LIMIT limit	No	The number of load jobs that are allowed to display. If this parameter is not specified, the information of all load jobs that match the filter conditions are displayed. If this parameter is specified, for example, `LIMIT 10`, only the information of 10 load jobs that match filter conditions are returned.
OFFSET offset	No	The `offset` parameter defines the number of load jobs to be skipped. For example, `OFFSET 5` skips the first five load jobs and returns the rest. The value of the `offset` parameter defaults to `0`.

Output

```
+-------+-------+-------+----------+------+---------+----------+-----
-----+-----------+-------------+--------------+--------------+---
------------+-----+------------+
| JobId | Label | State | Progress | Type | Priority | EtlInfo |
TaskInfo | ErrorMsg | CreateTime | EtlStartTime | EtlFinishTime |
```

```
LoadStartTime | LoadFinishTime | URL | JobDetails |
+-------+-------+-------+---------+------+--------+---------+-----
-----+-----------+-------------+-------------+--------------+---
------------+-----+-----------+
```

The output of this statement varies based on loading methods.

Field	Broker Load	Spark Lo
JobId	The unique ID assigned by StarRocks to identify the load job in your StarRocks cluster.	The field has the meaning in a Spar as it does in a Bro job.
Label	The label of the load job. The label of a load job is unique within a database but can be duplicate across different databases.	The field has the meaning in a Spar as it does in a Bro job.
State	The state of the load job. • `PENDING`: The load job is created. • `QUEUEING`: The load job is in the queue waiting to be scheduled. • `LOADING`: The load job is running. • `PREPARED`: The transaction has been committed. • `FINISHED`: The load job succeeded. • `CANCELLED`: The load job failed.	The state of the lo • `PENDING`: Yo StarRocks clu preparing cor related to ETL submits an E your Spark cl • `ETL`: Your Sp is executing t and then writ into the corre HDFS cluster • `LOADING`: Th the HDFS clu being loaded StarRocks clu means the lo running.

Field	Broker Load	Spark Lo
		• `PREPARED`: T transaction h committed. • `FINISHED`: T succeeded. • `CANCELLED`: job failed.
Progress	The stage of the load job. A Broker Load job only has the `LOAD` stage, which ranges from 0% to 100% to describe the progress of the stage. When the load job enters the `LOAD` stage, `LOADING` is returned for the `State` parameter. A Broker Load job does not have the `ETL` stage. The `ETL` parameter is valid only for a Spark Load job. **Note** • The formula to calculate the progress of the `LOAD` stage: Number of StarRocks tables that complete data loading/Number of StarRocks tables that you plan to load data into * 100%. • When all data is loaded into StarRocks, `99%` is returned for the `LOAD` parameter. Then, loaded data starts taking effect in StarRocks. After the data takes effect, `100%` is returned for the `LOAD` parameter. • The progress of the `LOAD` stage is not linear. Therefore, the value of the `LOAD` parameter may not change over a period of time even if data loading is still ongoing.	The stage of the l Spark Load job ha stages: • `ETL`: ranges 100% to desc progress of th stage. • `LOAD`: ranges to 100% to d progress of th stage. When the load job `ETL` stage, `ETL` i for the `State` par When the load job the `LOAD` stage, returned for the `S` parameter. The **Note** is the sa those for Broker L
Type	The method of the load job. The value of this parameter defaults to `BROKER`.	The method of th The value of this p defaults to `SPARK`

Field	Broker Load	Spark Lo
Priority	The priority of the load job. Valid values: LOWEST, LOW, NORMAL, HIGH, and HIGHEST.	-
EtlInfo	The metrics related to ETL. • `unselected.rows`: The number of rows that are filtered out by the WHERE clause. • `dpp.abnorm.ALL`: The number of rows that are filtered out due to data quality issues, which refers to mismatches between source tables and StarRocks tables in, for example, the data type and the number of columns. • `dpp.norm.ALL`: The number of rows that are loaded into your StarRocks cluster. The sum of the preceding metrics is the total number of rows of raw data. You can use the following formula to calculate whether the percentage of unqualified data exceeds the value of the `max-filter-ratio` parameter: `dpp.abnorm.ALL`/(`unselected.rows` + `dpp.abnorm.ALL` + `dpp.norm.ALL`).	The field has the meaning in a Spa as it does in a Bro job.
TaskInfo	The parameters that are specified when you create the load job. • `resource`: This parameter is valid only in a Spark Load job. • `timeout`: The time period that a load job is allowed to run. Unit: seconds. • `max-filter-ratio`: The largest percentage of rows that are filtered out due to data quality issues. For more information, see BROKER LOAD.	The parameters t specified when yo the load job. • `resource`: T resource nam • `timeout`: Th period that a allowed to ru seconds. • `max-filter` The largest p of rows that a out due to da issues.

Field	Broker Load	Spark Lo
		For more informat SPARK LOAD.
ErrorMsg	The error message returned when the load job fails. When the state of the loading job is `PENDING`, `LOADING`, or `FINISHED`, `NULL` is returned for the `ErrorMsg` field. When the state of the loading job is `CANCELLED`, the value returned for the `ErrorMsg` field consists of two parts: `type` and `msg`. • The `type` part can be any of the following values: ◦ `USER_CANCEL`: The load job was manually canceled. ◦ `ETL_SUBMIT_FAIL`: The load job failed to be submitted. ◦ `ETL_QUALITY_UNSATISFIED`: The load job failed because the percentage of unqualified data exceeds the value of the `max_filter_ratio` parameter. ◦ `LOAD_RUN_FAIL`: The load job failed in the `LOAD` stage. ◦ `TIMEOUT`: The load job failed to finish within the specified timeout period. ◦ `UNKNOWN`: The load job failed due to an unknown error. • The `msg` part provides the detailed cause of the load failure.	The error messag when the load job When the state of loading job is `PEN` `LOADING`, or `FIN` `NULL` is returned `ErrorMsg` field. state of the loadir `CANCELLED`, the returned for the `E` field consists of t `type` and `msg`. • The `type` pa any of the fol values: ◦ `USER_CA` load job manually ◦ `ETL_SUB` StarRock submit a to Spark ◦ `ETL_RUN` Spark fai execute job. ◦ `ETL_QUA` `UNSATIS` load job because percenta unqualifi

Field	Broker Load	Spark Lo
		exceeds of the `ma` `filter-` paramet ◦ `LOAD-RU` The load in the `LO` ◦ `TIMEOUT` job failed within th timeout ◦ `UNKNOWN` job failed unknowr • The `msg` par the detailed the load failu
CreateTime	The time at which the load job was created.	The field has the meaning in a Spa as it does in a Bro job.
EtlStartTime	A Broker Load job does not have the `ETL` stage. Therefore, the value of this field is the same as the value of the `LoadStartTime` field.	The time at which stage starts.
EtlFinishTime	A Broker Load job does not have the `ETL` stage. Therefore, the value of this field is the same as the value of the `LoadStartTime` field.	The time at which stage finishes.
LoadStartTime	The time at which the `LOAD` stage starts.	The field has the meaning in a Spa

Field	Broker Load	Spark Lo
		as it does in a Bro job.
LoadFinishTime	The time at which the load job finishes.	The field has the meaning in a Spar as it does in a Bro job.
URL	The URL that is used to access the unqualified data detected in the load job. You can use the `curl` or `wget` command to access the URL and obtain the unqualified data. If no unqualified data is detected, `NULL` is returned.	The field has the meaning in a Spar as it does in a Bro job.
JobDetails	Other information related to the load job. • `Unfinished backends`: The ID of the BE that does not complete data loading. • `ScannedRows`: The total number of rows that are loaded into StarRocks and the number of rows that are filtered out. • `TaskNumber`: A load job can be split into one or more tasks that concurrently run. This field indicates the number of load tasks. • `All backends`: The ID of the BE that is executing data loading. • `FileNumber`: The number of source data files. • `FileSize`: The data volume of source data files. Unit: bytes.	The field has the meaning in a Spar as it does in a Bro job.

Usage notes

- The information returned by the SHOW LOAD statement is valid for 3 days from `LoadFinishTime` of a load job. After 3 days, the information cannot be displayed. You

can use the `label_keep_max_second` parameter to modify the default validity period.

```
ADMIN SET FRONTEND CONFIG ("label_keep_max_second" = "value");
```

- If the value of the `LoadStartTime` field is `N/A` for a long time, it means that load jobs heavily pile up. We recommended that you reduce the frequency of creating load jobs.
- Total time period consumed by a load job = `LoadFinishTime` - `CreateTime`.
- Total time period a load job consumed in the `LOAD` stage = `LoadFinishTime` - `LoadStartTime`.

Examples

Example 1: Vertically display all load jobs in your current database.

```
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}
```

Example 2: Display two load jobs whose labels contain the string `null` in your current database.

```
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}
|
+-------+-------------------------+---------+--------------------
+--------+-------------------------------------------------------------+-
-------------------------------------------------------------------------
-----------------------------------+----------+--------------------
+--------------------+--------------------+--------------------+--
------------------+-------------------------------------------------
-----------------------------+-----------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
------------------------+
```

Example 3: Display the load jobs whose labels contain the string `table` in `example_db`. In addition, the load jobs returned are displayed in descending order of the `LoadStartTime` field.

```
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} |
+-------+-------------------------+---------+------------------
+--------+-----------------------------------------------------+-
-------------------------------------------------------------------
---------------------------------+---------+------------------
+-------------------+-------------------+-------------------+--
-----------------+-----------------------------------------------
-----------------------------+-----------------------------------
-------------------------------------------------------------------
-------------------------------------------------------------------
-------------------------+
```

Example 4: Display the load job whose label is `duplicate_table_with_null` and state is `FINISHED` in `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} |
+-------+--------------------------+----------+--------------------
+--------+-----------------------------------------------------------+-
-------------------------------------------------+----------+------
--------------+---------------------+--------------------+---------
-----------+--------------------+------+------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
----------------------------------+
```

Example 5: Skip the first load job and display the next two load jobs. In addition, these two load jobs are sorted in ascending order.

```
SHOW LOAD FROM example_db
ORDER BY CreateTime ASC
LIMIT 2 OFFSET 1;
```

Or

```
SHOW LOAD FROM example_db
ORDER BY CreateTime ASC
LIMIT 1,2;
```

The output of the preceding statements is as follows.

```
+-------+----------------------------------------------+----------+---
-----------------+--------+----------------------------------------
---------------+------------------------------------------------------
-------------------------------------------------+----------+----
----------------+--------------------+-------------------+-------
-------------+--------------------+-------------------------------
---------------------------------------------+-------------------
------------------------------------------------------------------
------------------------------------------------------------------
--------------------------------------+
| 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 MATERIALIZED VIEW

Description

Shows all or one specific asynchronous materialized view.

Syntax

```
SHOW MATERIALIZED VIEW
[FROM db_name]
[
WHERE
[NAME { = "mv_name" | LIKE "mv_name_matcher"}
]
```

Parameters in brackets [] is optional.

Parameters

Parameter	Required	Description
db_name	no	The name of the database to which the materialized view resides. If this parameter is not specified, the current database is used by default.
mv_name	no	The name of the materialized view to show.
mv_name_matcher	no	The matcher used to filter materialized views.

Returns

Return	Description
id	The ID of the materialized view.
database_name	The name of the database in which the materialized view resides.
name	The name of the materialized view.
rows	The number of data rows in the materialized view.
text	The statement used to create the materialized view.

Examples

```
MySQL > SHOW MATERIALIZED VIEW\G
*************************** 1. row ***************************
           id: 10099
         name: order_mv
database_name: test
         text: CREATE MATERIALIZED VIEW `order_mv` (`order_id`,
`total`)
DISTRIBUTED BY HASH(`order_id`)
REFRESH ASYNC START("2022-09-01 10:00:00") EVERY(INTERVAL 1 DAY)
PROPERTIES (
"replication_num" = "3",
"storage_medium" = "HDD"
)
AS SELECT `order_list`.`order_id`, sum(`goods`.`price`) AS `total`
FROM `wlc`.`order_list` INNER JOIN `wlc`.`goods` ON
`goods`.`item_id1` = `order_list`.`item_id2`
GROUP BY `order_list`.`order_id`;
         rows: 3
1 row in set (0.02 sec)
```

SHOW PARTITIONS

Description

Displays partition information, including common partitions and temporary partitions.

Syntax:

```
SHOW [TEMPORARY] PARTITIONS FROM [db_name.]table_name [WHERE] [ORDER
BY] [LIMIT]
```

> NOTE
>
> - This syntax only supports StarRocks tables (`"ENGINE" = "OLAP"`). For Elasticsearch and Hive tables, use SHOW PROC '/dbs/db_id/table_id/partitions'.
> - This operation requires the SELECT__PRIV privilege on the specified table.

Description of return fields

```
+-------------+--------------+---------------+--------------------
+--------------------+--------+-------------+-------+--------------
-----+---------+---------------+--------------+--------------------
-+-------------------------+----------+-----------+----------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime
| VisibleVersionHash | State  | PartitionKey | Range |
DistributionKey    | Buckets | ReplicationNum | StorageMedium |
CooldownTime        | LastConsistencyCheckTime | DataSize |
IsInMemory | RowCount |
+-------------+--------------+---------------+--------------------
+--------------------+--------+-------------+-------+--------------
-----+---------+---------------+--------------+--------------------
-+-------------------------+----------+-----------+----------+
```

Field	Description
PartitionId	The ID of the partition.
PartitionName	The name of the partition.
VisibleVersion	The version number of the last successful load transaction. The version number increases by 1 with each successful load transaction.
VisibleVersionTime	The timestamp of the last successful load transaction.
VisibleVersionHash	The hash value for the version number of the last successful load transaction.
State	The status of the partition. Fixed value: `Normal`.
PartitionKey	The partition key that consists of one or more partition columns.
Range	The range of the partition, which is a right half-open interval.
DistributionKey	The bucket key of hash bucketing.
Buckets	The number of buckets for the partition.
ReplicationNum	The number of replicas per tablet in the partition.
StorageMedium	The storage medium to store the data in the partition. The value `HHD` indicates hard disk drives, and the value `SSD` indicates solid-state drives.
CooldownTime	The cooldown time for data in the partition. If the initial storage medium is SSD, the storage medium is switched from SSD to HDD after the time specified by this parameter. Format: "yyyy-MM-dd HH:mm:ss".

Field	Description
LastConsistencyCheckTime	The time of the last consistency check. `NULL` indicates no consistency check was performed.
DataSize	The size of data in the partition.
IsInMemory	Whether all data in the partition is stored in memory.
RowCount	The number of data rows of the partition.

Examples

1. Display information of all regular partitions from the specified table `site_access` under the specified database `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. Display information of all temporary partitions from the specified table `site_access` under the specified database `test`.

```
SHOW TEMPORARY PARTITIONS FROM test.site_access;
```

3. Display the information of the specified partition `p1` of the specified table `site_access` under the specified database `test`.

```
-- Regular partition
SHOW PARTITIONS FROM test.site_access WHERE PartitionName = "p1";
-- Temporary partition
SHOW TEMPORARY PARTITIONS FROM test.site_access WHERE
PartitionName = "p1";
```

4. Display the latest partition information of the specified table `site_access` under the specified database `test`.

```
-- Regular partition
SHOW PARTITIONS FROM test.site_access ORDER BY PartitionId DESC
LIMIT 1;
-- Temporary partition
SHOW TEMPORARY PARTITIONS FROM test.site_access ORDER BY
PartitionId DESC LIMIT 1;
```

SHOW REPOSITORIES

Description

Views the repositories created in StarRocks.

Syntax

```
SHOW REPOSITORIES
```

Return

Return	Description
RepoId	Repository ID.
RepoName	Repository name.
CreateTime	Repository creation time.
IsReadOnly	If the repository is read-only.
Location	Location of the repository in the remote storage system.
Broker	Broker used to create the repository.
ErrMsg	Error message during connection to the repository.

Examples

Example 1: Views the repositories created in StarRocks.

```
SHOW REPOSITORIES;
```

SHOW RESTORE

Description

Views the last RESTORE task in a specified database. For more information, see data backup and restoration.

> **NOTE**
>
> Only the information of the last RESTORE task is saved in StarRocks.

Syntax

```
SHOW RESTORE [FROM <db_name>]
```

Parameters

Parameter	Description
db_name	Name of the database that the RESTORE task belongs to.

Return

Return	Description
JobId	Unique job ID.
Label	Name of the data snapshot.
Timestamp	Backup timestamp.

Return	Description
DbName	Name of the database that the RESTORE task belongs to.
State	Current state of the RESTORE task: • PENDING: Initial state after submitting a job. • SNAPSHOTING: Executing the local snapshot. • DOWNLOAD: Submitting snapshot download task. • DOWNLOADING: Downloading the snapshot. • COMMIT: To commit the downloaded snapshot. • COMMITTING: Committing the downloaded snapshot. • FINISHED: RESTORE task finished. • CANCELLED: RESTORE task failed or cancelled.
AllowLoad	If loading data is allowed during the RESTORE task.
ReplicationNum	Number of replicas to be restored.
RestoreObjs	The restored objects (tables and partitions).
CreateTime	Task submission time.
MetaPreparedTime	Local metadata completion time.
SnapshotFinishedTime	Snapshot completion time.
DownloadFinishedTime	Snapshot download completion time.
FinishedTime	Task completion Time.
UnfinishedTasks	Unfinished subtask IDs in the SNAPSHOTTING, DOWNLOADING, and COMMITTING phases.
Progress	The progress of snapshot downloading tasks.
TaskErrMsg	Error messages.

Return	Description
Status	Status information.
Timeout	Task timeout. Unit: second.

Examples

Example 1: Views the last RESTORE task in the database `example_db`.

```
SHOW RESTORE FROM example_db;
```

SHOW ROUTINE LOAD

Description

Shows the execution information of Routine Load jobs.

Syntax

```
SHOW [ALL] ROUTINE LOAD [ FOR [<db_name>.]<job_name> | FROM <db_name>
]
[ WHERE [ STATE = { "NEED_SCHEDULE" | "RUNNING" | "PAUSED" |
"UNSTABLE" | "STOPPED" | "CANCELLED"  } ] ]
[ ORDER BY field_name [ ASC | DESC ] ]
[ LIMIT { [offset, ] limit | limit OFFSET offset } ]
```

> **TIP**
>
> You can add the `\G` option to the statement (such as `SHOW ROUTINE LOAD FOR <job_name>\G`) to vertically display the return result rather than in the usual horizontal table format.

Parameters

Parameter	Required	Description
db_name	No	The name of the database to which the load job belongs. Note that this parameter is required if the `FROM` clause is used.
job_name	No	The name of the load job. Note that this parameter is required if the `FOR` clause is used.

Parameter	Required	Description
ALL	No	Displays all load jobs, including those in the `STOPPED` or `CANCELLED` states.
STATE	No	The status of the load job.
ORDER BY field_name [ASC | DESC]	No	Sorts the return result in ascending or descending order based on the specified field. The following fields are supported: `Id`, `Name`, `CreateTime`, `PauseTime`, `EndTime`, `TableName`, `State`, and `CurrentTaskNum`. • To sort the return result in ascending order, specify `ORDER BY field_name ASC`. • To sort the return result in descending order, specify `ORDER BY field_name DESC`. If you do not specify the field or the sort order, the return result is sorted in ascending order of `Id` by default.
LIMIT limit	No	The number of load jobs that will be returned. For example, if `LIMIT 10` is specified, only the information of 10 load jobs that match filter conditions will be returned. If this parameter is not specified, the information of all load jobs that match the filter conditions are displayed.
OFFSET offset	No	The `offset` parameter defines the number of load jobs to be skipped. For example, `OFFSET 5` skips the first five load jobs and returns the rest. The value of the `offset` parameter defaults to `0`.

Output

Parameter	Description
Id	The globally unique ID of the load job, generated automatically by StarRocks.
Name	The name of the load job.
CreateTime	The date and time when the load job was created.
PauseTime	The date and time when the load job entered `PAUSED` state.
EndTime	The date and time when the load job entered `STOPPED` state.
DbName	The database to which the target table of the load job belongs.
TableName	The target table of the load job.
State	The status of the load job, including: • `NEED_SCHEDULE`: The load job is waiting to be scheduled. After you use CREATE ROUTINE LOAD or RESUME ROUTINE LOAD to create or resume a Routine Load job, the load job first enters the `NEED_SCHEDULE` state. • `RUNNING`: The load job is running. You can view the consumption progress of the Routine Load job through `Statistic` and `Progress`. • `PAUSED`: The load job is paused. You can refer to `ReasonOfStateChanged` and `ErrorLogUrls` for troubleshooting. After fixing the error, you can use RESUME ROUTINE LOAD to resume the Routine Load job. • `CANCELLED`: The load job is cancelled. You can refer to `ReasonOfStateChanged` and `ErrorLogUrls` for troubleshooting. However, after fixing the error, you cannot recover the load job in this state.

Parameter	Description
	• `STOPPED`: The load job is stopped. You cannot recover the load job in this state. • `UNSTABLE`: The load job is unstable. A Routine Load job is set in the `UNSTABLE` state if any task within the Routine Load job lags (namely, the difference between the timestamp of the message being consumed and the current time exceeds this FE parameter `routine_load_unstable_threshold_second`, and unconsumed messages exist in the data source.)
DataSourceType	The type of the data source. Fixed value: `KAFKA`.
CurrentTaskNum	The current number of tasks in the load job.
JobProperties	The properties of the load job, such as partitions to be consumed and column mapping.
DataSourceProperties	The properties of the data source, such as Topic and the list of addresses and ports of brokers in the Kafka cluster.
CustomProperties	The additional data source-related properties defined in the load job.
Statistic	The statistic of loading data, such as successfully loaded rows, total rows, and received data volume.
Progress	The progress (measured in the offset) of consuming messages in partitions of the topic.
TimestampProgress	The progress (measured in the timestamp) of consuming messages in partitions of the topic.
ReasonOfStateChanged	The reasons for the load job being in CANCELLED or PAUSED state.

Parameter	Description
ErrorLogUrls	The URL of error logs. You can use the `curl` or `wget` command to access the URL.
TrackingSQL	The SQL command that you can directly run to query the error log information recorded in the `information_schema` database.
OtherMsg	The information about all failed load tasks of the Routine Load job.
LatestSourcePosition	The latest consumer position of messages in partitions of the topic.

Examples

If a load job is successfully started and stays in the RUNNING state, the returned result might be as follows:

```
MySQL [example_db]> SHOW ROUTINE LOAD FOR example_tbl_ordertest1\G
*************************** 1. row ***************************
                  Id: 10204
                Name: example_tbl_ordertest1
          CreateTime: 2023-12-21 21:01:31
           PauseTime: NULL
             EndTime: NULL
              DbName: example_db
           TableName: example_tbl
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 1
       JobProperties:
{"partitions":"*","rowDelimiter":"\t","partial_update":"false","columnT
DataSourceProperties: {"topic":"lilyliuyitest4csv","currentKafkaPartiti
    CustomProperties: {"kafka_default_offsets":"OFFSET_BEGINNING","grou
           Statistic: {"receivedBytes":313,"errorRows":0,"committedTaskl
            Progress: {"0":"5"}
   TimestampProgress: {"0":"1686143856061"}
ReasonOfStateChanged:
```

```
        ErrorLogUrls:
         TrackingSQL:
            OtherMsg:
LatestSourcePosition: {"0":"6"}
1 row in set (0.01 sec)
```

If a load job is in the `PAUSED` or `CANCELLED` state due to exceptions, you can troubleshoot based on the `ReasonOfStateChanged`, `ErrorLogUrls`, `TrackingSQL`, and `OtherMsg` fields in the return result.

```
MySQL [example_db]> SHOW ROUTINE LOAD FOR example_tbl_ordertest2\G
*************************** 1. row ***************************
                  Id: 10204
                Name: example_tbl_ordertest2
          CreateTime: 2023-12-22 12:13:18
           PauseTime: 2023-12-22 12:13:38
             EndTime: NULL
              DbName: example_db
           TableName: example_tbl
               State: PAUSED
      DataSourceType: KAFKA
      CurrentTaskNum: 0
       JobProperties:
{"partitions":"*","rowDelimiter":"\t","partial_update":"false","columnT
DataSourceProperties: {"topic":"mytest","currentKafkaPartitions":"0","b
    CustomProperties: {"kafka_default_offsets":"OFFSET_BEGINNING","grou
           Statistic: {"receivedBytes":541,"errorRows":10,"committedTas
            Progress: {"0":"19"}
   TimestampProgress: {"0":"1702623900871"}
ReasonOfStateChanged: ErrorReason{errCode = 102, msg='current error row
        ErrorLogUrls: http://xxx.xx.xx.xxx:8040/api/_load_error_log?fil
         TrackingSQL: select tracking_log from information_schema.load_
            OtherMsg:
LatestSourcePosition: {"0":"20"}
1 row in set (0.00 sec)
```

SHOW ROUTINE LOAD TASK

Description

Shows the execution information of load tasks within a Routine Load job.

> **NOTE**
>
> For the relationship between a Routine Load job and the load tasks in it, see Load data using Routine Load

Syntax

```
SHOW ROUTINE LOAD TASK
[  FROM <db_name>]
[  WHERE JobName = <job_name> ]
```

> **NOTE**
>
> You can add the `\G` option to the statement (such as `SHOW ROUTINE LOAD TASK WHERE JobName = <job_name>\G`) to vertically display the return result rather than in the usual horizontal table format.

Parameters

Parameter	Required	Description
db_name	No	The name of the database to which the Routine Load job belongs.
JobName	No	The name of the Routine Load job.

Output

Parameter	Description
TaskId	The globally unique ID of the load task, automatically generated by StarRocks.
TxnId	The ID of the transaction to which the load task belongs.
TxnStatus	The status of the transaction to which the load task belongs. `UNKNOWN` indicates that the transaction is not started yet, possibly because the load task is not dispatched or executed.
JobId	The ID of the load job.
CreateTime	The date and time when the load task was created.
LastScheduledTime	The date and time when the load task was last scheduled.
ExecuteStartTime	The date and time when the load task was executed.
Timeout	The timeout period for the load task, controlled by the FE parameter `routine_load_task_timeout_second` and the `task_timeout_second` parameter in job_properties for the Routine Load job.
BeId	The ID of the BE that executes the load task.
DataSourceProperties	The load task's progress (measured in the offset) of consuming messages in partitions of the topic.
Message	The information returned for the load task, including task error information.

Examples

View all load tasks in the Routine Load job `example_tbl_ordertest`.

```
MySQL [example_db]> SHOW ROUTINE LOAD TASK WHERE JobName =
"example_tbl_ordertest";
+--------------------------------------+-------+-----------+-------+-
------------------+-------------------+-----------------+-------
--+------+--------------------------------------+--------------------
-----------------------------------------------------+
| TaskId                               | TxnId | TxnStatus | JobId |
CreateTime          | LastScheduledTime   | ExecuteStartTime |
Timeout | BeId | DataSourceProperties                | Message
|
+--------------------------------------+-------+-----------+-------+-
------------------+-------------------+-----------------+-------
--+------+--------------------------------------+--------------------
-----------------------------------------------------+
| abde6998-c19a-43d6-b48c-6ca7e14144a3 | -1    | UNKNOWN   | 10208 |
2023-12-22 12:46:10 | 2023-12-22 12:47:00 | NULL              | 60
| -1   | Progress:{"0":6},LatestOffset:null | there is no new data in
kafka/pulsar, wait for 10 seconds to schedule again |
+--------------------------------------+-------+-----------+-------+-
------------------+-------------------+-----------------+-------
--+------+--------------------------------------+--------------------
-----------------------------------------------------+
1 row in set (0.00 sec)
```

SHOW SNAPSHOT

Description

Views data snapshots in a specified repository. For more information, see data backup and restoration.

Syntax

```
SHOW SNAPSHOT ON <repo_name>
[WHERE SNAPSHOT = <snapshot_name> [AND TIMESTAMP =
<backup_timestamp>]]
```

Parameters

Parameter	Description
repo_name	Name of the repository that the snapshot belongs to.
snapshot_nam	Name of the snapshot.
backup_timestamp	Backup timestamp of the snapshot.

Return

Return	Description
Snapshot	Name of the snapshot.
Timestamp	Backup timestamp of the snapshot.

Return	Description
Status	Displays `OK` if the snapshot is okay. Displays error message if the snapshot is not okay.
Database	Name of the database that the snapshot belongs to.
Details	JSON-formatted directory and structure of the snapshot.

Example

Example 1: Views snapshots in repository `example_repo`.

```
SHOW SNAPSHOT ON example_repo;
```

Example 2: Views the snapshot named `backup1` in repository `example_repo`.

```
SHOW SNAPSHOT ON example_repo
WHERE SNAPSHOT = "backup1";
```

Example 3: Views the snapshot named `backup1` with backup timestamp `2018-05-05-15-34-26` in repository `example_repo`.

```
SHOW SNAPSHOT ON example_repo
WHERE SNAPSHOT = "backup1" AND TIMESTAMP = "2018-05-05-15-34-26";
```

SHOW TABLES

Description

Displays all tables in a StarRocks database or a database in an external data source, for example, Hive, Iceberg, Hudi, or Delta Lake.

> **NOTE**
>
> To view tables in an external data source, you must have the USAGE privilege on the external catalog that corresponds to that data source.

Syntax

```
SHOW TABLES [FROM <catalog_name>.<db_name>]
```

Parameters

Parameter	Required	Description
catalog_name	No	The name of the internal catalog or an external catalog. • If you do not specify this parameter or set it to `default_catalog`, tables in StarRocks databases are returned. • If you set this parameter to the name of an external catalog, tables in databases of an external data source are returned. You can run SHOW CATALOGS to view internal and external catalogs.

Parameter	Required	Description
db_name	No	The database name. If not specified, the current database is used by default.

Examples

Example 1: View tables in database `example_db` of the `default_catalog` after connecting to the StarRocks cluster. The following two statements are equivalent.

```
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                     |
+---------------------------+
```

Example 2: View tables in the current database `example_db` after connecting to this database.

```
show tables;
+---------------------------+
| Tables_in_example_db      |
+---------------------------+
| depts                     |
| depts_par                 |
| emps                      |
```

```
| emps2                      |
+----------------------------+
```

Example 2: View tables in database `hudi_db` of the external catalog `hudi_catalog`.

```
show tables from hudi_catalog.hudi_db;
+----------------------------+
| Tables_in_hudi_db          |
+----------------------------+
| hudi_sync_mor              |
| hudi_table1                |
+----------------------------+
```

Alternatively, you can run SET CATALOG to switch to the external catalog `hudi_catalog` and then run `SHOW TABLES FROM hudi_db;`.

References

- SHOW CATALOGS: Views all catalogs in a StarRocks cluster.
- SHOW DATABASES: Views all databases in the internal catalog or an external catalog.

SHOW TABLET

Description

Displays tablet-related information.

> **NOTE**
>
> This operation requires the ADMIN_PRIV privilege.

Syntax

Query information of tablets in a table or a 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>]
```

Parameter	Required	Description
db_name	No	The database name. If you do not specify this parameter, the current database is used by default.
table_name	Yes	The name of the table from which you want to query tablet information. You must specify this parameter. Otherwise, an error is returned.

Parameter	Required	Description
partition_name	No	The name of the partition from which you want to query tablet information.
version_number	No	The data version number.
backend_id	No	The ID of the BE where the replica of the tablet is located.
STATE	No	The status of tablet replicas. • `NORMAL`: The replica is normal. • `ALTER`: A Rollup or schema change is being performed on the replica. • `CLONE`: The replica is being cloned. (Replicas in this state are not available for use). • `DECOMMISSION`: The replica is being decommissioned.
field_name	No	The field by which the results are sorted. All the fields returned by `SHOW TABLET FROM <table_name>` are sortable. • If you want to display results in ascending order, use `ORDER BY field_name ASC`. • If you want to display results in descending order, use `ORDER BY field_name DESC`.
offset	No	The number of tablets to skip from the results. For example, `OFFSET 5` means to skip the first five tablets. Default value: 0.
limit	No	The number of tablets to return. For example, `LIMIT 10` means to return only 10 tablets. If this parameter is not specified, all the tablets that meet the filter conditions are returned.

Query information of a single tablet

After obtaining all tablet IDs using `SHOW TABLET FROM <table_name>`, you can query the information of a single tablet.

```
SHOW TABLET <tablet_id>
```

Parameter	Required	Description
tablet_id	Yes	Tablet ID

Description of return fields

Query information of tablets in a table or a partition

```
+----------+-----------+-----------+------------+---------+----------
---+------------------+----------------------+-----------------+--
------------------+--------------+----------+----------+--------+-
----------------------+-------------+-----------------+----------
----+----------+----------+------------------+
| TabletId | ReplicaId | BackendId | SchemaHash | Version |
VersionHash | LstSuccessVersion | LstSuccessVersionHash |
LstFailedVersion | LstFailedVersionHash | LstFailedTime | DataSize |
RowCount | State  | LstConsistencyCheckTime | CheckVersion |
CheckVersionHash | VersionCount | PathHash | MetaUrl  |
CompactionStatus  |
+----------+-----------+-----------+------------+---------+----------
---+------------------+----------------------+-----------------+--
------------------+--------------+----------+----------+--------+-
----------------------+-------------+-----------------+----------
----+----------+----------+------------------+
```

Field	Description
TabletId	Table ID.
ReplicaId	Replica ID.

Field	Description
BackendId	The ID of the BE where the replica is located.
SchemaHash	Schema hash (randomly generated).
Version	Data version number.
VersionHash	Hash of data version number.
LstSuccessVersion	The last successfully loaded version.
LstSuccessVersionHash	The hash of the last successfully loaded version.
LstFailedVersion	The version of the last failed loading. `-1` indicates no version failed to be loaded.
LstFailedVersionHash	The hash of the last failed version.
LstFailedTime	The time of the last failed loading. `NULL` indicates there is no load failure.
DataSize	Data size of the tablet.
RowCount	The number of data rows of the tablet.
State	Replica status of the tablet.
LstConsistencyCheckTime	The time of the last consistency check. `NULL` indicates no consistency check was performed.
CheckVersion	The data version on which consistency check was performed. `-1` indicates no version was checked.
CheckVersionHash	The hash of the version on which consistency check was performed.
VersionCount	The total number of data versions.

Field	Description
PathHash	The hash of the directory in which the tablet is stored.
MetaUrl	The URL used to query more meta information.
CompactionStatus	The URL used to query data version compaction status.

Query information of a specific tablet

```
+--------+-----------+--------------+-----------+------+---------+--
-----------+---------+--------+-----------+
| DbName | TableName | PartitionName | IndexName | DbId | TableId |
PartitionId | IndexId | IsSync | DetailCmd |
+--------+-----------+--------------+-----------+------+---------+--
-----------+---------+--------+-----------+
```

Field	Description
DbName	The name of the database to which the tablet belongs.
TableName	The name of the table to which the tablet belongs.
PartitionName	The name of the partition to which the tablet belongs.
IndexName	The index name.
DbId	The database ID.
TableId	The table ID.
PartitionId	The partition ID.
IndexId	The index ID.
IsSync	Whether data on the tablet is consistent with table meta. `true` indicates data is consistent and the tablet is normal. `false` indicates

Field	Description
	data is missing on the tablet.
DetailCmd	The URL used to query more information.

Examples

Create table `test_show_tablet` in the database `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`);
```

- Example 1: Query information of all the tablets in the specified table. The following example excerpts information of only one tablet from the return information.

```
    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
```

- Example 2: Query information of 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';
```

- Example 3: Query information of tablets in partition `p20210103`.

```
SHOW TABLET FROM test_show_tablet partition(p20210103);
```

- Example 4: Return information of 10 tablets.

```
    SHOW TABLET FROM test_show_tablet limit 10;
```

- Example 5: Return information of 10 tablets with an offset 5.

```
SHOW TABLET FROM test_show_tablet limit 5,10;
```

- Example 6: Filter tablets by `backendid`, `version`, and `state`.

```
    SHOW TABLET FROM test_show_tablet
    WHERE backendid = 10004 and version = 1 and state = "NORMAL";
```

- Example 7: Sort tablets by `version`.

```
    SHOW TABLET FROM table_name where backendid = 10004 order by
version;
```

- Example 8: Return information of tablets whose index name is `test_show_tablet`.

```
SHOW TABLET FROM test_show_tablet where indexname =
"test_show_tablet";
```

SHOW TRANSACTION

Description

This syntax is used to view the transaction details of the specified transaction id.

Syntax:

```
SHOW TRANSACTION
[FROM <db_name>]
WHERE id = transaction_id
```

Examples of returned results:

```
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
CommitTime: 2020-01-09 14:59:09
FinishTime: 2020-01-09 14:59:09
Reason:
ErrorReplicasCount: 0
ListenerId: -1
TimeoutMs: 300000
```

- TransactionId: transaction id
- Label: import the label corresponding to the task
- Coordinator: the node responsible for transaction coordination
- TransactionStatus: transaction status
- PREPARE: preparation stage
- COMMITTED: the transaction succeeded, but the data is not visible
- VISIBLE: the transaction is successful and the data is visible
- ABORTED: transaction failed

- LoadJobSourceType: type of import task.
- PrepareTime: transaction start time
- CommitTime: the time when the transaction is successfully committed
- FinishTime: the time when the data is visible
- Reason: error message
- ErrorReplicasCount: number of replicas with errors
- ListenerId: id of the related import job
- TimeoutMs: transaction timeout, in milliseconds

Examples

1. To view a transaction with id 4005:

```
SHOW TRANSACTION WHERE ID=4005;
```

2. In the specified dB, view the transaction with id 4005:

```
SHOW TRANSACTION FROM db WHERE ID=4005;
```

SPARK LOAD

Description

Spark load preprocesses the imported data through external spark resources, improves the import performance of a large amount of StarRocks data, and saves the computing resources of StarRocks cluster. It is mainly used in the scenario of initial migration and large amount of data import into StarRocks.

Spark load is an asynchronous import method. Users need to create Spark type import tasks through MySQL protocol and view the import results through `SHOW LOAD`.

Syntax

```
LOAD LABEL load_label
(
data_desc1[, data_desc2, ...]
)
WITH RESOURCE resource_name
[resource_properties]
[opt_properties]
```

1.load_label

Label of the currently imported batch. Unique within a database.

Syntax:

```
[database_name.]your_label
```

2.data_desc

Used to describe a batch of imported data.

Syntax:

```
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]
```

Note

```
file_path:

The file path can be specified to one file, or the * wildcard can be
used to specify all files in a directory. Wildcards must match files,
not directories.

hive_external_tbl:

hive external table name.
It is required that the columns in the imported starrocks table must
exist in the hive external table.
Each load task only supports loading from one Hive external table.
Cannot be used with file_ path mode at the same time.

PARTITION:

If this parameter is specified, only the specified partition will be
imported, and the data outside the imported partition will be
filtered out.
If not specified, all partitions of table will be imported by
default.
```

NEGATIVE:

If this parameter is specified, it is equivalent to loading a batch of "negative" data. Used to offset the same batch of previously imported data.
This parameter is only applicable when the value column exists and the aggregation type of the value column is SUM only.

column_separator:

Specifies the column separator in the import file. Default is \ t
If it is an invisible character, you need to prefix it with \ \ x and use hexadecimal to represent the separator.
For example, the separator of hive file \ x01 is specified as "\ \ x01"

file_type:

Used to specify the type of imported file. Currently, supported file types are csv, orc, and parquet.

column_list:

Used to specify the correspondence between the columns in the import file and the columns in the table.
When you need to skip a column in the import file, specify the column as a column name that does not exist in the table.

Syntax:
(col_name1, col_name2, ...)

SET:

If specify this parameter, you can convert a column of the source file according to the function, and then import the converted results into table. Syntax is column_name = expression.
Only Spark SQL build_in functions are supported. Please refer to https://spark.apache.org/docs/2.4.6/api/sql/index.html.
Give a few examples to help understand.
Example 1: there are three columns "c1, c2, c3" in the table, and the first two columns in the source file correspond to (c1, c2), and the sum of the last two columns corresponds to C3; then columns (c1, c2, tmp_c3, tmp_c4) set (c3 = tmp_c3 + tmp_c4) needs to be specified;
Example 2: there are three columns "year, month and day" in the table, and there is only one time column in the source file in the

```
format of "2018-06-01 01:02:03".
Then you can specify columns (tmp_time) set (year = year (tmp_time),
month = month (tmp_time), day = day (tmp_time)) to complete the
import.

WHERE:

Filter the transformed data, and only the data that meets the where
condition can be imported. Only the column names in the table can be
referenced in the WHERE statement
```

3.resource_name

The name of the spark resource used can be viewed through `SHOW RESOURCES` command.

4.resource_properties

When you have a temporary need, such as modifying the Spark and HDFS configurations, you can set the parameters here, which takes effect only in this specific spark loading job and not to affect the existing configurations in the StarRocks cluster.

5.opt_properties

Used to specify some special parameters.

Syntax:

```
[PROPERTIES ("key"="value", ...)]
```

You can specify the following parameters: timeout: specifies the timeout of the import operation. The default timeout is 4 hours. In seconds. max_filter_ratio:the maximum allowable data proportion that can be filtered (for reasons such as non-standard data). The default is zero tolerance. strict mode: whether to strictly restrict the data. The default is false. timezone: specifies the time zone of some functions affected by the time zone, such as strftime / alignment_timestamp/from_unixtime, etc. Please refer to the [time zone] document for details. If not specified, the "Asia / Shanghai" time zone is used.

6.Import data format example

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. (Note: for other date formats, you can use strftime or time_format function to convert in the Import command) string class (CHAR/VARCHAR): "I am a student", "a"

NULL value: \ N

Examples

1. Import a batch of data from HDFS, and specify the timeout time and filtering ratio. Use the name my_ spark resources for spark.

```
LOAD LABEL example_db.label1
(
DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/starRocks/data/input/file")
INTO TABLE `my_table`
)
WITH RESOURCE 'my_spark'
PROPERTIES
(
    "timeout" = "3600",
    "max_filter_ratio" = "0.1"
);
```

   Where hdfs_host is the host of namenode, hdfs_port is fs.defaultfs port (default 9000)

2. Import a batch of "negative" data from HDFS, specify the separator as comma, use the wildcard * to specify all files in the directory, and specify the temporary parameters of spark resources.

```
LOAD LABEL example_db.label3
(
DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/starRocks/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. Import a batch of data from HDFS, specify the partition, and do some conversion on the columns of the imported file, as follows:

```
The table structure is:
k1 varchar(20)
k2 int

Assume that the data file has only one line of data:

Adele,1,1

Each column in the data file corresponds to each column specified
in the import statement:
k1,tmp_k2,tmp_k3

The conversion is as follows:

1. k1: no conversion
2. k2:is the sum of tmp_ k2 and tmp_k3

LOAD LABEL example_db.label6
(
DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/starRocks/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. Extract the partition field in the file path

   If necessary, the partitioned fields in the file path will be resolved according to the field types defined in the table, similar to the function of Partition Discovery in Spark

```
```

```
LOAD LABEL example_db.label10
(
DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/starRocks/data/input/dir/cit
INTO TABLE `my_table`
(k1, k2, k3)
COLUMNS FROM PATH AS (city, utc_date)
SET (uniq_id = md5sum(k1, city))
)
WITH RESOURCE 'my_spark';
```

`hdfs://hdfs_host:hdfs_port/user/starRocks/data/input/dir/city=beijing` The directory includes the following files:

`[hdfs://hdfs_host:hdfs_port/user/starRocks/data/input/dir/city=beijing/utc_date=2019-06-26/0000.csv, hdfs://hdfs_host:hdfs_port/user/starRocks/data/input/dir/city=beijing/utc_date=2019-06-26/0001.csv, ...]`

The city and utc_date field in the file path are extracted

5. Filter the data to be imported. Only columns with k1 value greater than 10 can be imported.

```
LOAD LABEL example_db.label10
(
DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/starRocks/data/input/file")
INTO TABLE `my_table`
WHERE k1 > 10
)
WITH RESOURCE 'my_spark';
```

6. Import from the hive external table and convert the uuid column in the source table to the bitmap type through the global dictionary.

```
LOAD LABEL db1.label1
(
DATA FROM TABLE hive_t1
INTO TABLE tbl1
SET
```

```
(
uuid=bitmap_dict(uuid)
)
)
WITH RESOURCE 'my_spark';
```

STOP ROUTINE LOAD

Description

Stops a Routine Load job.

You can manage Routine Load jobs that run on StarRocks tables only as a user who has the INSERT privilege on those StarRocks tables. If you do not have the INSERT privilege, follow the instructions provided in GRANT to grant the INSERT privilege to the user that you use to connect to your StarRocks cluster.

::: warning

- A stopped Routine Load job cannot be resumed. Therefore, please proceed with caution when executing this statement.
- If you only need to pause the Routine Load job, you can execute PAUSE ROUTINE LOAD.

:::

Syntax

```
STOP ROUTINE LOAD FOR [db_name.]<job_name>
```

Parameters

Parameter	Required	Description
db_name		The name of the database to which the Routine Load job belongs.
job_name	✅	The name of the Routine Load job.

Examples

Stop the Routine Load job `example_tbl1_ordertest1` in the database `example_db`.

```
STOP ROUTINE LOAD FOR example_db.example_tbl1_ordertest1;
```

STREAM LOAD

Description

StarRocks provides the loading method HTTP-based Stream Load to help you load data from a local file system or a streaming data source. After you submit a load job, StarRocks synchronously runs the job, and returns the result of the job after the job finishes. You can determine whether the job is successful based on the job result. For information about the application scenarios, limits, principles, and supported data file formats of Stream Load, see Loading from a local file system via Stream Load.

Note that Stream Load operations not only load data into StarRocks tables but also update the data in the materialized views that are created on the tables.

Syntax

```
curl --location-trusted -u <username>:<password> -XPUT <url>
(
    data_desc
)
[opt_properties]
```

This topic uses curl as an example to describe how to load data by using Stream Load. In addition to curl, you can also use other HTTP-compatible tools or languages to perform Stream Load. Load-related parameters are included in HTTP request header fields. When you input these parameters, take note of the following points:

- You can use chunked transfer encoding, as demonstrated in this topic. If you do not choose chunked transfer encoding, you must input a `Content-Length` header field to indicate the length of content to be transferred, thereby ensuring data integrity.

  > **NOTE**
  >
  > If you use curl to perform Stream Load, StarRocks automatically adds a `Content-Length` header field and you do not need manually input it.

- You must add an `Expect` header field and specify its value as `100-continue`, as in `"Expect:100-continue"`. This helps prevent unnecessary data transfers and reduce resource overheads in case your job request is denied.

Note that in StarRocks some literals are used as reserved keywords by the SQL language. Do not directly use these keywords in SQL statements. If you want to use such a keyword in an SQL statement, enclose it in a pair of backticks (`). See Keywords.

Parameters

username and password

Specify the username and password of the account that you use to connect to your StarRocks cluster. This is a required parameter. If you use an account for which no password is set, you need to input only `<username>:`.

XPUT

Specifies the HTTP request method. This is a required parameter. Stream Load supports only the PUT method.

url

Specifies the URL of the StarRocks table. Syntax:

```
http://<fe_host>:
<fe_http_port>/api/<database_name>/<table_name>/_stream_load
```

The following table describes the parameters in the URL.

Parameter	Required	Description
fe_host	Yes	The IP address of the FE node in your StarRocks cluster. **NOTE** If you submit the load job to a specific BE node, you must input the IP address of the BE node.

Parameter	Required	Description
fe_http_port	Yes	The HTTP port number of the FE node in your StarRocks cluster. The default port number is `8030`. **NOTE** If you submit the load job to a specific BE node, you must input the HTTP port number of the BE node. The default port number is `8030`.
database_name	Yes	The name of the database to which the StarRocks table belongs.
table_name	Yes	The name of the StarRocks table.

data_desc

Describes the data file that you want to load. The `data_desc` descriptor can include the data file's name, format, column separator, row separator, destination partitions, and column mapping against the StarRocks table. Syntax:

```
-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>"
```

The parameters in the `data_desc` descriptor can be divided into three types: common parameters, CSV parameters, and JSON parameters.

Common parameters

Parameter	Required	Description
file_path	Yes	The save path of the data file. You can optionally include the extension of the file name.
format	No	The format of the data file. Valid values: `CSV` and `JSON`. Default value: `CSV`.
partitions	No	The partitions into which you want to load the data file. By default, if you do not specify this parameter, StarRocks loads the data file into all partitions of the StarRocks table.
temporary_partitions	No	The name of the temporary partition into which you want to load data file. You can specify multiple temporary partitions, which must be separated by commas (,).
columns	No	The column mapping between the data file and the StarRocks table. If the fields in the data file can be mapped in sequence onto the columns in the StarRocks table, you do not need to specify this parameter. Instead, you can use this parameter to implement data conversions. For example, if you load a CSV data file and the file consists of two columns that can be mapped in sequence onto the two columns, `id` and `city`, of the StarRocks table, you can specify `"columns: city,tmp_id, id = tmp_id * 100"`. For more information, see the "Column mapping" section in this topic.

CSV parameters

Parameter	Required	Description
column_separator	No	The characters that are used in the data file to separate fields. If you do not specify this parameter,

Parameter	Required	Description
		this parameter defaults to `\t`, which indicates tab. Make sure that the column separator you specify by using this parameter is the same as the column separator used in the data file. **NOTE** For CSV data, you can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
row_delimiter	No	The characters that are used in the data file to separate rows. If you do not specify this parameter, this parameter defaults to `\n`.

NOTE

- For CSV data, you can use a UTF-8 string, such as a comma (,), tab, or pipe (|), whose length does not exceed 50 bytes as a text delimiter.
- Null values are denoted by using `\N`. For example, a data file consists of three columns, and a record from that data file holds data in the first and third columns but no data in the second column. In this situation, you need to use `\N` in the second column to denote a null value. This means the record must be compiled as `a,\N,b` instead of `a,,b`. `a,,b` denotes that the second column of the record holds an empty string.

JSON parameters

Parameter	Required	Description
jsonpaths	No	The names of the keys that you want to load from the JSON data file. You need to specify this parameter only when you load JSON data by using the matched mode. The value of this parameter is in JSON format. See Configure column mapping for JSON data loading.
strip_outer_array	No	Specifies whether to strip the outermost array structure. Valid values: `true` and `false`. Default

Parameter	Required	Description
		value: `false`. In real-world business scenarios, the JSON data may have an outermost array structure as indicated by a pair of square brackets `[]`. In this situation, we recommend that you set this parameter to `true`, so StarRocks removes the outermost square brackets `[]` and loads each inner array as a separate data record. If you set this parameter to `false`, StarRocks parses the entire JSON data file into one array and loads the array as a single data record. For example, the JSON data is `[ {"category" : 1, "author" : 2}, {"category" : 3, "author" : 4} ]`. If you set this parameter to `true`, `{"category" : 1, "author" : 2}` and `{"category" : 3, "author" : 4}` are parsed into separate data records that are loaded into separate StarRocks table rows.
json_root	No	The root element of the JSON data that you want to load from the JSON data file. You need to specify this parameter only when you load JSON data by using the matched mode. The value of this parameter is a valid JsonPath string. By default, the value of this parameter is empty, indicating that all data of the JSON data file will be loaded. For more information, see the "Load JSON data using matched mode with root element specified" section of this topic.
ignore_json_size	No	Specifies whether to check the size of the JSON body in the HTTP request. **NOTE** By default, the size of the JSON body in an HTTP request cannot exceed 100 MB. If the JSON body exceeds 100 MB in size, an error "The size of this batch exceed the max size [104857600] of json type data data [8617627793]. Set ignore_json_size to skip

Parameter	Required	Description
		check, although it may lead huge memory consuming." is reported. To prevent this error, you can add `"ignore_json_size:true"` in the HTTP request header to instruct StarRocks not to check the JSON body size.

When you load JSON data, also note that the size per JSON object cannot exceed 4 GB. If an individual JSON object in the JSON data file exceeds 4 GB in size, an error "This parser can't support a document that big." is reported.

opt_properties

Specifies some optional parameters, which are applied to the entire load job. Syntax:

```
-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 "merge_condition: <column_name>"
```

The following table describes the optional parameters.

Parameter	Required	Description
label	No	The label of the load job. If you do not specify this parameter, StarRocks automatically generates a label for the load job. StarRocks does not allow you to use one label to load a data batch multiple times. As such, StarRocks prevents the same data from being repeatedly loaded. For label naming conventions, see System limits. By default, StarRocks retains the labels of load jobs that were successfully completed over the most recent three days. You can use the FE parameter

Parameter	Required	Description
		`label_keep_max_second` to change the label retention period.
where	No	The conditions based on which StarRocks filters the pre-processed data. StarRocks loads only the pre-processed data that meets the filter conditions specified in the WHERE clause.
max_filter_ratio	No	The maximum error tolerance of the load job. The error tolerance is the maximum percentage of data records that can be filtered out due to inadequate data quality in all data records requested by the load job. Valid values: `0` to `1`. Default value: `0`. We recommend that you retain the default value `0`. This way, if unqualified data records are detected, the load job fails, thereby ensuring data correctness. If you want to ignore unqualified data records, you can set this parameter to a value greater than `0`. This way, the load job can succeed even if the data file contains unqualified data records. **NOTE** Unqualified data records do not include data records that are filtered out by the WHERE clause.
timeout	No	The timeout period of the load job. Valid values: `1` to `259200`. Unit: second. Default value: `600`. **NOTE**In addition to the `timeout` parameter, you can also use the FE parameter `stream_load_default_timeout_second` to centrally control the timeout period for all Stream Load jobs in your StarRocks cluster. If you specify the `timeout` parameter, the timeout period specified by the `timeout` parameter prevails. If you do not specify the `timeout` parameter, the timeout period specified by the `stream_load_default_timeout_second` parameter prevails.

Parameter	Required	Description
strict_mode	No	Specifies whether to enable the strict mode. Valid values: `true` and `false`. Default value: `false`. The value `true` specifies to enable the strict mode, and the value `false` specifies to disable the strict mode.
timezone	No	The time zone used by the load job. Default value: `Asia/Shanghai`. The value of this parameter affects the results returned by functions such as strftime, alignment_timestamp, and from_unixtime. The time zone specified by this parameter is a session-level time zone. For more information, see Configure a time zone.
load_mem_limit	No	The maximum amount of memory that can be provisioned to the load job. Unit: bytes. By default, the maximum memory size for a load job is 2 GB. The value of this parameter cannot exceed the maximum amount of memory that can be provisioned to each BE.
merge_condition	No	Specifies the name of the column you want to use as the condition to determine whether updates can take effect. The update from a source record to a destination record takes effect only when the source data record has a greater or equal value than the destination data record in the specified column. StarRocks supports conditional updates since v2.5. For more information, see Change data through loading. **NOTE** The column that you specify cannot be a primary key column. Additionally, only tables that use the Primary Key tables support conditional updates.

Column mapping

Configure column mapping for CSV data loading

If the columns of the data file can be mapped one on one in sequence to the columns of the StarRocks table, you do not need to configure the column mapping between the data file and the StarRocks table.

If the columns of the data file cannot be mapped one on one in sequence to the columns of the StarRocks table, you need to use the `columns` parameter to configure the column mapping between the data file and the StarRocks table. This includes the following two use cases:

- **Same number of columns but different column sequence. Also, the data from the data file does not need to be computed by functions before it is loaded into the matching StarRocks table columns.**

  In the `columns` parameter, you need to specify the names of the StarRocks table columns in the same sequence as how the data file columns are arranged.

  For example, the StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence, and the data file also consists of three columns, which can be mapped to the StarRocks table columns `col3`, `col2`, and `col1` in sequence. In this case, you need to specify `"columns: col3, col2, col1"`.

- **Different number of columns and different column sequence. Also, the data from the data file needs to be computed by functions before it is loaded into the matching StarRocks table columns.**

  In the `columns` parameter, you need to specify the names of the StarRocks table columns in the same sequence as how the data file columns are arranged and specify the functions you want to use to compute the data. Two examples are as follows:

  - The StarRocks table consists of three columns, which are `col1`, `col2`, and `col3` in sequence. The data file consists of four columns, among which the first three columns can be mapped in sequence to the StarRocks table columns `col1`, `col2`, and `col3` and the fourth column cannot be mapped to any of the StarRocks table columns. In this case, you need to temporarily specify a name for the fourth column of the data file, and the temporary name must be different from any of the StarRocks table column names. For example, you can specify `"columns: col1, col2, col3, temp"`, in which the fourth column of the data file is temporarily named `temp`.

    - The StarRocks table consists of three columns, which are `year`, `month`, and `day` in sequence. The data file consists of only one column that accommodates date and time values in `yyyy-mm-dd hh:mm:ss` format. In this case, you can specify `"columns: col, year = year(col), month=month(col), day=day(col)"`, in which `col` is the temporary name of the data file column and the functions `year = year(col)`, `month=month(col)`, and `day=day(col)` are used to extract data from the data file column `col` and loads the data into the mapping StarRocks table columns. For example, `year = year(col)` is used to extract the `yyyy` data from the data file column `col` and loads the data into the StarRocks table column `year`.

For detailed examples, see Configure column mapping.

Configure column mapping for JSON data loading

If the keys of the JSON document have the same names as the columns of the StarRocks table, you can load the JSON-formatted data by using the simple mode. In simple mode, you do not need to specify the `jsonpaths` parameter. This mode requires that the JSON-formatted data must be an object as indicated by curly brackets `{}`, such as `{"category": 1, "author": 2, "price": "3"}`. In this example, `category`, `author`, and `price` are key names, and these keys can be mapped one on one by name to the columns `category`, `author`, and `price` of the StarRocks table.

If the keys of the JSON document have different names than the columns of the StarRocks table, you can load the JSON-formatted data by using the matched mode. In matched mode, you need to use the `jsonpaths` and `COLUMNS` parameters to specify the column mapping between the JSON document and the StarRocks table:

- In the `jsonpaths` parameter, specify the JSON keys in the sequence as how they are arranged in the JSON document.
- In the `COLUMNS` parameter, specify the mapping between the JSON keys and the StarRocks table columns:
    - The column names specified in the `COLUMNS` parameter are mapped one on one in sequence to the JSON keys.
    - The column names specified in the `COLUMNS` parameter are mapped one on one by name to the StarRocks table columns.

For examples about loading JSON-formatted data by using the matched mode, see Load JSON data using matched mode.

Return value

After the load job finishes, StarRocks returns the job result in JSON format. Example:

```
{
    "TxnId": 1003,
    "Label": "label123",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 999999,
    "NumberFilteredRows": 1,
    "NumberUnselectedRows": 0,
    "LoadBytes": 40888898,
    "LoadTimeMs": 2144,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMS": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 11,
    "CommitAndPublishTimeMs": 16,
}
```

The following table describes the parameters in the returned job result.

Parameter	Description
TxnId	The transaction ID of the load job.
Label	The label of the load job.
Status	The final status of the data loaded. • `Success`: The data is successfully loaded and can be queried. • `Publish Timeout`: The load job is successfully submitted, but the data still cannot be queried. You do not need to retry to load the data.

Parameter	Description
	• `Label Already Exists`: The label of the load job has been used for another load job. The data may have been successfully loaded or is being loaded. • `Fail`: The data failed to be loaded. You can retry the load job.
Message	The status of the load job. If the load job fails, the detailed failure cause is returned.
NumberTotalRows	The total number of data records that are read.
NumberLoadedRows	The total number of data records that are successfully loaded. This parameter is valid only when the value returned for `Status` is `Success`.
NumberFilteredRows	The number of data records that are filtered out due to inadequate data quality.
NumberUnselectedRows	The number of data records that are filtered out by the WHERE clause.
LoadBytes	The amount of data that is loaded. Unit: bytes.
LoadTimeMs	The amount of time that is taken by the load job. Unit: ms.
BeginTxnTimeMs	The amount of time that is taken to run a transaction for the load job.
StreamLoadPutTimeMS	The amount of time that is taken to generate a execution plan for the load job.
ReadDataTimeMs	The amount of time that is taken to read data for the load job.
WriteDataTimeMs	The amount of time that is taken to write data for the load job.

Parameter	Description
CommitAndPublishTimeMs	The amount of time that is taken to commit and publish data for the load job.

If the load job fails, StarRocks also returns `ErrorURL`. Example:

```
{"ErrorURL": "http://172.26.195.68:8045/api/_load_error_log?
file=error_log_3a4eb8421f0878a6_9a54df29fd9206be"}
```

`ErrorURL` provides a URL from which you can obtain details about unqualified data records that have been filtered out. StarRocks retains 1,000 unqualified data records.

You can run `curl "url"` to directly view details about the filtered-out, unqualified data records. You can also run `wget "url"` to export the details about these data records:

```
wget http://172.26.195.68:8045/api/_load_error_log?
file=error_log_3a4eb8421f0878a6_9a54df29fd9206be
```

The exported data record details are saved to a local file with a name similar to `_load_error_log?file=error_log_3a4eb8421f0878a6_9a54df29fd9206be`. You can use the `cat` command to view the file.

Then, you can adjust the configuration of the load job, and submit the load job again.

Examples

Load CSV data

This section CSV data as an example to describe how you can employ various parameter settings and combinations to meet your diverse loading requirements.

Set timeout period

Your StarRocks database `test_db` contains a table named `table1`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example1.csv` also consists of three columns, which can be mapped in sequence onto `col1`, `col2`, and `col3` of `table1`.

If you want to load all data from `example1.csv` into `table1` within up to 100 seconds, run the following command:

```
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
```

Set error tolerance

Your StarRocks database `test_db` contains a table named `table2`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example2.csv` also consists of three columns, which can be mapped in sequence onto `col1`, `col2`, and `col3` of `table2`.

If you want to load all data from `example2.csv` into `table2` with a maximum error tolerance of `0.2`, run the following command:

```
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
```

Configure column mapping

Your StarRocks database `test_db` contains a table named `table3`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example3.csv` also consists of three columns, which can be mapped in sequence onto `col2`, `col1`, and `col3` of `table3`.

If you want to load all data from `example3.csv` into `table3`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, the columns of `example3.csv` cannot be mapped onto the columns of `table3` in the same sequence as how these columns are arranged in `table3`. Therefore, you need to use the `columns` parameter to configure the column mapping between `example3.csv` and `table3`.

Set filter conditions

Your StarRocks database `test_db` contains a table named `table4`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example4.csv` also consists of three columns, which can be mapped in sequence onto `col1`, `col2`, and `col3` of `table4`.

If you want to load only the data records whose values in the first column of `example4.csv` are equal to `20180601` into `table4`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, `example4.csv` and `table4` have the same number of columns that can be mapped in sequence, but you need to use the WHERE clause to specify column-based filter conditions. Therefore, you need to use the `columns` parameter to define temporary names for the columns of `example4.csv`.

Set destination partitions

Your StarRocks database `test_db` contains a table named `table5`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example5.csv` also consists of three columns, which can be mapped in sequence onto `col1`, `col2`, and `col3` of `table5`.

If you want to load all data from `example5.csv` into partitions `p1` and `p2` of `table5`, run the following command:

```
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
```

Set strict mode and time zone

Your StarRocks database `test_db` contains a table named `table6`. The table consists of three columns, which are `col1`, `col2`, and `col3` in sequence.

Your data file `example6.csv` also consists of three columns, which can be mapped in sequence onto `col1`, `col2`, and `col3` of `table6`.

If you want to load all data from `example6.csv` into `table6` by using the strict mode and the time zone `Africa/Abidjan`, run the following command:

```
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
```

Load data into tables containing HLL-type columns

Your StarRocks database `test_db` contains a table named `table7`. The table consists of two HLL-type columns, which are `col1` and `col2` in sequence.

Your data file `example7.csv` also consists of two columns, among which the first column can be mapped onto `col1` of `table7` and the second column cannot be mapped onto any

column of `table7`. The values in the first column of `example7.csv` can be converted into HLL-type data by using functions before they are loaded into `col1` of `table7`.

If you want to load data from `example7.csv` into `table7`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, the two columns of `example7.csv` are named `temp1` and `temp2` in sequence by using the `columns` parameter. Then, functions are used to convert data as follows:
>
> - The `hll_hash` function is used to convert the values in `temp1` of `example7.csv` into HLL-type data and map `temp1` of `example7.csv` onto `col1` of `table7`.
> - The `hll_empty` function is used to fill the specified default value into `col2` of `table7`.

For usage of the functions `hll_hash` and `hll_empty`, see hll_hash and hll_empty.

Load data into tables containing BITMAP-type columns

Your StarRocks database `test_db` contains a table named `table8`. The table consists of two BITMAP-type columns, which are `col1` and `col2`, in sequence.

Your data file `example8.csv` also consists of two columns, among which the first column can be mapped onto `col1` of `table8` and the second column cannot be mapped onto any column of `table8`. The values in the first column of `example8.csv` can be converted by using functions before they are loaded into `col1` of `table8`.

If you want to load data from `example8.csv` into `table8`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, the two columns of `example8.csv` are named `temp1` and `temp2` in sequence by using the `columns` parameter. Then, functions are used to convert data as follows:
>
> - The `to_bitmap` function is used to convert the values in `temp1` of `example8.csv` into BITMAP-type data and map `temp1` of `example8.csv` onto `col1` of `table8`.
> - The `bitmap_empty` function is used to fill the specified default value into `col2` of `table8`.

For usage of the functions `to_bitmap` and `bitmap_empty`, see to_bitmap and bitmap_empty.

Load JSON data

This section describes the parameter settings that you need to take note of when you load JSON data.

Your StarRocks database `test_db` contains a table named `tbl1`, whose schema is as follows:

```
`category` varchar(512) NULL COMMENT "",`author` varchar(512) NULL
COMMENT "",`title` varchar(512) NULL COMMENT "",`price` double NULL
COMMENT ""
```

Load JSON data using simple mode

Suppose that your data file `example1.json` consists of the following data:

```
{"category":"C++","author":"avc","title":"C++ primer","price":895}
```

To load all data from `example1.json` into `tbl1`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, the parameters `columns` and `jsonpaths` are not specified. Therefore, the keys in `example1.json` are mapped by name onto the columns of `tbl1`.

To increase throughput, Stream Load supports loading multiple data records all at once. Example:

```
[{"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}]
```

Load JSON data using matched mode

StarRocks performs the following steps to match and process JSON data:

1. (Optional) Strips the outermost array structure as instructed by the `strip_outer_array` parameter setting.

   > **NOTE**
   >
   > This step is performed only when the outermost layer of the JSON data is an array structure as indicated by a pair of square brackets `[]`. You need to set `strip_outer_array` to `true`.

2. (Optional) Matches the root element of the JSON data as instructed by the `json_root` parameter setting.

   > **NOTE**

> This step is performed only when the JSON data has a root element. You need to specify the root element by using the `json_root` parameter.

3. Extracts the specified JSON data as instructed by the `jsonpaths` parameter setting.

Load JSON data using matched without root element specified

Suppose that your data file `example2.json` consists of the following data:

```
[{"category":"xuxb111","author":"1avc","title":"SayingsoftheCentury","p
{"category":"xuxb222","author":"2avc","title":"SayingsoftheCentury","pr
{"category":"xuxb333","author":"3avc","title":"SayingsoftheCentury","pr
```

To load only `category`, `author`, and `price` from `example2.json`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, the outermost layer of the JSON data is an array structure as indicated by a pair of square brackets `[]`. The array structure consists of multiple JSON objects that each represent a data record. Therefore, you need to set `strip_outer_array` to `true` to strip the outermost array structure. The key **title** that you do not want to load is ignored during loading.

Load JSON data using matched mode with root element specified

Suppose your data file `example3.json` consists of the following data:

```
{"id": 10001,"RECORDS":
[{"category":"11","title":"SayingsoftheCentury","price":895,"timestamp"
{"category":"22","author":"2avc","price":895,"timestamp":1589191487},
```

```
{"category":"33","author":"3avc","title":"SayingsoftheCentury","timesta
["3 records", "there will be 3 rows"]}
```

To load only `category`, `author`, and `price` from `example3.json`, run the following command:

```
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
```

> **NOTE**
>
> In the preceding example, the outermost layer of the JSON data is an array structure as indicated by a pair of square brackets `[]`. The array structure consists of multiple JSON objects that each represent a data record. Therefore, you need to set `strip_outer_array` to `true` to strip the outermost array structure. The keys `title` and `timestamp` that you do not want to load are ignored during loading. Additionally, the `json_root` parameter is used to specify the root element, which is an array, of the JSON data.

SUBMIT TASK

Description

Submits an ETL statement as an asynchronous task. This feature has been supported since StarRocks v2.5.

StarRocks v2.5 supports submitting asynchronous tasks for CREATE TABLE AS SELECT.

Syntax

```
SUBMIT TASK [task_name] AS <etl_statement>
```

Parameters

Parameter	Description
task_name	The task name.
etl_statement	The ETL statement that you want to submit as an asynchronous task. StarRocks currently supports submitting asynchronous tasks for CREATE TABLE AS SELECT.

Usage notes

This statement creates a Task, which is a template for storing a task that executes the ETL statement. You can check the information of the Task by querying the metadata table `tasks` in Information Schema.

```
SELECT * FROM INFORMATION_SCHEMA.tasks;
SELECT * FROM information_schema.tasks WHERE task_name =
'<task_name>';
```

After you run the Task, a TaskRun is generated accordingly. A TaskRun indicates a task that executes the ETL statement. A TaskRun has the following states:

- `PENDING`: The task waits to be run.
- `RUNNING`: The task is running.
- `FAILED`: The task failed.
- `SUCCESS`: The task runs successfully.

You can check the state of a TaskRun by querying the metadata table `task_runs` in Information Schema.

```
SELECT * FROM INFORMATION_SCHEMA.task_runs;
SELECT * FROM information_schema.task_runs WHERE task_name =
'<task_name>';
```

Configure via FE configuration items

You can configure asynchronous ETL tasks using the following FE configuration items:

Parameter	Default value	Description
task_ttl_second	86400	The period during which a Task is valid. Unit: seconds. Tasks that exceed the validity period are deleted.
task_check_interval_second	14400	The time interval to delete invalid Tasks. Unit: seconds.
task_runs_ttl_second	86400	The period during which a TaskRun is valid. Unit: seconds. TaskRuns that exceed the validity period are deleted

Parameter	Default value	Description
		automatically. Additionally, TaskRuns in the `FAILED` and `SUCCESS` states are also deleted automatically.
task_runs_concurrency	4	The maximum number of TaskRuns that can be run in parallel.
task_runs_queue_length	500	The maximum number of TaskRuns that are pending for running. If the number exceeds the default value, the incoming tasks will be suspended.
task_runs_max_history_number	10000	The maximum number of TaskRun records to retain.

Examples

Example 1: Submit an asynchronous task for `CREATE TABLE tbl1 AS SELECT * FROM src_tbl`, and specify the task name as `etl0`:

```
SUBMIT TASK etl0 AS CREATE TABLE tbl1 AS SELECT * FROM src_tbl;
```

UPDATE

Updates rows in a Primary Key table.

StarRocks supports the UPDATE statement since v2.3, which only supports single-table UPDATE and does not support common table expressions (CTEs).

Syntax

```
UPDATE <table_name>
    SET <column_name> = <expression> [, ...]
    WHERE <where_condition>
```

Parameters

`table_name`

The name of the table to be updated.

`column_name`

The name of the column to be updated. It cannot include the table name. For example, 'UPDATE t1 SET col = 1' is not valid.

`expression`

The expression that assigns new values to the column.

`where_condition`

The condition based on which you want to update rows. Only rows that meet the WHERE condition can be updated. This parameter is required, because it helps prevent you from accidentally updating the entire table. If you want to update the entire table, you can use 'WHERE true'.

Examples

Create a table `Employees` to record employee information and insert five data rows into the table.

```
CREATE TABLE Employees (
    EmployeeID INT,
    Name VARCHAR(50),
    Salary DECIMAL(10, 2)
)
PRIMARY KEY (EmployeeID)
DISTRIBUTED BY HASH (EmployeeID) BUCKETS 1
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);
```

If you need to give a 10% raise to all employees, you can execute the following statement:

```
UPDATE Employees
SET Salary = Salary * 1.1  -- Increase the salary by 10%.
WHERE true;
```

If you need to give a 10% raise to employees with salaries lower than the average salary, you can execute the following statement:

```
UPDATE Employees
SET Salary = Salary * 1.1   -- Increase the salary by 10%.
WHERE Salary < (SELECT AVG(Salary) FROM Employees);
```

Limitations

From v2.3 to v2.5, the UPDATE statement only supports single-table UPDATE and does not support common table expressions (CTEs).

DESC

Description

You can use the statement to perform the following operations:

- View the schema of a table stored in your StarRocks cluster, along with the type of the sort key and materialized view of the table.
- View the schema of a table stored in the following external data sources, such as Apache Hive™. Note that you can perform this operation only in StarRocks 2.4 and later versions.

Syntax

```
DESC[RIBE] [catalog_name.][db_name.]table_name [ALL];
```

Parameters

Parameter	Required	Description
catalog_name	No	The name of the internal catalog or an external catalog. • If you set the value of the parameter to the name of the internal catalog, which is `default_catalog`, you can view the schema of the table stored in your StarRocks cluster. • If you set the value of the parameter to the name of an external catalog, you can view the schema of the table stored in the external data source.
db_name	No	The database name.

Parameter	Required	Description
table_name	Yes	The table name.
ALL	No	• If this keyword is specified, you can view the type of the sort key, materialized view, and schema of a table stored in your StarRocks cluster. If this keyword is not specified, you only view the table schema. • Do not specify this keyword when you view the schema of a table stored in an external data source.

Output

```
+-----------+--------------+-------+------+------+-----+---------+--
-----+
| IndexName | IndexKeysType | Field | Type | Null | Key | Default |
Extra |
+-----------+--------------+-------+------+------+-----+---------+--
-----+
```

The following table describes the parameters returned by this statement.

Parameter	Description
IndexName	The table name. If you view the schema of a table stored in an external data source, this parameter is not returned.
IndexKeysType	The type of the sort key of the table. If you view the schema of a table stored in an external data source, this parameter is not returned.
Field	The column name.
Type	The data type of the column.
Null	Whether the column values can be NULL. • `yes`: indicates the values can be NULL.

Parameter	Description
	• `no`: indicates the values cannot be NULL.
Key	Whether the column is used as the sort key. • `true`: indicates the column is used as the sort key. • `false`: indicates the column is not used as the sort key.
Default	The default value for the data type of the column. If the data type does not have a default value, a NULL is returned.

Extra	• If you see the schema of a table stored in your StarRocks cluster, this field displays the following information about the column:
	  ◦ The aggregate function used by the column, such as `SUM` and `MIN`.
	  ◦ Whether a bloom filter index is created on the column. If so, the value of `Extra` is `BLOOM_FILTER`.
	• If you see the schema of a table stored in external data sources, this field displays whether the column is the partition column. If the column is the partition column, the value of `Extra` is `partition key`.

> Note: For information about how a materialized view is displayed in the output, see Example 2.

Examples

Example 1: View the schema of `example_table` stored in your StarRocks cluster.

```
DESC example_table;
```

Or

```
DESC default_catalog.example_db.example_table;
```

The output of the preceding statements is as follows.

```
+-------+--------------+------+-------+---------+-------+
| 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    |       |
+-------+--------------+------+-------+---------+-------+
```

Example 2: View the schema, type of the sort key, and materialized view of `sales_records` stored in your StarRocks cluster. In the following example, one materialized view `store_amt` is created based on `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   |
+--------------+--------------+-----------+--------+------+-------
+---------+-------+
```

Example 3: View the schema of `hive_table` stored in your 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 |
+-------+----------------+------+-------+---------+---------------+
```

References

- CREATE DATABASE
- SHOW CREATE DATABASE
- USE
- SHOW DATABASES
- DROP DATABASE

Data type overview

StarRocks supports the following data types: Numeric, String, Date, Semi-structured, and Others. You can define columns of these types when you create a table, load data into these columns, and query the data.

Numeric

- TINYINT
- SMALLINT
- INT
- BIGINT
- LARGEINT
- DECIMAL
- DOUBLE
- FLOAT
- BOOLEAN
- PERCENTILE

String

- CHAR
- STRING
- VARCHAR

Date

- DATE
- DATETIME

Semi-structured

- ARRAY
- JSON

Others

- BITMAP
- HLL

TINYINT

Description

SMALLINT

Description

INT

Description

BIGINT

Description

LARGEINT

Description

DECIMAL

Description

DOUBLE

Description

FLOAT

Description

BOOLEAN

Description

TINYINT

Description

TINYINT

1 byte signed integer, range [-128, 127]

SMALLINT

Description

SMALLINT

2-byte signed integer, range [-32768, 32767]

INT

Description

INT

4-byte signed integer, range [-2147483648, 2147483647]

BIGINT

Description

BIGINT

BIGINT 8-byte signed integer, range[-9223372036854775808, 9223372036854775807]

LARGEINT

Description

LARGEINT

16 byte signed integer, range [-2^127 + 1 ~ 2^127 - 1]

DECIMAL

Description

DECIMAL(P[,S])

High-precision fixed-point, P stands for the total number of significant numbers (precision), S stands for the maximum number of decimal points (scale).

- Decimal V2

  The range of P is [1,27], the range of S is [0,9], in addition, P must be greater than or equal to the value of S. The default value of S is 0.

- Fast Decimal (version 1.18 default)

  The range of P is [1,38], and the range of S is [0, P]. The default value of S is 0. Since starrocks-1.18, the decimal type supports FastDecimal with higher precision.

  The main optimizations are:

  1. Multiple width integers are used internally to represent decimal. Decimal (P < = 18, S) uses 64bit integers. Compared with the original decimal V2 implementation, 128bit integers are used uniformly. Arithmetic operations and conversion operations use fewer instructions on 64bit processors, so the performance is greatly improved.

  2. Compared with decimal V2, the Fast Decimal implementation has made extreme optimization of specific algorithms, especially multiplication, and the performance is improved by about 4 times.

  Current restrictions:

  1. At present, fast decimal does not support array type. If users want to use array (decimal) type, please use array (double) type, or use array (decimal) type after closing decimal v3.

  2. In hive direct connection, orc and parquet data formats do not support decimal yet.

DOUBLE

Description

DOUBLE

8-byte floating point number

FLOAT

Description

FLOAT

4-byte floating point number

BOOLEAN

Description

BOOL, BOOLEAN

Like TINYINT, 0 stands for false and 1 for true.

CHAR

Description

STRING

Description

VARCHAR

Description

CHAR

Description

CHAR(M)

A fixed-length string, M represents the length of a fixed-length string. The range of M is 1-255.

STRING

Description

STRING

A variable length string, maximum length is 65533 byte

VARCHAR

Description

VARCHAR(M)

A variable-length string. `M` indicates the length of the string. The default value is `1`. Unit: bytes.

- In versions earlier than StarRocks 2.1, the value range of `M` is 1–65533.
- [Preview] In StarRocks 2.1 and later versions, the value range of `M` is 1–1048576.

Examples

Create a table and specify the column type as 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) BUCKETS 4;
```

Management

DATE

Description

DATETIME

Description

DATE

Description

DATE type. The current value range is ['0000-01-01','9999-12-31'] and the default format is `YYYY-MM-DD`.

Examples

Example 1: Specify a column as the DATE type when you create a table.

```
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) BUCKETS 4;
```

Example 2: Convert a DATETIME value into a DATE value.

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

For more information, see the date function.

DATETIME

Description

DATETIME

Date and time type, value range is ['0000-01-01 00:00:00','9999-12-31 23:59:59'].

The form of printing is 'YYYY-MM-DD HH:MM:SS'

Work with semi-structured data

ARRAY

ARRAY, as an extended type of database, is supported in various database systems such as Postgr...

JSON

StarRocks starts to support the JSON data type since v2.2.0. This topic describes the basic conce...

ARRAY

ARRAY, as an extended type of database, is supported in various database systems such as PostgreSQL, ClickHouse, and Snowflake. ARRAY is widely used in scenarios such as A/B tests, user tag analysis, and user profiling. StarRocks supports multidimensional array nesting, array slicing, comparison, and filtering.

Define ARRAY columns

You can define an ARRAY column when you create a table.

```
-- Define a one-dimensional array.
ARRAY<type>

-- Define a nested array.
ARRAY<ARRAY<type>>

-- Define an array column as NOT NULL.
ARRAY<type> NOT NULL
```

`type` specifies the data types of elements in an array. StarRocks supports the following element types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, CHAR, DATETIME, and DATE.

Elements in an array are nullable by default, for example, `[null, 1 ,2]`. You cannot specify elements in an array as NOT NULL. However, you can specify an ARRAY column as NOT NULL when you create a table, such as the third example in the following code snippet.

Examples:

```
-- Define c1 as a one-dimensional array whose element type is INT.
create table t0(
  c0 INT,
  c1 ARRAY<INT>
)
duplicate key(c0)
```

```
distributed by hash(c0) buckets 3;

-- Define c1 as an nested array whose element type is VARCHAR.
create table t1(
  c0 INT,
  c1 ARRAY<ARRAY<VARCHAR(10)>>
)
duplicate key(c0)
distributed by hash(c0) buckets 3;

-- Define c1 as a NOT NULL array column.
create table t2(
  c0 INT,
  c1 ARRAY<INT> NOT NULL
)
duplicate key(c0)
distributed by hash(c0) buckets 3;
```

Limits

The following limits apply when you create ARRAY columns in StarRocks tables:

- In versions earlier than v2.1, you can create ARRAY columns only in Duplicate Key tables. From v2.1 onwards, you can also create ARRAY columns in other types of tables (Primary Key, Unique Key, Aggregate). Note that in an Aggregate table, you can create an ARRAY column only when the function used to aggregate data in that column is replace() or replace_if_not_null(). For more information, see Aggregate table.
- ARRAY columns cannot be used as key columns.
- ARRAY columns cannot be used as partition keys (included in PARTITION BY) or bucketing keys (included in DISTRIBUTED BY).
- DECIMAL V3 is not supported in ARRAY.
- An array can have a maximum of 14-level nesting.

Construct arrays in SQL

Arrays can be constructed in SQL using brackets `[]`, with each array element separated by a comma (`,`).

```
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]    |
+----------+
```

StarRocks automatically infers data types if an array consists of elements of multiple types:

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

mysql> select [12, "100"];

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

You can use pointed brackets (`<>`) to show the declared array type.

```
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]               |
+------------------------+
```

NULLs can be included in the element.

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

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

For an empty array, you can use pointed brackets to show the declared type, or you can write [] directly for StarRocks to infer the type based on the context. If StarRocks cannot infer the type, it will report an error.

```
mysql> select [];

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

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

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

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

mysql> select array_append([], 10);

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

Load Array data

StarRocks supports loading Array data in three ways:

- INSERT INTO is suitable for loading small-scale data for testing.
- Broker Load is suitable for loading ORC or Parquet files with large-scale data.
- Stream Load and Routine Load are suitable for loading CSV files with large-scale data.

Use INSERT INTO to load arrays

You can use INSERT INTO to load small-scale data column by column, or perform ETL on data before loading the data.

```
create table t0(
c0 INT,
c1 ARRAY<INT>
)
duplicate key(c0)
distributed by hash(c0) buckets 3;

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

Use Broker Load to load arrays from ORC or Parquet files

The array type in StarRocks corresponds to the list structure in ORC and Parquet files, which eliminates the need for you to specify different data types in StarRocks. For more information about data loading, see Broker load.

Use Stream Load or Routine Load to load CSV-formatted arrays

Arrays in CSV files are separated with comma by default. You can use Stream Load or Routine Load to load CSV text files or CSV data in Kafka.

Query ARRAY data

You can access elements in an array using `[]` and subscripts, starting from `1`.

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

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

If the subscript is 0 or a negative number, **no error is reported and NULL is returned**.

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

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

If the subscript exceeds the length of the array (the number of elements in the array), **NULL will be returned**.

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

+------------+
| [1,2,3][4] |
+------------+
|       NULL |
```

```
+------------+
1 row in set (0.01 sec)
```

For multidimensional arrays, the elements can be accessed **recursively**.

```
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

StarRocks starts to support the JSON data type since v2.2.0. This topic describes the basic concepts of JSON. It also describes how to create a JSON column, load JSON data, query JSON data, and use JSON functions and operators to construct and process JSON data.

What is JSON

JSON is a lightweight, data-interchange format that is designed for semi-structured data. JSON presents data in a hierarchical tree structure, which is flexible and easy to read and write in a wide range of data storage and analytics scenarios. JSON supports `NULL` values and the following data types: NUMBER, STRING, BOOLEAN, ARRAY, and OBJECT.

For more information about JSON, visit the JSON website. For information about the input and output syntax of JSON, see JSON specifications at RFC 7159.

StarRocks supports both storage and efficient querying and analytics of JSON data. StarRocks does not directly store the input text. Instead, it stores JSON data in a binary format to reduce the cost of parsing and increase query efficiency.

Use JSON data

Create a JSON column

When you create a table, you can use the `JSON` keyword to specify the `j` column as a JSON column.

```
CREATE TABLE `tj` (
    `id` INT(11) NOT NULL COMMENT "",
    `j`  JSON NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`) BUCKETS 1
```

```
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);
```

Load data and store the data as JSON data

StarRocks provides the following methods for you to load data and store the data as JSON data:

- Method 1: Use `INSERT INTO` to write data to a JSON column of a table. In the following example, a table named `tj` is used, and the `j` column of the table is a JSON column.

```
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));
```

> The parse_json function can interpret STRING data as JSON data. The json_object function can construct a JSON object or convert an existing table to a JSON file. For more information, see parse_json and json_object.

- Method 2: Use Stream Load to load a JSON file and store the file as JSON data. For more information, see Load JSON data.
    - If you want to load a root JSON object, set `jsonpaths` to `$`.
    - If you want to load specific values of a JSON object, set `jsonpaths` to `$.a`, in which `a` specifies a key. For more information about JSON path expressions supported in StarRocks, see JSON path.
- Method 3: Use Broker Load to load a Parquet file and store the file as JSON data. For more information, see Broker Load.

StarRocks supports the following data type conversions at Parquet file loading.

Data type of Parquet file	JSON data type
INTEGER (INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, and UINT64)	NUMBER
FLOAT and DOUBLE	NUMBER
BOOLEAN	BOOLEAN
STRING	STRING
MAP	OBJECT
STRUCT	OBJECT
LIST	ARRAY
Other data types such as UNION and TIMESTAMP	Not supported

- Method 4: Use Routine load to continuously load JSON data from Kafka into StarRocks.

Query and process JSON data

StarRocks supports the querying and processing of JSON data and the use of JSON functions and operators.

In the following examples, a table named `tj` is used, and the `j` column of the table is specified as the JSON column.

```
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} |
+------+--------------------+
```

Example 1: Filter the data of the JSON column to retrieve the data that meets the `id=1` filter condition.

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

Example 2: Filter data of the JSON column `j` to retrieve the data that meets the specified filter condition.

> `j->'a'` returns JSON data. You can use the first example to compare data (Note that implicit conversion is performed in this example). Alternatively, you can convert JSON data to INT by using the CAST function and then compare the data.

```
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} |
+------+--------------------+
```

Example 3: Use the CAST function to convert the values in the JSON column of the table to BOOLEAN values. Then, filter the data of the JSON column to retrieve the data that meets the specified filter condition.

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

```
| 3    | {"a": 3, "b": true} |
+------+---------------------+
```

Example 4: Use the CAST function to convert the values in the JSON column of the table to BOOLEAN values. Then, filter the data of the JSON column to retrieve the data that meets the specified filter condition, and perform arithmetic operations on the data.

```
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            |
+--------------------------+
```

Example 5: Sort the data of the table by using the JSON column as a sort key.

```
mysql> select * from tj
    ->        where j->'a' <= parse_json('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} |
+------+---------------------+
4 rows in set (0.05 sec)
```

JSON functions and operators

You can use JSON functions and operators to construct and process JSON data. For more information, see Overview of JSON functions and operators.

Limits and usage notes

- The maximum length of a JSON value is 16 MB.
- The ORDER BY, GROUP BY, and JOIN clauses do not support references to JSON columns. If you want to create references to JSON columns, use the CAST function to convert JSON columns to SQL columns before you create the references. For more information, see cast.
- JSON columns are supported in Duplicate Key, Primary Key, and Unique Key tables. They are not supported in Aggregate tables.
- JSON columns cannot be used as partition keys, bucketing keys, or dimension columns (DUPLICATE KEY, PRIMARY KEY, UNIQUE KEY). They cannot be used in ORDER BY, GROUP BY, and JOIN clauses.
- StarRocks allows you to use the following JSON comparison operators to query JSON data: `<`, `<=`, `>`, `>=`, `=`, and `!=`. It does not allow you to use `IN` to query JSON data.

Other

BITMAP

BITMAP is often used to accelerate count distinct. It is similar but more accurate in count distinct t...

HLL (HyperLogLog)

Description

BITMAP

BITMAP is often used to accelerate count distinct. It is similar but more accurate in count distinct than HyperLogLog (HLL). BITMAP consumes more memory and disk resources. It supports only aggregation of INT data. If you want to apply bitmap to string data, you must map the data using the low-cardinality dictionary.

This topic provides a simple example on how to create a BITMAP column and use bitmap functions to aggregate data for that column. For detailed function definitions or more Bitmap functions, see "Bitmap functions".

Create a table

- Create an Aggregate table, in which the data type of the `user_id` column is BITMAP and the bitmap_union() function is used to aggregate data.

```
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`) BUCKETS 2;
```

- Create a Primary Key table, in which the data type of the `userid` column is 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`) BUCKETS 1;
```

Before you insert data into BITMAP columns, you must first use the to_bitmap() function to convert data.

For details about how to use BITMAP, for example, load BITMAP data into a table, see bitmap.

HLL (HyperLogLog)

Description

HLL is used for approximate count distinct.

HLL enables the development of programmes based on the HyperLogLog algorithm. It is used to store intermediate results of the HyperLogLog calculation process. It can only be used as the value column type of the table. HLL reduces the amount of data through aggregation to speed up the query process. There may be a 1% deviation in the estimated results.

HLL column is generated based on the imported data or data from other columns. When data is imported, the hll_hash function specifies which column will be used to generated the HLL column. HLL is often used to replace COUNT DISTINCT and quickly calculate unique views (UVs) with rollup.

The storage space used by HLL is determined by the distinct values in the hash value. The storage space varies depending on three conditions:

- HLL is empty. No value is inserted into HLL and the storage cost is the lowest, which is 80 bytes.
- The number of distinct hash values in HLL is less than or equal to 160. The highest storage cost is 1360 bytes (80 + 160 * 8 = 1360).
- The number of distinct hash values in HLL is greater than 160. The storage cost is fixed at 16,464 bytes (80 + 16 * 1024 = 16464).

In actual business scenarios, data volume and data distribution affect the memory usage of queries and the accuracy of the approximate result. You need to consider these two factors:

- Data volume: HLL returns an approximate value. A larger data volume results in a more accurate result. A smaller data volume results in larger deviation.
- Data distribution： In the case of large data volume and high-cardinality dimension column for GROUP BY， data computation will use more memory. HLL is not recommended in this situation. It is recommended when you perform no-group-by count distinct or GROUP BY on low-cardinality dimension columns.

- Query granularity: If you query data at a large query granularity, we recommend you use the Aggregate table or materialized view to pre-aggregate data to reduce data volume.

Related functions

- HLL_UNION_AGG(hll): This function is an aggregate function used to estimate the cardinality of all data that meet the conditions. This can also be used to analyze functions. It only supports default window and does not support window clause.
- HLL_RAW_AGG(hll): This function is an aggregate function used to aggregate fields of hll type and returns with hll type.
- HLL_CARDINALITY(hll): This function is used to estimate the cardinality of a single hll column.
- HLL_HASH(column_name): This generates HLL column type and is used for inserts or imports. See the instructions for the use of imports.
- HLL_EMPTY: This generates empty HLL column and is used to fill in default values during inserts or imports. See the instructions for the use of imports.

Examples

1. Create a table with HLL columns `set1` and `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. Load data by using Stream Load.

```
a. Use table columns to generate an HLL column.
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. Use data columns to generate an HLL column.
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. Aggregate data in the following three ways: (Without aggregation, direct query on base table may be as slow as using approx_count_distinct)

```
-- a. Create a rollup to aggregate HLL column.
alter table test add rollup test_rollup(dt, set1);

-- b. Create another table to calculate uv and insert data into
it

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. Create another table to calculate UV. Insert data and
generate HLL column by testing other columns through 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. Query data. HLL columns do not support direct query to its original values. It can be queried by matching functions.

```
a. Calculate the total UV.
select HLL_UNION_AGG(uv_set) from test_uv;

b. Calculate the UV for each day.
select dt, HLL_CARDINALITY(uv_set) from test_uv;

c. Calculate the aggregation value of set1 in the test table.
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;
```

Keywords

This topic describes non-reserved keywords and reserved keywords. It provides a list of reserved keywords in StarRocks.

Introduction

Keywords in SQL statements, such as `CREATE` and `DROP`, have special meanings when parsed by StarRocks. Keywords are classified into non-reserved keywords and reserved keywords.

- **Non-reserved keywords** can be directly used as identifiers without special treatment, such as table names and column names. For example, `DB` is a non-reserved keyword. You can create a database named `DB`.

```
CREATE DATABASE DB;
Query OK, 0 rows affected (0.00 sec)
```

- **Reserved keywords** can be used as identifiers only after special treatment. For example, `LIKE` is a reserved keyword. If you want to use it to identify a database, enclose it in a pair of backticks (`).

```
CREATE DATABASE `LIKE`;
Query OK, 0 rows affected (0.01 sec)
```

If it is not enclosed in backticks, an error is returned:

```
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}.
```

Reserved keywords

Following are StarRocks reserved keywords arranged in alphabetical order. You must enclose them in a pair of backticks (`) if you want to use them as identifiers. The reserved keywords may vary with StarRocks versions.

A

- ADD
- ALL
- ALTER
- ANALYZE
- AND
- ANTI
- ARRAY
- AS
- ASC
- AUTHENTICATION

B

- BETWEEN
- BIGINT
- BITMAP
- BOTH
- BY

C

- CANCEL
- CASE
- CHAR
- CHARACTER
- CHECK
- COLLATE

- COLUMN
- CONVERT
- CREATE
- CROSS
- CUBE
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER

D

- DATABASE
- DATABASES
- DECIMAL
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128
- DEFAULT
- DELETE
- DENSE_RANK
- DESC
- DESCRIBE
- DISTINCT
- DISTRIBUTED
- DOUBLE
- DROP
- DUAL

E

- ELSE
- EXCEPT

- EXISTS
- EXPLAIN
- EXPORT

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

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
- NODES
- NOT
- NULL

O

- ON
- OR
- ORDER
- OUTER
- OUTFILE
- OVER

P

- PARTITION
- PERCENTILE
- PRIMARY
- PROCEDURE

Q

- QUALIFY

R

- RANGE
- RANK
- READ

- REGEXP
- RELEASE
- RENAME
- REPLACE
- REVOKE
- RIGHT
- RLIKE
- ROW
- ROWS
- ROW_NUMBER

S

- SCHEMA
- SCHEMAS
- SELECT
- SEMI
- SET
- SET_VAR
- SHOW
- SMALLINT
- SYSTEM
- SWAP

T

- TABLE
- TERMINATED
- THEN
- TINYINT
- TO
- TRUE

U

- UNION
- UNIQUE
- UNSIGNED
- UPDATE
- USE
- USING

V

- VALUES
- VARCHAR

W

- WHEN
- WHERE
- WITH

Function list

StarRocks offers a rich set of functions to facilitate data queries and analysis. In addition to commonly used functions, StarRocks supports semi-structured functions such as ARRAY, JSON, and MAP functions. It also supports higher-order Lambda functions. If these functions cannot meet your business requirements, you can use Java UDF to compile functions.

You can find a function based on the following categories:

- Function list
  - Aggregate functions
  - Array functions
  - Bit functions
  - Bitmap functions
  - Conditional functions
  - Cryptographic functions
  - Date and time functions
  - Geographic functions
  - Hash functions
  - JSON functions
  - String functions
  - Map functions
  - Math functions
  - Pattern matching functions
  - Percentile functions
  - Scalar functions
  - Table functions
  - Utility functions

Aggregate functions

- any_value
- approx_count_distinct
- array_agg
- avg
- bitmap
- bitmap_agg
- corr
- covar_pop
- covar_samp
- count
- group_concat
- grouping
- grouping_id
- group_concat
- hll_empty
- hll_hash
- hll_raw_agg
- hll_union
- hll_union_agg
- max
- max_by
- min
- multi_distinct_sum
- multi_distinct_count
- percentile_approx
- percentile_cont
- percentile_disc
- retention
- stddev，stddev_pop
- stddev_samp
- sum
- variance, variance_pop, var_pop
- var_samp

- window_funnel

Array functions

- array_agg
- array_append
- array_avg
- array_concat
- array_contains
- array_contains_all
- array_cum_sum
- array_difference
- array_distinct
- array_filter
- array_intersect
- array_join
- array_length
- array_map
- array_max
- array_min
- array_position
- array_remove
- array_slice
- array_sort
- array_sortby
- array_sum
- arrays_overlap
- array_to_bitmap
- reverse
- unnest

Bit functions

- bitand
- bitnot
- bitor
- bitxor
- bit_shift_left
- bit_shift_right
- bit_shift_right_logical

Bitmap functions

- base64_to_bitmap
- bitmap_agg
- bitmap_and
- bitmap_andnot
- bitmap_contains
- bitmap_count
- bitmap_from_string
- bitmap_empty
- bitmap_has_any
- bitmap_hash
- bitmap_intersect
- bitmap_max
- bitmap_min
- bitmap_or
- bitmap_remove
- bitmap_to_array
- bitmap_to_base64
- bitmap_to_string
- bitmap_union
- bitmap_union_count
- bitmap_union_int
- bitmap_xor
- intersect_count

- sub_bitmap
- to_bitmap
- subdivide_bitmap
- sub_bitmap
- to_bitmap

Conditional functions

- case
- coalesce
- if
- ifnull
- nullif

Cryptographic functions

- aes_encrypt
- aes_decrypt
- from_base64
- md5
- md5sum
- md5sum_numeric
- sha2
- sm3
- to_base64

Date and time functions

- add_months
- adddate, days_ad
- convert_tz
- current_date，curdate
- current_time，curtime

- current_timestamp
- date
- date_add
- date_format
- date_slice
- date_sub, subdate
- date_trunc
- datediff
- day
- dayname
- dayofmonth
- dayofweek
- dayofyear
- days_add
- days_diff
- days_sub
- from_days
- from_unixtime
- hour
- hours_add
- hours_diff
- hours_sub
- microseconds_add
- microseconds_sub
- minute
- minutes_add
- minutes_diff
- minutes_sub
- month
- monthname
- months_add
- months_diff
- months_sub

- now
- quarter
- second
- seconds_add
- seconds_diff
- seconds_sub
- str_to_date
- str2date
- time_slice
- time_to_sec
- timediff
- days_diff
- timestamp
- timestampadd
- timestampdiff
- to_date
- to_days
- unix_timestamp
- utc_timestamp
- week
- weekofyear
- weeks_add
- weeks_diff
- weeks_sub
- year
- years_add
- years_diff
- years_sub

Geographic functions

- ST_AsText, ST_AsWKT
- st_circle

- st_contains
- st_distance_sphere
- st_geometryfromtext
- st_linefromtext, ST_LineStringFromText
- st_point
- st_polygon
- st_x
- st_y

Hash functions

- murmur_hash3_32

JSON functions

- json_array
- json_object
- parse_json
- Arrow function
- cast
- get_json_double
- get_json_int
- get_json_strin,get_json_object
- json_each
- json_exists
- json_length
- json_keys
- json_query
- json_string

String functions

- append_trailing_char_if_absent

- ascii
- char
- char_length
- character_length
- concat
- concat_ws
- ends_with
- find_in_set
- group_concat
- hex
- instr
- lcase
- left
- length
- locate
- lower
- lpad
- ltrim
- money_format
- null_or_empty
- parse_url
- repeat
- reverse
- right
- rpad
- rtrim
- space
- split
- split_part
- starts_with
- strleft
- strright
- substr, substring

- trim
- ucase
- unhex
- upper

Map functions

- map_keys
- map_size
- map_values

Math functions

- abs
- acos
- asin
- atan
- atan2
- bin
- ceil, dceil
- ceiling
- conv
- cos
- cosine_similarity
- cosine_similarity_norm
- cot
- degrees
- divide
- e
- expexp, dexp
- floor, dfloor
- fmod
- greatest

- least
- ln, dlog1, log
- log
- log2
- log10, dlog10
- mod
- multiply
- negative
- pi
- pmod
- positive
- pow, power, dpow, fpow
- radians
- rand, random
- round, dround
- sign
- sin
- sqrt, dsqrt
- square
- tan
- truncate

Pattern matching functions

- like
- regexp
- regexp_extract
- regexp_replace

Percentile functions

- percentile_approx_raw
- percentile_empty

- percentile_hash
- percentile_union

Scalar functions

- hll_cardinality

Table functions

- json_each
- subdivide_bitmap
- unnest

Utility functions

- catalog
- current_version
- database
- host_name
- isnull
- last_query_id
- sleep
- uuid
- uuid_numeric
- version

Java UDFs

From v2.2.0 onwards, you can compile user-defined functions (UDFs) to suit your specific business needs by using the Java programming language.

This topic describes the types of UDFs supported by StarRocks, how to develop UDFs, and how to use the UDFs you have developed.

Currently, StarRocks supports scalar UDFs, user-defined aggregate functions (UDAFs), user-defined window functions (UDWFs), and user-defined table functions (UDTFs).

Prerequisites

- You have installed Apache Maven, so you can create and compile Java projects.
- You have installed JDK 1.8 on your servers.
- The Java UDF feature is enabled. You can set the FE configuration item `enable_udf` to `true` in the FE configuration file **fe/conf/fe.conf** to enable this feature, and then restart the FE nodes to make the settings take effect. For more information, see Parameter configuration.

Develop and use UDFs

You need to create a Maven project and compile the UDF you need by using the Java programming language.

Step 1: Create a Maven project

Create a Maven project, whose basic directory structure is as follows:

```
project
|--pom.xml
|--src
|  |--main
```

```
|   |   |--java
|   |   |--resources
|   |--test
|--target
```

Step 2: Add dependencies

Add the following dependencies to the **pom.xml** file:

```
<?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>8</maven.compiler.source>
        <maven.compiler.target>8</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>
```

Step 3: Compile a UDF

Use the Java programming language to compile a UDF.

Compile a scalar UDF

A scalar UDF operates on a single row of data and returns a single value. When you use a scalar UDF in a query, each row corresponds to a single value in the result set. Typical scalar functions include `UPPER`, `LOWER`, `ROUND`, and `ABS`.

Suppose that the values of a field in your JSON data are JSON strings rather than JSON objects. When you use an SQL statement to extract JSON strings, you need to run `GET_JSON_STRING` twice, for example, `GET_JSON_STRING(GET_JSON_STRING('{"key":"{\\"k0\\":\\"v0\\"}"}', "$.key"), "$.k0")`.

To simplify the SQL statement, you can compile a scalar UDF that can directly extract JSON strings, for example, `MY_UDF_JSON_GET('{"key":"{\\"k0\\":\\"v0\\"}"}', "$.key.k0")`.

```
package com.starrocks.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 {
            // The JSONPath library can be fully expanded even if the
values of a field are JSON strings.
            return JSONPath.read(obj, key).toString();
        } catch (Exception e) {
            return null;
        }
    }
}
```

The user-defined class must implement the method described in the following table.

> **NOTE**
>
> The data types of the request parameters and return parameters in the method must be the same as those declared in the `CREATE FUNCTION` statement that is to be executed in Step 6 and conform to the mapping that is provided in the "Mapping between SQL data types and Java data types" section of this topic.

Method	Description
TYPE1 evaluate(TYPE2, ...)	Runs the UDF. The evaluate() method requires the public member access level.

Compile a UDAF

A UDAF operates on multiple rows of data and returns a single value. Typical aggregate functions include `SUM`, `COUNT`, `MAX`, and `MIN`, which aggregate multiple rows of data specified in each GROUP BY clause and return a single value.

Suppose that you want to compile a UDAF named `MY_SUM_INT`. Unlike the built-in aggregate function `SUM`, which returns BIGINT-type values, the `MY_SUM_INT` function supports only request parameters and returns parameters of the INT data type.

```
package com.starrocks.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;
    }
}
```

The user-defined class must implement the methods described in the following table.

> **NOTE**
>
> The data types of the request parameters and return parameters in the methods must be the same as those declared in the `CREATE FUNCTION` statement that is to be executed in Step 6 and conform to the mapping that is provided in the "Mapping between SQL data types and Java data types" section of this topic.

Method	Description
State create()	Creates a state.
void destroy(State)	Destroys a state.
void update(State, ...)	Updates a state. In addition to the first parameter `State`, you can also specify one or more request parameters in the UDF declaration.
void serialize(State, ByteBuffer)	Serializes a state into the byte buffer.
void merge(State, ByteBuffer)	Deserializes a state from the byte buffer, and merges the byte buffer into the state as the first parameter.
TYPE finalize(State)	Obtains the final result of the UDF from a state.

During compilation, you must also use the buffer class `java.nio.ByteBuffer` and the local variable `serializeLength`, which are described in the following table.

Class and local variable	Description
java.nio.ByteBuffer()	The buffer class, which stores intermediate results. Intermediate results may be serialized or deserialized when they are transmitted between nodes for execution. Therefore, you must also use the `serializeLength` variable to specify the length that is allowed for the deserialization of intermediate results.

Class and local variable	Description
serializeLength()	The length that is allowed for the deserialization of intermediate results. Unit: bytes. Set this local variable to an INT-type value. For example, `State { int counter = 0; public int serializeLength() { return 4; }}` specifies that intermediate results are of the INT data type and the length for deserialization is 4 bytes. You can adjust these settings based on your business requirements. For example, if you want to specify the data type of intermediate results as LONG and the length for deserialization as 8 bytes, pass `State { long counter = 0; public int serializeLength() { return 8; }}`.

Take note of the following points for the deserialization of intermediate results stored in the `java.nio.ByteBuffer` class:

- The `remaining()` method that is dependent on the `ByteBuffer` class cannot be called to deserialize a state.
- The `clear()` method cannot be called on the `ByteBuffer` class.
- The value of `serializeLength` must be the same as the length of the written-in data. Otherwise, incorrect results are generated during serialization and deserialization.

Compile a UDWF

Unlike regular aggregate functions, a UDWF operates on a set of multiple rows, which are collectively called a window, and returns a value for each row. A typical window function includes an `OVER` clause that divides rows into multiple sets. It performs a calculation across each set of rows and returns a value for each row.

Suppose that you want to compile a UDWF named `MY_WINDOW_SUM_INT`. Unlike the built-in aggregate function `SUM`, which returns BIGINT-type values, the `MY_WINDOW_SUM_INT` function supports only request parameters and returns parameters of the INT data type.

```
package com.starrocks.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];
        }
    }
}
```

The user-defined class must implement the method required by UDAFs (because a UDWF is a special aggregate function) and the `windowUpdate()` method described in the following table.

> **NOTE**
>
> The data types of the request parameters and return parameters in the method must be the same as those declared in the `CREATE FUNCTION` statement that is to be executed in Step 6 and conform to the mapping that is provided in the "Mapping between SQL data types and Java data types" section of this topic.

Method	Description
void windowUpdate(State state, int, int, int , int, ...)	Updates the data of the window. For more information about UDWFs, see Window functions. Every time when you enter a row as input, this method obtains the window information and updates intermediate results accordingly. • `peer_group_start`: the start position of the current partition. `PARTITION BY` is used in the OVER clause to specify a partition column. Rows with the same values in the partition column are considered to be in the same partition. • `peer_group_end`: the end position of the current partition. • `frame_start`: the start position of the current window frame. The window frame clause specifies a calculation range, which covers the current row and the rows that are within a specified distance to the current row. For example, `ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING` specifies a calculation range that covers the current row, the previous row before the current row, and the following row after the current row.

Method	Description
	• `frame_end`: the end position of the current window frame. • `inputs`: the data that is entered as the input to a window. The data is an array package that supports only specific data types. In this example, INT values are entered as input, and the array package is Integer[].

Compile a UDTF

A UDTF reads one row of data and returns multiple values that can be considered to be a table. Table-valued functions are typically used to transform rows into columns.

> **NOTE**
>
> StarRocks allows a UDTF to return a table that consists of multiple rows and one column.

Suppose that you want to compile a UDTF named `MY_UDF_SPLIT`. The `MY_UDF_SPLIT` function allows you to use spaces as delimiters and supports request parameters and return parameters of the STRING data type.

```
package com.starrocks.udf.sample;

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

The method defined by the user-defined class must meet the following requirements:

> **NOTE**
>
> The data types of the request parameters and return parameters in the method must be the same as those declared in the `CREATE FUNCTION` statement that is to be executed in Step 6 and conform to the mapping that is provided in the "Mapping between SQL data types and Java data types" section of this topic.

Method	Description
TYPE[] process()	Runs the UDTF and returns an array.

Step 4: Package the Java project

Run the following command to package the Java project:

```
mvn package
```

The following JAR files are generated in the **target** folder: **udf-1.0-SNAPSHOT.jar** and **udf-1.0-SNAPSHOT-jar-with-dependencies.jar**.

Step 5: Upload the Java project

Upload the JAR file **udf-1.0-SNAPSHOT-jar-with-dependencies.jar** to an HTTP server that keeps up and running and is accessible to all FEs and BEs in your StarRocks cluster. Then, run the following command to deploy the file:

```
mvn deploy
```

You can set up a simple HTTP server by using Python and upload the JAR file to that HTTP server.

> **NOTE**
>
> In Step 6, the FEs will check the JAR file that contains the code for the UDF and calculate the checksum, and the BEs will download and execute the JAR file.

Step 6: Create the UDF in StarRocks

After you upload the JAR file mentioned above, you can create the UDF you have compiled in StarRocks.

Syntax

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

Parameters

Parameter	Required	Description
AGGREGATE	No	Whether to create a UDAF or UDWF.
TABLE	No	Whether to create a UDTF. If both `AGGREGATE` and `TABLE` are not specified, a Scalar function is created.
function_name	Yes	The name of the function you want to create. You can include the name of the database in this parameter, for example, `db1.my_func`. If `function_name` includes the database name, the UDF is created in that database. Otherwise, the UDF is created in the current database. The name of the new function and its parameters cannot be the same as an existing name in the destination database. Otherwise, the function cannot be created. The creation succeeds if the function name is the same but the parameters are different.
arg_type	Yes	Argument type of the function. The added argument can be represented by `, ...`. For the supported data types, see Mapping between SQL data types and Java data types.
return_type	Yes	The return type of the function. For the supported data types, see Java UDF.
PROPERTIES	Yes	Properties of the function, which vary depending on the type of the UDF to create.

Create a scalar UDF

Run the following command to create the scalar UDF you have compiled in the preceding example:

```
CREATE FUNCTION MY_UDF_JSON_GET(string, string)
RETURNS string
properties (
    "symbol" = "com.starrocks.udf.sample.UDFJsonGet",
    "type" = "StarrocksJar",
    "file" = "http://http_host:http_port/udf-1.0-SNAPSHOT-jar-with-
dependencies.jar"
);
```

Parameter	Description
symbol	The name of the class for the Maven project to which the UDF belongs. The value of this parameter is in the `<package_name>.<class_name>` format.
type	The type of the UDF. Set the value to `StarrocksJar`, which specifies that the UDF is a Java-based function.
file	The HTTP URL from which you can download the JAR file that contains the code for the UDF. The value of this parameter is in the `http://<http_server_ip>:<http_server_port>/<jar_package_name>` format.

Create a UDAF

Run the following command to create the UDAF you have compiled in the preceding example:

```
CREATE AGGREGATE FUNCTION MY_SUM_INT(INT)
RETURNS INT
PROPERTIES
(
    "symbol" = "com.starrocks.udf.sample.SumInt",
    "type" = "StarrocksJar",
    "file" = "http://http_host:http_port/udf-1.0-SNAPSHOT-jar-with-
```

```
dependencies.jar"
);
```

The descriptions of the parameters in PROPERTIES are the same as those in Create a scalar UDF.

Create a UDWF

Run the following command to create the UDWF you have compiled in the preceding example:

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

`analytic`: Whether the UDF is a window function. Set the value to `true`. The descriptions of other properties are the same as those in Create a scalar UDF.

Create a UDTF

Run the following command to create the UDTF you have compiled in the preceding example:

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

The descriptions of the parameters in PROPERTIES are the same as those in Create a scalar UDF.

Step 7: Use the UDF

After you create the UDF, you can test and use it based on your business needs.

Use a scalar UDF

Run the following command to use the scalar UDF you have created in the preceding example:

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

Use a UDAF

Run the following command to use the UDAF you have created in the preceding example:

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

Use a UDWF

Run the following command to use the UDWF you have created in the preceding example:

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

Use a UDTF

Run the following command to use the UDTF you have created in the preceding example:

```
-- Suppose that you have a table named t1, and the information about
its columns a, b, and c1 is as follows:
SELECT t1.a,t1.b,t1.c1 FROM t1;
> output:
```

```
1,2.1,"hello world"
2,2.2,"hello UDTF."

-- Run the MY_UDF_SPLIT() function.
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."
```

> **NOTE**
>
> - The first `MY_UDF_SPLIT` in the preceding code snippet is the alias of the column that is returned by the second `MY_UDF_SPLIT`, which is a function.
> - You cannot use `AS t2(f1)` to specify the aliases of the table and its columns that are to be returned.

View UDFs

Run the following command to query UDFs:

```
SHOW FUNCTIONS;
```

For more information, see SHOW FUNCTIONS.

Drop UDF

Run the following command to drop a UDF:

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

For more information, see DROP FUNCTION.

Mapping between SQL data types and Java data types

SQL TYPE	Java TYPE
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

Parameter settings

Configure the following environment variable in the **be/conf/be.conf** file of each Java virtual machine (JVM) in your StarRocks cluster to control memory usage. If you use JDK 8, configure `JAVA_OPTS`. If you use JDK 9 or later, configure `JAVA_OPTS_FOR_JDK_9_AND_LATER`.

```
JAVA_OPTS="-Xmx12G"

JAVA_OPTS_FOR_JDK_9_AND_LATER="-Xmx12G"
```

FAQ

Can I use static variables when I create UDFs? Do the static variables of different UDFs have mutual impacts on each other?

Yes, you can use static variables when you compile UDFs. The static variables of different UDFs are isolated from each other and do not affect each other even if the UDFs have classes with identical names.

Window function

Background

The window function is a special class of built-in functions. Similar to the aggregation function, it also does calculations on multiple input rows to get a single data value. The difference is that the window function processes the input data within a specific window, rather than using the "group by" method. The data in each window can be sorted and grouped using the over() clause. The window function **computes a separate value for each row**, rather than computing one value for each group. This flexibility allows users to add additional columns to the select clause and further filter the result set. The window function can only appear in the select list and the outermost position of a clause. It takes effect at the end of the query, that is, after the `join`, `where`, and `group by` operations are performed. The window function is often used to analyze trends, calculate outliers, and perform bucketing analyses on large-scale data.

Usage

Syntax of the window function:

```
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]
...]
```

Functions

Currently supported functions include:

- MIN(), MAX(), COUNT(), SUM(), AVG()
- FIRST_VALUE(), LAST_VALUE(), LEAD(), LAG()
- ROW_NUMBER(), RANK(), DENSE_RANK(), QUALIFY()

- NTILE()
- VARIANCE(), VAR_SAMP(), STD(), STDDEV_SAMP(), COVAR_SAMP(), COVAR_POP(), CORR()

PARTITION BY clause

The Partition By clause is similar to Group By. It groups the input rows by one or more specified columns. Rows with the same value are grouped together.

ORDER BY clause

The `Order By` clause is basically the same as the outer `Order By`. It defines the order of the input rows. If `Partition By` is specified, `Order By` defines the order within each Partition grouping. The only difference is that `Order By n` (n is a positive integer) in the `OVER` clause is equivalent to no operation, whereas `n` in the outer `Order By` indicates sorting by the nth column.

Example:

This example shows adding an id column to the select list with values of 1, 2, 3, etc., sorted by the `date_and_time` column in the events table.

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

Window clause

The window clause is used to specify a range of rows for operations (the previous and later lines based on the current line). It supports the following syntaxes – AVG(), COUNT(), FIRST_VALUE(), LAST_VALUE() and SUM(). For MAX() and MIN(), the window clause can specify the start to `UNBOUNDED PRECEDING`.

Syntax:

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

Example:

Suppose we have the following stock data, the stock symbol is JDR, and the closing price is the daily closing price.

```
create table stock_ticker (
    stock_symbol string,
    closing_price decimal(8,2),
    closing_date timestamp);

-- ...load some data...

select *
from stock_ticker
order by stock_symbol, closing_date
```

The raw data was shown as follows:

```
+--------------+---------------+---------------------+
| stock_symbol | closing_price | closing_date        |
+--------------+---------------+---------------------+
| 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 |
+--------------+---------------+---------------------+
```

This query uses the window function to generate the moving_average column whose value is the 3-day (previous day, current day, and next day) average stock price. The first day does not have the value of its previous day, and the last day does not have the value of the day after, so these two rows only calculate the average value of two days. Here `Partition By` does not take effect, because all the data is JDR data. However, if there is other stock information, `Partition By` will ensure that the window function is operated within each Partition.

```
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;
```

The following data is obtained:

```
+--------------+---------------------+---------------+---------------
-+
| stock_symbol | closing_date        | closing_price | moving_average
|
+--------------+---------------------+---------------+---------------
-+
| JDR          | 2014-10-02 00:00:00 | 12.86         | 12.87
|
| JDR          | 2014-10-03 00:00:00 | 12.89         | 12.89
|
| JDR          | 2014-10-04 00:00:00 | 12.94         | 12.79
|
| JDR          | 2014-10-05 00:00:00 | 12.55         | 13.17
|
| JDR          | 2014-10-06 00:00:00 | 14.03         | 13.77
|
| JDR          | 2014-10-07 00:00:00 | 14.75         | 14.25
|
| JDR          | 2014-10-08 00:00:00 | 13.98         | 14.36
|
+--------------+---------------------+---------------+---------------
-+
```

Function Examples

This section describes the window functions supported in StarRocks.

AVG()

Syntax:

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

Example:

Calculate the x-average of the current row and each row before and after it.

```
select x, property,
    avg(x)
        over (
            partition by property
            order by x
            rows between 1 preceding and 1 following
        ) as 'moving average'
from int_t
where property in ('odd','even');
```

```
+----+----------+----------------+
| x  | property | moving average |
+----+----------+----------------+
| 2  | even     | 3              |
| 4  | even     | 4              |
| 6  | even     | 6              |
| 8  | even     | 8              |
| 10 | even     | 9              |
| 1  | odd      | 2              |
| 3  | odd      | 3              |
| 5  | odd      | 5              |
| 7  | odd      | 7              |
| 9  | odd      | 8              |
+----+----------+----------------+
```

COUNT()

Syntax:

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

Example:

Count the occurrence of x from the current row to the first row.

```
select x, property,
    count(x)
        over (
            partition by property
            order by x
            rows between unbounded preceding and current row
        ) as 'cumulative total'
from int_t where property in ('odd','even');
```

```
+----+----------+------------------+
| x  | property | cumulative count |
+----+----------+------------------+
| 2  | even     | 1                |
| 4  | even     | 2                |
| 6  | even     | 3                |
| 8  | even     | 4                |
| 10 | even     | 5                |
| 1  | odd      | 1                |
| 3  | odd      | 2                |
| 5  | odd      | 3                |
| 7  | odd      | 4                |
| 9  | odd      | 5                |
+----+----------+------------------+
```

DENSE_RANK()

The DENSE_RANK() function is used to represent rankings. Unlike RANK(), DENSE_RANK()**does not have vacant** numbers. For example, if there are two tied 1s, the third number of DENSE_RANK() is still 2, whereas the third number of RANK() is 3.

Syntax:

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

The following example shows the ranking of column x according to the property column grouping.

```
select x, y,
    dense_rank()
        over (
```

```
            partition by x
            order by y
        ) as `rank`
from int_t;
```

```
+---+---+------+
| x | y | rank |
+---+---+------+
| 1 | 1 | 1    |
| 1 | 2 | 2    |
| 1 | 2 | 2    |
| 2 | 1 | 1    |
| 2 | 2 | 2    |
| 2 | 3 | 3    |
| 3 | 1 | 1    |
| 3 | 1 | 1    |
| 3 | 2 | 2    |
+---+---+------+
```

NTILE

NTILE() function divides the sorted rows in a partition by the specified number of `num_buckets` as equally as possible, stores the divided rows in the respective buckets, starting from 1 `[1, 2, ..., num_buckets]`, and returns the bucket number that each row is in.

About the size of the bucket:

- If the row counts can be divided by the specified number of `num_buckets` exactly, all the buckets will be of the same size.
- If the row counts cannot be divided by the specified number of `num_buckets` exactly, there will be buckets of two different sizes. The difference between sizes is 1. The buckets with more rows will be listed ahead of the one with fewer rows.

Syntax:

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

`num_buckets`: Number of the buckets to be created. The value must be a constant positive integer whose maximum is `2^63 - 1`.

Window clause is not allowed in NTILE() function.

NTILE() function returns BIGINT type of data.

Example:

The following example divides all rows in the partition into 2 buckets.

```
select id, x, y,
    ntile(2)
        over (
            partition by x
            order by y
        ) as bucket_id
from t1;
```

Return data:

```
+------+------+------+-----------+
| id   | x    | y    | bucket_id |
+------+------+------+-----------+
|    1 |    1 |   11 |         1 |
|    2 |    1 |   11 |         1 |
|    3 |    1 |   22 |         1 |
|    4 |    1 |   33 |         2 |
|    5 |    1 |   44 |         2 |
|    6 |    1 |   55 |         2 |
|    7 |    2 |   66 |         1 |
|    8 |    2 |   77 |         1 |
|    9 |    2 |   88 |         2 |
|   10 |    3 |   99 |         1 |
+------+------+------+-----------+
```

As the above example shown, when `num_buckets` is `2`:

- Rows of No.1 to No.6 were classified into the first partition; rows of No.1 to No.3 were stored in the first bucket, and rows of No.4 to No.6 were stored in the second one.

- Rows of No.7 to No.9 were classified into the second partition; rows of No.7 and No.8 were stored in the first bucket, and row No.9 was stored in the second one.
- Row No.10 was classified into the third partition and stored in the first bucket.

FIRST_VALUE()

FIRST_VALUE() returns the **first** value of the window range.

Syntax:

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

`IGNORE NULLS` is supported from v2.5.0. It is used to determine whether NULL values of `expr` are eliminated from the calculation. By default, NULL values are included, which means NULL is returned if the first value in the filtered result is NULL. If you specify IGNORE NULLS, the first non-null value in the filtered result is returned. If all the values are NULL, NULL is returned even if you specify IGNORE NULLS.

Example:

We have the following data:

```
 select name, country, greeting
 from mail_merge;
```

```
+---------+---------+--------------+
| name    | country | greeting     |
+---------+---------+--------------+
| Pete    | USA     | Hello        |
| John    | USA     | Hi           |
| Boris   | Germany | Guten tag    |
| Michael | Germany | Guten morgen |
| Bjorn   | Sweden  | Hej          |
| Mats    | Sweden  | Tja          |
+---------+---------+--------------+
```

Use FIRST_VALUE() to return the first greeting value in each grouping, based on the country grouping.

```
select country, name,
    first_value(greeting)
        over (
            partition by country
            order by name, greeting
        ) as greeting
from mail_merge;
```

```
+---------+---------+-----------+
| country | name    | greeting  |
+---------+---------+-----------+
| Germany | Boris   | Guten tag |
| Germany | Michael | Guten tag |
| Sweden  | Bjorn   | Hej       |
| Sweden  | Mats    | Hej       |
| USA     | John    | Hi        |
| USA     | Pete    | Hi        |
+---------+---------+-----------+
```

LAG()

Returns the value of the row that lags the current row by `offset` rows. This function is often used to compare values between rows and filter data.

`LAG()` can be used to query data of the following types:

- Numeric: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL
- String: CHAR, VARCHAR
- Date: DATE, DATETIME
- BITMAP and HLL are supported from StarRocks 2.5.

Syntax:

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

Parameters:

- `expr`: the field you want to compute.

- `offset`: the offset. It must be a **positive integer**. If this parameter is not specified, 1 is the default.
- `default`: the default value returned if no matching row is found. If this parameter is not specified, NULL is the default. `default` supports any expression whose type is compatible with `expr`.

Example:

Calculate the `closing_price` of the previous day. In this example, `default` is set to 0, which means 0 is returned if no matching row is found.

```
select stock_symbol, closing_date, closing_price,
    lag(closing_price, 1, 0)
    over(
    partition by stock_symbol
    order by closing_date
    ) as "yesterday closing"
from stock_ticker
order by closing_date;
```

Output:

```
+--------------+---------------------+---------------+---------------
----+
| stock_symbol | closing_date        | closing_price | yesterday
closing |
+--------------+---------------------+---------------+---------------
----+
| JDR          | 2014-09-13 00:00:00 | 12.86         | 0
|
| JDR          | 2014-09-14 00:00:00 | 12.89         | 12.86
|
| JDR          | 2014-09-15 00:00:00 | 12.94         | 12.89
|
| JDR          | 2014-09-16 00:00:00 | 12.55         | 12.94
|
| JDR          | 2014-09-17 00:00:00 | 14.03         | 12.55
|
| JDR          | 2014-09-18 00:00:00 | 14.75         | 14.03
|
| JDR          | 2014-09-19 00:00:00 | 13.98         | 14.75
|
```

```
+-------------+-------------------+--------------+--------------
----+
```

The first row shows what happens when there is no previous row. The function returns the `default` value 0.

LAST_VALUE()

LAST_VALUE() returns the **last** value of the window range. It is the opposite of FIRST_VALUE().

Syntax:

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

`IGNORE NULLS` is supported from v2.5.0. It is used to determine whether NULL values of `expr` are eliminated from the calculation. By default, NULL values are included, which means NULL is returned if the last value in the filtered result is NULL. If you specify IGNORE NULLS, the last non-null value in the filtered result is returned. If all the values are NULL, NULL is returned even if you specify IGNORE NULLS.

Use the data from the example:

```
select country, name,
    last_value(greeting)
        over (
            partition by country
            order by name, greeting
        ) as greeting
from mail_merge;
```

```
+---------+---------+--------------+
| country | name    | greeting     |
+---------+---------+--------------+
| Germany | Boris   | Guten morgen |
| Germany | Michael | Guten morgen |
| Sweden  | Bjorn   | Tja          |
| Sweden  | Mats    | Tja          |
```

```
| USA      | John     | Hello         |
| USA      | Pete     | Hello         |
+---------+---------+--------------+
```

LEAD()

Returns the value of the row that leads the current row by `offset` rows. This function is often used to compare values between rows and filter data.

Data types that can be queried by `lead()` are the same as those supported by lag().

Syntax

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

Parameters:

- `expr`: the field you want to compute.
- `offset`: the offset. It must be a positive integer. If this parameter is not specified, 1 is the default.
- `default`: the default value returned if no matching row is found. If this parameter is not specified, NULL is the default. `default` supports any expression whose type is compatible with `expr`.

Example:

Calculate the trending of closing prices between two days, that is, whether the price of the next day is higher or lower. `default` is set to 0, which means 0 is returned if no matching row is found.

```
select stock_symbol, closing_date, closing_price,
    case(lead(closing_price, 1, 0)
         over (partition by stock_symbol
         order by closing_date)- closing_price) > 0
        when true then "higher"
        when false then "flat or lower" end
    as "trending"
```

```
from stock_ticker
order by closing_date;
```

Output

```
+--------------+---------------------+---------------+---------------
+
| stock_symbol | closing_date        | closing_price | trending
|
+--------------+---------------------+---------------+---------------
+
| JDR          | 2014-09-13 00:00:00 | 12.86         | higher
|
| JDR          | 2014-09-14 00:00:00 | 12.89         | higher
|
| JDR          | 2014-09-15 00:00:00 | 12.94         | flat or lower
|
| JDR          | 2014-09-16 00:00:00 | 12.55         | higher
|
| JDR          | 2014-09-17 00:00:00 | 14.03         | higher
|
| JDR          | 2014-09-18 00:00:00 | 14.75         | flat or lower
|
| JDR          | 2014-09-19 00:00:00 | 13.98         | flat or lower
|
+--------------+---------------------+---------------+---------------
+
```

MAX()

Returns the maximum value of the specified rows in the current window.

Syntax

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

Example:

Calculate the maximum value of rows from the first row to the row after the current row.

```
select x, property,
    max(x)
        over (
            order by property, x
            rows between unbounded preceding and 1 following
        ) as 'local maximum'
from int_t
where property in ('prime','square');
```

```
+---+----------+---------------+
| x | property | local maximum |
+---+----------+---------------+
| 2 | prime    | 3             |
| 3 | prime    | 5             |
| 5 | prime    | 7             |
| 7 | prime    | 7             |
| 1 | square   | 7             |
| 4 | square   | 9             |
| 9 | square   | 9             |
+---+----------+---------------+
```

From StarRocks 2.4 onwards, you can specify the row range as `rows between n preceding and n following`, which means you can capture `n` rows before the current row and `n` rows after the current row.

Example statement:

```
select x, property,
    max(x)
        over (
            order by property, x
            rows between 3 preceding and 2 following) as 'local
maximum'
from int_t
where property in ('prime','square');
```

MIN()

Returns the minimum value of the specified rows in the current window.

Syntax:

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

Example:

Calculate the minimum value of rows from the first row to the row after the current row.

```
select x, property,
    min(x)
        over (
            order by property, x desc
            rows between unbounded preceding and 1 following
        ) as 'local minimum'
from int_t
where property in ('prime','square');
```

```
+---+----------+---------------+
| x | property | local minimum |
+---+----------+---------------+
| 7 | prime    | 5             |
| 5 | prime    | 3             |
| 3 | prime    | 2             |
| 2 | prime    | 2             |
| 9 | square   | 2             |
| 4 | square   | 1             |
| 1 | square   | 1             |
+---+----------+---------------+
```

From StarRocks 2.4 onwards, you can specify the row range as `rows between n preceding and n following`, which means you can capture `n` rows before the current row and `n` rows after the current row.

Example statement:

```
select x, property,
    min(x)
    over (
          order by property, x desc
          rows between 3 preceding and 2 following) as 'local
```

```
minimum'
from int_t
where property in ('prime','square');
```

RANK()

The RANK() function is used to represent rankings. Unlike DENSE_RANK(), RANK() will **appear as a vacant** number. For example, if two tied 1s appear, the third number of RANK() will be 3 instead of 2.

Syntax:

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

Example:

Ranking according to column x:

```
select x, y, rank() over(partition by x order by y) as `rank`
from int_t;
```

```
+---+---+------+
| x | y | rank |
+---+---+------+
| 1 | 1 | 1    |
| 1 | 2 | 2    |
| 1 | 2 | 2    |
| 2 | 1 | 1    |
| 2 | 2 | 2    |
| 2 | 3 | 3    |
| 3 | 1 | 1    |
| 3 | 1 | 1    |
| 3 | 2 | 3    |
+---+---+------+
```

ROW_NUMBER()

Returns a continuously increasing integer starting from 1 for each row of a Partition. Unlike RANK() and DENSE_RANK(), the value returned by ROW_NUMBER() **does not repeat or**

have gaps and is **continuously incremented**.

Syntax:

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

Example:

```
select x, y, row_number() over(partition by x order by y) as `rank`
from int_t;
```

```
+---+---+------+
| x | y | rank |
+---+---+------+
| 1 | 1 | 1    |
| 1 | 2 | 2    |
| 1 | 2 | 3    |
| 2 | 1 | 1    |
| 2 | 2 | 2    |
| 2 | 3 | 3    |
| 3 | 1 | 1    |
| 3 | 1 | 2    |
| 3 | 2 | 3    |
+---+---+------+
```

QUALIFY()

The QUALIFY clause filters the results of window functions. In a SELECT statement, you can use the QUALIFY clause to apply conditions to a column to filter results. QUALIFY is analogous to the HAVING clause in aggregate functions. This function is supported from v2.5.

QUALIFY simplifies the writing of SELECT statements.

Before QUALIFY is used, a SELECT statement may go like this:

```
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;
```

After QUALIFY is used, the statement is shortened to:

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

QUALIFY supports only the following three window functions: ROW_NUMBER(), RANK(), and DENSE_RANK().

Syntax:

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

Parameters:

`<column_list>`: columns from which you want to obtain data.

`<data_source>`: The data source is generally a table.

`<window_function>`: The `QUALIFY` clause can only be followed by a window function, including ROW_NUMBER(), RANK(), and DENSE_RANK().

Examples:

```
-- Create a table.
CREATE TABLE sales_record (
   city_id INT,
   item STRING,
   sales INT
```

```
) DISTRIBUTED BY HASH(`city_id`) BUCKETS 1;

-- Insert data into the table.
insert into sales_record values
(1,'fruit',95),
(2,'drinks',70),
(3,'fruit',87),
(4,'drinks',98);

-- Query data from the table.
select * from sales_record order by city_id;
+---------+--------+-------+
| city_id | item   | sales |
+---------+--------+-------+
|       1 | fruit  |    95 |
|       2 | drinks |    70 |
|       3 | fruit  |    87 |
|       4 | drinks |    98 |
+---------+--------+-------+
```

Example 1: Obtain records whose row number is greater than 1 from the table.

```
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 |
+---------+--------+-------+
```

Example 2: Obtain records whose row number is 1 from each partition of the table. The table is divided into two partitions by `item` and the first row in each partition is returned.

```
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)
```

Example 3: Obtain records whose sales rank No.1 from each partition of the table. The table is divided into two partitions by `item` and the row with the highest sales in each partition is returned.

```
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 |
+---------+--------+-------+
```

Usage notes:

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()

Syntax:

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

Example:

Group by property and calculate the sum of the **current, preceding, and following rows** within the group.

```
select x, property,
    sum(x)
        over (
            partition by property
            order by x
            rows between 1 preceding and 1 following
        ) as 'moving total'
from int_t where property in ('odd','even');
```

```
+----+----------+--------------+
| x  | property | moving total |
+----+----------+--------------+
| 2  | even     | 6            |
| 4  | even     | 12           |
| 6  | even     | 18           |
| 8  | even     | 24           |
| 10 | even     | 18           |
| 1  | odd      | 4            |
| 3  | odd      | 9            |
| 5  | odd      | 15           |
| 7  | odd      | 21           |
+----+----------+--------------+
```

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 v2.5.10.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the VARIANCE() function.

```
mysql> select variance(k) over (partition by no) FROM agg;
+-----------------------------------+
| variance(k) OVER (PARTITION BY no ) |
+-----------------------------------+
|                                 0 |
|                           54.6875 |
|                           54.6875 |
|                           54.6875 |
|                           54.6875 |
+-----------------------------------+

mysql> select variance(k) over(
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
```

```
+------------------+
| window_test      |
+------------------+
|                0 |
|               25 |
| 38.88888888888889 |
|          54.6875 |
|          54.6875 |
+------------------+
```

VAR_SAMP, VARIANCE_SAMP

Returns the sample variance of an expression. These functions can be used as window functions since v2.5.10.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the VAR_SAMP() window function.

```
mysql> select VAR_SAMP(k) over (partition by no) FROM agg;
+-------------------------------------+
| var_samp(k) OVER (PARTITION BY no ) |
+-------------------------------------+
|                                   0 |
|                   72.91666666666667 |
|                   72.91666666666667 |
|                   72.91666666666667 |
|                   72.91666666666667 |
+-------------------------------------+

mysql> select VAR_SAMP(k) over(
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
+--------------------+
| window_test        |
+--------------------+
|                  0 |
|                 50 |
| 58.333333333333336 |
|  72.91666666666667 |
|  72.91666666666667 |
+--------------------+
```

STD, STDDEV, STDDEV_POP

Returns the standard deviation of an expression. These functions can be used as window functions since v2.5.10.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the STD() window function.

```
mysql> select STD(k) over (partition by no) FROM agg;
+-------------------------------+
| std(k) OVER (PARTITION BY no ) |
+-------------------------------+
|                              0 |
|               7.39509972887452 |
|               7.39509972887452 |
|               7.39509972887452 |
|               7.39509972887452 |
+-------------------------------+

mysql> select std(k) over (
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
```

```
+------------------+
| window_test      |
+------------------+
|                0 |
|                5 |
| 6.236095644623236 |
|  7.39509972887452 |
|  7.39509972887452 |
+------------------+
```

STDDEV_SAMP

Returns the sample standard deviation of an expression. This function can be used as a window function since v2.5.10.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the STDDEV_SAMP() window function.

```
mysql> select STDDEV_SAMP(k) over (partition by no) FROM agg;
+---------------------------------------+
| stddev_samp(k) OVER (PARTITION BY no ) |
+---------------------------------------+
|                                     0 |
|                      8.539125638299666 |
|                      8.539125638299666 |
|                      8.539125638299666 |
|                      8.539125638299666 |
+---------------------------------------+

mysql> select STDDEV_SAMP(k) over (
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
+--------------------+
| window_test        |
+--------------------+
|                  0 |
| 7.0710678118654755 |
|  7.637626158259733 |
|  8.539125638299666 |
|  8.539125638299666 |
+--------------------+
```

COVAR_SAMP

Returns the sample covariance of two expressions. This function is supported from v2.5.10. It is also an aggregate function.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the COVAR_SAMP() window function.

```
mysql> select COVAR_SAMP(k, v) over (partition by no) FROM agg;
+-----------------------------------------+
| covar_samp(k, v) OVER (PARTITION BY no ) |
+-----------------------------------------+
|                                    NULL |
|                     119.99999999999999 |
|                     119.99999999999999 |
|                     119.99999999999999 |
|                     119.99999999999999 |
+-----------------------------------------+

mysql> select COVAR_SAMP(k,v) over (
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
```

```
+--------------------+
| window_test        |
+--------------------+
|               NULL |
|                 55 |
|                 55 |
| 119.99999999999999 |
| 119.99999999999999 |
+--------------------+
```

COVAR_POP

Returns the population covariance of two expressions. This function is supported from v2.5.10. It is also an aggregate function.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the COVAR_POP() window function.

```
mysql> select COVAR_POP(k, v) over (partition by no) FROM agg;
+----------------------------------------+
| covar_pop(k, v) OVER (PARTITION BY no ) |
+----------------------------------------+
|                                   NULL |
|                       79.99999999999999 |
|                       79.99999999999999 |
|                       79.99999999999999 |
|                       79.99999999999999 |
+----------------------------------------+

mysql> select COVAR_POP(k,v) over (
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
+-------------------+
| window_test       |
+-------------------+
|              NULL |
|              27.5 |
|              27.5 |
| 79.99999999999999 |
| 79.99999999999999 |
+-------------------+
```

CORR

Returns the Pearson correlation coefficient between two expressions. This function is supported from v2.5.10. It is also an aggregate function.

Syntax:

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

> **NOTE**
>
> 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:

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Use the CORR() window function.

```
mysql> select CORR(k, v) over (partition by no) FROM agg;
+-----------------------------------+
| corr(k, v) OVER (PARTITION BY no ) |
+-----------------------------------+
|                              NULL |
|                 0.9988445981121532 |
|                 0.9988445981121532 |
|                 0.9988445981121532 |
|                 0.9988445981121532 |
+-----------------------------------+

mysql> select CORR(k,v) over (
    partition by no
    order by k
    rows between unbounded preceding and 1 following) AS window_test
FROM agg order by no,k;
```

```
+--------------------+
| window_test        |
+--------------------+
|               NULL |
|                  1 |
|                  1 |
| 0.9988445981121532 |
| 0.9988445981121532 |
+--------------------+
```

Lambda expression

Lambda expressions are anonymous functions that can be passed as parameters into higher-order SQL functions. Lambda expressions allow you to develop code that is more concise, elegant, and extensible.

Lambda expressions are written with the `->` operator, which reads "goes to". The left side of `->` is the input parameters (if any) and the right side is an expression.

From v2.5 onwards, StarRocks supports using lambda expressions in the following higher-order SQL functions: array_map(), array_filter(), array_sum(), and array_sortby().

Syntax

```
parameter -> expression
```

Parameters

- `parameter`: the input parameters for the lambda expression, which can accept zero, one, or more parameters. Two or more input parameters are enclosed in parentheses.
- `expression`: a simple expression that references `parameter`. The expression must be valid for the input parameters.

Return value

The type of the return value is determined by the result type of `expression`.

Usage notes

Almost all scalar functions can be used in a lambda body. But there are a few exceptions:

- Subqueries are not supported, for example, `x -> 5 + (SELECT 3)`.
- Aggregate functions are not supported, for example, `x -> min(y)`.
- Window functions are not supported.
- Table functions are not supported.
- Correlated columns cannot occur in lambda functions.

Examples

Simple examples of lambda expressions:

```
-- Accepts no parameters and returns 5.
() -> 3
-- Takes x and returns the value of (x + 2).
x -> x + 2
-- Takes x and y, and returns their sum.
(x, y) -> x + y
-- Takes x and applies a function to x.
x -> COALESCE(x, 0)
x -> day(x)
x -> split(x,",")
x -> if(x>0,"positive","negative")
```

An example of using lambda expressions in higher-order functions:

```
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)
```

any_value

Description

Obtains an arbitrary row from each aggregated group. You can use this function to optimize a query that has a `GROUP BY` clause.

Syntax

```
ANY_VALUE(expr)
```

Parameters

`expr`: the expression that gets aggregated.

Return value

Returns an arbitrary row from each aggregated group. The return value is non-deterministic.

Examples

```
// Original data
mysql> select * from any_value_test;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 |    1 |    1 |
|    1 |    2 |    1 |
|    2 |    1 |    1 |
|    2 |    2 |    2 |
|    3 |    1 |    1 |
+------+------+------+
```

```
5 rows in set (0.01 sec)

// After ANY_VALUE is used
mysql> select a,any_value(b),sum(c) from any_value_test 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 c,any_value(a),sum(b) from any_value_test group by c;
+------+---------------+----------+
| c    | any_value(`a`) | sum(`b`) |
+------+---------------+----------+
|    1 |             1 |        5 |
|    2 |             2 |        2 |
+------+---------------+----------+
2 rows in set (0.01 sec)
```

Keywords

ANY_VALUE

approx_count_distinct

Description

Returns the approximate value of aggregate function similar to the result of COUNT(DISTINCT col).

It is faster than the COUNT and DISTINCT combination and uses a fixed-size memory, so less memory is used for columns of high cardinality.

Syntax

```
APPROX_COUNT_DISTINCT(expr)
```

Examples

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

keyword

APPROX_COUNT_DISTINCT

avg

Description

Returns the average value of selected fields.

Optional field DISTINCT parameters can be used to return the weighted average.

Syntax

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

Examples

```
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 |
+---------------------+---------------------------+
```

keyword

AVG

bitmap

Here is a simple example to illustrate the usage of several aggregate functions in Bitmap. For detailed function definitions or more Bitmap functions, see bitmap-functions.

Create table

The aggregation model is needed when creating table. The data type is bitmap and the aggregation function is 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`) BUCKETS 2;
```

Note: With an amount of data, you'd better create a rollup table corresponding to high-frequent bitmap_union.

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

Data Load

`TO_BITMAP (expr)`: Convert 0 ~ 18446744073709551615 unsigned bigint to bitmap

`BITMAP_EMPTY ()`: Generate empty bitmap columns, used for the default value to be filled in when inserting or inputting

`BITMAP_HASH (expr)`: Convert columns of any type to a bitmap by hashing

Stream Load

When inputting data using Stream Load, you can convert the data to a BItmap field as follows:

```
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_empty()" \
    http://host:8410/api/test/testDb/_stream_load
```

Insert Into

When inputting data using Insert Into, you need to select the corresponding mode based on the type of columns in the source table.

- id2's column type in source table is bitmap

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

- id2's column type in target table is bitmap

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

- id2's column type in source table is bitmap, and is the result of aggregation using bit_map_union().

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

- id2's column type in source table is INT, and the bitmap type is generated by to_bitmap().

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

- id2's column type in source table is STRING, and the bitmap type is generated by bitmap_hash().

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

Data Query

Syntax

``BITMAP_UNION (expr)`: Calculate the union of the input Bitmaps, and returns the new Bitmap.

`BITMAP_UNION_COUNT (expr)`: Calculate the union of the input Bitmaps, and returns its cardinality, equivalent to BITMAP_COUNT (BITMAP_UNION (expr)). It is recommended to use the BITMAP_UNION_COUNT function first, for its performance is better than BITMAP_COUNT (BITMAP_UNION (expr)).

`BITMAP_UNION_INT (expr)`: Calculate the number of different values in columns of type TINYINT, SMALLINT and INT, return the value same as COUNT (DISTINCT expr).

`INTERSECT_COUNT (bitmap_column_to_count, filter_column, filter_values ...)`: Calculate the cardinality of the intersection of multiple bitmaps that satisfy filter_column condition. bitmap_column_to_count is a column of type bitmap, filter_column is a column of varying dimensions, and filter_values is a list of dimension values.

`BITMAP_INTERSECT(expr)`: Calculate the intersection of this group of bitmap values and returns a new bitmap.

Example

The following SQL uses the `pv_bitmap` table above as an example:

Calculate the deduplicated value for `user_id`:

```
select bitmap_union_count(user_id)
from pv_bitmap;

select bitmap_count(bitmap_union(user_id))
from pv_bitmap;
```

Calculate the deduplicated value of `id`:

```
select bitmap_union_int(id)
from pv_bitmap;
```

Calculate the retention of `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 -
- Number of users that access both the 'game' and 'shopping' pages
from pv_bitmap
where page in ('game', 'shopping');
```

keyword

BITMAP,BITMAP_COUNT,BITMAP_EMPTY,BITMAP_UNION,BITMAP_UNION_INT,TO_BITMAP
,BITMAP_UNION_COUNT,INTERSECT_COUNT,BITMAP_INTERSECT

corr

Description

Returns the Pearson correlation coefficient between two expressions. This function is supported from v2.5.10. It can also be used as a window function.

Syntax

```
CORR(expr1, expr2)
```

Parameters

`expr1` and `expr2` must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

If `expr1` and `expr2` are table columns, this function calculates the correlation coefficient of these two columns.

Return value

Returns a DOUBLE value. The formula is as follows, where `n` represents the row count of the table:

$$\frac{\sum_{i=1}^{n}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{n}((x_i - \bar{x})^2) \cdot \sum_{i=1}^{n}((y_i - \bar{y})^2)}}$$

Usage notes

- A data row is counted only when the two columns in this row are non-null values. Otherwise, this data row is eliminated from the result.
- 0 is returned if `n` is 1.
- NULL is returned in any input is NULL.

Examples

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Calculate the correlation coefficient of the `k` and `v` columns:

```
mysql> select no,CORR(k,v) from agg group by no;
+------+--------------------+
| no   | corr(k, v)         |
+------+--------------------+
|    1 |               NULL |
|    2 | 0.9988445981121532 |
+------+--------------------+
```

count

Description

Returns the total number of rows specified by an expression.

This function has three variations:

- `COUNT(*)` counts all rows in a table, no matter whether they contain NULL values.
- `COUNT(expr)` counts the number of rows with non-NULL values in a specific column.
- `COUNT(DISTINCT expr)` counts the number of distinct non-NULL values in a column.

`COUNT(DISTINCT expr)` is used for exact count distinct. If you require higher count distinct performance, see Use bitmap for exact count discount.

From StarRocks 2.4 onwards, you can use multiple COUNT(DISTINCT) in one statement.

Syntax

```
COUNT(expr)
COUNT(DISTINCT expr [,expr,...])`
```

Parameters

`expr`: the column or expression based on which `count()` is performed. If `expr` is a column name, the column can be of any data type.

Return value

Returns a numeric value. If no rows can be found, 0 is returned. This function ignores NULL values.

Examples

Suppose there is a table named `test`. Query the country, category, and supplier of each order by `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 |
+------+----------+----------+------------+
```

Example 1: Count the number of rows in table `test`.

```
    select count(*) from test;
    +----------+
    | count(*) |
    +----------+
    |        7 |
    +----------+
```

Example 2: Count the number of values in the `id` column.

```
    select count(id) from test;
    +-----------+
    | count(id) |
    +-----------+
    |         7 |
    +-----------+
```

Example 3: Count the number of values in the `category` column while ignoring NULL values.

```
select count(category) from test;
  +-----------------+
  | count(category) |
  +-----------------+
  |         6       |
  +-----------------+
```

Example 4: Count the number of distinct values in the `category` column.

```
select count(distinct category) from test;
+------------------------+
| count(DISTINCT category) |
+------------------------+
|                      4 |
+------------------------+
```

Example 5: Count the number of combinations that can be formed by `category` and `supplier`.

```
select count(distinct category, supplier) from test;
+-----------------------------------+
| count(DISTINCT category, supplier) |
+-----------------------------------+
|                                 5 |
+-----------------------------------+
```

In the output, the combination with `id` 1004 duplicates with the combination with `id` 1002. They are counted only once. The combination with `id` 1007 has a NULL value and is not counted.

Example 6: Use multiple COUNT(DISTINCT) in one statement.

```
select count(distinct country, category), count(distinct
country,supplier) from test;
+---------------------------------+------------------------------
---+
| count(DISTINCT country, category) | count(DISTINCT country,
supplier) |
+---------------------------------+------------------------------
---+
```

```
|                                    6 |
7 |
+-----------------------------------+-------------------------------
---+
```

covar_pop

Description

Returns the population covariance of two expressions. This function is supported from v2.5.10. It can also be used as a window function.

Syntax

```
COVAR_POP(expr1, expr2)
```

Parameters

`expr1` and `expr2` must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

If `expr1` and `expr2` are table columns, this function calculates the population covariance of these two columns.

Return value

Returns a DOUBLE value. The formula is as follows, where `n` represents the row count of the table:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n}$$

Usage notes

- A data row is counted only when the two columns in this row are non-null values. Otherwise, this data row is eliminated from the result.
- NULL is returned in any input is NULL.

Examples

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Calculate the population covariance of the `k` and `v` columns:

```
mysql> select no,COVAR_POP(k,v) from agg group by no;
+------+------------------+
| no   | covar_pop(k, v)   |
```

```
+------+-------------------+
|    1 |              NULL |
|    2 | 79.99999999999999 |
+------+-------------------+
```

covar_samp

Description

Returns the sample covariance of two expressions. This function is supported from v2.5.10. It can also be used as a window function.

Syntax

```
COVAR_SAMP(expr1, expr2)
```

Parameters

`expr1` and `expr2` must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

If `expr1` and `expr2` are table columns, this function calculates the sample covariance of these two columns.

Return value

Returns a DOUBLE value. The formula is as follows, where `n` represents the row count of the table:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Usage notes

- A data row is counted only when the two columns in this row are non-null values. Otherwise, this data row is eliminated from the result.
- 0 is returned if `n` is 1.
- NULL is returned in any input is NULL.

Examples

Suppose table `agg` has the following data:

```
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 |
+------+-------+-------+
```

Calculate the sample covariance of the `k` and `v`` columns:

```
mysql> select no,COVAR_SAMP(k,v) from agg group by no;
+------+--------------------+
| no   | covar_samp(k, v)   |
+------+--------------------+
|    1 |               NULL |
|    2 | 119.99999999999999 |
+------+--------------------+
```

group_concat

Description

This is an aggregate function similar to sum(). group_concat concatenates non-null values into one string, with the second argument `sep` being the separator. The second argument can also be omitted. This function usually needs to be used along with `group by`.

> Please note that strings may not be concatenated in sequence because it uses distributed computing.

Syntax

```
VARCHAR group_concat(VARCHAR str[, VARCHAR sep])
```

Parameters

- `str`: the values to concatenate. It must evaluate to VARCHAR.
- `sep`: the separator, optional. If it is not specified, a comma and a space (`, `) is used by default.

Return value

Returns a VARCHAR value.

Examples

```
CREATE TABLE IF NOT EXISTS group_concat (
    id        tinyint(4)      NULL,
    value   varchar(65533)  NULL
```

```
) ENGINE=OLAP
DISTRIBUTED BY HASH(id);

INSERT INTO group_concat VALUES
(1,'fruit'),
(2,'drinks'),
(3,null),
(4,'fruit'),
(5,'meat'),
(6,'seafood');

select * from group_concat order by id;
+------+---------+
| id   | value   |
+------+---------+
|    1 | fruit   |
|    2 | drinks  |
|    3 | NULL    |
|    4 | fruit   |
|    5 | meat    |
|    6 | seafood |
```

```
select group_concat(value) from group_concat;
+-----------------------------------+
| group_concat(value)               |
+-----------------------------------+
| meat, fruit, seafood, fruit, drinks |
+-----------------------------------+

MySQL > select group_concat(value, " ") from group_concat;
+---------------------------------+
| group_concat(value, ' ')        |
+---------------------------------+
| fruit meat fruit drinks seafood |
+---------------------------------+
```

keyword

GROUP_CONCAT,GROUP,CONCAT

grouping

Description

Indicates whether a column is an aggregate column. If it is an aggregate column, 0 is returned. Otherwise, 1 is returned.

Syntax

```
GROUPING(col_expr)
```

Parameters

`col_expr`: An expression for a dimension column specified in the expression-list of the ROLLUP, CUBE, or GROUPING SETS expansion of the GROUP BY clause.

Examples

```
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

Description

grouping_id is used to distinguish the grouping statistics results of the same grouping standard.

Syntax

```
GROUPING_ID(expr)
```

Examples

```
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_empty

Description

Generates an empty HLL column to supplement the default values when inserting or loading data.

Syntax

```
HLL_EMPTY()
```

Return value

Returns an empty HLL.

Examples

Supplement the default values when inserting data.

```
insert into hllDemo(k1,v1) values(10,hll_empty());
```

Supplement the default values when loading data.

```
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

Description

Converts a value to an hll type. Typically used in imports to map a value in the source data to an HLL column type in the StarRocks table.

Syntax

```
HLL_HASH(column_name)
```

Parameters

`column_name`: The name of the generated HLL column.

Return value

Returns a value of the HLL type.

Example

```
mysql> select hll_cardinality(hll_hash("a"));
+-------------------------------+
| hll_cardinality(hll_hash('a')) |
+-------------------------------+
|                             1 |
+-------------------------------+
```

hll_raw_agg

Description

This function is an aggregate function that is used to aggregate HLL fields. It returns an HLL value.

Syntax

```
hll_raw_agg(hll)
```

Parameters

`hll`: the HLL column that is generated by other columns or based on the loaded data.

Return value

Returns a value of the HLL type.

Examples

```
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

Description

Returns the concatenation of a set of HLL values.

Syntax

```
hll_union(hll)
```

Examples

```
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

Description

HLL is an engineering implementation based on the HyperLogLog algorithm, which is used to save the intermediate results of HyperLogGog calculation process.

It can only be used as the value column of a table and reduce the amount of data through aggregation to achieve the purpose of speeding up the query.

An estimated result with an error of about 1% based on HLL. The HLL column is generated by other columns or based on data loaded into the table.

During loading, the hll_hash function is used to specify which column is used to generate the HLL column. It is often used to replace Count Distinct, and to calculate UVs quickly in business by combining rollup.

Syntax

```
HLL_UNION_AGG(hll)
```

Examples

```
MySQL > select HLL_UNION_AGG(uv_set) from test_uv;
+-------------------------+
| HLL_UNION_AGG(`uv_set`) |
+-------------------------+
| 17721                   |
+-------------------------+
```

keyword

HLL_UNION_AGG,HLL,UNION,AGG

max

Description

Returns the maximum value of the expr expression.

Syntax

```
MAX(expr)
```

Examples

```
MySQL > select max(scan_rows)
from log_statis
group by datetime;
+------------------+
| max(`scan_rows`) |
+------------------+
|          4671587 |
+------------------+
```

keyword

MAX

max_by

Description

Returns the value of `x` associated with the maximum value of `y`.

For example, `SELECT max_by(subject, exam_result) FROM exam;` is to return the subject that has the highest exam score.

This function is supported from v2.5.

Syntax

```
max_by(x,y)
```

Parameters

- `x`: an expression of any type.
- `y`: an expression of a type that can be ordered.

Return value

Returns a value that has the same type as `x`.

Usage notes

- `y` must be a sortable type. If you use an unsortable type of `y`, such as `bitmap` or `hll`, an error is returned.
- If `y` contains a null value, the row that corresponds to the null value is ignored.

- If more than one value of `x` has the same maximum value of `y`, this function returns the first value of `x` encountered.

Examples

1. Create a table `exam`.

```
CREATE TABLE exam (
    subject_id INT,
    subject STRING,
    exam_result INT
) DISTRIBUTED BY HASH(`subject_id`) BUCKETS 1;
```

2. Insert values into this table and query data from this table.

```
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. Obtain the subject that has the highest score. Two subjects `physics` and `music` have the same highest score `95` and the first subject encountered (`physics`) is returned.

```
SELECT max_by(subject, exam_result) FROM exam;
+-----------------------------+
| max_by(subject, exam_result) |
+-----------------------------+
| physics                     |
+-----------------------------+
1 row in set (0.01 sec)
```

min

Description

Returns the minimum value of the expr expression.

Syntax

```
MIN(expr)
```

Examples

```
MySQL > select min(scan_rows)
from log_statis
group by datetime;
+------------------+
| min(`scan_rows`) |
+------------------+
|                0 |
+------------------+
```

keyword

MIN

multi_distinct_count

Description

Returns the total number of rows of the `expr`, equivalent to count(distinct expr).

Syntax

```
multi_distinct_count(expr)
```

Parameters

`expr`: the column or expression based on which `multi_distinct_count()` is performed. If `expr` is a column name, the column can be of any data type.

Return value

Returns a numeric value. If no rows can be found, 0 is returned. This function ignores NULL values.

Examples

Suppose there is a table named `test`. Query the category and supplier of each order by `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 |
+------+----------+----------+------------+
```

Example 1: Count the number of distinct values in the `category` column.

```
select multi_distinct_count(category) from test;
+-------------------------------+
| multi_distinct_count(category) |
+-------------------------------+
|                             4 |
+-------------------------------+
```

Example 2: Count the number of distinct values in the `supplier` column.

```
select multi_distinct_count(supplier) from test;
+-------------------------------+
| multi_distinct_count(supplier) |
+-------------------------------+
|                             5 |
+-------------------------------+
```

multi_distinct_sum

Description

Returns the sum of distinct values in `expr`, equivalent to sum(distinct expr).

Syntax

```
multi_distinct_sum(expr)
```

Parameters

- `expr`: the columns involved in the calculation. Column values can be of the following types: TINYINT, SMALLINT, INT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Return value

The mapping between column values and return value types is as follows:

- BOOLEAN -> BIGINT
- TINYINT -> BIGINT
- SMALLINT -> BIGINT
- INT -> BIGINT
- BIGINT -> BIGINT
- FLOAT -> DOUBLE
- DOUBLE -> DOUBLE
- LARGEINT -> LARGEINT
- DECIMALV2 -> DECIMALV2
- DECIMAL32 -> DECIMAL128
- DECIMAL64 -> DECIMAL128

- DECIMAL128 -> DECIMAL128

Examples

1. Create a table with `k0` as the INT field.

```
CREATE TABLE tabl
(k0 BIGINT NOT NULL) ENGINE=OLAP
DUPLICATE KEY(`k0`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`k0`) BUCKETS 1
PROPERTIES(
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);
```

2. Insert values into the table.

```
INSERT INTO tabl VALUES ('0'), ('1'), ('1'), ('1'), ('2'), ('2');
```

3. Use multi_distinct_sum() to calculate the sum of distinct values in the `k0` column.

```
MySQL > select multi_distinct_sum(k0) from tabl;
+-----------------------+
| multi_distinct_sum(k0) |
+-----------------------+
|                     3 |
+-----------------------+
1 row in set (0.03 sec)
```

   The distinct values of `k0` is 0, 1, 2 and we can get 3 after adding them together.

percentile_approx

Description

Returns the approximation of the pth percentile, where the value of p is between 0 and 1.

Compression parameter is optional and has a setting range of [2048, 10000]. The larger the value, the higher the accuracy, the larger the memory consumption, and the longer the calculation time. If it is not specified or not beyond the range of [2048, 10000], the function runs with a default compression parameter of 10000.

This function uses fixed size memory, so less memory can be used for columns with high cardinality, and can be used to calculate statistics such as tp99.

Syntax

```
PERCENTILE_APPROX(expr, DOUBLE p[, DOUBLE compression])
```

Examples

```
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 |
+----------+-----------------------------------------------+
```

keyword

PERCENTILE_APPROX,PERCENTILE,APPROX

percentile_cont

Description

Computes the percentile value of `expr` with linear interpolation.

Syntax

```
PERCENTILE_CONT (expr, percentile)
```

Parameters

- `expr`: the expression by which to order the values. It must be of numeric data types, DATE, or DATETIME. For example, if you want to find the median score of physics, specify the column that contains the physics scores.
- `percentile`: the percentile of the value you want to find. It is a constant floating-point number from 0 to 1. For example, if you want to find the median value, set this parameter to `0.5`.

Return value

Returns a value that is at the specified percentile. If no input value lies exactly at the desired percentile, the result is calculated using linear interpolation of the two nearest input values.

The data type is the same as `expr`.

Usage notes

This function ignores NULLs.

Examples

Suppose there is a table named `exam` with the following data.

```
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 |
+-----------+-------+
```

Calculate the median score of each subject while ignoring NULLs.

Query:

```
SELECT Subject, PERCENTILE_CONT (Score, 0.5)  FROM exam group by
Subject;
```

Result:

```
+-----------+----------------------------+
| Subject   | percentile_cont(Score, 0.5) |
+-----------+----------------------------+
| chemistry |                         90 |
| math      |                         70 |
| physics   |                       82.5 |
+-----------+----------------------------+
```

percentile_disc

Description

Returns a percentile value based on a discrete distribution of the input column `expr`. If the exact percentile value cannot be found, this function returns the larger value between the two closest values.

This function is supported from v2.5 onwards.

Syntax

```
PERCENTILE_DISC (expr, percentile)
```

Parameters

- `expr`: the column for which you want to calculate the percentile value. The column can be of any data type that is sortable.
- `percentile`: the percentile of the value you want to find. It must be a constant floating-point number between 0 and 1. For example, if you want to find the median value, set this parameter to `0.5`. If you want to find the value at the 70th percentile, specify 0.7.

Return value

The data type of the return value is the same as `expr`.

Usage notes

NULL values are ignored in the calculation.

Examples

Create table `exam` and insert data into this table.

```
CREATE TABLE exam (
    subject STRING,
    score INT
)
DISTRIBUTED BY HASH(`subject`) BUCKETS 1;

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 |
+-----------+-------+
```

Calculate the median of each subject.

```
select subject, percentile_disc(score, 0.5)
```

```
from exam group by subject;
```

Output

```
+-----------+-----------------------------+
| subject   | percentile_disc(score, 0.5) |
+-----------+-----------------------------+
| chemistry |                         100 |
| math      |                          70 |
| physics   |                          85 |
+-----------+-----------------------------+
```

retention

Description

Calculates the user retention rate within a specified period of time. This function accepts 1 to 31 conditions and evaluates whether each condition is true. If the condition evaluates to true, 1 is returned. Otherwise, 0 is returned. It eventually returns an array of 0 and 1. You can calculate the user retention rate based on this data.

Syntax

```
ARRAY retention(ARRAY input)
```

Parameters

`input`: an array of conditions. A maximum of 31 conditions can be passed in. Separate multiple conditions with commas.

Return value

Returns an array of 0 and 1. The number of 0 and 1 is the same as the number of input conditions.

The evaluation starts from the first condition.

- If the condition evaluates to true, 1 is returned. Otherwise, 0 is returned.
- If the first condition is not true, the current position and its following positions are all set to 0.

Examples

1. Create a table named `test` and insert data.

```
CREATE TABLE test(
    id TINYINT,
    action STRING,
    time DATETIME
)
ENGINE=olap
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 8;

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. Query data from `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. Use `retention` to calculate user retention rate.

Example 1: Evaluate user behavior against the following conditions: view commodity page on 2022-01-01 (action='pv') and place an order on 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)
```

In the result:

- Users 1 and 5 meets two conditions and [1,1] is returned.
- User 2 does not meet the second condition and [1,0] is returned.
- User 3 meets the second condition but does not meet the first condition. [0,0] is returned.
- User 4 meets no condition and [0,0] is returned.

Example 2: Calculate the percentage of users who have viewed commodity page on 2022-01-01 (action='pv') and placed an order on 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)
```

The return value is the user retention rate on 2022-01-02.

keyword

retention, retention rate, RETENTION

std

Description

Returns the standard deviation of an expression. Since v2.5.10, this function can also be used as a window function.

Syntax

```
STD(expr)
```

Parameters

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

Return value

Returns a DOUBLE value.

Examples

Sample dataset

```
=======
```plaintext
>>>>>>> 41ea837d3 ([Doc] add corr related functions and update window
functions (#28776))
MySQL > select * from std_test;
+------+------+
| col0 | col1 |
```
```

```
+------+------+
|    0 |    0 |
|    1 |    2 |
|    2 |    4 |
|    3 |    6 |
|    4 |    8 |
+------+------+
```

Calculate the standard deviation of `col0` and `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 |
+-------------------+-------------------+
```

keyword

STD

stddev,stddev_pop,std

Description

Returns the population standard deviation of the expr expression. Since v2.5.10, this function can also be used as a window function.

Syntax

```
STDDEV(expr)
```

Parameters

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

Return value

Returns a DOUBLE value. The formula is as follows, where `n` represents the row count of the table:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Examples

```
mysql> SELECT stddev(lo_quantity), stddev_pop(lo_quantity) from
lineorder;
+--------------------+------------------------+
| stddev(lo_quantity) | stddev_pop(lo_quantity) |
+--------------------+------------------------+
|   14.43100708360797 |       14.43100708360797 |
+--------------------+------------------------+
```

See also

stddev_samp

keyword

STDDEV,STDDEV_POP,POP

stddev_samp

Description

Returns the sample standard deviation of an expression. Since v2.5.10, this function can also be used as a window function.

Syntax

```
STDDEV_SAMP(expr)
```

Parameters

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

Return value

Returns a DOUBLE value. The formula is as follows, where `n` represents the row count of the table:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Examples

```
MySQL > select stddev_samp(scan_rows)
from log_statis
group by datetime;
+-------------------------+
| stddev_samp(`scan_rows`) |
+-------------------------+
|        2.372044195280762 |
+-------------------------+
```

See also

stddev

keyword

STDDEV_SAMP,STDDEV,SAMP

sum

Description

Returns the sum of non-null values for `expr`. You can use the DISTINCT keyword to compute the sum of distinct non-null values.

Syntax

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

Parameters

`expr`: the expression that evaluates to a numeric value. Supported data types are TINYINT, SMALLINT, INT, FLOAT, DOUBLE, or DECIMAL.

Return value

Data type mapping between input value and return value:

- TINYINT -> BIGINT
- SMALLINT -> BIGINT
- INT -> BIGINT
- FLOAT -> DOUBLE
- DOUBLE -> DOUBLE
- DECIMAL -> DECIMAL

Usage notes

- This function ignores nulls.

- An error is returned if `expr` does not exist.
- If a VARCHAR expression is passed, this function implicitly casts the input into DOUBLE values. If the cast fails, an error is returned.

Examples

1. Create a table named `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) BUCKETS 8;
```

2. Insert data into `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. Query data from `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. Use this function to compute sum.

   Example 1: Calculate the total sales of each region.

```
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)
```

   Example 2: Calculate the total employee income of each region. This function ignores nulls and the income of employee id `777326` is not counted.

```
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)
```

   Example 3: Calculate the total number of hobbies. The `hobby` column is of the STRING type and will be implicitly converted to DOUBLE during computation.

```
MySQL > select sum(DISTINCT hobby) from employees;
```

```
+---------------------+
| sum(DISTINCT hobby) |
+---------------------+
|                  15 |
+---------------------+
1 row in set (0.01 sec)
```

Example 4: Use `sum` with the WHERE clause to calculate the total income of employees whose monthly income is higher than 30000.

```
MySQL > select sum(income) from employees
WHERE income > 30000;

+-------------+
| sum(income) |
+-------------+
|      118448 |
+-------------+
1 row in set (0.00 sec)
```

keyword

SUM, sum

var_samp,variance_samp

Description

Returns the sample variance of an expression. Since v2.5.10, this function can also be used as a window function.

Syntax

```
VAR_SAMP(expr)
```

Parameters

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

Return value

Returns a DOUBLE value.

Examples

```
MySQL > select var_samp(scan_rows)
from log_statis
group by datetime;
+----------------------+
| var_samp(`scan_rows`) |
+----------------------+
|    5.6227132145741789 |
+----------------------+
```

keyword

VAR_SAMP,VARIANCE_SAMP,VAR,SAMP,VARIANCE

variance,var_pop,variance_pop

Description

Returns the population variance of an expression. Since v2.5.10, this function can also be used as a window function.

Syntax

```
VARIANCE(expr)
```

Parameters

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

Return value

Returns a DOUBLE value.

Examples

```
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       |
+--------------------------+------------------+
```

keyword

VARIANCE,VAR_POP,VARIANCE_POP,VAR,POP

window_funnel

Description

Searches for an event chain in a sliding window and calculates the maximum number of consecutive events in the event chain. This function is commonly used for analyzing conversion rate. It is supported from v2.3.

This function works according to the following rules:

- It starts the count from the first event in the event chain. If the first event is found, the event counter is set to 1 and the sliding window starts. If the first event is not found, 0 is returned.
- In the sliding window, the counter is incremented if the events in the event chain occur in sequence. If the sliding window is exceeded, the event counter is no longer incremented.
- If multiple event chains match the specified condition, the longest event chain is returned.

Syntax

```
BIGINT window_funnel(BIGINT window, DATE|DATETIME time, INT mode,
array[cond1, cond2, ..., condN])
```

Parameters

- `window`: The length of the sliding window. The supported data type is BIGINT. The unit depends on the `time` parameter. If the data type of `time` is DATE, the unit is days. If the data type of `time` is DATETIME, the unit is seconds.
- `time`: The column containing timestamps. DATE and DATETIME types are supported.

- `mode`: The mode in which the event chain is filtered. The supported data type is INT. Value range: 0, 1, 2.
  - `0` is the default value, which indicates common funnel calculation.
  - `1` indicates the `DEDUPLICATION` mode, that is, the filtered event chain cannot have repeated events. Suppose the `array` parameter is `[event_type = 'A', event_type = 'B', event_type = 'C', event_type = 'D']` and the original event chain is "A-B-C-B-D". Event B is repeated and the filtered event chain is "A-B-C".
  - `2` indicates the `FIXED` mode, that is, the filtered event chain cannot have events that disrupt the specified sequence. Suppose the previous `array` parameter is used and the original event chain is "A-B-D-C". Event D interrupts the sequence and the filtered event chain is "A-B".
  - `4` indicates the `INCREASE` mode, which means the filtered events must have strictly increasing timestamps. Duplicate timestamp disrupts the event chain. This mode is supported since version 2.5.
- `array`: The defined event chain. It must be an array.

Return value

Returns a value of the BIGINT type.

Examples

Example 1: Calculate the maximum number of consecutive events based on `uid`. The sliding window is 1800s and the filtering mode is `0`.

This example uses table `action`, in which data is sorted by `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)
```

Execute the following statement:

```
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 |
+------+-------+
```

Description of the results:

- The matching event chain for `uid = 1` is "Browse-Click-Order-Pay", and `4` is returned. The time of the last "Browse" event (2020-01-02 11:00:00) does not meet the condition and is not counted.

- The event chain of `uid = 2` does not start from the first event "Browse", and `0` is returned.
- The matching event chain for `uid = 3` is "Browse", and `1` is returned. The "Click" event exceeds the 1800s time window and is not counted.
- The matching event chain for `uid = 4` is "Browse-Click", and `2` is returned.
- The matching event chain for `uid = 5` is "Browse-Click", and `2` is returned. The "Order" event (2020-01-02 11:10:00) does not belong to the event chain and is not counted.
- The matching event chain for `uid = 6` is "Browse-Click-Order", and `3` is returned.

Example 2: Calculate the maximum number of consecutive events based on `uid`. The sliding window is 1800s, and filtering modes `0` and `1` are used.

This example uses table `action1`, in which data is sorted by `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)
```

Execute the following statement:

```
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)
```

For `uid = 1`, the "Click" event (2020-01-02 11:29:50) is a repeated event but it is still counted because mode `0` is used. Therefore, `4` is returned.

Change `mode` to `1` and execute the statement again.

```
+------+-------+
| uid  | level |
+------+-------+
| 1    |     3 |
| 2    |     1 |
+------+-------+
2 rows in set (0.05 sec)
```

The longest event chain filtered after deduplication is "Browse-Click-Order", and `3` is returned.

Example 3: Calculate the maximum number of consecutive events based on `uid`. The sliding window is 1900s, and filter modes `0` and `2` are used.

This example uses table `action2`, in which data is sorted by `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)
```

Execute the following statement:

```
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)
```

`3` is returned for `uid = 1` because mode `0` is used and the "Pay" event (2020-01-02 11:30:00) does not disrupt the event chain.

Change `mode` to `2` and execute the statement again.

```
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)
```

`2` is returned because the "Pay" event disrupts the event chain and the event counter stops. The filtered event chain is "Browse-Click".

Example 4: Calculate the maximum number of consecutive events based on `uid`. The sliding window is 1900s, and filter modes `0` and `4` are used.

This example uses table `action3`, in which data is sorted by `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)
```

Execute the following statement:

```
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 |
+------+-------+
```

`3` is returned for `uid = 1` and `uid = 2`.

Change `mode` to `4` and execute the statement again.

```
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)
```

`1` is returned for `uid = 2` because mode `4` (strictly increasing) is used. "Click" happens at the same second as "BROWSE". Therefore, "Click" and "Order" are not counted.

Keywords

window funnel, funnel, window_funnel

array_agg

Description

Aggregates the values including `NULL` in a column into an array.

Syntax

```
ARRAY_AGG(col)
```

Parameters

`col`: the column whose values you want to aggregate. Supported data types are BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, CHAR, DATETIME, and DATE.

Return value

Returns a value of the ARRAY data type.

Usage notes

- The order of the elements in an array is random, which means it may be different from the order of the values in the column.
- The data type of the elements in the returned array is the same as the data type of the values in the column.

Examples

Take the following data table as an example:

```
mysql> select * from test;
+------+------+
| c1   | c2   |
+------+------+
|    1 | a    |
|    1 | b    |
|    2 | c    |
|    2 | NULL |
|    3 | NULL |
+------+------+
```

Example 1: Group the values in column c1 and aggregate the values in column c2 into an array based on the grouping of column c1.

```
mysql> select c1, array_agg(c2) from test group by c1;
+------+----------------+
| c1   | array_agg(`c2`) |
+------+----------------+
|    1 | ["a","b"]       |
|    2 | [null,"c"]      |
|    3 | [null]          |
+------+----------------+
```

Example 2: Use the WHERE clause when the values in column c2 are aggregated into an array. If no data in column c2 meets the condition that is specified in the WHERE clause, a

`NULL` value is returned.

```
mysql> select array_agg(c2) from test where c1>4;

+-----------------+

| array_agg(`c2`) |

+-----------------+

| NULL            |

+-----------------+
```

Keywords

ARRAY_AGG, ARRAY

array_append

Description

Adds a new element to the end of the array. Returns an array.

Syntax

```
array_append(any_array, any_element)
```

Examples

```
mysql> select array_append([1, 2], 3);
+-----------------------+
| array_append([1,2], 3) |
+-----------------------+
| [1,2,3]                |
+-----------------------+
1 row in set (0.00 sec)
```

You can add NULL to the array.

```
mysql> select array_append([1, 2], NULL);
+--------------------------+
| array_append([1,2], NULL) |
+--------------------------+
| [1,2,NULL]                |
+--------------------------+
1 row in set (0.01 sec)
```

keyword

ARRAY_APPEND,ARRAY

array_avg

Description

Calculates the average value of all data in an ARRAY and return this result.

Syntax

```
array_avg(array(type))
```

`array(type)` supports the following types of elements: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMALV2.

Examples

```
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

Description

Concatenates multiple arrays into one array that contains all the elements in the arrays.

Elements in the arrays to concatenate can be of the same type or different types. However, we recommend that the elements be of the same type.

Nulls are processed as normal values.

Syntax

```
array_concat(input0, input1, ...)
```

Parameters

`input`: one or more arrays that you want to concatenate. Specify arrays in the `(input0, input1, ...)` format. This function supports the following types of array elements: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE, and JSON. **JSON is supported from 2.5.**

Return value

Returns an array that contains all the elements held in the arrays that are specified by the `input` parameter. The elements of the returned array are of the same data type as the elements of the input arrays. Additionally, the elements of the returned array follow the order of the input arrays and their elements.

Examples

Example 1: Concatenate arrays that contain numeric elements.

```
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] |

+-----------------------------------+
```

Example 2: Concatenate arrays that contain string elements.

```
select array_concat(["sql","storage","execute"], ["Query"],
["Vectorized", "cbo"]);

+-------------------------------------------------------------------
--------+

| array_concat(['sql','storage','execute'], ['Query'],
['Vectorized','cbo']) |

+-------------------------------------------------------------------
--------+

| ["sql","storage","execute","Query","Vectorized","cbo"]
|

+-------------------------------------------------------------------
--------+
```

Example 3: Concatenate two arrays of different types.

```
select array_concat([57,65], ["pear","apple"]);
+----------------------------------------+
| array_concat([57, 65], ['pear', 'apple']) |
+----------------------------------------+
```

```
| ["57","65","pear","apple"]                |
+-------------------------------------------+
```

Example 4: Process nulls as normal values.

```
select array_concat(["sql",null], [null], ["Vectorized", null]);

+-----------------------------------------------------------+

| array_concat(['sql',NULL], [NULL], ['Vectorized',NULL]) |

+-----------------------------------------------------------+

| ["sql",null,null,"Vectorized",null]                       |

+-----------------------------------------------------------+
```

array_contains

Description

Checks whether the array contains a certain element. If yes, it returns 1; otherwise, it returns 0.

Syntax

```
array_contains(any_array, any_element)
```

Examples

```
mysql> select array_contains(["apple","orange","pear"], "orange");
+----------------------------------------------------+
| array_contains(['apple','orange','pear'], 'orange') |
+----------------------------------------------------+
|                                                  1 |
+----------------------------------------------------+
1 row in set (0.01 sec)
```

You can also check whether the array contains NULL.

```
mysql> select array_contains([1, NULL], NULL);
+-------------------------------+
| array_contains([1,NULL], NULL) |
+-------------------------------+
|                             1 |
+-------------------------------+
1 row in set (0.00 sec)
```

You can check whether the multi-dimensional array contains a certain subarray. At this time, you need to ensure that the subarray elements match exactly, including the element

arrangement order.

```
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

Description

Checks whether `arr1` contains all the elements of `arr2`, that is, whether `arr2` is a subset of `arr1`. If yes, 1 is returned. If not, 0 is returned.

Syntax

```
BOOLEAN array_contains_all(arr1, arr2)
```

Parameters

`arr`: the two arrays to compare. This syntax checks whether `arr2` is a subset of `arr1`.

The data types of elements in the two arrays must be the same. For the data types of array elements supported by StarRocks, see ARRAY.

Return value

Returns a value of the BOOLEAN type.

1 is returned if `arr2` is a subset of `arr1`. Otherwise, 0 is returned.

If any of the two arrays is NULL, NULL is returned.

Usage notes

- If an array contains `null` elements, `null` is processed as a value.
- An empty array is a subset of any array.

- Elements in the two arrays can have different order.

Examples

1. Create a table named `t1` and insert data into this table.

```
CREATE TABLE t1 (
    c0 INT,
    c1 ARRAY<INT>,
    c2 ARRAY<INT>
) ENGINE=OLAP
DUPLICATE KEY(c0)
DISTRIBUTED BY HASH(c0) BUCKETS 8;

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. Query data from this table.

```
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. Check whether each row of `c2` is a subset of the corresponding row of `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 |
+------+-------------+----------+----------------------------+
```

In the output:

For row 1, `c2` is a subset of `c1` and 1 is returned.

For row 2, `c2` is not a subset of `c1` and 0 is returned.

For row 3, `c1` is NULL and NULL is returned.

For row 4, `c2` is NULL and NULL is returned.

For row 5, the two arrays contain `null` and `null` is processed as a normal value, `1` is returned.

For row 6, `c2` is an empty array and considered a subset of `c1`. Therefore, `1` is returned.

array_cum_sum

Description

Returns the cumulative sums of elements in an array.

Syntax

```
array_cum_sum(array(bigint))
array_cum_sum(array(double))
```

Parameters

`array`: The elements in the array must be of the BIGINT (8-byte signed integer) or DOUBLE (8-byte floating-point number) type.

Return values

Returns an array.

NULL is returned if the input array is NULL.

Examples

Example 1: Return the cumulative sums of an array.

```
select array_cum_sum([11, 11, 12]);
+--------------------------+
| array_cum_sum([11,11,12]) |
+--------------------------+
```

```
| [11,22,34]                |
+---------------------------+
```

Example 2: Use this function with 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]
|
+-------------------------------------------------------------------------
------------------+
```

Example 3: The input array contains a null.

```
select array_cum_sum([null,1,2]);
+--------------------------------+
| array_cum_sum([null,1,2])      |
+--------------------------------+
| [null,1,3]                     |
+--------------------------------+
```

Example 4: The input array is null.

```
select array_cum_sum(null);
+--------------------------------+
| array_cum_sum(NULL)            |
+--------------------------------+
| NULL                           |
+--------------------------------+
```

array_difference

Description

Calculates the difference between every two neighboring elements of an array by subtracting each element from its following element and returns an array that consists of the differences.

Syntax

```
array_difference(input)
```

Parameters

`input`: the array for which you want to calculate the differences between every two neighboring elements.

Return value

Returns an array of the same data type and length as the array that you specify in the `input` parameter.

Examples

Example 1:

```
mysql> SELECT array_difference([342, 32423, 213, 23432]);
+-----------------------------------------+
| array_difference([342,32423,213,23432]) |
+-----------------------------------------+
```

```
| [0,32081,-32210,23219]                    |
+-------------------------------------------+
```

Example 2:

```
mysql> SELECT array_difference([342, 32423, 213, null, 23432]);
+-----------------------------------------------+
| array_difference([342,32423,213,NULL,23432]) |
+-----------------------------------------------+
| [0,32081,-32210,null,null]                    |
+-----------------------------------------------+
```

Example 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]                        |
+-----------------------------------------------+
```

Example 4:

```
mysql> SELECT array_difference([false, true, false]);
+---------------------------------------+
| array_difference([FALSE, TRUE, FALSE]) |
+---------------------------------------+
| [0,1,-1]                              |
+---------------------------------------+
```

array_distinct

Description

Removes duplicate elements from an array.

Syntax

```
array_distinct(array)
```

Parameters

`array`: the array from which you want to remove duplicate elements. Only the ARRAY data type is supported.

Return value

Returns an array.

Usage notes

- The elements of the array that is returned may be sorted in a different order than the elements of the array that you specify.
- The elements of the array that is returned are of the same data type as the elements of the array that you specify.

Examples

In this section, the following table is used as an example:

```
mysql> select * from test;
+------+---------------+
| c1   | c2            |
+------+---------------+
|    1 | [1,1,2]       |
|    2 | [1,null,null] |
|    3 | NULL          |
|    4 | [null]        |
+------+---------------+
```

Remove duplicate values from column `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

Description

Returns values from an array that matches the given filter.

This function comes in the following two forms. The lambda adoption allows for more flexible array filtering. For more information about lambda functions, see Lambda expression. This function is supported from v2.5.

Syntax

```
array_filter(array, array<bool>)
array_filter(lambda_function, arr1,arr2...)
```

- `array_filter(array, array<bool>)`

  Returns values from an array that matches `array<bool>`.

- `array_filter(lambda_function, arr1,arr2...)`

  Returns values from an array that matches the lambda function.

Parameters

`array`: the array to filter values from.

`array<bool>`: the expression used to filter values.

`lambda_function`: the lambda function used to filter values.

Usage notes

- The two input parameters of `array_filter(array, array<bool>)` must be ARRAY and the filter expression can evaluate to `array<bool>`.
- The lambda function in `array_filter(lambda_function, arr1,arr2...)` follows the usage notes in array_map().
- If the input array is null, null is returned. If the filter array is null, an empty array is returned.

Examples

- Examples without using lambda functions

```
-- All the elements in the array match the filter.
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)

-- The filter is null and an empty array is returned.
select array_filter([1,2,3],null);
+-----------------------------+
| array_filter([1, 2, 3], NULL) |
+-----------------------------+
| []                          |
+-----------------------------+
1 row in set (0.01 sec)

-- The input array is null, null is returned.
select array_filter(null,[1]);
+------------------------+
| array_filter(NULL, [1]) |
+------------------------+
| NULL                   |
+------------------------+

-- Both the input array and filter are null. Null is returned.
select array_filter(null,null);
+-------------------------+
| array_filter(NULL, NULL) |
+-------------------------+
```

```
| NULL                       |
+----------------------------+
1 row in set (0.01 sec)

-- The filter contains a null element and an empty array is
returned.
select array_filter([1,2,3],[null]);
+--------------------------------+
| array_filter([1, 2, 3], [NULL]) |
+--------------------------------+
| []                             |
+--------------------------------+
1 row in set (0.01 sec)

-- The filter contains two null elements and an empty array is
returned.
select array_filter([1,2,3],[null,null]);
+-------------------------------------+
| array_filter([1, 2, 3], [NULL, NULL]) |
+-------------------------------------+
| []                                  |
+-------------------------------------+
1 row in set (0.00 sec)

-- Only one element matches the filter.
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)
```

- Examples using lambda functions

```
  -- Return the elements in x that are less than the elements in
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]                                                     |
  +-----------------------------------------------------------+
```

array_intersect

Description

Returns an array of the elements in the intersection of one or more arrays.

Syntax

```
array_intersect(input0, input1, ...)
```

Parameters

`input`: one or more arrays whose intersection you want to obtain. Specify arrays in the `(input0, input1, ...)` format and make sure that the arrays that you specify are of the same data type.

Return value

Returns an array of the same data type as the arrays that you specify.

Examples

Example 1:

```
mysql> SELECT array_intersect(["SQL", "storage"], ["mysql", "query",
"SQL"], ["SQL"])
AS no_intersect ;
+--------------+
| no_intersect |
+--------------+
```

```
| ["SQL"]      |
+--------------+
```

Example 2:

```
mysql> SELECT array_intersect(["SQL", "storage"], ["mysql", null],
[null]) AS no_intersect ;
+--------------+
| no_intersect |
+--------------+
| []           |
+--------------+
```

Example 3:

```
mysql> SELECT array_intersect(["SQL", null, "storage"], ["mysql",
null], [null]) AS no_intersect ;
+--------------+
| no_intersect |
+--------------+
| [null]       |
+--------------+
```

array_join

Description

Concatenates the elements of an array into a string.

Syntax

```
array_join(array, sep[, null_replace_str])
```

Parameters

- `array`: the array whose elements you want to concatenate. Only the ARRAY data type is supported.
- `sep`: the delimiter that is used to separate the concatenated array elements. Only the VARCHAR data type is supported.
- `null_replace_str`: the string that is used to substitute `NULL` values. Only the VARCHAR data type is supported.

Return value

Returns a value of the VARCHAR data type.

Usage notes

- The value of the `array` parameter must be a one-dimensional array.
- The `array` parameter does not support DECIMAL values.

- If you set the `sep` parameter to `NULL`, the return value is `NULL`.
- If you do not specify the `null_replace_str` parameter, `NULL` values are discarded.
- If you set the `null_replace_str` parameter to `NULL`, the return value is `NULL`.

Examples

Example 1: Concatenate the elements of an array. In this example, the `NULL` value in the array is discarded, and the concatenated array elements are separated by underscores (`_`).

```
mysql> select array_join([1, 3, 5, null], '_');

+------------------------------+

| array_join([1,3,5,NULL], '_') |

+------------------------------+

| 1_3_5                        |

+------------------------------+
```

Example 2: Concatenate the elements of an array. In this example, the `NULL` value in the array is substituted with `NULL` strings, and the concatenated array elements are separated by underscores (`_`).

```
mysql> select array_join([1, 3, 5, null], '_', 'NULL');

+-------------------------------------+

| array_join([1,3,5,NULL], '_', 'NULL') |

+-------------------------------------+

| 1_3_5_NULL                          |

+-------------------------------------+
```

array_length

Description

Returns the number of elements in the array. The result type is INT. if the parameter is NULL, the result is also NULL.

Syntax

```
array_length(any_array)
```

Examples

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

keyword

ARRAY_LENGTH,ARRAY

array_map

Description

array_map() is a higher-order SQL function that can take lambda functions as arguments.It applies the lambda function to the input arrays and returns a new array.

array_map() can accept multiple arrays and can be nested with lambda functions to capture a variable. For more information about lambda functions, see Lambda expression. This function is supported from v2.5.

The alias is transform().

Syntax

```
array_map(lambda_function, arr1, arr2...)
array_map(arr1, arr2..., lambda_function)
```

Usage notes

- Lambda functions can only be used as the **first** or **last** parameter of a higher-order function. Either position does not affect the calculation.
- The number of arrays must be the same as the input parameters in the lambda function. Otherwise, an error is returned.
- All the arrays must have the same number of elements. Otherwise, an error is returned.

Examples

Example 1: Increment every element of the array by 100.

```
select array_map(x -> x+100,[1,2,3]);
+----------------------------------+
```

```
| array_map(x -> x + 100, [1, 2, 3]) |
+------------------------------------+
| [101,102,103]                      |
+------------------------------------+
```

Example 2: Sum array1 [1,2,3] and array2 [11,12,13] element by element.

```
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]                                      |
+-------------------------------------------------+
```

Example 3: Determine whether the element in `x` is greater than 1.5. If yes, log(x) is returned for the element. If not, (x+y) is returned for the corresponding element.

```
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]
|
+-------------------------------------------------------------------
------+
```

Example 4: Use array_map() to capture variables. The returned element whose value is 1 indicates the condition is met.

```
-- In the example table, last_avg is the average score in the last
exam. score represents the scores of three subjects in the current
exam.

+------+----------+------------+
| id   | last_avg | score      |
+------+----------+------------+
|    1 |       55 | [50,60,70] |
|    2 |       73 | [70,65,75] |
```

```
|    3 |       89 | [88,92,90] |
+------+----------+------------+

-- Find the score that is higher than the average score of the last
exam.
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]                             |
+-------------------------------------+
```

Example 5: Use nested lambda functions.

```
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]]                                           |
+-----------------------------------------------------------------------+
```

Example 6: An error is returned because the numbers of arrays and lambda parameters are inconsistent. The lambda function requires only one array but two arrays are passed in.

```
select array_map(x -> x,[1],[2,4]);
ERROR 1064 (HY000): Lambda arguments should equal to lambda input
arrays.
```

array_max

Description

Obtains the maximum value of all data in an ARRAY and return this result.

Syntax

```
array_max(array(type))
```

Examples

```
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

Description

Obtains the minimum value of all data in an ARRAY and return this result.

Syntax

```
array_min(array(type))
```

Examples

```
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

Description

Obtains the position of an element in an array.

Syntax

```
array_position(any_array, any_element)
```

Parameters

- `any_array`: the array to be searched.
- `any_element`: an expression that matches the element in an array.

Return value

Returns the position of the specified element if the element can be found in the specified array and returns 0 if the element cannot be found in the array.

Examples

- Example 1: Obtain the position of an element in an array.

```
mysql> select array_position(["apple","orange","pear"], "orange");

+-------------------------------------------------+

| array_position(['apple','orange','pear'], 'orange') |

+-------------------------------------------------+
```

```
|                                                      2 |
+--------------------------------------------------------+
1 row in set (0.01 sec)
```

- Example 2: Obtain the position of `NULL` in an array.

```
mysql> select array_position([1, NULL], NULL);
```

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

- Example 3: Obtain the position of a sub-array in a multi-dimensional array. A position can be returned only when a sub-array that consists of the same elements in the same order as the specified sub-array can be found in the multi-dimensional array.

```
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

Description

Removes an element from an array.

Syntax

```
array_remove(any_array, any_element)
```

Parameters

- `any_array`: the array to be searched.
- `any_element`: an expression that matches the element in an array.

Return value

Returns the array from which specified element has been removed.

Examples

```
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_slice

Description

Returns a slice of an array. This function intercepts `length` elements from `input` from the position specified by `offset`.

Syntax

```
array_slice(input, offset, length)
```

Parameters

- `input`: the array whose slice you want to intercept. This function supports the following types of array elements: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE, and JSON. **JSON is supported from 2.5.**
- `offset`: the position from which to intercept elements. Valid values start from `1`. It must be a BIGINT value.
- `length`: the length of the slice that you want to intercept. It must be a BIGINT value.

Return value

Returns an array that have the same data type as the array specified by the `input` parameter.

Usage notes

- The offset starts from 1.

- If the specified length exceeds the actual number of elements that can be intercepted, all the matching elements are returned. See Example 4.

Examples

Example 1: Intercept 2 elements starting from the third element.

```
mysql> select array_slice([1,2,4,5,6], 3, 2) as res;
+-------+
| res   |
+-------+
| [4,5] |
+-------+
```

Example 2: Intercept 2 elements starting from the first element.

```
mysql> select array_slice(["sql","storage","query","execute"], 1, 2)
as res;
+------------------+
| res              |
+------------------+
| ["sql","storage"] |
+------------------+
```

Example 3: Null elements are treated as normal values.

```
mysql> select array_slice([57.73,97.32,128.55,null,324.2], 3, 3) as
res;
+--------------------+
| res                |
+--------------------+
| [128.55,null,324.2] |
+--------------------+
```

Example 4: Intercept 5 elements starting from the third element.

This function intends to intercept 5 elements but there are only 3 elements from the third element. As a result, all of these 3 elements are returned.

```
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

Description

Sorts the elements of an array in ascending order.

Syntax

```
array_sort(array)
```

Parameters

`array`: the array whose elements you want to sort. Only the ARRAY data type is supported.

Return value

Returns an array.

Usage notes

- This function sorts the elements of an array only in ascending order.
- `NULL` values are placed at the beginning of the array that is returned.
- If you want to sort the elements of an array in descending order, use the reverse function.
- The elements of the returned array have the same data type as the elements of the input array.

Examples

The following table is used as an example:

```
mysql> select * from test;

+------+--------------+

| c1   | c2           |

+------+--------------+

|    1 | [4,3,null,1] |

|    2 | NULL         |

|    3 | [null]       |

|    4 | [8,5,1,4]    |

+------+--------------+
```

Sort the values of column `c2` in ascending order.

```
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]        |

+------+------------------+
```

array_sortby

Description

Sorts elements in an array according to the ascending order of elements in another array or array converted from a lambda expression. For more information, see Lambda expression. This function is supported from v2.5.

Elements in the two arrays are like key-value pairs. For example, b = [7,5,6] is the sorting key of a = [3,1,4]. According to the key-value pair relationship, elements in the two arrays have the following one-to-one mapping.

Array	Element 1	Element 2	Element 3
a	3	1	4
b	7	5	6

After array `b` is sorted in ascending order, it becomes [5,6,7]. Array `a` becomes [1,4,3] accordingly.

Array	Element 1	Element 2	Element 3
a	1	4	3
b	5	6	7

Syntax

```
array_sortby(array0, array1)
array_sortby(<lambda function>, array0 [, array1...])
```

- `array_sortby(array0, array1)`

  Sorts `array0` according to the ascending order of `array1`.

- `array_sortby(<lambda function>, array0 [, array1...])`

  Sorts `array0` according to the array returned from the lambda function.

Parameters

- `array0`: the array you want to sort. It must be an array, array expression, or `null`. Elements in the array must be sortable.
- `array1`: the sorting array used to sort `array0`. It must be an array, array expression, or `null`.
- `lambda function`: the lambda expression used to generate the sorting array.

Return value

Returns an array.

Usage notes

- This function can sort elements of an array only in ascending order.
- `NULL` values are placed at the beginning of the array that is returned.
- If you want to sort elements of an array in descending order, use the reverse function.
- If the sorting array (`array1`) is null, data in `array0` remains unchanged.
- The elements of the returned array have the same data type as the elements of `array0`. The attribute of null values are also the same.
- The two arrays must have the same number of elements. Otherwise, an error is returned.

Examples

The following table is used to demonstrate how to use this function.

```
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`) BUCKETS 2
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT",
"enable_persistent_index" = "false",
"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)
```

Example 1: Sort `c3` according to `c2`. This example also provides the result of array_sort() for comparison.

```
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]
|
+------+-----------+------------+---------------+----------------
-----+
```

Example 2: Sort array `c3` based on `c2` generated from a lambda expression. This example is equivalent to Example 1. It also provides the result of array_sort() for comparison.

```
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]      |
+------+------------+-------------+----------------+-----------------+
```

Example 3: Sort array `c3` based on the ascending order of `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]         |
+----------+---------+----------+------------+------------------+
```

References

array_sort

array_sum

Description

Sums all the elements in an array.

From StarRocks 2.5, array_sum() can take a lambda expression as an argument. However, it cannot work directly with the Lambda expression. It must work on the result converted from array_map().

Syntax

```
array_sum(array(type))
array_sum(lambda_function, arr1,arr2...) =
array_sum(array_map(lambda_function, arr1,arr2...))
```

Parameters

- `array(type)`: the array you want to calculate the sum. Array elements support the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, and DECIMALV2.
- `lambda_function`: the lambda expression used to calculate the target array for array_sum().

Return value

Returns a numeric value.

Examples

Use array_sum without lambda function

```
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                         |
+--------------------------------+
```

Use array_sum with lambda function

```
-- Multiply [1,2,3] by [1,2,3] and sum the elements.
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

Description

Converts an array into BITMAP values. This function is supported from v2.3.

Syntax

```
BITMAP array_to_bitmap(array)
```

Parameters

`array`: Elements in an array can be of the BIGINT, INT, TINYINT, or SMALLINT type.

Return value

Returns a value of the BITMAP type.

Usage notes

- If the data type of elements in the input array is invalid, such as STRING or DECIMAL, an error is returned.
- If an empty array is entered, an empty BITMAP value is returned.
- If `NULL` is entered, `NULL` is returned.

Examples

Example 1: Convert an array into BITMAP values. This function must be nested in `bitmap_to_array` because BITMAP values cannot be displayed.

```
MySQL > select bitmap_to_array(array_to_bitmap([1,2,3]));
+-----------------------------------------+
| bitmap_to_array(array_to_bitmap([1,2,3])) |
+-----------------------------------------+
| [1,2,3]                                 |
+-----------------------------------------+
```

Example 2: Enter an empty array and an empty array is returned.

```
MySQL > select bitmap_to_array(array_to_bitmap([]));
+--------------------------------------+
| bitmap_to_array(array_to_bitmap([])) |
+--------------------------------------+
| []                                   |
+--------------------------------------+
```

Example 3: Enter `NULL` and `NULL` is returned.

```
MySQL > select array_to_bitmap(NULL);
+----------------------+
| array_to_bitmap(NULL) |
+----------------------+
| NULL                 |
+----------------------+
```

arrays_overlap

Description

Checks whether the intersection of two arrays is empty. The arrays that you specify must be of the same data type. If the intersection is empty, this function returns `0`. Otherwise, this function returns `1`.

Syntax

```
arrays_overlap(input0, input1)
```

Parameters

- `input0`: one of the two arrays that you want to compare.
- `input1`: the other of the two arrays that you want to compare. The value of this parameter must be of the same data type as the value of the `input0` parameter.

Return value

Returns a value of the BOOLEAN data type.

Examples

Example 1:

```
mysql> select arrays_overlap([11, 9, 3, 2], [null, 11]);
+-----------------------------------+
| arrays_overlap([11,9,3,2], [NULL,11]) |
+-----------------------------------+
```

```
|                                      1 |
+----------------------------------------+
```

Example 2:

```
mysql> select arrays_overlap([9, 3, 2], [null, 11]);
+-----------------------------------+
| arrays_overlap([9,3,2], [NULL,11]) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

Example 3:

```
mysql> select arrays_overlap([9, 3, null, 2], [null, 11]);
+-------------------------------------+
| arrays_overlap([9,3,NULL,2], [NULL,11]) |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

Example 4:

```
mysql> select arrays_overlap([9, 3, "SQL", 2], [null, "SQL"]);
+-----------------------------------------+
| arrays_overlap([9,3,'SQL',2], [NULL,'SQL']) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

reverse

Description

Reverses a string or array. Returns a string or array with the characters in the string or array elements in reverse order.

Syntax

```
reverse(param)
```

Parameters

`param`: the string or array to reverse. It can be of the VARCHAR, CHAR, or ARRAY type.

Currently, this function supports only one-dimensional arrays and the array elements cannot be of the DECIMAL type. This function supports the following types of array elements: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE, and JSON. **JSON is supported from 2.5.**

Return value

The return type is the same as `param`.

Examples

Example 1: Reverse a string.

```
MySQL > SELECT REVERSE('hello');
+------------------+
| REVERSE('hello') |
```

```
+------------------+
| olleh            |
+------------------+
1 row in set (0.00 sec)
```

Example 2: Reverse an array.

```
MYSQL> SELECT REVERSE([4,1,5,8]);
+--------------------+
| REVERSE([4,1,5,8]) |
+--------------------+
| [8,5,1,4]          |
+--------------------+
```

unnest

Description

UNNEST is a table function that takes an array and converts elements in that array into multiple rows of a table. The conversion is also known as "flattening".

You can use Lateral Join with UNNEST to implement common conversions, for example, from STRING, ARRAY, or BITMAP to multiple rows. For more information, see Lateral join.

From v2.5, UNNEST can take a variable number of array parameters. The arrays can vary in type and length (number of elements). If the arrays have different lengths, the largest length prevails, which means nulls will be added to arrays that are less than this length. See Example 2 for more information.

Syntax

```
unnest(array0[, array1 ...])
```

Parameters

`array`: the array you want to convert. It must be an array or expression that can evaluate to an ARRAY data type. You can specify one or more arrays or array expressions.

Return value

Returns the multiple rows converted from the array. The type of return value depends on the types of elements in the array.

For the element types supported in an array, see ARRAY.

Usage notes

- UNNEST is a table function. It must be used with Lateral Join but the keyword Lateral Join does not need to be explicitly specified.
- If the array expression evaluates to NULL or it is empty, no rows will be returned.
- If an element in the array is NULL, NULL is returned for that element.

Examples

Example 1: UNNEST takes one parameter

```
-- Create table student_score where scores is an ARRAY column.
CREATE TABLE student_score
(
    `id` bigint(20) NULL COMMENT "",
    `scores` ARRAY<int> NULL COMMENT ""
)
DUPLICATE KEY (id)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;

-- Insert data into this table.
INSERT INTO student_score VALUES
(1, [80,85,87]),
(2, [77, null, 89]),
(3, null),
(4, []),
(5, [90,92]);

-- Query data from this table.
SELECT * FROM student_score ORDER BY id;
+------+-------------+
| id   | scores      |
+------+-------------+
|    1 | [80,85,87]  |
|    2 | [77,null,89] |
|    3 | NULL        |
|    4 | []          |
|    5 | [90,92]     |
+------+-------------+

-- Use UNNEST to flatten the scores column into multiple rows.
```

```
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] corresponding to `id = 1` is converted into three rows.

[77,null,89] corresponding to `id = 2` retains the null value.

`scores` corresponding to `id = 3` and `id = 4` are NULL and empty, which are skipped.

Example 2: UNNEST takes multiple parameters

```
-- Create table example_table where the type and scores columns vary
in type.
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) BUCKETS 1
PROPERTIES (
"replication_num" = "3");

-- Insert data into the table.
INSERT INTO example_table VALUES
("1", "typeA;typeB", [80,85,88]),
("2", "typeA;typeB;typeC", [87,90,95]);

-- Query data from the table.
SELECT * FROM example_table;
+------+-----------------+------------+
```

```
| id   | type              | scores     |
+------+-------------------+------------+
| 1    | typeA;typeB       | [80,85,88] |
| 2    | typeA;typeB;typeC | [87,90,95] |
+------+-------------------+------------+

-- Use UNNEST to convert type and scores into multiple rows.
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` and `scores` in `UNNEST` vary in type and length.

`type` is a VARCHAR column while `scores` is an ARRAY column. The split() function is used to convert `type` into ARRAY.

For `id = 1`, `type` is converted into ["typeA","typeB"], which has two elements.

For `id = 2`, `type` is converted into ["typeA","typeB","typeC"], which has three elements.

To ensure consistent numbers of rows for each `id`, a null element is added to ["typeA","typeB"].

bit_shift_left

Description

Shifts the binary representation of a numeric expression to the left by a specified number of bits.

This function performs an **arithmetic left shift**, during which the bit length does not change, 0 is appended to the end, and the high bit remains unchanged. For example, shifting `10101` to the left by one bit results in `11010`.

Syntax

```
bit_shift_left(value, shift)
```

Parameters

`value`: the value or numeric expression to shift. Supported data types are TINYINT, SMALLINT, INT, BIGINT, and LARGEINT.

`shift`: the number of bits to shift. The supported data type is BIGINT.

Return value

Returns a value of the same type as `value`.

Usage notes

- If any input parameter is NULL, NULL is returned.
- If `shift` is less than 0, 0 is returned.

- Shifting a `value` by `0` always results in the original `value`.
- Shifting `0` by a `shift` always results in `0`.
- If the data type of `value` is numeric but not an integer, that value will be cast to an integer. See Examples.
- If the data type of `value` is a string, the value will be cast to an integer if possible. If the value cannot be cast to an integer, it will be processed as NULL. See Examples.

Examples

Use this function to shift numeric values.

```
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 |
+-----------------------+
```

References

- bit_shift_right
- bit_shift_right_logical

bit_shift_right_logical

Description

Shifts the binary representation of a numeric expression to the right by a specified number of bits.

This function performs a **logical right shift,** during which the bit length does not change, the low bit is dropped, and 0 is appended to the high bit despite whether the original bit is positive or negative. **Logical** shifts are unsigned shifts. For example, shifting `10101` by one bit results in `00101`.

bit_shift_right() and bit_shift_right_logical() return the same result for a positive value.

Syntax

```
bit_shift_right_logical(value, shift)
```

Parameters

`value`: the value or numeric expression to shift. Supported data types are TINYINT, SMALLINT, INT, BIGINT, and LARGEINT.

`shift`: the number of bits to shift. The supported data type is BIGINT.

Return value

Returns a value of the same type as `value`.

Usage notes

- If any input parameter is NULL, NULL is returned.
- If `shift` is less than 0, 0 is returned.
- Shifting a `value` by `0` always results in the original `value`.
- Shifting `0` by a `shift` always results in `0`.
- If the data type of `value` is numeric but not an integer, that value will be cast to an integer. See Examples.
- If the data type of `value` is a string, the value will be cast to an integer if possible. For example, the string "2.3" will be cast to 2. If the value cannot be cast to an integer, the value will be treated as NULL. See Examples.

Examples

Use this function to shift numeric values.

```
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 |
+-----------------------------------------------------+
```

References

- bit_shift_left
- bit_shift_right

bit_shift_right

Description

Shifts the binary representation of a numeric expression to the right by a specified number of bits.

This function performs an **arithmetic right shift**, during which the bit length does not change, the low bit is dropped, and the sign bit is used as the high bit. For example, shifting `10101` to the right by one bit results in `11010`.

Syntax

```
bit_shift_right(value, shift)
```

Parameters

`value`: the value or numeric expression to shift. Supported data types are TINYINT, SMALLINT, INT, BIGINT, and LARGEINT.

`shift`: the number of bits to shift. The supported data type is BIGINT. The supported data type is BIGINT.

Return value

Returns a value of the same type as `value`.

Usage notes

- If any input parameter is NULL, NULL is returned.
- If `shift` is less than 0, 0 is returned.

- Shifting a `value` by `0` always results in the original `value`.
- Shifting `0` by a `shift` always results in `0`.
- If the data type of `value` is numeric but not an integer, that value will be cast to an integer. See Examples.
- If the data type of `value` is a string, the value will be cast to an integer if possible. For example, the string "2.3" will be cast to 2. If the value cannot be cast to an integer, the value will be treated as NULL. See Examples.

Examples

Use this function to shift numeric values.

```
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 |
+-----------------------+
```

References

- bit_shift_left
- bit_shift_right_logical

bitand

Description

Returns the bitwise AND of two numeric expressions.

Syntax

```
BITAND(x,y);
```

Parameters

- `x`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.
- `y`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

> `x` and `y` must agree in data type.

Return value

The return value has the same type as `x`. If any value is NULL, the result is NULL.

Examples

```
mysql> select bitand(3,0);
+--------------+
| bitand(3, 0) |
+--------------+
|            0 |
```

```
+--------------+
1 row in set (0.01 sec)
```

bitnot

Description

Returns the bitwise negation of a numeric expression.

Syntax

```
BITNOT(x);
```

Parameters

`x`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

Return value

The return value has the same type as `x`. If any value is NULL, the result is NULL.

Examples

```
mysql> select bitnot(3);
+-----------+
| bitnot(3) |
+-----------+
|        -4 |
+-----------+
```

bitor

Description

Returns the bitwise OR of two numeric expressions.

Syntax

```
BITOR(x,y);
```

Parameters

- `x`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.
- `y`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

> `x` and `y` must agree in data type.

Return value

The return value has the same type as `x`. If any value is NULL, the result is NULL.

Examples

```
mysql> select bitor(3,0);
+-------------+
| bitor(3, 0) |
+-------------+
|           3 |
```

```
+--------------+
1 row in set (0.00 sec)
```

bitxor

Description

Returns the bitwise XOR of two numeric expressions.

Syntax

```
BITXOR(x,y);
```

Parameters

- `x`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.
- `y`: This expression must evaluate to any of the following data types: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

> `x` and `y` must agree in data type.

Return value

The return value has the same type as `x`. If any value is NULL, the result is NULL.

Examples

```
mysql> select bitxor(3,0);
+--------------+
| bitxor(3, 0) |
+--------------+
|            3 |
```

```
+--------------+
1 row in set (0.00 sec)
```

base64_to_bitmap

Description

Before you import bitmap data into StarRocks, you need to serialize the data and encode the data as a Base64 string. When you import the Base64 string into StarRocks, you need to convert the string into bitmap data. This function is used to convert Base64 strings into bitmap data.

This function is supported from v2.3.

Syntax

```
BITMAP base64_to_bitmap(VARCHAR bitmap)
```

Parameters

`bitmap`: The supported data type is VARCHAR. Before you load Bitmap data into StarRocks, you can use Java or C++ to create a BitmapValue object, add an element, serialize the data, and encode the data as a Base64 string. Then, pass the Base64 string as an input parameter into this function.

Return value

Returns a value of the BITMAP type.

Examples

Create a database named `bitmapdb` and a table named `bitmap`. Use Stream Load to import JSON data into `bitmap_table`. During this process, use base64_to_bitmap to convert the Base64 string in the JSON file into bitmap data.

1. Create a database and a table in StarRocks. In this example, a Primary Key table is created.

```
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`) BUCKETS 1
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);
```

2. Use Stream Load to import JSON data into `bitmap_table`.

   Suppose there is a JSON file named **simpledata**. This file has the following content and `userid` is a Base64-encoded string.

```
{
    "tagname": "Product", "tagvalue": "Insurance",
"userid":"AjowAAABAAAAAAACABAAAAABAAIAAwA="
}
```

   Use base64_to_bitmap to convert `userid` into a bitmap value.

```
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. Query data from `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

Description

Aggregates values (excluding NULLs) in a column into a bitmap (multiple rows into one row).

Syntax

```
BITMAP_AGG(col)
```

Parameters

`col`: the column whose values you want to aggregate. It must evaluate to BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, and LARGEINT.

Return value

Returns a value of the BITMAP type.

Usage notes

If the value in a row is less than 0 or greater than 18446744073709551615, the value will be ignored and not added to the Bitmap (see Example 3).

Examples

Take the following data table as an example:

```
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 |
+------+------+------+------+-------+----------------------+
```

Example 1: Aggregate values in column `c1` into one bitmap.

```
mysql> select bitmap_to_string(bitmap_agg(c1)) from t1_test;
+---------------------------------+
| bitmap_to_string(bitmap_agg(c1)) |
+---------------------------------+
| 1,2,3,4,5,6                     |
+---------------------------------+
```

Example 2: Aggregate values in column `c2` into one bitmap (NULLs are ignored).

```
mysql> SELECT BITMAP_TO_STRING(BITMAP_AGG(c2)) FROM t1_test;
+--------------------------------+
| bitmap_to_string(bitmap_agg(c2)) |
+--------------------------------+
| 0,1                            |
+--------------------------------+
```

Example 3: Aggregate values in column `c6` into one bitmap (the last two values that exceed the value range are ignored).

```
mysql> select bitmap_to_string(bitmap_agg(c6)) from t1_test;
+--------------------------------+
| bitmap_to_string(bitmap_agg(c6)) |
+--------------------------------+
| 11111,22222,33333              |
+--------------------------------+
```

Keywords

BITMAP_AGG, BITMAP

bitmap_and

Description

Calculates the intersection of two input bitmaps and returns the new bitmap.

Syntax

```
BITMAP BITMAP_AND(BITMAP lhs, BITMAP rhs)
```

Examples

```
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 |
+------+
```

keyword

BITMAP_AND,BITMAP

bitmap_andnot

Description

Returns bitmap values that exist in `lhs` but do not exist in `rhs`, and returns the new bitmap.

Syntax

```
bitmap_andnot(BITMAP lhs, BITMAP rhs)
```

Examples

```
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   |
+------+
```

keyword

BITMAP_ANDNOT, BITMAP

bitmap_contains

Description

Calculates whether the input value is in the bitmap column, and returns a Boolean value.

Syntax

```
B00LEAN BITMAP_CONTAINS(BITMAP bitmap, BIGINT input)
```

Examples

```
MySQL > select bitmap_contains(to_bitmap(1),2) cnt;
+------+
| cnt  |
+------+
|    0 |
+------+

MySQL > select bitmap_contains(to_bitmap(1),1) cnt;
+------+
| cnt  |
+------+
|    1 |
+------+
```

keyword

BITMAP_CONTAINS,BITMAP

bitmap_count

Description

Returns the 1-bit count for the input bitmap.

Syntax

```
INT BITMAP_COUNT(any_bitmap)
```

Examples

```
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 |
+--------------------+
```

keyword

BITMAP,BITMAP_COUNT

bitmap_empty

Description

Returns an empty bitmap. It is mainly used to fill in the default value during insert or stream load. For example:

```
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
```

Syntax

```
BITMAP BITMAP_EMPTY()
```

Examples

```
MySQL > select bitmap_count(bitmap_empty());
+-----------------------------+
| bitmap_count(bitmap_empty()) |
+-----------------------------+
|                            0 |
+-----------------------------+
```

keyword

BITMAP_EMPTY,BITMAP

bitmap_from_string

Description

Converts a string into a BITMAP. The string is composed of a set of UINT64 numbers separated by commas. For example, the "0, 1, 2" string will be converted into a Bitmap, in which bits 0, 1 and 2 are set. If the input field is invalid, NULL will be returned.

This function deduplicates the input string during the conversion. It must be used with other functions, such as bitmap_to_string to return results on terminals.

Syntax

```
BITMAP BITMAP_FROM_STRING(VARCHAR input)
```

Return value

Returns a BITMAP value. If the input string is invalid, NULL is returned. If the input string is empty, an empty value is returned.

Examples

```
-- The input is empty and an empty value is returned.
MySQL > select bitmap_to_string(bitmap_empty());
+----------------------------------+
| bitmap_to_string(bitmap_empty()) |
+----------------------------------+
|                                  |
+----------------------------------+

-- `0,1,2` is returned.
MySQL > select bitmap_to_string(bitmap_from_string("0, 1, 2"));
+-----------------------------------------------+
```

```
| bitmap_to_string(bitmap_from_string('0, 1, 2')) |
+-------------------------------------------------+
| 0,1,2                                           |
+-------------------------------------------------+

-- `-1` is an invalid input and NULL is returned.

MySQL > select bitmap_to_string(bitmap_from_string("-1, 0, 1, 2"));
+-----------------------------------+
| bitmap_from_string('-1, 0, 1, 2') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+

-- 2^64 is an invalid input and NULL is returned.
MySQL > select bitmap_to_string(bitmap_from_string("0,
18446744073709551616"));
+-------------------------------------------------------------------+
| bitmap_to_string(bitmap_from_string('0, 18446744073709551616')) |
+-------------------------------------------------------------------+
| NULL                                                              |
+-------------------------------------------------------------------+

-- The input string is deduplicated.

MySQL > select bitmap_to_string(bitmap_from_string("0, 1, 1"));
+-------------------------------------------------+
| bitmap_to_string(bitmap_from_string('0, 1, 1')) |
+-------------------------------------------------+
| 0,1                                             |
+-------------------------------------------------+
```

keywords

BITMAP_FROM_STRING,BITMAP

bitmap_has_any

Description

Calculates whether there are intersecting elements between two Bitmap columns, and the return value is Boolean value.

Syntax

```
B00LEAN BITMAP_HAS_ANY(BITMAP lhs, BITMAP rhs)
```

Examples

```
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 |
+------+
```

keyword

BITMAP_HAS_ANY,BITMAP

bitmap_hash

Description

Calculates a 32-bit hash value for any type of input and return the bitmap containing the hash value. It is mainly used for the stream load task to import non integer fields into the bitmap field of the StarRocks table. For example:

```
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
```

Syntax

```
BITMAP BITMAP_HASH(expr)
```

Examples

```
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                            |
+---------------------------------------+
```

keyword

BITMAP_HASH,BITMAP

bitmap_intersect

Description

Aggregation function, used to calculate the bitmap intersection after grouping. Common usage scenarios, such as calculating user retention rate.

Syntax

```
BITMAP BITMAP_INTERSECT(BITMAP value)
```

Enter a set of bitmap values, find the intersection of this set of bitmap values, and return the result.

Example

Table structure

```
KeysType: AGG_KEY
Columns: tag varchar, date datetime, user_id bitmap bitmap_union
```

```
-- Calculate users retention under different tags today and
yesterday.
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;
```

Use with bitmap_to_string function to obtain the specific data of the intersection.

```
--Find out users retained under different tags today and yesterday.
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;
```

keyword

BITMAP_INTERSECT, BITMAP

bitmap_max

Description

Obtains the maximum value of a bitmap. If the bitmap is `NULL`, this function returns `NULL`. If the bitmap is empty, this function returns `NULL` by default.

Syntax

```
bitmap_max(bitmap)
```

Parameters

`bitmap`: the bitmap whose maximum value you want to obtain. Only the BITMAP data type is supported. You can specify a bitmap that you construct by using functions such as bitmap_from_string.

Return value

Returns a value of the LARGEINT data type.

Examples

```
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

Description

Obtains the minimum value of a bitmap. If the bitmap is `NULL`, this function returns `NULL`. If the bitmap is empty, this function returns `NULL` by default.

Syntax

```
bitmap_min(bitmap)
```

Parameters

`bitmap`: the bitmap whose minimum value you want to obtain. Only the BITMAP data type is supported. You can specify a bitmap that you construct by using functions such as bitmap_from_string.

Return value

Returns a value of the LARGEINT data type.

Examples

```
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

Description

Calculates the union of two input bitmaps and return a new bitmap.

Syntax

```
BITMAP BITMAP_OR(BITMAP lhs, BITMAP rhs)
```

Examples

```
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 |
+------+
```

keyword

BITMAP_OR,BITMAP

bitmap_remove

Description

Removes `input` from the Bitmap value `lhs` and returns a result set.

Syntax

```
bitmap_remove(BITMAP lhs, BIGINT input)
```

Examples

```
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 |
+------+
```

keyword

BITMAP_REMOVE, BITMAP

bitmap_to_array

Description

Converts a BITMAP into a BIGINT array.

Syntax

```
ARRAY<BIGINT> BITMAP_TO_ARRAY (bitmap)
```

Parameters

`bitmap`: the bitmap you want to convert.

Return value

Returns a BIGINT array.

Examples

```
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

Description

Converts a bitmap to a Base64-encoded string. This function is supported from v2.5.

Syntax

```
VARCHAR bitmap_to_base64(BITMAP bitmap)
```

Parameters

`bitmap`: the bitmap to convert. This parameter is required. If the input value is invalid, an error is returned.

Return value

Returns a value of the VARCHAR type.

Examples

Example 1: Use this function with other bitmap functions.

```
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)
```

Example 2: Convert each value in a BITMAP column into Base64-encoded strings.

1. Create an Aggregate table `page_uv` where the `AGGREGATE KEY` is (`page_id`, `visit_date`). This table contains a BITMAP column `visit_users` whose values are to be aggregated.

```
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`) BUCKETS 1
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT"
);
```

2. Insert data into this table.

```
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. Convert each value in the `visit_users` column into a Base64-encoded string.

```
select page_id, bitmap_to_base64(visit_users) from page_uv;
+---------+-----------------------------------------+
| page_id | bitmap_to_base64(visit_users)           |
+---------+-----------------------------------------+
|       1 | CgMAAAANAAAAAAAABcAAAAAAAAIQAAAAAAAA= |
|       1 | AQ0AAAA=                                |
|       2 | ARcAAAA=                                |
+---------+-----------------------------------------+
```

bitmap_to_string

Description

Converts an input bitmap into a string that is separated by commas (,). This string contains all the bits in the bitmap. If the input is null, null is returned.

Syntax

```
VARCHAR BITMAP_TO_STRING(BITMAP input)
```

Parameters

`input`: the bitmap you want to convert.

Return value

Returns a value of the VARCHAR type.

Examples

Example 1: The input is null and null is returned.

```
MySQL > select bitmap_to_string(null);
+-----------------------+
| bitmap_to_string(NULL) |
+-----------------------+
| NULL                  |
+-----------------------+
```

Example 2: The input is empty and an empty string is returned.

```
MySQL > select bitmap_to_string(bitmap_empty());
+----------------------------------+
| bitmap_to_string(bitmap_empty()) |
+----------------------------------+
|                                  |
+----------------------------------+
```

Example 3: Convert a bitmap that contains one bit into a string.

```
MySQL > select bitmap_to_string(to_bitmap(1));
+--------------------------------+
| bitmap_to_string(to_bitmap(1)) |
+--------------------------------+
| 1                              |
+--------------------------------+
```

Example 4: Convert a bitmap that contains two bits into a string.

```
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

Description

Calculates the bitmap union of a set of values after grouping. Common usage scenarios include calculating PV and UV.

Syntax

```
BITMAP BITMAP_UNION(BITMAP value)
```

Examples

```
select page_id, bitmap_union(user_id)
from table
group by page_id;
```

Use this function with bitmap_count() to obtain the UV of a web page.

```
select page_id, bitmap_count(bitmap_union(user_id))
from table
group by page_id;
```

If `user_id` is an integer, the above query statement is equivalent to the following:

```
select page_id, count(distinct user_id)
from table
group by page_id;
```

keyword

BITMAP_UNION, BITMAP

bitmap_union_count

Description

Returns the union of a set of bitmap values and returns the cardinality of the union. This function is supported from v2.3.

Syntax

```
BIGINT bitmap_union_count(BITMAP value)
```

Parameters

`value`: a set of bitmap values. The supported data type is BITMAP.

Return value

Returns a value of the BIGINT type.

Examples

Calculate the unique views (UVs) of a web page. If `user_id` is of the INT type, the latter two queries are equivalent.

```
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

Description

Count the number of different values in columns of type TINYINT, SMALLINT and INT, return the sum of COUNT (DISTINCT expr) same.

Syntax

```
BIGINT bitmap_union_int(expr)
```

Parameters

`expr`: column expression. The supported column type is TINYINT, SMALLINT and INT.

Return value

Returns a value of the BIGINT type.

Examples

```
mysql> select bitmap_union_int(k1) from tbl1;
+------------------------+
| bitmap_union_int(`k1`) |
+------------------------+
|                      2 |
+------------------------+
```

bitmap_xor

Description

Calculates the set consisting elements unique to `lhs` and `rhs`. It is logically equivalent to `bitmap_andnot(bitmap_or(lhs, rhs), bitmap_and(lhs, rhs))` (complementary set).

Syntax

```
bitmap_xor(BITMAP lhs, BITMAP rhs)
```

Examples

```
mysql> select bitmap_to_string(bitmap_xor(bitmap_from_string('1, 3'),
bitmap_from_string('2'))) cnt;
+------+
|cnt   |
+------+
|1,2,3 |
+------+
```

keyword

BITMAP_XOR, BITMAP

intersect_count

Description

A function to find the intersection size (number of same elements) of two Bitmap values, without requiring the data distribution to be orthogonal. The first parameter is the Bitmap column, the second parameter is the dimension column used for filtering, and the third parameter is a variable-length parameter, meaning that different values of the filtering dimension column are taken.

If there is no intersection, 0 is returned.

Syntax

```
BIGINT INTERSECT_COUNT(bitmap_column, column_to_filter,
filter_values)
```

Parameters

- `bitmap_column`: the bitmap column to be computed.
- `column_to_filter`: the name of the column to be intersected.
- `filter_values`: the different values of the filtered dimension columns.

Return value

Returns a value of the BIGINT type.

Example

Create an Aggregate table with a Bitmap column `user_id` and insert data.

```
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                    |
+------------+--------------------------+
```

Calculate the number of same `user_id` corresponding to `'2020-10-01'` and `'2020-10-02'`. Two same elements (2 and 3) are found and the number `2` is returned.

```
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 |
+-----------------------------------------------------------+
```

Calculate the number of same `user_id` corresponding to `'2020-10-01'` and `'2020-10-03'`. One element `5` is found and the number `1` is returned.

```
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

Description

Intercepts `len` elements from a BITMAP value `src` starting from the position specified by `offset`. The output elements are a subset of `src`.

This function is mainly used for scenarios such as paginated queries. It is supported from v2.5.

Syntax

```
BITMAP sub_bitmap(BITMAP src, BIGINT offset, BIGINT len)
```

Parameters

- `src`: the BITMAP value from which you want to obtain elements.
- `offset`: the starting position. It must be a BIGINT value. Note the following points when you use `offset`:
  - Offsets start from 0.
  - Negative offsets are counted from right to left. See Examples 3 and 4.
  - If the starting position specified by `offset` exceeds the actual length of the BITMAP value, NULL is returned. See Example 6.
- `len`: the number of elements to obtain. It must be a BIGINT value greater than or equal to 1. If the number of matching elements is less than the value of `len`, all the matching elements are returned. See Examples 2, 3, and 7.

Return value

Returns a value of the BITMAP type. NULL is returned if any of the input parameters is invalid.

Examples

In the following examples, the input of sub_bitmap() is the output of bitmap_from_string(). For example, `bitmap_from_string('1,1,3,1,5,3,5,7,7,9')` returns `1, 3, 5, 7, 9`. sub_bitmap() takes this BITMAP value as the input. For more information about bitmap_from_string(), see bitmap_from_string.

Example 1: Obtain two elements from the BITMAP value with the offset set to 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   |
+-------+
```

Example 2: Obtain 100 elements from the BITMAP value with the offset set to 0. 100 exceeds the length of the BITMAP value and all the matching elements are returned.

```
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 |
+-----------+
```

Example 3: Obtain 100 elements from the BITMAP value with the offset set to -3. 100 exceeds the length of the BITMAP value and all the matching elements are returned.

```
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 |
+-------+
```

Example 4: Obtain two elements from the BITMAP value with the offset set to -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   |
+-------+
```

Example 5: NULL is returned because `-10` is an invalid input of `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  |
+-------+
```

Example 6: The starting position specified by offset 5 exceeds the length of the BITMAP value `1,3,5,7,9`. NULL is returned.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'),
5, 1)) value;
+-------+
| value |
+-------+
| NULL  |
+-------+
```

Example 7: `len` is set to 5 but only two elements match the condition. All of these two elements are returned.

```
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

Description

Splits a large bitmap into multiple sub-bitmaps.

This function is mainly used to export bitmaps. Bitmaps that are too large will exceed the maximum packet size allowed in the MySQL protocol.

This function is supported from v2.5.

Syntax

```
BITMAP subdivide_bitmap(bitmap, length)
```

Parameters

`bitmap`: The bitmap that needs to be split, required. `length`: The maximum length of each sub-bitmap, required. Bitmaps larger than this value will be split into multiple small bitmaps.

Return value

Returns multiple sub-bitmaps not larger than `length`.

Examples

Suppose there is a table `t1`, in which the `c2` column is a BITMAP column.

```
-- Use bitmap_to_string() to convert values in `c2` into a string.
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 |
+------+----------------------+

-- Split `c2` into small bitmaps whose maximum length is 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

Description

The input is unsigned bigint with the value ranging from 0 to 18446744073709551615, and the output is bitmap containing this element. This function is mainly used for the stream load task to import integer fields into the bitmap field of the StarRocks table. For example:

```
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
```

Syntax

```
BITMAP TO_BITMAP(expr)
```

Examples

```
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

Overview of JSON functions and operators

This topic provides an overview of the JSON constructor functions, query functions, and processing functions, operators, and path expressions that are supported by StarRocks.

JSON constructor functions

JSON constructor functions are used to construct JSON data, such as JSON objects and JSON arrays.

Function	Description	Example	Return value
json_object	Converts one or more key-value pairs to a JSON object that consists of the key-value pairs, which are sorted by key in dictionary order.	`SELECT JSON_OBJECT('Daniel Smith', 26, 'Lily Smith', 25);`	`{"Daniel Smith": 26, "Lily Smith": 25}`
json_array	Converts each element of an SQL array to a JSON value and returns a JSON array that consists of those JSON values.	`SELECT JSON_ARRAY(1, 2, 3);`	`[1,2,3]`
parse_json	Converts a string to a JSON value.	`SELECT PARSE_JSON('{"a": 1}');`	`{"a": 1}`

JSON query functions and processing functions

JSON query functions and processing functions are used to query and process JSON data. For example, you can use a path expression to locate an element in a JSON object.

Function	Description	Example	Return value
arrow function	Queries the element that can be located by a path expression in a JSON object.	`SELECT parse_json('{"a": {"b": 1}}') -> '$.a.b';`	`1`
cast	Converts data between a JSON data type and an SQL data type.	`SELECT CAST(1 AS JSON);`	`1`
get_json_double	Analyzes and gets the floating point value from a specified path in a JSON string.	`SELECT get_json_double('{"k1":1.3, "k2":"2"}', "$.k1");`	`1.3`

Function	Description	Example	Return value
get_json_int	Analyzes and gets the integer value from a specified path in a JSON string.	`SELECT get_json_int('{"k1":1, "k2":"2"}', "$.k1");`	`1`
get_json_string	Analyzes and gets the strings from a specified path in a JSON string.	`SELECT get_json_string('{"k1":"v1", "k2":"v2"}', "$.k1");`	`v1`
json_query	Queries the value of an element that can be located by a path expression in a JSON object.	`SELECT JSON_QUERY('{"a": 1}', '$.a');`	`1`
json_each	Expands the top-level elements of a JSON object into key-value pairs.	`SELECT * FROM tj_test, LATERAL JSON_EACH(j);`	`!json_ea`

Function	Description	Example	Return value
json_exists	Checks whether a JSON object contains an element that can be located by a path expression. If the element exists, this function returns 1. If the element does not exist, the function returns 0.	`SELECT JSON_EXISTS('{"a": 1}', '$.a');`	`1`
json_keys	Returns the top-level keys from a JSON object as a JSON array, or, if a path is specified, the top-level keys from the path.	`SELECT JSON_KEYS('{"a": 1, "b": 2, "c": 3}');`	`["a", "b", "c"`

Function	Description	Example	Return value
json_length	Returns the length of a JSON document.	`SELECT json_length('{"Name": "Alice"}');`	`1`
json_string	Converts the JSON object to a JSON string	`SELECT json_string(parse_json('{"Name": "Alice"}'));`	`{"Name": "Alice"}`

JSON operators

StarRocks supports the following JSON comparison operators: `<`, `<=`, `>`, `>=`, `=`, and `!=`. You can use these operators to query JSON data. However, it does not allow you to use `IN` to query JSON data. For more information about JSON operators, see JSON operators.

JSON path expressions

You can use a JSON path expression to query an element in a JSON object. JSON path expressions are of the STRING data type. In most cases, they are used with various JSON functions, such as JSON_QUERY. In StarRocks, JSON path expressions do not completely comply with the SQL/JSON path specifications. For information about the JSON path syntax that is supported in StarRocks, see the following table, in which the following JSON object is used as an example.

```
{
    "people": [{
        "name": "Daniel",
        "surname": "Smith"
    }, {
        "name": "Lily",
        "surname": "Smith",
        "active": true
```

```
    }]
}
```

JSON path symbol	Description	JSON path example	Return value
`$`	Denotes a root JSON object.	`'$'`	`{ "people": [ { "name": "Daniel", "surname": "Smith" }, { "name": "Lily", "surname": Smith, "active": true } ] }`
`.`	Denotes a child JSON object.	`' $.people'`	`[ { "name": "Daniel", "surname": "Smith" }, { "name": "Lily", "surname": Smith, "active": true } ]`
`[]`	Denotes one or more array indexes. `[n]` denotes the nth element in an array. Indexes start from 0. StarRocks 2.5 supports querying multi-dimensional arrays, for example, `["Lucy", "Daniel"], ["James", "Smith"]`.	`'$.people [0]'`	`{ "name": "Daniel", "surname": "Smith" }`

JSON path symbol	Description	JSON path example	Return value
	To query the "Lucy" element, you can use `$.people[0][0]`.		
`[*]`	Denotes all elements in an array.	`'$.people[*].name'`	`["Daniel", "Lily"]`
`[start: end]`	Denotes a subset of elements from an array. The subset is specified by the `[start, end]` interval, which excludes the element that is denoted by the end index.	`'$.people[0: 1].name'`	`["Daniel"]`

JSON operators

StarRocks supports the following JSON comparison operators: `<`, `<=`, `>`, `>=`, `=`, and `!=`. You can use these operators to query JSON data. However, StarRocks does not allow you to use `IN` to query JSON data.

- > The operands of an operator must both be JSON values.
- > If one operand of an operator is a JSON value while the other is not, the operand that is not a JSON value is converted to a JSON value during the arithmetic operation. For more information about the conversion rules, see CAST.

Arithmetic rules

JSON operators comply with the following arithmetic rules:

- When the operands of an operator are JSON values of the same data type:
  - If both operands are JSONs values of a basic data type, such as NUMBER, STRING, or BOOLEAN, the operator performs the arithmetic operation in compliance with the arithmetic rules for the basic data type.

> Note: If both operands are numbers but one is a DOUBLE value and the other is an INT value, the operator converts the INT value to a DOUBLE value.

- If both operands are JSON values of a composite data type, such as OBJECT or ARRAY, the operator sorts the keys in the operands in dictionary order based on the sequence of the keys in the first operand and then compares the values of the keys between the operands.

Example 1:

The first operand is `{"a": 1, "c": 2}`, and the second operand is `{"b": 1, "a": 2}`. In this example, the operator compares the values of the keys `a` between the operands. The value of the key `a` in the first operand is `1`, whereas the value of the key `a` in the second operand is `2`. The value `1` is greater than the value `2`. Therefore, the operator concludes

that the first operand `{"a": 1, "c": 2}` is less than the second operand `{"b": 1, "``a``": 2}`.

```
mysql> SELECT PARSE_JSON('{"a": 1, "c": 2}') < PARSE_JSON('{"b": 1, "a": 2} ');

       -> 1
```

Example 2:

The first operand is `{"a": 1, "c": 2}`, and the second operand is `{"b": 1, "a": 1}`. In this example, the operator first compares the values of the keys `a` between the operands. The values of the keys `a` in the operands are both `1`. Then, the operator compares the values of the keys `c` in the operands. The second operand does not contain the key `c`. Therefore, the operator concludes that the first operand `{"a": 1, "c": 2}` is greater than the second operand `{"b": 1, "a": 1}`.

```
mysql> SELECT PARSE_JSON('{"a": 1, "c": 2}') < PARSE_JSON('{"b": 1, "a": 1}');

       -> 0
```

- When the operands of an operator are JSON values of two distinct data types, the operator compares the operands in compliance with the following arithmetic rules: NULL < BOOLEAN < ARRAY < OBJECT < DOUBLE < INT < STRING.

```
mysql> SELECT PARSE_JSON('"a"') < PARSE_JSON('{"a": 1, "c": 2}');

       -> 0
```

json_array

Description

json_object

Description

parse_json

Description

json_array

Description

Converts each element of an SQL array to a JSON value and returns a JSON array that consists of the JSON values.

Syntax

```
json_array(value, ...)
```

Parameters

`value`: an element in the SQL array. Only `NULL` values and the following data types are supported: STRING, VARCHAR, CHAR, JSON, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, FLOAT, and BOOLEAN.

Return value

Returns a JSON array.

Examples

Example 1: Construct a JSON array that consists of values of different data types.

```
mysql> SELECT json_array(1, true, 'starrocks', 1.1);

       -> [1, true, "starrocks", 1.1]
```

Example 2: Construct an empty JSON array.

```
mysql> SELECT json_array();

       -> []
```

json_object

Description

Converts one or more key-value pairs to a JSON object that consists of the key-value pairs. The key-value pairs are sorted by key in dictionary order.

Syntax

```
json_object(key, value, ...)
```

Parameters

- `key`: a key in the JSON object. Only the VARCHAR data type is supported.
- `value`: a value in the JSON object. Only `NULL` values and the following data types are supported: STRING, VARCHAR, CHAR, JSON, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, FLOAT, and BOOLEAN.

Return value

Returns a JSON object.

> If the total number of keys and values is an odd number, the JSON_OBJECT function fills `NULL` in the last field.

Examples

Example 1: Construct a JSON object that consists of values of different data types.

```
mysql> SELECT json_object('name', 'starrocks', 'active', true,
'published', 2020);

       -> {"active": true, "name": "starrocks", "published": 2020}
```

Example 2: Construct a JSON object by using nested JSON_OBJECT functions.

```
mysql> SELECT json_object('k1', 1, 'k2', json_object('k2', 2), 'k3',
json_array(4, 5));

       -> {"k1": 1, "k2": {"k2": 2}, "k3": [4, 5]}
```

Example 3: Construct an empty JSON object.

```
mysql> SELECT json_object();

       -> {}
```

parse_json

Description

Converts a string to a JSON value.

Syntax

```
parse_json(string_expr)
```

Parameters

`string_expr`: the expression that represents the string. Only the STRING, VARCHAR, and CHAR data types are supported.

Return value

Returns a JSON value.

> Note: If the string cannot be parsed into a standard JSON value, the PARSE_JSON function returns `NULL` (see Example 5). For information about the JSON specifications, see RFC 7159.

Examples

Example 1: Convert a STRING value of `1` to a JSON value of `1`.

```
mysql> SELECT parse_json('1');
+-----------------+
| parse_json('1') |
```

```
+-----------------+
| "1"             |
+-----------------+
```

Example 2: Convert an array of the STRING data type to a JSON array.

```
mysql> SELECT parse_json('[1,2,3]');
+----------------------+
| parse_json('[1,2,3]') |
+----------------------+
| [1, 2, 3]             |
+----------------------+
```

Example 3: Convert an object of the STRING data type to a JSON object.

```
mysql> SELECT parse_json('{"star": "rocks"}');
+--------------------------------+
| parse_json('{"star": "rocks"}') |
+--------------------------------+
| {"star": "rocks"}               |
+--------------------------------+
```

Example 4: Construct a JSON value of `NULL`.

```
mysql> SELECT parse_json('null');
+-------------------+
| parse_json('null') |
+-------------------+
| "null"             |
+-------------------+
```

Example 5: If the string cannot be parsed into a standard JSON value, the PARSE_JSON function returns `NULL`. In this example, `star` is not enclosed in double quotation marks ("). Therefore, the PARSE_JSON function returns `NULL`.

```
mysql> SELECT parse_json('{star: "rocks"}');
+-----------------------------+
| parse_json('{star: "rocks"}') |
+-----------------------------+
```

```
| NULL                          |
+-------------------------------+
```

Example 6: If a JSON key contains a '.', for example, 'a.1', it must be escaped with '\' or you need to enclose the entire key value along with double quotes in single quotes.

```
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"                                       |
+-------------------------------------------+
```

Keywords

parse_json, parse json

Arrow function

Description

cast

Description

get_json_double

Description

get_json_int

Description

get_json_string,get_json_object

Description

json_each

Description

json_exists

Description

json_keys

Description

json_length

Description

json_query

Description

json_string

Description

Arrow function

Description

Queries an element that can be located by the `json_path` expression in a JSON object and returns a JSON value. The arrow function `->` is more compact and easier to use than the json_query function.

Syntax

```
json_object_expr -> json_path
```

Parameters

- `json_object_expr`: the expression that represents the JSON object. The object can be a JSON column, or a JSON object that is produced by a JSON constructor function such as PARSE_JSON.
- `json_path`: the expression that represents the path to an element in the JSON object. The value of this parameter is a string. For information about the JSON path syntax that is supported by StarRocks, see Overview of JSON functions and operators.

Return value

Returns a JSON value.

> If the element does not exist, the arrow function returns an SQL value of `NULL`.

Examples

Example 1: Query an element that can be located by the `'$.a.b'` expression in the specified JSON object.

```
mysql> SELECT parse_json('{"a": {"b": 1}}') -> '$.a.b';

       -> 1
```

Example 2: Use nested arrow functions to query an element. The arrow function in which another arrow function is nested queries an element based on the result that is returned by the nested arrow function.

> In this example, the root element $ is omitted from the `json_path` expression.

```
mysql> SELECT parse_json('{"a": {"b": 1}}')->'a'->'b';

       -> 1
```

Example 3: Query an element that can be located by the `'a'` expression in the specified JSON object.

> In this example, the root element $ is omitted from the `json_path` expression.

```
mysql> SELECT parse_json('{"a": "b"}') -> 'a';

       -> "b"
```

cast

Description

Converts a value between the JSON type and SQL type.

Syntax

- Conversion from JSON to SQL

```
cast(json_expr AS sql_data_type)
```

- Conversion from SQL to JSON

```
cast(sql_expr AS JSON)
```

Parameters

- `json_expr`: the expression that represents the JSON value you want to convert to an SQL value.
- `sql_data_type`: the SQL data type to which you want to convert the JSON value. Only the STRING, VARCHAR, CHAR, BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, and FLOAT data types are supported.
- `sql_expr`: the expression that represents the SQL value you want to convert to a JSON value. This parameter supports all SQL data types that are supported by the `sql_data_type` parameter.

Return value

- If you use the `cast(json_expr AS sql_data_type)` syntax, the cast function returns a value of the SQL data type that is specified by the `sql_data_type` parameter.
- If you use the `cast(sql_expr AS JSON)` syntax, the cast function returns a JSON value.

Usage notes

- Conversion from SQL to JSON
  - If the SQL value exceeds the precision that is supported by JSON, the cast function returns `NULL` to prevent an arithmetic overflow.
  - If the SQL value is `NULL`, the cast function does not convert the SQL value `NULL` to a JSON value of `NULL`. The return value is still an SQL value of `NULL`.
- Conversion from JSON to SQL
  - The cast function supports only conversions between compatible JSON and SQL data types. For example, you can convert a JSON string to an SQL string.
  - The cast function does not support conversions between incompatible JSON and SQL data types. For example, if you convert a JSON number to an SQL string, the function returns `NULL`.
  - If an arithmetic overflow occurs, the cast function returns an SQL value of `NULL`.
  - If you convert a JSON value of `NULL` to an SQL value, the function returns an SQL value of `NULL`.
  - If you convert a JSON string to a VARCHAR value, the function returns a VARCHAR value that is not enclosed in double quotation marks (").

Examples

Example 1: Convert a JSON value to an SQL value.

```
-- Convert a JSON value to an INT value.
mysql> select cast(parse_json('{"a": 1}') -> 'a' as int);
+-------------------------------------------+
| CAST((parse_json('{"a": 1}')->'a') AS INT) |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+

-- Convert a JSON string to a VARCHAR value.
mysql> select cast(parse_json('"star"') as varchar);
+--------------------------------------+
| cast(parse_json('"star"') AS VARCHAR) |
+--------------------------------------+
| star                                 |
+--------------------------------------+

-- Convert a JSON object to a VARCHAR value.
mysql> select cast(parse_json('{"star": 1}') as varchar);
+-----------------------------------------+
| cast(parse_json('{"star": 1}') AS VARCHAR) |
+-----------------------------------------+
| {"star": 1}                             |
+-----------------------------------------+

-- Convert a JSON array to a VARCHAR value.

mysql> select cast(parse_json('[1,2,3]') as varchar);
+--------------------------------------+
| cast(parse_json('[1,2,3]') AS VARCHAR) |
+--------------------------------------+
| [1, 2, 3]                            |
+--------------------------------------+
```

Example 2: Convert an SQL value to a JSON value.

```
-- Convert an INT value to a JSON value.
mysql> select cast(1 as json);
+----------------+
| cast(1 AS JSON) |
+----------------+
| 1              |
+----------------+

-- Convert a VARCHAR value to a JSON value.
```

```
mysql> select cast("star" as json);
+----------------------+
| cast('star' AS JSON) |
+----------------------+
| "star"               |
+----------------------+

-- Convert a BOOLEAN value to a JSON value.
mysql> select cast(true as json);
+--------------------+
| cast(TRUE AS JSON) |
+--------------------+
| true               |
+--------------------+
```

get_json_double

Description

This function analyzes and gets the floating point value from a specified path in json string. json_path must start with `$` and use `.` as the path separator.

If the path includes `.`, it could be circled by `"` and `"`.

`[ ]` is used as the array subscripts which start from 0.

Content in the path should not contain `"` , `[` and `]`.

If the format of json_string or json_path is wrong, this function will return NULL.

Syntax

```
DOUBLE get_json_double(VARCHAR json_str, VARCHAR json_path)
```

Examples

1. Get the value whose key is "k1"

```
MySQL > SELECT get_json_double('{"k1":1.3, "k2":"2"}', "$.k1");
+-----------------------------------------------+
| get_json_double('{"k1":1.3, "k2":"2"}', '$.k1') |
+-----------------------------------------------+
|                                           1.3 |
+-----------------------------------------------+
```

2. Get the second element in the array whose key is "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. Get the first element in the array whose path is 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 |
+-----------------------------------------------------------------
-----+
```

keyword

GET_JSON_DOUBLE,GET,JSON,DOUBLE

get_json_int

Description

This function analyzes and gets the integer value from a specified path in json string.

json_path must start with `$` and use `.` as the path separator.

If the path includes `.`, it could be circled by `"` and `"`.

`[ ]` is used as the array subscripts which start from 0.

Content in the path should not contain `"` , `[` and `]`.

If the format of json_string or json_path is wrong, this function will return NULL.

Syntax

```
INT get_json_int(VARCHAR json_str, VARCHAR json_path)
```

Examples

1. Get the value whose key is "k1"

```
MySQL > SELECT get_json_int('{"k1":1, "k2":"2"}', "$.k1");
+-----------------------------------------+
| get_json_int('{"k1":1, "k2":"2"}', '$.k1') |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

2. Get the second element in the array whose key is "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. Get the first element in the array whose path is 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 |
+-----------------------------------------------------------+
```

keyword

GET_JSON_INT,GET,JSON,INT

get_json_string,get_json_object

Description

Analyzes and gets strings from the specified path (`json_path`) in a JSON string. If the format of `json_string` or `json_path` is wrong or if no matching value is found, this function will return NULL.

The alias is get_json_object.

Syntax

```
VARCHAR get_json_string(VARCHAR json_str, VARCHAR json_path)
```

Parameters

- `json_str`: the JSON string. Supported data type is VARCHAR.
- `json_path`: the JSON path. Supported data type is VARCHAR. `json_path` starts with `$` and uses `.` as the path separator. `[ ]` is used as the array subscripts which starts from 0. For example, `$."my.key"[1]` indicates to obtain the second value form element `my.key`.

Return value

Returns a value of the VARCHAR type. If no matching object is found, NULL is returned.

Examples

Example 1: Get the value whose key is `k1`.

```
MySQL > SELECT get_json_string('{"k1":"v1", "k2":"v2"}', "$.k1");
+-----------------------------------------------+
| get_json_string('{"k1":"v1", "k2":"v2"}', '$.k1') |
+-----------------------------------------------+
| v1                                            |
+-----------------------------------------------+
```

Example 2: Get the value whose key is `a` from the first element.

```
MySQL > 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
|
+-------------------------------------------------------------------
----------+
```

Example 3: Get the second element in the array whose key is `my.key`

```
MySQL > 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
|
+-------------------------------------------------------------------
----------+
```

Example 4: Get the first element in the array whose path is `k1.key -> k2`.

```
MySQL > 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
|
+-----------------------------------------------------------------
---+
```

Example 5: Get all values whose key is `k1` from the array.

```
MySQL > 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"]
|
+-----------------------------------------------------------------
------------+
```

keyword

GET_JSON_STRING,GET,JSON,STRING

json_each

Description

Expands the outermost elements of a JSON object into a set of key-value pairs held in two columns and returns a table that consists of one row for each element.

Syntax

```
json_each(json_object_expr)
```

Parameters

`json_object_expr`: the expression that represents the JSON object. The object can be a JSON column, or a JSON object that is produced by a JSON constructor function such as PARSE_JSON.

Return value

Returns two columns: one named key and one named value. The key column stores VARCHAR values, and the value column stores JSON values.

Usage notes

The json_each function is a table function that returns a table. The returned table is a result set that consists of multiple rows. Therefore, a lateral join must be used in the FROM clause to join the returned table to the original table. The lateral join is mandatory, but the LATERAL keyword is optional. The json_each function cannot be used in the SELECT clause.

Examples

```
-- A table named tj is used as an example. In the tj table, the j
column is a JSON object.
mysql> SELECT * FROM tj;
+------+------------------+
| id   | j                |
+------+------------------+
|    1 | {"a": 1, "b": 2} |
|    3 | {"a": 3}         |
+------+------------------+

-- Expand the j column of the tj table into two columns by key and
value to obtain a result set that consists of multiple rows. In this
example, the LATERAL keyword is used to join the result set to the tj
table.

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

Description

Checks whether a JSON object contains an element that can be located by the `json_path` expression. If the element exists, the JSON_EXISTS function returns `1`. Otherwise, the JSON_EXISTS function returns `0`.

Syntax

```
json_exists(json_object_expr, json_path)
```

Parameters

- `json_object_expr`: the expression that represents the JSON object. The object can be a JSON column, or a JSON object that is produced by a JSON constructor function such as PARSE_JSON.
- `json_path`: the expression that represents the path to an element in the JSON object. The value of this parameter is a string. For more information about the JSON path syntax that is supported by StarRocks, see Overview of JSON functions and operators.

Return value

Returns a BOOLEAN value.

Examples

Example 1: Check whether the specified JSON object contains an element that can be located by the `'$.a.b'` expression. In this example, the element exists in the JSON object. Therefore, the json_exists function returns `1`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": {"b": 1}}'), '$.a.b') ;

       -> 1
```

Example 2: Check whether the specified JSON object contains an element that can be located by the `'$.a.c'` expression. In this example, the element does not exist in the JSON object. Therefore, the json_exists function returns `0`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": {"b": 1}}'), '$.a.c') ;

       -> 0
```

Example 3: Check whether the specified JSON object contains an element that can be located by the `'$.a[2]'` expression. In this example, the JSON object, which is an array named a, contains an element at index 2. Therefore, the json_exists function returns `1`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": [1,2,3]}'), '$.a[2]') ;

       -> 1
```

Example 4: Check whether the specified JSON object contains an element that can be located by the `'$.a[3]'` expression. In this example, the JSON object, which is an array named a, does not contain an element at index 3. Therefore, the json_exists function returns `0`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": [1,2,3]}'), '$.a[3]') ;

       -> 0
```

json_keys

Description

Returns the top-level keys from a JSON object as a JSON array, or, if a `path` is specified, the top-level keys from the path.

Syntax

```
json_keys(json_doc[, path])
```

Parameters

`json_doc`: required. The JSON document for which to return the keys. It must be a JSON object.

`path`: optional. The path generally starts with `$` and uses `.` as the path separator. `[]` is used as the array subscript, which starts from 0.

Return value

Returns a JSON array.

An empty array is returned if the JSON object is empty.

`NULL` is returned if the JSON document is not a JSON object or the path does not identify a value in the document.

If the JSON document is an array nested with a JSON object, you can use the `path` parameter to obtain keys from that object.

Examples

Example 1: Return an empty array because the input JSON object is empty.

```
select json_keys('{}');
+-----------------+
| json_keys('{}') |
+-----------------+
| []              |
+-----------------+
```

Example 2: Return the keys of a JSON object.

```
select json_keys('{"a": 1, "b": 2, "c": 3}');
+---------------+
| json_keys('1') |
+---------------+
|["a", "b", "c"] |
+---------------+
```

Example 3: Return the keys of a JSON object that matches the specified path.

```
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"]
|
+-------------------------------------------------------------------
-+
```

Example 4: The path does not exist.

```
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
|
+-------------------------------------------------------------------
-+
```

Example 5: The JSON document is not a JSON object.

```
select json_keys('[1, 2, {"a": 1, "b": 2}]');
+---------------------------------------+
| json_keys('[1, 2, {"a": 1, "b": 2}]') |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
```

Example 6: The JSON document is an array nested with a JSON object. A path is specified to obtain keys from that object.

```
select json_keys('[0, 1, {"a": 1, "b": 2}]', '$[2]');
+-----------------------------------------------+
| json_keys('[0, 1, {"a": 1, "b": 2}]', '$[2]') |
+-----------------------------------------------+
| ["a", "b"]                                    |
+-----------------------------------------------+
```

json_length

Description

Returns the length of a JSON document. If a path is specified, this function returns the length of the value identified by the path.

The length of a document is determined according to the following rules:

- The length of a scalar value is 1. For example, the length of `1`, `"a"`, `true`, `false`, and `null` is 1.
- The length of an array is the number of array elements. For example, the length of `[1, 2]` is 2.
- The length of an object is the number of object members. For example, the length of `{"a": 1}` is 1.
- The length of nested arrays or objects does not count. For example, the length of `{"a": [1, 2]}` is 1, because the nested array `[1, 2]` is not calculated into length.

Syntax

```
json_length(json_doc[, path])
```

Parameters

`json_doc`: required, the JSON document for which to return the length.

`path`: optional. It is used to return the length of a value within the document. The path generally starts with `$` and uses `.` as the path separator. `[]` is used as the array subscript, which starts from 0.

Return value

Returns a value of the INT type.

An error is returned if the JSON document is not a valid document.

0 is returned in any of the following scenarios:

- The **`path`** does not identify a value in the document.
- The path is not a valid path expression.
- The path contains the `*` or `**` wildcard.

Examples

Example 1: Return the length of a scalar value.

```
select json_length('1');
+------------------+
| json_length('1') |
+------------------+
|                1 |
+------------------+
```

Example 2: Return the length of an empty object.

```
select json_length('{}');
+--------------------+
| json_length('{}') |
+--------------------+
|                  0 |
+--------------------+
```

Example 3: Return the length of an object that has data.

```
select json_length('{"Name": "Homer"}');
+---------------------------------+
| json_length('{"Name": "Homer"}') |
```

```
+----------------------------------+
|                                1 |
+----------------------------------+
```

Example 4: Return the length of a JSON array.

```
select json_length('[1, 2, 3]');
+--------------------------+
| json_length('[1, 2, 3]') |
+--------------------------+
|                        3 |
+--------------------------+
```

Example 5: Return the length of a JSON array in which one element has a nested array.

The nested array `[3, 4]` is not calculated into length.

```
select json_length('[1, 2, [3, 4]]');
+-----------------------------+
| json_length('[1, 2, [3, 4]]') |
+-----------------------------+
|                           3 |
+-----------------------------+
```

Example 6: Return the length of an object specified by path `$.Person`.

```
SET @file = '{
    "Person": {
       "Name": "Homer",
       "Age": 39,
       "Hobbies": ["Eating", "Sleeping"]
    }
 }';
select json_length(@file, '$.Person') 'Result';
```

Example 7: Return the length of the value specified by path `$.y`.

```
select json_length('{"x": 1, "y": [1, 2]}', '$.y');
+------------------------------------------+
| json_length('{"x": 1, "y": [1, 2]}', '$.y') |
```


2

json_query

Description

Queries the value of an element that can be located by the `json_path` expression in a JSON object and returns a JSON value.

Syntax

```
json_query(json_object_expr, json_path)
```

Parameters

- `json_object_expr`: the expression that represents the JSON object. The object can be a JSON column, or a JSON object that is produced by a JSON constructor function such as PARSE_JSON.
- `json_path`: the expression that represents the path to an element in the JSON object. The value of this parameter is a string. For information about the JSON path syntax that is supported by StarRocks, see Overview of JSON functions and operators.

Return value

Returns a JSON value.

> If the element does not exist, the json_query function returns an SQL value of `NULL`.

Examples

Example 1: Query the value of an element that can be located by the `'$.a.b'` expression in the specified JSON object. In this example, the json_query function returns a JSON value of `1`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": {"b": 1}}'), '$.a.b') ;

       -> 1
```

Example 2: Query the value of an element that can be located by the `'$.a.c'` expression in the specified JSON object. In this example, the element does not exist. Therefore, the json_query function returns an SQL value of `NULL`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": {"b": 1}}'), '$.a.c') ;

       -> NULL
```

Example 3: Query the value of an element that can be located by the `'$.a[2]'` expression in the specified JSON object. In this example, the JSON object, which is an array named a, contains an element at index 2, and the value of the element is 3. Therefore, the JSON_QUERY function returns a JSON value of `3`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": [1,2,3]}'), '$.a[2]') ;

       -> 3
```

Example 4: Query an element that can be located by the `'$.a[3]'` expression in the specified JSON object. In this example, the JSON object, which is an array named a, does not contain an element at index 3. Therefore, the json_query function returns an SQL value of `NULL`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": [1,2,3]}'), '$.a[3]') ;

       -> NULL
```

json_string

Description

Converting JSON object to JSON string

Syntax

```
json_string(json_object_expr)
```

Parameters

- `json_object_expr`: the expression that represents the JSON object. The object can be a JSON column, or a JSON object that is produced by a JSON constructor function such as PARSE_JSON.

Return value

Returns a VARCHAR value.

Examples

Example 1: Converting the JSON object to a JSON string

```
select json_string('{"Name": "Alice"}');
+-----------------------------------+
| json_string('{"Name": "Alice"}') |
+-----------------------------------+
```

```
| {"Name": "Alice"}                |
+----------------------------------+
```

Example 1: Convert the result of PARSE_JSON to a JSON string

```
select json_string(parse_json('{"Name": "Alice"}'));
+----------------------------------+
| json_string('{"Name": "Alice"}') |
+----------------------------------+
| {"Name": "Alice"}                |
+----------------------------------+
```

Map functions

map_keys

Description

map_size

Description

map_values

Description

map_keys

Description

Returns an array of all the keys in the specified map.

From version 2.5, StarRocks supports querying complex data types MAP and STRUCT from data lakes. MAP is an unordered collection of key-value pairs, for example, `{"a":1, "b":2}`.

You can use external catalogs provided by StarRocks to query MAP and STRUCT data from Apache Hive™, Apache Hudi, and Apache Iceberg. You can only query data from ORC and Parquet files. For more information about how to use external catalogs to query external data sources, see Overview of catalogs and topics related to the required catalog type.

Syntax

```
map_keys(any_map)
```

Parameters

`any_map`: the MAP value from which you want to retrieve keys.

Return value

The return value is in the format of `array<keyType>`. The element type in the array matches the key type in the map.

If the input is NULL, NULL is returned. If a key or value in the MAP value is NULL, NULL is processed as a normal value and contained in the result.

Examples

This example uses the Hive table `hive_map`, which contains the following data:

```
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} |
+---------+---------------+
3 rows in set (0.05 sec)
```

After a Hive catalog is created in your database, you can use this catalog and the map_keys() function to obtain all the keys from each row of the `col_map` column.

```
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

Description

Returns the number of elements in a MAP value.

From version 2.5, StarRocks supports querying complex data types MAP and STRUCT from data lakes. MAP is an unordered collection of key-value pairs, for example, `{"a":1, "b":2}`. One key-value pair constitutes one element, for example, `{"a":1, "b":2}` contains two elements.

You can use external catalogs provided by StarRocks to query MAP and STRUCT data from Apache Hive™, Apache Hudi, and Apache Iceberg. You can only query data from ORC and Parquet files. For more information about how to use external catalogs to query external data sources, see Overview of catalogs and topics related to the required catalog type.

Syntax

```
INT map_size(any_map)
```

Parameters

`any_map`: the MAP value from which you want to retrieve the number of elements.

Return value

Returns a value of the INT value.

If the input is NULL, NULL is returned.

If a key or value in the MAP value is NULL, NULL is processed as a normal value.

Examples

This example uses the Hive table `hive_map`, which contains the following data:

```
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} |
+---------+---------------+
3 rows in set (0.05 sec)
```

After a Hive catalog is created in your database, you can use this catalog and the map_size() function to obtain the number of elements in each row of the `col_map` column.

```
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

Description

Returns an array of all the values in the specified map.

From version 2.5, StarRocks supports querying complex data types MAP and STRUCT from data lakes. MAP is an unordered collection of key-value pairs, for example, `{"a":1, "b":2}`.

You can use external catalogs provided by StarRocks to query MAP and STRUCT data from Apache Hive™, Apache Hudi, and Apache Iceberg. You can only query data from ORC and Parquet files. For more information about how to use external catalogs to query external data sources, see Overview of catalogs and topics related to the required catalog type.

Syntax

```
map_values(any_map)
```

Parameters

`any_map`: the MAP value from which you want to retrieve values.

Return value

The return value is in the format of `array<valueType>`. The element type in the array matches the value type in the map.

If the input is NULL, NULL is returned. If a key or value in the MAP value is NULL, NULL is processed as a normal value and contained in the result.

Examples

This example uses the Hive table `hive_map`, which contains the following data:

```
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} |
+---------+---------------+
3 rows in set (0.05 sec)
```

After a Hive catalog is created in your database, you can use this catalog and the map_values() function to obtain all the values from each row of the `col_map` column.

```
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)
```

case

Description

coalesce

Description

if

Description

ifnull

Description

nullif

Description

case

Description

CASE is a conditional expression. It returns the result in the THEN clause if a condition in the WHEN clause evaluates to true. If none of the conditions evaluate to true, it returns the result in the optional ELSE clause. If ELSE is not present, NULL is returned.

Syntax

The CASE expression comes in two forms:

- Simple CASE

```
CASE expression
    WHEN expression1 THEN result1
    [WHEN expression2 THEN result2]
    ...
    [WHEN expressionN THEN resultN]
    [ELSE result]
END
```

For this syntax, `expression` is compared to each expression in the WHEN clause. If an equal expression is found, the result in the THEN clause is returned. If no equal expression is found, the result in the ELSE clause is returned if ELSE is present.

- Searched CASE

```
CASE WHEN condition1 THEN result1
    [WHEN condition2 THEN result2]
    ...
    [WHEN conditionN THEN resultN]
    [ELSE result]
END
```

For this syntax, each condition in the WHEN clause is evaluated until one is true and the corresponding result in the THEN clause is returned. If no condition evaluates to true, the result in the ELSE clause is returned, if ELSE is present.

The first CASE equals the second one as follows:

```
CASE WHEN expression = expression1 THEN result1
    [WHEN expression = expression2 THEN result2]
    ...
    [WHEN expression = expressionN THEN resultN]
    [ELSE result]
END
```

Parameters

- `expressionN`: the expression to compare. Multiple expressions must be compatible in data types. This expression must evaluate to any of the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATETIME, DATE, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128, VARCHAR, BITMAP, PERCENTILE, HLL, TIME, and JSON.
- `conditionN`: the condition that can evaluate to a BOOLEAN value.
- `resultN` must be compatible in data types.

Return value

The return value is of the common type of all types in the THEN clause.

Examples

Suppose table `test_case` has the following data:

```
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 |
+-------+--------+-------+
```

Use simple CASE

- ELSE is specified and the result in ELSE is returned if no equal expression is found.

```
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 is not specified and NULL is returned if no condition evaluates to true.

```
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     |
+--------+------------+
```

Use searched CASE with no ELSE specified

```
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

Description

Returns the first non-NULL expression among the input parameters. Returns NULL if non-NULL expressions cannot be found.

Syntax

```
coalesce(expr1,...);
```

Parameters

`expr1`: the input expressions, which must evaluate to compatible data types. The supported data types are BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATETIME, DATE, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128, VARCHAR, BITMAP, PERCENTILE, HLL, and TIME.

Return value

The return value has the same type as `expr1`.

Examples

```
mysql> select coalesce(3,NULL,1,1);
+------------------------+
| coalesce(3, NULL, 1, 1) |
+------------------------+
|                      3 |
```

```
+--------------------------+
1 row in set (0.00 sec)
```

if

Description

If `expr1` evaluates to TRUE, returns `expr2`. Otherwise, returns `expr3`.

Syntax

```
if(expr1,expr2,expr3);
```

Parameters

`expr1`: the condition. It must be a BOOLEAN value.

`expr2`: This value is returned if the condition is true. This expression must evaluate to any of the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATETIME, DATE, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128, VARCHAR, BITMAP, PERCENTILE, HLL, TIME.

`expr3`: This value is returned if the condition is false. The data type is the same as `expr2`.

> `expr2` and `expr3` must agree in data type.

Return value

The return value has the same type as `expr2`.

Examples

```
mysql> select if(false,1,2);
+-----------------+
```

```
| if(FALSE, 1, 2) |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.00 sec)
```

ifnull

Description

If `expr1` is NULL, returns expr2. If `expr1` is not NULL, returns `expr1`.

Syntax

```
ifnull(expr1,expr2);
```

Parameters

`expr1`: This expression must evaluate to any of the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATETIME, DATE, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128, VARCHAR, BITMAP, PERCENTILE, HLL, TIME.

`expr2`: This expression must evaluate to any of the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATETIME, DATE, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128, VARCHAR, BITMAP, PERCENTILE, HLL, TIME.

> `expr1` and `expr2` must agree in data type.

Return value

The return value has the same type as `expr1`.

Examples

```
mysql> select ifnull(NULL,2);
+-----------------+
| ifnull(NULL, 2) |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.01 sec)
```

nullif

Description

Returns NULL if `expr1` is equal to `expr2`. Otherwise, returns `expr1`.

Syntax

```
nullif(expr1,expr2);
```

Parameters

`expr1`: This expression must evaluate to any of the following data types: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATETIME, DATE, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128, VARCHAR, BITMAP, PERCENTILE, HLL, TIME.

`expr2`: An expression that evaluates to the same data type as `expr1`.

> `expr1` and `expr2` must agree in data type.

Return value

The return value has the same type as `expr1`.

Examples

```
mysql> select nullif(1,2);
+--------------+
| nullif(1, 2) |
+--------------+
```

```
|            1 |
+--------------+

select nullif(1,1);
+--------------+
| nullif(1, 1) |
+--------------+
|         NULL |
+--------------+
```

aes_decrypt

Description

aes_encrypt

Description

from_base64

Description

md5

Uses the MD5 Message-Digest Algorithm to calculate the 128-bit checksum of a string. The check...

md5sum

Calculates the 128-bit checksum of multiple strings. The checksum is represented by a 32-charact...

md5sum_numeric

Calculates the 128-bit checksum of multiple strings. The checksum is represented by a decimal str...

sha2

Calculates the SHA-224 hash value, SHA-256 hash value, SHA-384 hash value, or SHA-512 hash v...

sm3

Description

to_base64

Description

aes_decrypt

Description

Uses the AES_128_ECB algorithm to decrypt a string and returns a binary string.

AES is short for advanced encryption standard and ECB is short for electronic code book. The key used to encrypt the string is a 128-bit string.

Syntax

```
aes_decrypt(str,key_str);
```

Parameters

`str`: the string to decrypt. It must be of the VARCHAR type.

`key_str`: the key used to encrypt `str`. It must be of the VARCHAR type.

Return value

Returns a value of the VARCHAR type. If the input is invalid, NULL is returned.

Examples

Decode a Base64 string and use this function to decrypt the decoded string into the original string.

```
mysql> select AES_DECRYPT(from_base64('uv/Lhzm74syo8JlfWarwKA==
'),'F3229A0B371ED2D9441B830D21A390C3');
+--------------------------------------------------------------
```

```
------------------------+
| aes_decrypt(from_base64('uv/Lhzm74syo8JlfWarwKA==  '),
'F3229A0B371ED2D9441B830D21A390C3') |
+-------------------------------------------------------------------
------------------------+
| starrocks
|
+-------------------------------------------------------------------
------------------------+
1 row in set (0.00 sec)
```

aes_encrypt

Description

Uses the AES_128_ECB algorithm to encrypt a string and returns a binary string.

AES is short for advanced encryption standard and ECB is short for electronic code book. The key used to encrypt the string is a 128-bit string.

Syntax

```
aes_encrypt(str,key_str);
```

Parameters

`str`: the string to encrypt. It must be of the VARCHAR type.

`key_str`: the key used to encrypt `str`. It must be of the VARCHAR type.

Return value

Returns a value of the VARCHAR type. If the input is NULL, NULL is returned.

Examples

Use this function to AES encrypt `starrocks` and convert the encrypted string into a Base64-encoded string.

```
mysql> select
to_base64(AES_ENCRYPT('starrocks','F3229A0B371ED2D9441B830D21A390C3'));
+-----------------------------------------------------------------
```

```
---+
| to_base64(aes_encrypt('starrocks',
'F3229A0B371ED2D9441B830D21A390C3')) |
+-------------------------------------------------------------------------
---+
| uv/Lhzm74syo8JlfWarwKA==
|
+-------------------------------------------------------------------------
---+
1 row in set (0.01 sec)
```

from_base64

Description

Decodes a Base64-encoded string. This function is an inverse of to_base64.

Syntax

```
from_base64(str);
```

Parameters

`str`: the string to decode. It must be of the VARCHAR type.

Return value

Returns a value of the VARCHAR type. If the input is NULL or an invalid Base64 string, NULL is returned. If the input is empty, an error is returned.

This function accepts only one string. More than one input string causes an error.

Examples

```
mysql> select from_base64("starrocks");
+-------------------------+
| from_base64('starrocks') |
+-------------------------+
| ²®$                      |
+-------------------------+
1 row in set (0.00 sec)

mysql> select from_base64('c3RhcnJvY2tz');
```

```
+----------------------------+
| from_base64('c3RhcnJvY2tz') |
+----------------------------+
| starrocks                  |
+----------------------------+
```

md5

Uses the MD5 Message-Digest Algorithm to calculate the 128-bit checksum of a string. The checksum is represented by a 32-character hexadecimal string.

Syntax

```
md5(expr)
```

Parameters

`expr`: the string to calculate. It must be of the VARCHAR type.

Return value

Returns a checksum of the VARCHAR type, which is a 32-character hexadecimal string.

If the input is NULL, NULL is returned.

Examples

```
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)
```

Keywords

MD5, ENCRYPTION

md5sum

Calculates the 128-bit checksum of multiple strings. The checksum is represented by a 32-character hexadecimal string.

If only one string is passed in, the result is the same as that of md5.

Compared to md5(), md5sum() accepts more parameters and therefore, is more efficient in calculating the checksum of multiple files.

Syntax

```
md5sum(expr,...);
```

Parameters

`expr`: the string to calculate. It must be of the VARCHAR type.

Return value

Returns a checksum of the VARCHAR type.

Examples

```
mysql> select md5sum("starrocks");
+----------------------------------+
| md5sum('starrocks')              |
+----------------------------------+
| f75523a916caf65f1ad487a9f8017f75 |
+----------------------------------+
1 row in set (0.01 sec)
```

```
mysql> select md5sum("starrocks","star");
+----------------------------------+
| md5sum('starrocks', '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

Calculates the 128-bit checksum of multiple strings. The checksum is represented by a decimal string of digits.

Syntax

```
LARGEINT md5sum_numeric(VARCHAR expr,...);
```

Parameters

`expr`: the string to calculate. It must be of the VARCHAR type.

Return value

Returns a checksum of the LARGEINT type. If the input is empty, an error is returned.

Examples

```
mysql> select md5sum_numeric("starrocks");
+-----------------------------------------+
| md5sum_numeric('starrocks')             |
+-----------------------------------------+
| 328761383472799310362963866384446095221 |
+-----------------------------------------+
1 row in set (0.01 sec)

mysql> select md5sum_numeric("starrocks","star");
+-----------------------------------------+
| md5sum_numeric('starrocks', '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

Calculates the SHA-224 hash value, SHA-256 hash value, SHA-384 hash value, or SHA-512 hash value of a string.

Syntax

```
VARCHAR sha2(VARCHAR expr, INT hash_length)
```

Parameters

- `Expr`: the string whose value you want to calculate.
- `hash_length`: the length of a hash value. The value of this parameter can be 224, 256, 384, 512, or 0. The value 0 is equivalent to 256. If you set this parameter to any other value, this function returns `NULL`.

Return value

Returns a hash value of the VARCHAR type. If any of the two input parameters is `Null`, `Null` is returned.

Examples

```
mysql> select sha2('abc',224);

+----------------------------------------------------------+

| sha2('abc', 224)                                         |

+----------------------------------------------------------+
```

```
| 23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7 |

+------------------------------------------------------------+

1 row in set (0.01 sec)



mysql> select sha2('abc', 384);

+--------------------------------------------------------------------------------
---+

| sha2('abc', 384)
|

+--------------------------------------------------------------------------------
---+

|
cb00753f45a35e8bb5a03d699ac65007272c32ab0eded1631a8b605a43ff5bed8086072
|

+--------------------------------------------------------------------------------
---+

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)
```

Keywords

SHA2, ENCRYPTION

sm3

Description

Uses the SM3 digest algorithm to encrypt a string into a 256-bit hexadecimal string. Every 32 bits are separated by a space.

Digest algorithms have a broad range of use cases in digital signature, message authentication, and data integrity check. The SM3 algorithm is an enhancement of SHA-256.

Syntax

```
SM3(str);
```

Parameters

`str`: the string to encrypt. It must be of the VARCHAR type.

Return value

Returns a value of the VARCHAR type. If the input is NULL, NULL is returned. If the input is empty, an error is returned.

This function accepts only one string. More than one input string causes an error.

Examples

```
mysql> select sm3('l');
+------------------------------------------------------------------
-----+
```

```
| sm3('l')
|
+-----------------------------------------------------------------
-----+
| 1dec1a7a 94236240 49db411e 2c32c62d c0c93856 8208ac3a 09d395eb
2468b445 |
+-----------------------------------------------------------------
-----+
1 row in set (0.01 sec)
```

to_base64

Description

Converts a string into a Base64-encoded string. This function is an inverse of from_base64.

Syntax

```
to_base64(str);
```

Parameters

`str`: the string to encode. It must be of the VARCHAR type.

Return value

Returns a value of the VARCHAR type. If the input is NULL, NULL is returned. If the input is empty, an error is returned.

This function accepts only one string. More than one input string causes an error.

Examples

```
mysql> select to_base64("starrocks");
+-----------------------+
| to_base64('starrocks') |
+-----------------------+
| c3RhcnJvY2tz           |
+-----------------------+
1 row in set (0.00 sec)

mysql> select to_base64(123);
```

```
+----------------+
| to_base64(123) |
+----------------+
| MTIz           |
+----------------+
```

add_months

Description

adddate,days_add

Description

convert_tz

Description

curdate,current_date

Description

current_timestamp

Description

curtime,current_time

Description

date

Description

date_add

Description

date_format

Description

date_slice

Description

date_sub

Description

date_trunc

Description

datediff

Description

day

Description

dayname

Description

dayofmonth

Description

dayofweek

Description

dayofyear

Description

days_diff

Description

days_sub

Description

from_days

Description

from_unixtime

Description

hour

Description

hours_add

Description

hours_diff

Description

hours_sub

Description

microseconds_add

Description

microseconds_sub

Description

minute

Description

minutes_add

Description

minutes_diff

Description

minutes_sub

Description

month

Description

monthname

Description

months_add

Description

months_diff

Description

months_sub

Description

now, current_timestamp, localtime, localtimestamp

Description

quarter

Description

second

Description

seconds_add

Description

seconds_diff

Description

seconds_sub

Description

str2date

Description

str_to_date

Description

time_slice

Description

time_to_sec

Description

timediff

Description

timestamp

Description

timestampadd

Description

timestampdiff

Description

to_date

Description

to_days

Description

unix_timestamp

Description

utc_timestamp

Description

week

Description

weekofyear

Description

weeks_add

Description

weeks_diff

Description

weeks_sub

Description

year

Description

years_add

Description

years_diff

Description

years_sub

Description

add_months

Description

Adds a specified number of months to a given date (DATE or DATETIME). The months_add function provides similar functionalities.

The day component in the resulting month remains the same as that specified in `date`, unless the resulting month has fewer days than the day component of the given date, in which case the day will be the last day of the resulting month. For example, `select add_months('2022-01-31', 1);` returns `2022-02-28 00:00:00`. If the resulting month has more days than the day component of the given date, the result has the same day component as `date`. For example, `select add_months('2022-02-28', 1)` returns `2022-03-28 00:00:00`.

Difference with Oracle: In Oracle, if `date` is the last day of the month, then the result is the last day of the resulting month.

Returns NULL if an invalid date or a NULL argument is passed in.

Syntax

```
ADD_MONTH(date, months)
```

Parameters

- `date`: It must be a valid date or datetime expression.
- `months`: the months you want to add. It must be an integer. A positive integer adds months to `date`. A negative integer subtracts months from `date`.

Return value

Returns a DATETIME value. If the date does not exist, for example, `2020-02-30`, NULL is returned.

If the date is a DATE value, it will be converted into a DATETIME value.

Examples

```
> 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,days_add

Description

Adds a specified time interval to a date.

Syntax

```
DATETIME ADDDATE(DATETIME|DATE date,INTERVAL expr type)
```

Parameters

- `date`: It must be a valid date or datetime expression.
- `expr`: the time interval you want to add. It must be of the INT type.
- `type`: the unit of the time interval. It can only be set to any of the following values: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.

Return value

Returns a DATETIME value. If the date does not exist, for example, `2020-02-30`, NULL is returned. If the date is a DATE value, it will be converted into a DATETIME value.

Examples

```
select adddate('2010-11-30 23:59:59', INTERVAL 2 DAY);
+-----------------------------------------------+
| adddate('2010-11-30 23:59:59', INTERVAL 2 DAY) |
+-----------------------------------------------+
| 2010-12-02 23:59:59                           |
+-----------------------------------------------+
```

```
select adddate('2010-12-03', INTERVAL 2 DAY);
+---------------------------------------+
| adddate('2010-12-03', INTERVAL 2 DAY) |
+---------------------------------------+
| 2010-12-05 00:00:00                   |
+---------------------------------------+
```

convert_tz

Description

Converts a DATE or DATETIME value from one time zone to another.

This function may return different results for different time zones. For more information, see Configure a time zone.

Syntax

```
DATETIME CONVERT_TZ(DATETIME|DATE dt, VARCHAR from_tz, VARCHAR to_tz)
```

Parameters

- `dt`: the DATE or DATETIME value to convert.
- `from_tz`: the source time zone. VARCHAR is supported. The time zone can be represented in two formats: one is Time Zone Database (for example, Asia/Shanghai) and the other is UTC offset (for example, +08:00).
- `to_tz`: the destination time zone. VARCHAR is supported. Its format is the same as `from_tz`.

Return value

Returns a value of the DATETIME data type. If the input is a DATE value, it will be converted into a DATETIME value. This function returns NULL if any of the input parameters is invalid or NULL.

Usage notes

For Time Zone Database, see List of tz database time zones (in Wikipedia).

Examples

Example 1: Convert a datetime in Shanghai to Los_Angeles.

```
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)
|
```

Example 2: Convert a date in Shanghai to Los_Angeles.

```
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)
```

Example 3: Convert a datetime in UTC+08:00 to Los_Angeles.

```
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)
```

Keywords

CONVERT_TZ, timezone, time zone

curdate,current_date

Description

Obtains the current date and returns a value of the DATE type.

Syntax

```
DATE CURDATE()
```

Examples

```
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

Description

Obtains the current date and returns a value if the DATETIME type.

Syntax

```
DATETIME CURRENT_TIMESTAMP()
```

Examples

Example 1: Return the current time.

```
MySQL > select current_timestamp();
+---------------------+
| current_timestamp() |
+---------------------+
| 2019-05-27 15:59:33 |
+---------------------+
```

Example 2: When you create a table, you can use this function for a column so that current time is the default value of the column.

```
CREATE TABLE IF NOT EXISTS sr_member (
    sr_id            INT,
    name             STRING,
    city_code        INT,
    reg_date         DATETIME DEFAULT current_timestamp,
    verified         BOOLEAN
);
```

keyword

CURRENT_TIMESTAMP,CURRENT,TIMESTAMP

curtime,current_time

Description

Obtains the current time and returns a value of the TIME type.

This function may return different results for different time zones. For more information, see Configure a time zone.

Syntax

```
TIME CURTIME()
```

Examples

```
MySQL > select current_time();
+----------------+
| current_time() |
+----------------+
| 15:25:47       |
+----------------+
```

keyword

CURTIME,CURRENT_TIME

date

Description

Extracts the date part of a date or datetime expression.

Syntax

```
DATE date(DATETIME|DATE expr)
```

Parameters

`expr`: the date or datetime expression.

Return value

Returns a value of the DATE type. NULL is returned if the input is NULL or invalid.

Examples

Example 1: Extract the date part of a datetime value.

```
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)
```

Example 2: Extract the date part of a date value.

```
SELECT DATE('2017-12-31');
+--------------------+
| date('2017-12-31') |
+--------------------+
| 2017-12-31         |
+--------------------+
1 row in set (0.08 sec)
```

Example 3: Extract the date part of the current timestamp.

```
SELECT DATE(current_timestamp());
+--------------------------+
| date(current_timestamp()) |
+--------------------------+
| 2022-11-08               |
+--------------------------+
1 row in set (0.05 sec)
```

date_add

Description

Adds a specified time interval to a date.

Syntax

```
DATETIME DATE_ADD(DATETIME|DATE date,INTERVAL expr type)
```

Parameters

- `date`: It must be a valid date or datetime expression.
- `expr`: the time interval you want to add. It must be of the INT type.
- `type`: the unit of the time interval. It can only be set to any of the following values: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.

Return value

Returns a DATETIME value. If the date does not exist, for example, `2020-02-30`, NULL is returned. If the date is a DATE value, it will be converted into a DATETIME value.

Examples

```
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                   |
+---------------------------------------+
```

date_format

Description

Converts a date into a string according to the specified format. Currently it supports strings with a maximum of 128 bytes. If the length of the returned value exceeds 128, NULL is returned.

Syntax

```
VARCHAR DATE_FORMAT(DATETIME date, VARCHAR format)
```

Parameters

- The `date` parameter must be a valid date or date expression.
- `format` specifies the output format of the date or time.

Here are the formats available:

```
%a | Abbreviated weekday name (Sun to Sat)
%b | Abbreviated month name (Jan to Dec)
%c | Numeric month name (0-12)
%D | Day of the month as a numeric value, followed by suffix in
English
%d | Day of the month as a numeric value (00-31)
%e | Day of the month as a numeric value (0-31)
%f | Microseconds
%H | Hour (00-23)
%h | Hour (01-12)
%I | Hour (01-12)
%i | Minutes (00-59)
%j | Day of the year (001-366)
%k | Hour (0-23)
%l | Hour (1-12)
```

```
%M | Month name in full
%m | Month name as a numeric value (00-12)
%p | AM or PM
%r | Time in 12 hour (hh:mm:ss AM or PM)
%S | Seconds (00-59)
%s | Seconds (00-59)
%T | Time in 24 hour format (hh:mm:ss)
%U | Week (00-53) where Sunday is the first day of the week
%u | Week (00-53) where Monday is the first day of the week
%V | Week (01-53)  where Sunday is the first day of the week. Used
with %X.
%v | Week (01-53) where Monday is the first day of the week. Used
with %x.
%W | Weekday name in full
%w | Day of the week where Sunday=0 and Saturday=6
%X |  Year for the week where Sunday is the first day of the week. 4-
digital value. Used with  %V.
%x | Year for the week where Monday is the first day of the week. 4-
digital value. Used with  %v.
%Y | Year. 4-digital value.
%y | Year. 2-digital value.
%% |  Represent %.
```

Examples

```
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

Description

Converts a given time into the beginning or end of a time interval based on the specified time granularity.

This function is supported from v2.5.

Syntax

```
DATE time_slice(DATE dt, INTERVAL N type[, boundary])
```

Parameters

- `dt`: the time to convert, DATE.
- `INTERVAL N type`: the time granularity, for example, `interval 5 day`.
  - `N` is the length of time interval. It must be an INT value.
  - `type` is the unit, which can be YEAR, QUARTER, MONTH, WEEK, DAY. If `type` is set to HOUR, MINUTE, or SECOND for a DATE value, an error is returned.
- `boundary`: optional. It is used to specify whether to return the beginning (`FLOOR`) or end (`CEIL`) of the time interval. Valid values: FLOOR, CEIL. If this parameter is not specified, `FLOOR` is the default.

Return value

Returns a value of the DATE type.

Usage notes

The time interval starts from A.D. `0001-01-01 00:00:00`.

Examples

Example 1: Convert a given time to the beginning of a 5-year time interval without specifying the `boundary` parameter.

```
select date_slice('2022-04-26', interval 5 year);
+-----------------------------------------------+
| date_slice('2022-04-26', INTERVAL 5 year, floor) |
+-----------------------------------------------+
| 2021-01-01                                    |
+-----------------------------------------------+
```

Example 2: Convert a given time to the end of a 5-day time interval.

```
select date_slice('0001-01-07', interval 5 day, CEIL);
+---------------------------------------------+
| date_slice('0001-01-07', INTERVAL 5 day, ceil) |
+---------------------------------------------+
| 0001-01-11                                  |
+---------------------------------------------+
```

The following examples are provided based on the `test_all_type_select` table.

```
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)
```

Example 3: Convert a given DATE value to the beginning of a 5-second time interval.

```
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)
```

An error is returned because the system cannot find the second part of a DATE value.

Example 4: Convert a given DATE value to the beginning of a 5-day time interval.

```
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)
```

Example 5: Convert a given DATE value to the end of a 5-day time interval.

```
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

Description

Subtracts the specified time interval from a date.

Syntax

```
DATETIME DATE_SUB(DATETIME|DATE date,INTERVAL expr type)
```

Parameters

- `date`: It must be a valid DATE or DATETIME expression.
- `expr`: the time interval you want to subtract. It must be of the INT type.
- `type`: the unit of the time interval. It can only be set to any of the following values: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.

Return value

Returns a DATETIME value. If the date does not exist, for example, `2020-02-30`, or if the date is not a DATE or DATETIME value, NULL is returned.

Examples

```
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                                   |
+---------------------------------------+
```

date_trunc

Description

Truncates a time value based on the specified date part, such as year, day, hour, or minute.

StarRocks also provides the year, quarter, month, week, day, and hour functions for you to extract the specified date part.

Syntax

```
DATETIME date_trunc(VARCHAR fmt, DATETIME|DATE datetime)
```

Parameters

- `datetime`: the time to truncate, which can be of the DATETIME or DATE type. The date and time must exist. Otherwise, NULL will be returned. For example, `2021-02-29 11:12:13` does not exist as a date and NULL will be returned.
- `fmt`: the date part, that is, to which precision `datetime` will be truncated. The value must be a VARCHAR constant.

  `fmt` must be set to a value listed in the following table. If the value is incorrect, an error will be returned.

  If `datetime` is a DATE value, `fmt` can only be `year`, `quarter`, `month`, `week`, or `day`. If you set `fmt` to other date units, for example `hour`, an error is reported. See Example 5.

Value	Description
second	Truncates to the second.

Value	Description
minute	Truncates to the minute. The second part will be zero out.
hour	Truncates to the hour. The minute and second parts will be zero out.
day	Truncates to the day. The time part will be zero out.
week	Truncates to the first date of the week that `datetime` falls in. The time part will be zero out.
month	Truncates to the first date of the month that `datetime` falls in. The time part will be zero out.
quarter	Truncates to the first date of the quarter that `datetime` falls in. The time part will be zero out.
year	Truncates to the first date of the year that `datetime` falls in. The time part will be zero out.

Return value

Returns a value of the DATETIME type.

If `datetime` is of the DATE type and `fmt` is set to `hour`, `minute`, or `second`, the time part of the returned value defaults to `00:00:00`.

Examples

Example 1: Truncate the input time to the minute.

```
select date_trunc("minute", "2020-11-04 11:12:13");
+-----------------------------------------+
| date_trunc('minute', '2020-11-04 11:12:13') |
+-----------------------------------------+
```

```
| 2020-11-04 11:12:00                         |
+---------------------------------------------+
```

Example 2: Truncate the input time to the hour.

```
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                     |
+-----------------------------------------+
```

Example 3: Truncate the input time to the first day of a week.

```
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                     |
+-----------------------------------------+
```

Example 4: Truncate the input time to the first day of a year.

```
select date_trunc("year", "2020-11-04 11:12:13");
+-----------------------------------------+
| date_trunc('year', '2020-11-04 11:12:13') |
+-----------------------------------------+
| 2020-01-01 00:00:00                     |
+-----------------------------------------+
```

Example 5: Truncate a DATE value to the hour. An error is returned.

```
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.
```

datediff

Description

Calculates the date from `expr1` to `expr2` and returns a result in days.

`expr1` and `expr2` are valid DATE or DATETIME expressions.

Note: Only the date parts of the values are used in the calculation.

Syntax

```
INT DATEDIFF(DATETIME expr1,DATETIME expr2)`
```

Examples

```
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 |
+-------------------------------------------------------------------------
---------------+
```

day

Description

Extracts the day part of a date or datetime expression and returns a value that ranges from 1 to 31.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT DAY(DATETIME|DATE date)
```

Examples

```
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

dayname

Description

Returns the day corresponding to a date.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
VARCHAR DAYNAME(date)
```

Examples

```
MySQL > select dayname('2007-02-03 00:00:00');
+------------------------------+
| dayname('2007-02-03 00:00:00') |
+------------------------------+
| Saturday                     |
+------------------------------+
```

keyword

DAYNAME

dayofmonth

Description

Obtains the day part in a date and returns a value that ranges from 1 to 31.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT DAYOFMONTH(DATETIME date)
```

Examples

```
MySQL > select dayofmonth('1987-01-31');
+-----------------------------------+
| dayofmonth('1987-01-31 00:00:00') |
+-----------------------------------+
|                                31 |
+-----------------------------------+
```

keyword

DAYOFMONTH

dayofweek

Description

Returns the weekday index for a given date. For example, the index for Sunday is 1, for Monday is 2, for Saturday is 7.

The `date` parameter must be of the DATE or DATETIME type, or a valid expression that can be cast into a DATE or DATETIME value.

Syntax

```
INT dayofweek(DATETIME date)
```

Examples

```
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

Description

Returns the day of the year for a given date.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT DAYOFYEAR(DATETIME|DATE date)
```

Examples

```
MySQL > select dayofyear('2007-02-03 00:00:00');
+----------------------------------+
| dayofyear('2007-02-03 00:00:00') |
+----------------------------------+
|                               34 |
+----------------------------------+
```

keyword

DAYOFYEAR

days_diff

Description

Returns the day difference between two date expressions (`expr1` – `expr2`), accurate to the day.

Difference between days_diff and datediff:

Function	Behavior	Example
days_diff	The calculation is accurate to the second and rounded down to the nearest integer.	days_diff between '2020-12-25 23:00:00' and '2020-12-24 23:00:01' is 0.
datediff	The calculation is accurate to the day.	datediff between '2020-12-25 23:00:00' and '2020-12-24 23:00:01' is 1.

Syntax

```
BIGINT days_diff(DATETIME expr1,DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME or DATE type.
- `expr2`: the start time. It must be of the DATETIME or DATE type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29. days_diff() takes the hour, minute, and second parts into calculation. If the difference is less than 1 day, 0 is returned.

Examples

```
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

Description

Subtracts specified days from a date or datetime to get a new datetime.

Syntax

```
DATETIME days_sub(DATETIME|DATE d, INT n);
```

Parameters

`d`: the date or datetime expression.

`n`: the number of days to subtract from `d`.

Return value

Returns a value of the datetime type, NULL is returned if any parameter is NULL or invalid.

NULL is returned if the result exceeds the range [0000-01-01 00:00:00, 9999-12-31 00:00:00].

Examples

```
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

Description

Returns a date from 0000-01-01.

Syntax

```
DATE FROM_DAYS(INT N)
```

Examples

```
MySQL > select from_days(730669);
+-------------------+
| from_days(730669) |
+-------------------+
| 2000-07-03        |
+-------------------+
```

keyword

FROM_DAYS,FROM,DAYS

from_unixtime

Description

Converts a UNIX timestamp into the required time format. The default format is `yyyy-MM-dd HH:mm:ss`. It also supports the formats in date_format.

Currently, `string_format` supports the following formats:

```
%Y: Year  e.g.: 2014, 1900
%m: Month   e.g.: 12, 09
%d: Day  e.g.: 11, 01
%H: Hour  e.g.: 23, 01, 12
%i: Minute  e.g.: 05, 11
%s: Second  e.g.: 59, 01
```

Other formats are invalid and NULL will be returned.

Syntax

```
VARCHAR from_unixtime(INT unix_timestamp[, VARCHAR string_format])
```

Parameters

- `unix_timestamp`: the UNIX timestamp you want to convert. It must be of the INT type. If the specified timestamp is less than 0 or greater than 2147483647, NULL will be returned. That is, the range for timestamp is `1970-01-01 00:00:00` to `2038-01-19 11:14:07`.
- `string_format`: the required time format.

Return value

Returns a DATETIME or DATE value of the VARCHAR type. If `string_format` specifies the DATE format, a DATE value of the VARCHAR type is returned.

If the timestamp exceeds the value range or if `string_format` is invalid, NULL will be returned.

Examples

```
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                            |
+------------------------------------------------+
```

keyword

FROM_UNIXTIME,FROM,UNIXTIME

hour

Description

Returns the hour for a given date. The return value ranges from 0 to 23.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT HOUR(DATETIME|DATE date)
```

Examples

```
MySQL > select hour('2018-12-31 23:59:59');
+-----------------------------+
| hour('2018-12-31 23:59:59') |
+-----------------------------+
|                          23 |
+-----------------------------+
```

keyword

HOUR

hours_add

Description

Adds hours to a date or datetime.

Syntax

```
DATETIME hours_add(DATETIME|DATE date, INT hours);
```

Parameters

`date`: Base date or datetime.

`hours`: Hours to be added.

Return value

Returns a DATETIME value.

NULL is returned when any of the input parameters is NULL.

Examples

```
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

Description

Returns the hour difference between two date expressions (`expr1` – `expr2`), accurate to the hour.

Syntax

```
BIGINT hours_diff(DATETIME expr1, DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME type.
- `expr2`: the start time. It must be of the DATETIME type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29.

Examples

```
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

Description

Reduces the specified date and time by a specified number of hours.

Syntax

```
DATETIME hours_sub(DATETIME|DATE date, INT hours);
```

Parameters

- `date`: It must be a valid DATE or DATETIME expression.
- `hours`: The number of hours reduced. The supported data type is INT.

Return value

Returns a DATETIME value. If the date does not exist, for example, `2022-21-01`, or if the date is not a DATE or DATETIME value, NULL is returned.

Examples

```
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         |
+-----------------------------+

Error case:
select hours_sub('2022-21-01', -1);
+-------------------------------------+
| hours_sub('2022-21-01', -1) |
+-------------------------------------+
| NULL                                 |
+-------------------------------------+
```

microseconds_add

Description

Adds a time interval to a date value. The time interval is in microseconds.

Syntax

```
DATETIME microseconds_add(DATETIME expr1,INT expr2);
```

Parameters

`expr1`: the time expression. It must be of the DATETIME type.

`expr2`: the time interval you want to add, in microseconds. It must be of the INT type.

Return value

Returns a value of the DATETIME type. If the input value is of the DATE type, the hour, minute, and seconds parts are processed as `00:00:00`.

Examples

```
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

Description

Subtracts a time interval from a date value. The time interval is in microseconds.

Syntax

```
DATETIME microseconds_sub(DATETIME expr1,INT expr2);
```

Parameters

`expr1`: the time expression. It must be of the DATETIME type.

`expr2`: the time interval you want to substract, in microseconds. It must be of the INT type.

Return value

Returns a value of the DATETIME type. If the input value is of the DATE type, the hour, minute, and seconds parts are processed as `00:00:00`.

Examples

```
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        |
+-----------------------------------+
```

minute

Description

Returns the minute for a given date. The return value ranges from 0 to 59.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT MINUTE(DATETIME|DATE date)
```

Examples

```
MySQL > select minute('2018-12-31 23:59:59');
+-----------------------------+
|minute('2018-12-31 23:59:59')|
+-----------------------------+
|                          59 |
+-----------------------------+
```

keyword

MINUTE

minutes_add

Description

Adds the specified minutes to the date, accurate to the minute.

Syntax

```
DATETIME minutes_add(DATETIME|DATE date, INT minutes);
```

Parameters

- `date`: the start time. It must be of the DATETIME or DATE type.
- `minutes`: the minutes to add. It must be of the INT type, it could be greater, equal or less than zero.

Return value

Returns a DATETIME value.

Returns NULL if either `date` or `minutes` is NULL.

Examples

```
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

Description

Returns the minute difference between two date expressions (`expr1` – `expr2`), accurate to the minute.

Syntax

```
BIGINT minutes_diff(DATETIME expr1,DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME type.
- `expr2`: the start time. It must be of the DATETIME type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29.

Examples

```
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

Description

Subtracts specified minutes from the date, accurate to the minute.

Syntax

```
DATETIME minutes_sub(DATETIME|DATE date, INT minutes);
```

Parameters

- `date`: the base date. It must be of the DATETIME or DATE type.
- `minutes`: the minutes to reduce. It must be of the INT type, it could be greater, equal or less than zero.

Return value

Returns a DATETIME value.

Returns NULL if either `date` or `minutes` is NULL.

Examples

```
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

Description

Returns the month for a given date. The return value ranges from 1 to 12.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT MONTH(DATETIME date)
```

Examples

```
MySQL > select month('1987-01-01');
+-----------------------------+
|month('1987-01-01 00:00:00') |
+-----------------------------+
|                           1 |
+-----------------------------+
```

keyword

MONTH

monthname

Description

Returns the name of the month for a given date.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
VARCHAR MONTHNAME(date)
```

Examples

```
MySQL > select monthname('2008-02-03 00:00:00');
+----------------------------------+
| monthname('2008-02-03 00:00:00') |
+----------------------------------+
| February                         |
+----------------------------------+
```

keyword

MONTHNAME, monthname

months_add

Description

Adds a specified number of months to the date, accurate to the month.

The add_months function provides similar functionalities.

Syntax

```
DATETIME months_add(DATETIME expr1, INT expr2);
```

Parameters

- `expr1`: the start time. It must be of the DATETIME or DATE type.
- `expr2`: the months to add. It must be of the INT type. It can be greater, equal, or less than zero. A negative value subtracts months from `date`.

Return value

Returns a DATETIME value.

Examples

```
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

Description

Returns the month difference between two date expressions (`expr1` – `expr2`), accurate to the month.

Syntax

```
BIGINT months_diff(DATETIME expr1,DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME or DATE type.
- `expr2`: the start time. It must be of the DATETIME or DATE type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29.

Examples

```
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

Description

Subtracts specified months from the date.

Syntax

```
DATETIME months_sub(DATETIME expr1, INT expr2);
```

Parameters

- `expr1`: the specified date. It must be of the DATETIME or DATE type.
- `expr2`: the month to subtract. It must be INT type.

Return value

Returns a DATETIME value.

Examples

```
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         |
+-----------------------------+
```

now, current_timestamp, localtime, localtimestamp

Description

Returns the current date and time.

This function may return different results for different time zones. For more information, see Configure a time zone.

Syntax

```
DATETIME NOW()
```

Examples

```
MySQL > select now();
+---------------------+
| now()               |
+---------------------+
| 2019-05-27 15:58:25 |
+---------------------+
```

keyword

NOW, now

quarter

Description

Returns the quarter part of a date, in the range of 1 to 4.

Syntax

```
INT quarter(DATETIME|DATE date);
```

Parameters

`date`: It must be of the DATETIME or DATE type.

Return value

Returns an INT value.

NULL is returned in any of the following scenarios:

- The date is not a valid DATETIME or DATE value.
- The input is empty.
- The date does not exist, for example, 2022-02-29.

Examples

Example 1: Return the quarter part of a DATETIME value.

```
SELECT QUARTER("2022-10-09 15:59:33");
+------------------------------+
| quarter('2022-10-09 15:59:33') |
+------------------------------+
|                            4 |
+------------------------------+
```

Example 2: Return the quarter part of a DATE value.

```
SELECT QUARTER("2022-10-09");
+----------------------+
| quarter('2022-10-09') |
+----------------------+
|                    4 |
+----------------------+
```

Example 3: Return the quarter part that corresponds to the current time or date.

```
SELECT QUARTER(NOW());
+---------------+
| quarter(now()) |
+---------------+
|             4 |
+---------------+

SELECT QUARTER(CURDATE());
+-------------------+
| quartr(curdate()) |
+-------------------+
|                 4 |
+-------------------+
```

second

Description

Returns the second part for a given date. The return value ranges from 0 to 59.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT SECOND(DATETIME date)
```

Examples

```
MySQL > select second('2018-12-31 23:59:59');
+-----------------------------+
|second('2018-12-31 23:59:59')|
+-----------------------------+
|                           59 |
+-----------------------------+
```

keyword

SECOND

seconds_add

Description

Adds a specified time interval to the datetime. Unit is second.

Syntax

```
DATETIME SECONDS_ADD(DATETIME expr1,INT expr2)
```

The `expr1` parameter is a valid datetime expression.

The `expr2` parameter is the seconds you want to add.

Examples

```
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          |
+-----------------------------+
```

keyword

SECONDS_ADD,ADD

seconds_diff

Description

Returns the second difference between two date expressions (`expr1` – `expr2`), accurate to the second.

Syntax

```
BIGINT seconds_diff(DATETIME expr1,DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME type.
- `expr2`: the start time. It must be of the DATETIME type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29.

Examples

```
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

Description

Subtracts a time interval from a date value. The time interval is in seconds.

Syntax

```
DATETIME seconds_sub(DATETIME|DATE date, INT seconds);
```

Parameters

`date`: the time expression. It must be of the DATETIME type.

`seconds`: the time interval you want to substract, in seconds. It must be of the INT type.

Return value

Returns a value of the DATETIME type.

If the input value is of the DATE type, the hour, minute, and seconds parts are processed as `00:00:00`.

Examples

```
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          |
+-----------------------------+v
```

str2date

Description

Converts a string into a DATE value according to the specified format. If the conversion fails, NULL is returned.

The format must be consistent with that described in date_format.

This function is equivalent to str_to_date but has a different return type.

Syntax

```
DATE str2date(VARCHAR str, VARCHAR format);
```

Parameters

`str`: the time expression you want to convert. It must be of the VARCHAR type.

`format`: the format used to return the value. For the supported format, see date_format.

Return value

Returns a value of the DATE type.

NULL is returned if `str` or `format` is NULL.

Examples

```
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

Description

Converts a string into a DATETIME value according to the specified format. If the conversion fails, NULL is returned.

The format must be consistent with that described in date_format.

This function is inverse to date_format.

Syntax

```
DATETIME STR_TO_DATE(VARCHAR str, VARCHAR format)
```

Parameters

`str`: the time expression you want to convert. It must be of the VARCHAR type.

`format`: the format used to return the value. For the supported format, see date_format.

Return value

Returns a value of the DATETIME type. If `format` specifies a date, a DATE value is returned.

NULL is returned if `str` or `format` is NULL.

Examples

Example 1: Convert the input into a DATETIME value.

```
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                                 |
+-----------------------------------------------------+
```

Example 2: Convert the input into a DATE value.

```
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                                                |
+-----------------------------------------------------------+
```

Example 3: Convert the input "200442 Monday" into a DATE value.

```
MySQL > select str_to_date('200442 Monday', '%X%V %W');
+-----------------------------------------+
| str_to_date('200442 Monday', '%X%V %W') |
+-----------------------------------------+
| 2004-10-18                              |
+-----------------------------------------+
```

keyword

STR_TO_DATE,STR,TO,DATE

time_slice

Description

Converts a given time into the beginning or end of a time interval based on the specified time granularity.

This function is supported from v2.3. v2.5 supports converting a given time into the end of a time interval.

Syntax

```
DATETIME time_slice(DATETIME dt, INTERVAL N type[, boundary])
```

Parameters

- `dt`: the time to convert, DATETIME.
- `INTERVAL N type`: the time granularity, for example, `interval 5 second`.
  - `N` is the length of time interval. It must be an INT value.
  - `type` is the unit, which can be YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND.
- `boundary`: optional. It is used to specify whether to return the beginning (`FLOOR`) or end (`CEIL`) of the time interval. Valid values: FLOOR, CEIL. If this parameter is not specified, `FLOOR` is the default. This parameter is supported from v2.5.

Return value

Returns a value of the DATETIME type.

Usage notes

The time interval starts from A.D. `0001-01-01 00:00:00`.

Examples

The following examples are provided based on the `test_all_type_select` table.

```
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)
```

Example 1: Convert a given DATETIME value to the beginning of a 5-second time interval without specifying the `boundary` parameter.

```
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)
```

Example 2: Convert a given DATETIME value to the beginning of a 5-day time interval with `boundary` set to 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)
```

Example 3: Convert a given DATETIME value to the end of a 5-day time interval.

```
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                           |
+-----------------------------------------------+
5 rows in set (0.12 sec)
```

time_to_sec

Description

Converts a time value into the number of seconds. The formula used for the conversion is as follows:

Hour x 3600 + Minute x 60 + Second

Syntax

```
BIGINT time_to_sec(TIME time)
```

Parameters

`time`: It must be of the TIME type.

Return value

Returns a value of the BIGINT type. If the input is invalid, NULL is returned.

Examples

```
select time_to_sec('12:13:14');
+-----------------------------+
| time_to_sec('12:13:14')     |
+-----------------------------+
|                       43994|
+-----------------------------+
```

timediff

Description

Returns the difference between two DATETIME expressions.

The return value must be of the TIME type.

Syntax

```
TIME TIMEDIFF(DATETIME expr1, DATETIME expr2)`
```

Examples

```
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

Description

Returns the DATETIME value of a date or datetime expression.

Syntax

```
DATETIME timestamp(DATETIME|DATE expr);
```

Parameters

`expr`: the time expression you want to convert. It must be of the DATETIME or DATE type.

Return value

Returns a DATETIME value. If the input time is empty or does not exist, such as `2021-02-29`, NULL is returned.

Examples

```
select timestamp("2019-05-27");
+------------------------+
| timestamp('2019-05-27') |
+------------------------+
| 2019-05-27 00:00:00     |
+------------------------+
1 row in set (0.00 sec)
```

timestampadd

Description

Adds an integer expression interval to the date or datetime expression `datetime_expr`.

The unit for the interval as mentioned must be one of the following:

SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

Syntax

```
DATETIME TIMESTAMPADD(unit, interval, DATETIME datetime_expr)
```

Examples

```
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                            |
+------------------------------------------------+
```

keyword

TIMESTAMPADD

timestampdiff

Description

Returns the interval from `datetime_expr2` to `datetime_expr1`. `datetime_expr1` and `datetime_expr2` must be of the DATE or DATETIME type.

The unit for the integer result and the interval should be one of the following:

SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

Syntax

```
INT TIMESTAMPDIFF(unit,DATETIME datetime_expr1, DATETIME
datetime_expr2)
```

Examples

```
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
|
+-----------------------------------------------------------------------
-+
```

keyword

TIMESTAMPDIFF

to_date

Description

Converts a DATETIME value into a date.

Syntax

```
DATE TO_DATE(DATETIME datetime)
```

Examples

```
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_days

Description

Returns the number of days between a date and 0000-01-01.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT TO_DAYS(DATETIME date)
```

Examples

```
MySQL > select to_days('2007-10-07');
+-----------------------+
| to_days('2007-10-07') |
+-----------------------+
|                733321 |
+-----------------------+
```

keyword

TO_DAYS,TO,DAYS

unix_timestamp

Description

Converts a DATE or DATETIME value into a UNIX timestamp.

If no parameter is specified, this function converts the current time into a UNIX timestamp.

The `date` parameter must be of the DATE or DATETIME type.

For time before 1970-01-01 00:00:00 or after 2038-01-19 11:14:07, this function returns 0.

For more information about the date format, see date_format.

This function may return different results for different time zones. For more information, see Configure a time zone.

Syntax

```
INT UNIX_TIMESTAMP()
INT UNIX_TIMESTAMP(DATETIME date)
INT UNIX_TIMESTAMP(DATETIME date, STRING fmt)
```

Examples

```
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_timestamp

Description

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or 'YYYYMMDDHHMMSS' format depending on the usage of the function, for example, in a string or numeric context.

Syntax

```
DATETIME UTC_TIMESTAMP()
```

Examples

```
MySQL > select utc_timestamp(),utc_timestamp() + 1;
+---------------------+---------------------+
| utc_timestamp()     | utc_timestamp() + 1 |
+---------------------+---------------------+
| 2019-07-10 12:31:18 |      20190710123119 |
+---------------------+---------------------+
```

`utc_timestamp() + N` means adding `N` seconds to the current time.

keyword

UTC_TIMESTAMP,UTC,TIMESTAMP

week

Description

Returns the week number for a given date. This function works in the same way as the WEEK function in MySQL.

This function is supported from v2.3.

Syntax

```
INT WEEK(DATETIME|DATE date, INT mode)
```

Parameters

- `Date`: The supported data types are DATETIME and DATE.
- `Mode`: optional. The supported data type is INT. This parameter is used to specify the logic for calculating the week number, that is, whether the week starts on Sunday or Monday, and whether the return value is in the range of 0–53 or 1–53. Value range: 0–7. Default value: `0`. If this parameter is not specified, mode `0` is used by default. The following table describes how this parameter works.

Mode	First day of week	Range	Week 1 is the first week ...
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year

Mode	First day of week	Range	Week 1 is the first week ...
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

Return value

Returns a value of the INT type. Value range: 0–53. The specific range is determined based on the `mode` parameter. `NULL` is returned if the value of `date` is invalid or the input value is empty.

Examples

Calculate the week number for `2007-01-01`. `2007-01-01` is Monday on the calendar.

- `Mode` is set to `0` and `0` is returned. The week starts on Sunday. `2007-01-01` is Monday and cannot be week 1. Therefore, `0` is returned.

```
mysql> SELECT WEEK('2007-01-01', 0);
+----------------------+
| week('2007-01-01', 0) |
+----------------------+
|                    0 |
+----------------------+
1 row in set (0.02 sec)
```

- `Mode` is set to `1` and `1` is returned. The week starts on Monday and `2007-01-01` is Monday.

```
mysql> SELECT WEEK('2007-01-01', 1);
+----------------------+
| week('2007-01-01', 1) |
```

```
+----------------------+
|                    1 |
+----------------------+
1 row in set (0.02 sec)
```

- `Mode` is set to `2` and `53` is returned. The week starts on Sunday. However, `2007-01-01` is Monday and the value range is 1–53. Therefore, `53` is returned, indicating that this is the last week of the previous year.

```
mysql> SELECT WEEK('2007-01-01', 2);
+----------------------+
| week('2007-01-01', 2) |
+----------------------+
|                   53 |
+----------------------+
1 row in set (0.01 sec)
```

weekofyear

Description

Returns the week number for a given date within a year.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT WEEKOFYEAR(DATETIME date)
```

Examples

```
MySQL > select weekofyear('2008-02-20 00:00:00');
+----------------------------------+
| weekofyear('2008-02-20 00:00:00') |
+----------------------------------+
|                                8 |
+----------------------------------+
```

keyword

WEEKOFYEAR

weeks_add

Description

Adds a specified number of weeks to a date.

Syntax

```
DATETIME weeks_add(DATETIME|DATE expr1, INT expr2);
```

Parameters

- `expr1`: the original date. It must be of the DATETIME or DATE type.
- `expr2`: the number of weeks to add. It must be of the `INT` type.

Return value

Returns a DATETIME value.

`NULL` is returned if the date does not exist.

Examples

```
select weeks_add('2022-12-20',2);
+---------------------------+
| weeks_add('2022-12-20', 2) |
+---------------------------+
|        2023-01-03 00:00:00 |
+---------------------------+
```

weeks_diff

Description

Returns the week difference between two date expressions (`expr1` – `expr2`), accurate to the week.

Syntax

```
BIGINT weeks_diff(DATETIME expr1,DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME or DATE type.
- `expr2`: the start time. It must be of the DATETIME or DATE type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29.

Examples

```
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

Description

Returns the value with the number of weeks minused to date.

Syntax

```
DATETIME weeks_sub(DATETIME expr1, INT expr2);
```

Parameters

- `expr1`: the original date. It must be of the `DATETIME` type.
- `expr2`: the number of weeks. It must be of the `INT` type.

Return value

returns `DATETIME`.

`NULL` is returned if the date does not exist.

Examples

```
select weeks_sub('2022-12-22',2);
+--------------------------+
| weeks_sub('2022-12-22', 2) |
+--------------------------+
|        2022-12-08 00:00:00 |
+--------------------------+
```

year

Description

Returns the year part in a date and returns a value that ranges from 1000 to 9999.

The `date` parameter must be of the DATE or DATETIME type.

Syntax

```
INT YEAR(DATETIME date)
```

Examples

```
MySQL > select year('1987-01-01');
+----------------------------+
| year('1987-01-01 00:00:00') |
+----------------------------+
|                       1987 |
+----------------------------+
```

keyword

YEAR

years_add

Description

Adds a specified time interval to the date. Unit is year.

Syntax

```
DATETIME YEARS_ADD(DATETIME|DATE expr1,INT expr2)
```

The expr1 parameter is a valid datetime or date expression.

The expr2 parameter is the years you want to add.

Examples

```
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        |
+---------------------------+
```

keyword

YEARS_ADD,ADD

years_diff

Description

Returns the year difference between two date expressions (`expr1` – `expr2`), accurate to the year.

Syntax

```
BIGINT years_diff(DATETIME expr1,DATETIME expr2);
```

Parameters

- `expr1`: the end time. It must be of the DATETIME or DATE type.
- `expr2`: the start time. It must be of the DATETIME or DATE type.

Return value

Returns a BIGINT value.

NULL is returned if the date does not exist, for example, 2022-02-29.

Examples

```
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

Description

Subtracts the specified number of years from the specified datetime or date.

Syntax

```
DATETIME YEARS_SUB(DATETIME date, INT years)
```

Parameters

- `date`: The original date time, of type DATETIME or DATE.
- `years`: The number of years to subtract. The value can be negative, but date year minus years can't exceed 10000. For example, if the year of date is 2022, then years can't be less than -7979. At the same time, the years cann't exceed the year value of date, for example, if the year value of date is 2022, then years can't be greater than 2022.

Return value

The return value type is the same as the parameter `date`. Returns NULL if the result year is out of range [0, 9999].

Examples

```
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                 |
+-------------------------------------+
```

ST_AsText,ST_AsWKT

Description

ST_Circle

Description

ST_Contains

Description

ST_Distance_Sphere

Description

ST_GeometryFromText,ST_GeomFromText

Description

ST_LineFromText,ST_LineStringFromText

Description

ST_Point

Description

ST_Polygon,ST_PolyFromText,ST_PolygonFromText

Description

ST_X

Description

ST_Y

Description

ST_AsText,ST_AsWKT

Description

Converts a geometric figure to WKT (Well Known Text) format.

Syntax

```
VARCHAR ST_AsText(GEOMETRY geo)
```

Examples

```
MySQL > SELECT ST_AsText(ST_Point(24.7, 56.7));
+---------------------------------+
| st_astext(st_point(24.7, 56.7)) |
+---------------------------------+
| POINT (24.7 56.7)               |
+---------------------------------+
```

keyword

ST_ASTEXT,ST_ASWKT,ST,ASTEXT,ASWKT

ST_Circle

Description

Converts a WKT (WEll Known Text) to a circle on the sphere of the earth.

Syntax

```
GEOMETRY ST_Circle(DOUBLE center_lng, DOUBLE center_lat, DOUBLE
radius)
```

Parameters

`center_lng` indicates the longitude of the center of the circle.

`center_lat` indicates the latitude of the center of the circle.

`radius` indicates the radius of a circle, in meters. A maximum of 9999999 radius is supported.

Examples

```
MySQL > SELECT ST_AsText(ST_Circle(111, 64, 10000));
+-----------------------------------------+
| st_astext(st_circle(111.0, 64.0, 10000.0)) |
+-----------------------------------------+
| CIRCLE ((111 64), 10000)                |
+-----------------------------------------+
```

keyword

ST_CIRCLE,ST,CIRCLE

ST_Contains

Description

Checks whether the geometric figure shape1 can fully contain shape2.

Syntax

```
BOOL ST_Contains(GEOMETRY shape1, GEOMETRY shape2)
```

Examples

```
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 |
```

```
+-----------------------------------------------------------------------------
----------------------+
```

keyword

ST_CONTAINS,ST,CONTAINS

ST_Distance_Sphere

Description

Calculates the spherical distance between two points on the Earth in "meters". The parameters inputted are longitude at X, latitude at X, longitude at Y, latitude at Y.

Syntax

```
DOUBLE ST_Distance_Sphere(DOUBLE x_lng, DOUBLE x_lat, DOUBLE y_lng,
DOUBLE x_lat)
```

Examples

```
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 |
+-----------------------------------------------------------------
--------+
```

keyword

ST_DISTANCE_SPHERE,ST,DISTANCE,SPHERE

ST_GeometryFromText,ST_GeomFromText

Description

Converts a WKT (Well Known Text) to the corresponding memory geometry.

Syntax

```
GEOMETRY ST_GeometryFromText(VARCHAR wkt)
```

Examples

```
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)                                     |
+-----------------------------------------------------------+
```

keyword

ST_GEOMETRYFROMTEXT,ST_GEOMFROMTEXT,ST,GEOMETRYFROMTEXT,GEOMFROMTEXT

ST_LineFromText,ST_LineStringFromText

Description

Converts a WKT (Well Known Text) to a memory representation in the form of Line.

Syntax

```
GEOMETRY ST_LineFromText(VARCHAR wkt)
```

Examples

```
MySQL > SELECT ST_AsText(ST_LineFromText("LINESTRING (1 1, 2 2)"));
+-----------------------------------------------------------+
| st_astext(st_geometryfromtext('LINESTRING (1 1, 2 2)')) |
+-----------------------------------------------------------+
| LINESTRING (1 1, 2 2)                                     |
+-----------------------------------------------------------+
```

keyword

ST_LINEFROMTEXT,ST_LINESTRINGFROMTEXT,ST,LINEFROMTEXT,LINESTRINGFROMTEXT

ST_Point

Description

Returns the corresponding Point with the given X-coordinate and Y-coordinate. At the moment this value only makes sense on a spherical set. X/Y corresponds to longitude/latitude.

> **Caution**
>
> If you directly select ST_Point(), it may get stuck.

Syntax

```
POINT ST_Point(DOUBLE x, DOUBLE y)
```

Examples

```
MySQL > SELECT ST_AsText(ST_Point(24.7, 56.7));
+---------------------------------+
| st_astext(st_point(24.7, 56.7)) |
+---------------------------------+
| POINT (24.7 56.7)               |
+---------------------------------+
```

keyword

ST_POINT,ST,POINT

ST_Polygon,ST_PolyFromText,ST_PolygonFromText

Description

Converts a WKT (Well Known Text) to a corresponding polygon memory form.

Syntax

```
GEOMETRY ST_Polygon(VARCHAR wkt)
```

Examples

```
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))                        |
+----------------------------------------------------------------+
```

keyword

ST_POLYGON,ST_POLYFROMTEXT,ST_POLYGONFROMTEXT,ST,POLYGON,POLYFROMTEXT,POLYGONFROMTEXT

ST_X

Description

If point is of a valid Point type, return the corresponding X-coordinate value.

Syntax

```
DOUBLE ST_X(POINT point)
```

Examples

```
MySQL > SELECT ST_X(ST_Point(24.7, 56.7));
+----------------------------+
| st_x(st_point(24.7, 56.7)) |
+----------------------------+
|                       24.7 |
+----------------------------+
```

keyword

ST_X,ST,X

ST_Y

Description

If point is of a valid Point type, return the corresponding Y-coordinate value.

Syntax

```
DOUBLE ST_Y(POINT point)
```

Examples

```
MySQL > SELECT ST_Y(ST_Point(24.7, 56.7));
+----------------------------+
| st_y(st_point(24.7, 56.7)) |
+----------------------------+
|                       56.7 |
+----------------------------+
```

keyword

ST_Y,ST,Y

abs

Description

acos

Description

asin

Description

atan

Description

atan2

Description

bin

Description

ceil, dceil

Description

ceiling

Description

conv

Description

cos

Description

cos_similarity

Description

cos_similarity_norm

Description

cot

Description

degrees

Description

divide

Description

e

Description

exp,dexp

Description

floor, dfloor

Description

fmod

Description

greatest

Description

least

Description

ln, dlog1, log

Description

log

Description

log10, dlog10

Description

log2

Description

mod

Description

multiply

Description

negative

Description

pi

Description

pmod

Description

positive

Description

pow, power, dpow, fpow

Description

radians

Description

rand, random

Description

round, dround

Description

sign

Description

sin

Description

sqrt, dsqrt

Description

square

Description

tan

Description

truncate

Description

abs

Description

Returns the absolute value of the numeric value `x`. If the input value is NULL, NULL is returned.

Syntax

```
ABS(x);
```

Parameters

`x`: the numeric value or expression.

Supported data types: DOUBLE, FLOAT, LARGEINT, BIGINT, INT, SMALLINT, TINYINT, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128.

Return value

The data type of the return value is the same as the type of `x`.

Examples

```
mysql> select abs(-1);
+---------+
| abs(-1) |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

Keywords

abs, absolute

acos

Description

Computes the arc cosine of an argument.

Syntax

```
DOUBLE acos(DOUBLE arg)
```

Parameters

`arg`: You can specify only a numeric value. This function converts the numeric value into a DOUBLE value before it computes the arc cosine of the value.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

```
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 |
+----------+
```

keyword

ACOS

asin

Description

Computes the arc sine of the argument.

Syntax

```
DOUBLE asin(DOUBLE arg)
```

Parameters

`arg`: You can specify only a numeric value. This function converts the numeric value into a DOUBLE value before it computes the arc sine of the value.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

```
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 |
+-----------+
```

keyword

ASIN

atan

Description

Computes the arc tangent of the argument.

Syntax

```
DOUBLE atan(DOUBLE arg)
```

Parameters

`arg`: You can specify only a numeric value. This function converts the numeric value into a DOUBLE value before it computes the arc tangent of the value.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

```
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 |
+----------+
```

keyword

ATAN

atan2

Description

Returns the arc tangent of `x` divided by `y`, that is, the arc tangent of `x/y`. The signs of the two parameters are used to determine the quadrant of the result.

The return value is in the range of [-π, π].

Syntax

```
ATAN2(x,y);
```

Parameters

`x`: The supported data type is DOUBLE.

`y`: The supported data type is DOUBLE.

Return value

Returns a value of the DOUBLE type. Returns NULL if `x` or `y` is NULL.

Examples

```
mysql> select atan2(-0.8,2);
+--------------------+
| atan2(-0.8, 2)     |
+--------------------+
| -0.3805063771123649 |
```

```
+---------------------+
1 row in set (0.01 sec)
```

bin

Description

Transforms the input `arg` into a binary.

Syntax

```
bin(arg)
```

Parameter

`arg`: the input you want to transform into a binary. It supports the BIGINT data type.

Return value

Returns a value of the VARCHAR data type.

Examples

```
mysql> select bin(3);
+--------+
| bin(3) |
+--------+
| 11     |
+--------+
1 row in set (0.02 sec)
```

ceil, dceil

Description

Returns values from the input `arg` rounded to the nearest equal or larger integer.

Syntax

```
ceil(arg)
```

Parameter

`arg` supports the DOUBLE data type.

Return value

Returns a value of the BIGINT data type.

Examples

```
mysql> select ceil(3.14);
+------------+
| ceil(3.14) |
+------------+
|          4 |
+------------+
1 row in set (0.15 sec)
```

ceiling

Description

Returns values from the input `arg` rounded to the nearest equal or larger integer.

Syntax

```
ceiling(arg)
```

Parameter

`arg` supports the DOUBLE data type.

Return value

Returns a value of the BIGINT data type.

Examples

```
mysql> select ceiling(3.14);
+---------------+
| ceiling(3.14) |
+---------------+
|             4 |
+---------------+
1 row in set (0.00 sec)
```

conv

Description

Converts the number `x` from one numeric base system to another, and returns the result as a string value.

Syntax

```
CONV(x,y,z);
```

Parameters

- `x`: the number to be converted. VARCHAR or BIGINT is supported.
- `y`: the source base. TINYINT is supported.
- `z`: the target base. TINYINT is supported.

Return value

Returns a value of the VARCHAR data type.

Examples

Convert the number 8 from numeric base system 2 to numeric base system 10:

```
mysql> select conv(8,10,2);
+----------------+
| conv(8, 10, 2) |
+----------------+
| 1000           |
```

```
+----------------+
1 row in set (0.00 sec)
```

cos

Description

Computes the cosine of the argument.

Syntax

```
DOUBLE cos(DOUBLE arg)
```

Parameters

`arg`: You can specify only a numeric value. This function converts the numeric value into a DOUBLE value before it computes the cosine of the value.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

```
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

Description

Measures the similarity of two vectors by calculating the cosine of the angle between them. The angle is formed by the direction of the vectors while the difference in their magnitude is ignored.

The similarity is between -1 and 1. Smaller angles between vectors indicate greater cosine similarity.

- If two vectors have the same direction, they have a 0-degree angle and the cosine similarity is 1.
- Perpendicular vectors have a 90-degree angle and a cosine similarity of 0.
- Opposite vectors have a 180-degree angle and a cosine similarity of -1.

`cosine_similarity` is commonly used for assessing text and video similarity.

This function normalizes vectors before measuring the cosine similarity. If the input vectors have been normalized, you can use cosine_similarity_norm.

Syntax

```
cosine_similarity(a, b)
```

Parameters

`a` and `b` are the vectors to compare. They must have the same dimension. The supported data type is `Array<float>`. The two arrays must have the same number of elements. Otherwise, an error is returned.

Return value

Returns a FLOAT value within the range of [-1, 1]. If any input parameter is null or invalid, an error is reported.

Examples

1. Create a table to store vectors and insert data into this table.

```
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. Calculate the similarity of each row in the `data` column compared to the array `[0.1, 0.2, 0.3]` and list the result in descending order.

```
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

Description

Measures the similarity of two normalized vectors by calculating the cosine of the angle between them. The angle is formed by the direction of the vectors while the difference in their magnitude is ignored. This function assumes that the input vectors have been normalized. If you need to normalize vectors before calculating the cosine similarity, use cosine_similarity.

The similarity is between -1 and 1. Smaller angles between vectors indicate greater cosine similarity.

- If two vectors have the same direction, they have a 0-degree angle and the cosine similarity is 1.
- Perpendicular vectors have a 90-degree angle and a cosine similarity of 0.
- Opposite vectors have a 180-degree angle and a cosine similarity of -1.

Syntax

```
cosine_similarity_norm(a, b)
```

Parameters

`a` and `b` are the vectors to compare. They must have the same dimension. The supported data type is `Array<float>`. The two arrays must have the same number of elements. Otherwise, an error is returned.

Return value

Returns a FLOAT value within the range of [-1, 1]. If any input parameter is null or invalid, an error is reported.

Examples

1. Create a table to store vectors and insert data into this table.

```
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. Calculate the similarity of each row in the `data` column compared to the array `[0.1, 0.2, 0.3]` and list the result in descending order.

```
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 |
+------+--------------+------------+
```

cot

Description

Returns the cotangent of the angle in radians `x`. If the angle in radians `x` is `0` or `NULL`, `NULL` is returned.

Syntax

```
COT(x);
```

Parameters

`x`: the angle in radians. DOUBLE is supported.

Return value

Returns a value of the DOUBLE data type.

Examples

```
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

Description

Converts the angle in radians `x` to degrees.

Syntax

```
DEGREES(x);
```

Parameters

`x`: the angle in radians. DOUBLE is supported.

Return value

Returns a value of the DOUBLE data type.

Examples

```
mysql> select degrees(3.1415926535898);
+--------------------------+
| degrees(3.1415926535898) |
+--------------------------+
|        180.0000000000004 |
+--------------------------+
1 row in set (0.07 sec)
```

divide

Description

Returns the quotient of `x` divided `y`. If `y` is 0, this functions returns null.

Syntax

```
divide(x, y)
```

Parameters

- `x`: The supported types are DOUBLE, FLOAT, LARGEINT, BIGINT, INT, SMALLINT, TINYINT, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128.
- `y`: The supported types are the same as `x`.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

```
mysql> select divide(3, 2);
+--------------+
| divide(3, 2) |
+--------------+
```

```
|          1.5 |
+--------------+
1 row in set (0.00 sec)
```

e

Description

Returns the constant e (the base of natural logarithms).

Syntax

```
E();
```

Parameters

This function takes no parameters.

Return value

Returns the constant e as an DOUBLE.

Examples

```
mysql> select e();
+------------------+
| e()              |
+------------------+
| 2.718281828459045 |
+------------------+
1 row in set (0.01 sec)
```

exp,dexp

Description

Returns the value of e raised to the power of `x`. This function is called natural logarithms function.

Syntax

```
EXP(x);
```

Parameters

`x`: the power number. DOUBLE is supported.

Return value

Returns a value of the DOUBLE data type.

Examples

Return e raised to the power of 3.14:

```
mysql> select exp(3.14);
+-------------------+
| exp(3.14)         |
+-------------------+
| 23.103866858722185 |
+-------------------+
1 row in set (0.01 sec)
```

floor, dfloor

Description

Returns the largest integer that is not more than `x`.

Syntax

```
FLOOR(x);
```

Parameters

`x`: DOUBLE is supported.

Return value

Returns a value of the BIGINT data type.

Examples

```
mysql> select floor(3.14);
+-------------+
| floor(3.14) |
+-------------+
|           3 |
+-------------+
1 row in set (0.01 sec)
```

fmod

Description

Returns the floating point remainder of the division (`dividend`/`divisor`). It is a modulo function.

Syntax

```
fmod(dividend,devisor);
```

Parameters

- `dividend`: DOUBLE or FLOAT is supported.
- `devisor`: DOUBLE or FLOAT is supported.

> **Note**
>
> The data type of `devisor` needs to be the same as the data type of `dividend`. Otherwise, StarRocks performs implicit conversion to convert the data type.

Return value

The data type and sign of output need to be the same as the data type and sign of `dividend`. If `divisor` is `0`, `NULL` is returned.

Examples

```
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

Description

Returns the largest value from a list of one or more parameters.

Generally, the return value has the same data type as the input.

The parameters are compared based on the following rules:

- NULL is returned if any of the input parameters is NULL.
- If at least one parameter is of the DOUBLE type, all the parameters are compared as DOUBLE values. The same rule applies to the DECIMAL and FLOAT data types.
- If the parameters are a mix of numbers and strings but the strings can be converted into numbers, the parameters are compared as numbers. If the strings cannot be converted into numbers, the parameters are compared as strings.
- If the parameters are characters, they are compared based on the alphabetical order of the first letter.

Syntax

```
GREATEST(expr1,...);
```

Parameters

`expr1`: the expression to compare. It supports the following data types:

- SMALLINT
- TINYINT

- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128
- DATETIME
- VARCHAR

Examples

Example 1: Return the largest value for a single input.

```
mysql> select greatest(3);
+-------------+
| greatest(3) |
+-------------+
|           3 |
+-------------+
1 row in set (0.01 sec)
```

Example 2: Return the largest value from a list of values.

```
mysql> select greatest(3,4,5,5,6);
+-----------------------+
| greatest(3, 4, 5, 5, 6) |
+-----------------------+
|                     6 |
```

```
+------------------------+
1 row in set (0.00 sec)
```

Example 3: One parameter is of the DOUBLE type and a DOUBLE value is returned.

```
mysql> select greatest(7,4.5);
+------------------+
| greatest(7, 4.5) |
+------------------+
|              7.0 |
+------------------+
1 row in set (0.05 sec)
```

Example 4: The input parameters are a mix of number and string but the string can be converted into a number. The parameters are compared as numbers.

```
mysql> select greatest(7,'9');
+------------------+
| greatest(7, '9') |
+------------------+
| 9                |
+------------------+
1 row in set (0.04 sec)
```

Example 5: The input parameters are a mix of number and string but the string cannot be converted into a number. The parameters are compared as strings. The string `'1'` is less than `'at'`.

```
select greatest(1,'at');
+-------------------+
| greatest(1, 'at') |
+-------------------+
| at                |
+-------------------+
```

Example 6: The input parameters are characters. The letter `Z` has the largest value.

```
mysql> select greatest('A','B','Z');
+-----------------------+
| greatest('A', 'B', 'Z') |
```

```
+-------------------------+
| Z                       |
+-------------------------+
1 row in set (0.00 sec)
```

Keywords

GREATEST, greatest

least

Description

Returns the smallest value from a list of one or more parameters.

Generally, the return value has the same data type as the input.

The comparison rules are the same as the greatest function.

Syntax

```
LEAST(expr1,...);
```

Parameters

`expr1`: the expression to compare. It supports the following data types:

- SMALLINT
- TINYINT
- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE
- DECIMALV2
- DECIMAL32

- DECIMAL64
- DECIMAL128
- DATETIME
- VARCHAR

Examples

Example 1: Return the smallest value for a single input.

```
select least(3);
+----------+
| least(3) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

Example 2: Return the smallest value from a list of values.

```
select least(3,4,5,5,6);
+----------------------+
| least(3, 4, 5, 5, 6) |
+----------------------+
|                    3 |
+----------------------+
1 row in set (0.01 sec)
```

Example 3: One parameter is of the DOUBLE type and a DOUBLE value is returned.

```
select least(4,4.5,5.5);
+-------------------+
| least(4, 4.5, 5.5) |
+-------------------+
|                4.0 |
+-------------------+
```

Example 4: The input parameters are a mix of number and string but the string can be converted into a number. The parameters are compared as numbers.

```
select least(7,'5');
+--------------+
| least(7, '5') |
+--------------+
| 5             |
+--------------+
1 row in set (0.01 sec)
```

Example 5: The input parameters are a mix of number and string but the string cannot be converted into a number. The parameters are compared as strings. The string `'1'` is less than `'at'`.

```
select least(1,'at');
+---------------+
| least(1, 'at') |
+---------------+
| 1              |
+---------------+
```

Example 6: The input parameters are characters.

```
mysql> select least('A','B','Z');
+---------------------+
| least('A', 'B', 'Z') |
+---------------------+
| A                   |
+---------------------+
1 row in set (0.00 sec)
```

Keywords

LEAST, least

ln, dlog1, log

Description

Computes the natural (base `e`) logarithm of a number.

Syntax

```
ln(arg)
log(arg)
dlog1(arg)
```

Parameters

- `arg`: the value whose logarithm you want to calculate. Only the DOUBLE data type is supported.

> **NOTE**
>
> StarRocks returns NULL if `arg` is specified as a negative or 0.

Return value

Returns a value of the DOUBLE data type.

Example

Example 1: Calculate the natural logarithm of 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

Description

Calculates the logarithm of a number to a specified base (or radix). If the base is not specified, this function is equivalent to ln.

Syntax

```
log([base,] arg)
```

Parameters

- `base`: Optional. The base. Only DOUBLE data type is supported. If this parameter is not specified, this function is equivalent to ln.

> **NOTE**
>
> StarRocks returns NULL if `base` is specified as a negative, 0, or 1.

- `arg`: The value whose logarithm you want to calculate. Only DOUBLE data type is supported.

> **NOTE**
>
> StarRocks returns NULL if `arg` is specified as a negative or 0.

Return value

Returns a value of the DOUBLE data type.

Example

Example 1: Calculate the logarithm of 8 to the base 2.

```
mysql> select log(2,8);
+-----------+
| log(2, 8) |
+-----------+
|         3 |
+-----------+
1 row in set (0.01 sec)
```

Example 2: Calculate the logarithm of 10 to the base *e* (with the base not specified).

```
mysql> select log(10);
+-------------------+
| log(10)           |
+-------------------+
| 2.302585092994046 |
+-------------------+
1 row in set (0.09 sec)
```

log10, dlog10

Description

Calculates the base 10 logarithm of a number.

Syntax

```
log10(arg)
```

Parameters

- `arg`: The value whose logarithm you want to calculate. Only DOUBLE data type is supported.

> **NOTE**
>
> StarRocks returns NULL if `arg` is specified as a negative or 0.

Return value

Returns a value of the DOUBLE data type.

Example

Example 1: Calculate the base 10 logarithm of 100.

```
select log10(100);
+------------+
| log10(100) |
+------------+
```

```
|          2 |
+------------+
1 row in set (0.02 sec)
```

log2

Description

Calculates the base 2 logarithm of a number.

Syntax

```
log2(arg)
```

Parameters

- `arg`: The value whose logarithm you want to calculate. Only DOUBLE data type is supported.

> **NOTE**
>
> StarRocks returns NULL if `arg` is specified as a negative or 0.

Return value

Returns a value of the DOUBLE data type.

Example

Example 1: Calculate the base 2 logarithm of 8.

```
mysql> select log2(8);
+---------+
| log2(8) |
+---------+
```

```
|       3 |
+---------+
1 row in set (0.00 sec)
```

mod

Description

The modulus function that returns the remainder of `dividend` divided by `divisor`.

Syntax

```
mod(dividend, divisor)
```

Parameters

- `dividend`: The number to be divided.
- `divisor`: The number that divides.

Both `dividend` and `divisor` support the following data types:

- TINYINT
- SMALLINT
- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128

> **NOTE**

> `dividend` and `divisor` must agree in the data type. StarRocks performs an implicit conversion if they do not agree in the data type.

Return value

Returns a value of the same data type as the `dividend`. StarRocks returns NULL if `divisor` is specified as 0.

Examples

```
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

Description

Computes the product of the arguments.

Syntax

```
multiply(arg1, arg2)
```

Parameters

`arg1`: numeric source column or literal. `arg2`: numeric source column or literal.

Return value

Returns the product of the two arguments. The return type depends on the arguments.

Usage notes

If you specify a non-numeric value, this function will fail.

Examples

```
MySQL > 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 > select * from t;
+------+------+------+------+
| id   | name | job1 | job2 |
+------+------+------+------+
|    2 |    2 |    2 |    2 |
+------+------+------+------+
1 row in set (0.08 sec)

MySQL > select multiply(1.0,id) from t;
+------------------+
| multiply(1.0, id) |
+------------------+
|                2 |
+------------------+
1 row in set (0.01 sec)
```

keyword

multiply

negative

Description

Returns the negative of input `arg`.

Syntax

```
negative(arg)
```

Parameter

`arg` supports the following data types:

- BIGINT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128

Return value

Returns a value of the same data type as the input.

Examples

```
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

Description

Returns the value of Pi.

Syntax

```
pi()
```

Parameter

N/A

Return value

Returns the value of Pi as a DOUBLE type value.

Examples

```
mysql> select pi();
+------------------+
| pi()             |
+------------------+
| 3.141592653589793 |
+------------------+
1 row in set (0.00 sec)
```

pmod

Description

Returns the positive remainder of `dividend` divided by `divisor`.

Syntax

```
pmod(dividend, divisor)
```

Parameters

- `dividend`: the number to be divided.
- `divisor`: the number that divides.

Both `arg1` and `arg2` support the following data types:

- BIGINT
- DOUBLE

> **NOTE**
>
> `dividend` and `divisor` must agree in the data type. StarRocks performs an implicit conversion if they do not agree in the data type.

Return value

Returns a value of the same data type as the `dividend`. StarRocks returns NULL if `divisor` is specified as 0.

Examples

```
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

Description

Returns `x` as a value.

Syntax

```
POSITIVE(x);
```

Parameters

`x`: supports the BIGINT, DOUBLE, DECIMALV2, DECIMAL32, DECIMAL64, and DECIMAL128 data types.

Return value

Returns a value whose data type is the same as the data type of the `x` value.

Examples

```
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

Description

Returns the result of `x` raised to the power of `y`.

Syntax

```
POW(x,y);POWER(x,y);
```

Parameters

`x`: supports the DOUBLE data type.

`y`: supports the DOUBLE data type.

Return value

Returns a value of the DOUBLE data type.

Examples

```
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

Description

Converts `x` from an angle to a radian.

Syntax

```
REDIANS(x);
```

Parameters

`x`: supports the DOUBLE data type.

Return value

Returns a value of the DOUBLE data type.

Examples

```
mysql> select radians(90);
+-------------------+
| radians(90)       |
+-------------------+
| 1.5707963267948966 |
+-------------------+
1 row in set (0.00 sec)
```

rand, random

Description

Returns a random floating-point number between 0 (inclusive) and 1 (exclusive).

Syntax

```
RAND(x);
```

Parameters

`x`: optional. The data type is BIGINT. If `x` is specified, this function returns a repeatable random number. If `x` is not specified, this function returns a completely random number.

Return value

Returns a value of the DOUBLE type.

Examples

```
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)
```

Keywords

RAND, RANDOM

round, dround

Description

Rounds a number to a specified number of digits.

- If `n` is not specified, `x` is rounded to the nearest integer.
- If `n` is specified, `x` is rounded to `n` decimal places. If `n` is negative, `x` is rounded to the left of the decimal point. If overflow occurs, an error is returned.

Syntax

```
ROUND(x [,n]);
```

Parameters

`x`: the number to be rounded. It supports the DOUBLE and DECIMAL128 data types.

`n`: the number of decimal places to round the number to. It supports the INT data type. This parameter is optional.

Return value

- If only `x` is specified, the return value is of the following data type:

  ["DECIMAL128"] -> "DECIMAL128"

  ["DOUBLE"] -> "BIGINT"

- If both `x` and `n` are specified, the return value is of the following data type:

  ["DECIMAL128", "INT"] -> "DECIMAL128"

["DOUBLE", "INT"] -> "DOUBLE"

Examples

```
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

Description

Returns the sign of `x`. A negative number, 0, or a positive number as input corresponds to `-1`, `0`, or `1` as output, respectively.

Syntax

```
SIGN(x);
```

Parameters

`x`: supports the DOUBLE data type.

Return value

Returns a value of the FLOAT data type.

Examples

```
mysql> select sign(3.14159);
+---------------+
| sign(3.14159) |
+---------------+
|             1 |
+---------------+
1 row in set (0.02 sec)
```

sin

Description

Returns the sine of `x`, where `x` is in radians.

Syntax

```
SIN(x);
```

Parameters

`x`: supports the DOUBLE data type.

Return value

Returns a value of the DOUBLE data type.

Examples

```
mysql> select sin(3.14);
+----------------------+
| sin(3.14)            |
+----------------------+
| 0.0015926529164868282 |
+----------------------+
1 row in set (0.21 sec)
```

sqrt, dsqrt

Description

Calculates the square root of a value. dsqrt is same as sqrt.

Syntax

```
DOUBLE SQRT(DOUBLE x);
DOUBLE DSQRT(DOUBLE x);
```

Parameters

`x`: You can specify only a numeric value. This function converts the numeric value into a DOUBLE value before calculation.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

```
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

Description

Calculates the square of a value.

Syntax

```
square(arg)
```

Parameters

`arg`: the value whose square you want to calculate. You can specify only a numeric value. This function converts the numeric value into a DOUBLE value before it calculates the square of the value.

Return value

Returns a value of the DOUBLE data type.

Usage notes

If you specify a non-numeric value, this function returns `NULL`.

Examples

Example 1: Calculate the square of a numeric value.

```
mysql>  select square(11);
+------------+
```

```
| square(11) |
+------------+
|        121 |
+------------+
```

Example 2: Calculate the square of a non-numeric value. The return value is `NULL`.

```
mysql>  select square('2021-01-01');
+----------------------+
| square('2021-01-01') |
+----------------------+
|                 NULL |
+----------------------+
```

tan

Description

Returns the tangent of `x`, where `x` is in radians.

Syntax

```
TAN(x);
```

Parameters

`x`: supports the DOUBLE data type.

Return value

Returns a value of the DOUBLE data type.

Examples

```
mysql> select tan(3.14);
+----------------------+
| tan(3.14)            |
+----------------------+
| -0.001592654936407223 |
+----------------------+
1 row in set (0.12 sec)
```

truncate

Description

Rounds the input down to the nearest equal or smaller value with the specified number of places after the decimal point.

Syntax

```
truncate(arg1,arg2);
```

Parameter

- `arg1`: the input to be rounded. It supports the following data types:
  - DOUBLE
  - DECIMAL128
- `arg2`: the number of places to keep after the decimal point. It supports the INT data type.

Return value

Returns a value of the same data type as `arg1`.

Examples

```
mysql> select truncate(3.14,1);
+------------------+
| truncate(3.14, 1) |
+------------------+
|              3.1 |
```

```
+--------------------+
1 row in set (0.00 sec)
```

append_trailing_char_if_absent

Description

ascii

Description

char

Description

char_length

Description

character_length

Description

concat

Description

concat_ws

Description

ends_with

Description

find_in_set

Description

group_concat

Description

hex

Description

instr

Description

lcase

Description

left

Description

length

Description

locate

Description

lower

Description

lpad

Description

ltrim

Description

money_format

Description

null_or_empty

Description

parse_url

Description

repeat

Description

reverse

Description

right

Description

rpad

Description

rtrim

Description

space

Description

split

Description

split_part

Description

starts_with

Description

strleft

Description

strright

Description

substring, substr

Description

trim

Description

ucase

Description

unhex

Description

upper

Description

append_trailing_char_if_absent

Description

If the str string is not empty and does not contain trailing_char character in the end, it appends trailing_char character to the end. trailing_char can only contain one character. If it contains multiple characters, this function will return NULL.

Syntax

```
VARCHAR append_trailing_char_if_absent(VARCHAR str, VARCHAR
trailing_char)
```

Examples

```
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)
```

keyword

APPEND_TRAILING_CHAR_IF_ABSENT

ascii

Description

This function returns the ascii value of the leftmost character of a given string.

Syntax

```
INT ascii(VARCHAR str)
```

Examples

```
MySQL > select ascii('1');
+------------+
| ascii('1') |
+------------+
|         49 |
+------------+

MySQL > select ascii('234');
+--------------+
| ascii('234') |
+--------------+
|           50 |
+--------------+
```

keyword

ASCII

char

Description

CHAR() returns the character value of the given integer value according to the ASCII table.

Syntax

```
char(n)
```

Parameters

- `n`: integer value

Return value

Returns a VARCHAR value.

Examples

```
> select char(77);
+----------+
| char(77) |
+----------+
| M        |
+----------+
```

keyword

CHAR

char_length

Description

This function returns the length of a string. For multibytes characters, it returns the number of characters. It currently only supports utf8 coding. Note: This function is also named as character_length.

Syntax

```
INT char_length(VARCHAR str)
```

Examples

```
MySQL > select char_length("abc");
+--------------------+
| char_length('abc') |
+--------------------+
|                  3 |
+--------------------+
```

keyword

CHAR_LENGTH, CHARACTER_LENGTH

character_length

Description

This function is an alias of `char_length`, please check char_length function for detailed explanation.

keyword

CHAR_LENGTH,CHARACTER_LENGTH

concat

Description

This function combines multiple strings. If any of the parameter value is NULL, it will return NULL.

Syntax

```
VARCHAR concat(VARCHAR,...)
```

Examples

```
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                  |
+-----------------------+
```

keyword

CONCAT

concat_ws

Description

This function uses the first argument sep as the separator which combines the second with the rest to form a string. If the separator is NULL, the result is NULL. concat_ws does not skip empty strings, but it will skip NULL values.

Syntax

```
VARCHAR concat_ws(VARCHAR sep, VARCHAR str,...)
```

Examples

```
MySQL > select concat_ws("Rock", "Star", "s");
+------------------------------+
| concat_ws('Rock', 'Star', 's') |
+------------------------------+
| StarRocks                    |
+------------------------------+

MySQL > select concat_ws(NULL, "Star", "s");
+----------------------------+
| concat_ws(NULL, 'Star', 's') |
+----------------------------+
| NULL                       |
+----------------------------+
1 row in set (0.01 sec)

MySQL > StarRocks > select concat_ws("Rock", "Star", NULL, "s");
+-----------------------------------+
| concat_ws('Rock', 'Star', NULL, 's') |
+-----------------------------------+
| StarRocks                         |
```

```
+-------------------------------------+
1 row in set (0.04 sec)
```

keyword

CONCAT_WS,CONCAT,WS

ends_with

Description

Returns `true` if a string ends with a specified suffix. Otherwise, it returns `false`. If the argument is NULL, the result is NULL.

Syntax

```
BOOLEAN ENDS_WITH (VARCHAR str, VARCHAR suffix)
```

Examples

```
MySQL > select ends_with("Hello starrocks", "starrocks");
+-----------------------------------+
| ends_with('Hello starrocks', 'starrocks') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+

MySQL > select ends_with("Hello starrocks", "Hello");
+-----------------------------------+
| ends_with('Hello starrocks', 'Hello') |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

keyword

ENDS_WITH

find_in_set

Description

This function returns the position of the first str in strlist (start counting with 1). Strlist is a sting separated by commas. If it does not find any str, it returns 0. When the argument is NULL, the result is NULL.

Syntax

```
INT find_in_set(VARCHAR str, VARCHAR strlist)
```

Examples

```
MySQL > select find_in_set("b", "a,b,c");
+--------------------------+
| find_in_set('b', 'a,b,c') |
+--------------------------+
|                        2 |
+--------------------------+
```

keyword

FIND_IN_SET,FIND,IN,SET

group_concat

Description

This is an aggregate function similar to sum(). group_concat concatenates non-null values into one string, with the second argument `sep` being the separator. The second argument can also be omitted. This function usually needs to be used along with `group by`.

> Please note that strings may not be concatenated in sequence because it uses distributed computing.

Syntax

```
VARCHAR group_concat(VARCHAR str[, VARCHAR sep])
```

Parameters

- `str`: the values to concatenate. It must evaluate to VARCHAR.
- `sep`: the separator, optional. If it is not specified, a comma and a space (`, `) is used by default.

Return value

Returns a VARCHAR value.

Examples

```
CREATE TABLE IF NOT EXISTS group_concat (
    id        tinyint(4)      NULL,
    value   varchar(65533)  NULL
```

```
) ENGINE=OLAP
DISTRIBUTED BY HASH(id);

INSERT INTO group_concat VALUES
(1,'fruit'),
(2,'drinks'),
(3,null),
(4,'fruit'),
(5,'meat'),
(6,'seafood');

select * from group_concat order by id;
+------+---------+
| id   | value   |
+------+---------+
|    1 | fruit   |
|    2 | drinks  |
|    3 | NULL    |
|    4 | fruit   |
|    5 | meat    |
|    6 | seafood |
```

```
select group_concat(value) from group_concat;
+-----------------------------------+
| group_concat(value)               |
+-----------------------------------+
| meat, fruit, seafood, fruit, drinks |
+-----------------------------------+

MySQL > select group_concat(value, " ") from group_concat;
+---------------------------------+
| group_concat(value, ' ')        |
+---------------------------------+
| fruit meat fruit drinks seafood |
+---------------------------------+
```

keyword

GROUP_CONCAT,GROUP,CONCAT

hex

Description

If `x` is a numerical value, this function returns a hexadecimal string representation of the value.

If `x` is a string, this function returns a hexadecimal string representation of the string where each character in the string is converted into two hexadecimal digits.

Syntax

```
HEX(x);
```

Parameters

`x`: the string or number to convert. The supported data types are BIGINT and VARCHAR.

Return value

Returns a value of the VARCHAR type.

Examples

```
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   |
+--------------+
```

Keywords

HEX

instr

Description

This function returns the position where str first appeared in substr (start counting from 1 and measured in characters). If str is not found in substr, then this function will return 0.

Syntax

```
INT instr(VARCHAR str, VARCHAR substr)
```

Examples

```
MySQL > select instr("abc", "b");
+-----------------+
| instr('abc', 'b') |
+-----------------+
|                 2 |
+-----------------+

MySQL > select instr("abc", "d");
+-----------------+
| instr('abc', 'd') |
+-----------------+
|                 0 |
+-----------------+
```

keyword

INSTR

lcase

Description

This function converts a string to lower-case. It is analogous to the function lower.

Syntax

```
VARCHAR lcase(VARCHAR str)
```

Examples

```
mysql> SELECT lcase("AbC123");
+-----------------+
|lcase('AbC123')  |
+-----------------+
|abc123           |
+-----------------+
```

keyword

LCASE

left

Description

This function returns a specified number of characters from the left side of a given string. The unit for length: utf8 character. Note: This function is also named as strleft.

Syntax

```
VARCHAR left(VARCHAR str,INT len)
```

Examples

```
MySQL > select left("Hello starrocks",5);
+--------------------------+
| left('Hello starrocks', 5) |
+--------------------------+
| Hello                    |
+--------------------------+
```

keyword

LEFT

length

Description

This function returns the length of a string (in bytes).

Syntax

```
INT length(VARCHAR str)
```

Examples

```
MySQL > select length("abc");
+---------------+
| length('abc') |
+---------------+
|             3 |
+---------------+
```

keyword

LENGTH

locate

Description

This function is used for finding the location of a substring in a string (starting counting from 1 and measured in characters). If the third argument pos is specified, it will start to find positions of substr in strings below pos. If str is not found, it will return 0.

Syntax

```
INT locate(VARCHAR substr, VARCHAR str[, INT pos])
```

Examples

```
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 |
+-------------------------------+
```

keyword

LOCATE

lower

Description

Converts all strings in an argument to lower-case.

Syntax

```
INT lower(VARCHAR str)
```

Examples

```
MySQL > SELECT lower("AbC123");
+-----------------+
| lower('AbC123') |
+-----------------+
| abc123          |
+-----------------+
```

keyword

LOWER

lpad

Description

This function returns strings with a length of `len` (starting counting from the first syllable) in `str`. If `len` is longer than `str`, the return value is lengthened to `len` characters by adding pad characters in front of `str`. If `str` is longer than `len`, the return value is shortened to `len` characters. `len` means the length of characters, not bytes.

Syntax

```
VARCHAR lpad(VARCHAR str, INT len, VARCHAR pad)
```

Examples

```
MySQL > SELECT lpad("hi", 5, "xy");
+---------------------+
| lpad('hi', 5, 'xy') |
+---------------------+
| xyxhi               |
+---------------------+

MySQL > SELECT lpad("hi", 1, "xy");
+---------------------+
| lpad('hi', 1, 'xy') |
+---------------------+
| h                   |
+---------------------+
```

keyword

LPAD

ltrim

Description

Removes the leading spaces or specified characters from the beginning (left) of the `str` argument. Removing specified characters are supported from StarRocks 2.5.0.

Syntax

```
VARCHAR ltrim(VARCHAR str[, VARCHAR characters])
```

Parameters

`str`: required, the string to trim, which must evaluate to a VARCHAR value.

`characters`: optional, the characters to remove, which must be a VARCHAR value. If this parameter is not specified, spaces are removed from the string by default. If this parameter is set to an empty string, an error is returned.

Return value

Returns a VARCHAR value.

Examples

Example 1: Remove spaces from the beginning of the string.

```
MySQL > SELECT ltrim('   ab d');
+------------------+
| ltrim('   ab d') |
+------------------+
```

```
| ab d              |
+-------------------+
```

Example 2: Remove specified characters from the beginning of the string.

```
MySQL > SELECT ltrim("xxabcdxx", "x");
+-----------------------+
| ltrim('xxabcdxx', 'x') |
+-----------------------+
| abcdxx                |
+-----------------------+
```

References

- trim
- rtrim

keyword

LTRIM

money_format

Description

This function returns a string formatted as a currency string. The integer part is separated by comma every three bits and the decimal part is reserved for two bits.

Syntax

```
VARCHAR money_format(Number)
```

Examples

```
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             |
+---------------------+
```

keyword

MONEY_FORMAT,MONEY,FORMAT

null_or_empty

Description

This function returns true when the string is empty or NULL. Otherwise, it returns false.

Syntax

```
BOOLEAN NULL_OR_EMPTY (VARCHAR str)
```

Examples

```
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 |
+--------------------+
```

keyword

NULL_OR_EMPTY

parse_url

Description

Parses a URL and extracts a component from this URL.

Syntax

```
parse_url(expr1,expr2);
```

Parameters

`expr1`: the URL. The supported data type is VARCHAR.

`expr2`: the component to extract from this URL. The supported data type is VARCHAR. Valid values:

- PROTOCOL
- HOST
- PATH
- REF
- AUTHORITY
- FILE
- USERINFO
- QUERY. Parameters in QUERY cannot be returned. If you want to return specific parameters, use parse_url with trim to achieve this implementation. For details, see Examples.

`expr2` is **case-sensitive**.

Return value

Returns a value of the VARCHAR type. If the URL is invalid, an error is returned. If the requested information cannot be find, NULL is returned.

Examples

```
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
|
+-----------------------------------------------------------------------
-----------+
```

repeat

Description

This function repeats `str` by a number of times according to `count`. When `count` is below 1, it returns an empty string. When `str` or `count` is NULL, it returns NULL.

Syntax

```
VARCHAR repeat(VARCHAR str, INT count)
```

Examples

```
MySQL > SELECT repeat("a", 3);
+---------------+
| repeat('a', 3) |
+---------------+
| aaa            |
+---------------+

MySQL > SELECT repeat("a", -1);
+----------------+
| repeat('a', -1) |
+----------------+
|                 |
+----------------+
```

keyword

REPEAT,

reverse

Description

Reverses a string or array. Returns a string or array with the characters in the string or array elements in reverse order.

Syntax

```
reverse(param)
```

Parameters

`param`: the string or array to reverse. It can be of the VARCHAR, CHAR, or ARRAY type.

Currently, this function supports only one-dimensional arrays and the array elements cannot be of the DECIMAL type. This function supports the following types of array elements: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE, and JSON. **JSON is supported from 2.5.**

Return value

The return type is the same as `param`.

Examples

Example 1: Reverse a string.

```
MySQL > SELECT REVERSE('hello');
+------------------+
| REVERSE('hello') |
```

```
+------------------+
| olleh            |
+------------------+
1 row in set (0.00 sec)
```

Example 2: Reverse an array.

```
MYSQL> SELECT REVERSE([4,1,5,8]);
+--------------------+
| REVERSE([4,1,5,8]) |
+--------------------+
| [8,5,1,4]          |
+--------------------+
```

right

Description

This function returns a specified length of characters from the right side of a given string. Length unit: utf8 character. Note: This function is also named as strright.

Syntax

```
VARCHAR right(VARCHAR str,INT len)
```

Examples

```
MySQL > select right("Hello starrocks",9);
+----------------------------+
| right('Hello starrocks', 9) |
+----------------------------+
| starrocks                  |
+----------------------------+
```

keyword

RIGHT

rpad

Description

This function returns strings with a length of `len` (starting counting from the first syllable) in `str`. If `len` is longer than `str`, the return value is lengthened to `len` characters by adding pad characters behind `str`. If `str` is longer than `len`, the return value is shortened to `len` characters. `len` means the length of characters, not bytes.

Syntax

```
VARCHAR rpad(VARCHAR str, INT len, VARCHAR pad)
```

Examples

```
MySQL > SELECT rpad("hi", 5, "xy");
+--------------------+
| rpad('hi', 5, 'xy') |
+--------------------+
| hixyx               |
+--------------------+

MySQL > SELECT rpad("hi", 1, "xy");
+--------------------+
| rpad('hi', 1, 'xy') |
+--------------------+
| h                   |
+--------------------+
```

keyword

RPAD

rtrim

Description

Removes the trailing spaces or specified characters from the end (right) of the `str` argument. Removing specified characters are supported from StarRocks 2.5.0.

Syntax

```
VARCHAR rtrim(VARCHAR str[, VARCHAR characters]);
```

Parameters

`str`: required, the string to trim, which must evaluate to a VARCHAR value.

`characters`: optional, the characters to remove, which must be a VARCHAR value. If this parameter is not specified, spaces are removed from the string by default. If this parameter is set to an empty string, an error is returned.

Return value

Returns a VARCHAR value.

Examples

Example 1: Remove the three trailing spaces from the string.

```
mysql> SELECT rtrim('   ab d   ');
+--------------------+
| rtrim('   ab d   ') |
+--------------------+
```

```
|    ab d             |
+---------------------+
1 row in set (0.00 sec)
```

Example 2: Remove specified characters from the end of the string.

```
MySQL > SELECT rtrim("xxabcdxx", "x");
+-----------------------+
| rtrim('xxabcdxx', 'x') |
+-----------------------+
| xxabcd                 |
+-----------------------+
```

References

- trim
- ltrim

space

Description

Returns a string of the specified number of spaces.

Syntax

```
space(x);
```

Parameters

`x`: the number of spaces to return. The supported data type is INT.

Return value

Returns a value of the VARCHAR type.

Examples

```
mysql> select space(6);
+----------+
| space(6) |
+----------+
|          |
+----------+
1 row in set (0.00 sec)
```

Keywords

SPACE

split

Description

This function splits a given string according to the separators, and returns the split parts in ARRAY.

Syntax

```
ARRAY<VARCHAR> split(VARCHAR content, VARCHAR delimiter)
```

Examples

```
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

Description

This function splits a given string according to the separators and returns the requested part. (start counting from the beginning)

Syntax

```
VARCHAR split_part(VARCHAR content, VARCHAR delimiter, INT field)
```

Examples

```
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("abca", "a", 1);
+--------------------------+
| split_part('abca', 'a', 1) |
+--------------------------+
|                          |
+--------------------------+
```

keyword

SPLIT_PART,SPLIT,PART

starts_with

Description

This function returns 1 when a string starts with a specified prefix. Otherwise, it returns 0. When the argument is NULL, the result is NULL.

Syntax

```
BOOLEAN starts_with(VARCHAR str, VARCHAR prefix)
```

Examples

```
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

strleft

Description

This function extracts a number of characters from a string with specified length (starting from left). The unit for length: utf8 character. Note: This function is also named as left.

Syntax

```
VARCHAR strleft(VARCHAR str,INT len)
```

Examples

```
MySQL > select strleft("Hello starrocks",5);
+------------------------------+
| strleft('Hello starrocks', 5) |
+------------------------------+
| Hello                        |
+------------------------------+
```

keyword

STRLEFT

strright

Description

This function extracts a number of characters from a string with specified length (starting from right). The unit for length: utf-8 character. Note: This function is also named as right.

Syntax

```
VARCHAR strright(VARCHAR str,INT len)
```

Examples

```
MySQL > select strright("Hello starrocks",9);
+------------------------------+
| strright('Hello starrocks', 9) |
+------------------------------+
| starrocks                    |
+------------------------------+
```

keyword

STRRIGHT

substring, substr

Description

Extracts characters from a string and returns a substring.

If `len` is not specified, this function extracts characters from the position specified by `pos`. If `len` is specified, this function extracts `len` characters from the position specified by `pos`.

`pos` can be a negative integer. In this case, this function extracts characters starting from the end of the string.

Syntax

```
VARCHAR substr(VARCHAR str, pos[, len])
```

Parameter values

- `str`: the string to extract characters, required. It must be a VARCHAR value.
- `pos`: the start position, required. The first position in the string is 1.
- `length`: the number of characters to extract, optional. It must be a positive integer.

Return value

Returns a value of the VARCHAR type.

If the number of characters (`len`) to return exceeds the actual length of the matching characters, all the matching characters are returned.

If the position specified by `pos` exceeds the range of the string, an empty string is returned.

Examples

```
MySQL > select substring("starrockscluster", 1, 9);
+-----------------------------------+
| substring('starrockscluster', 1, 9) |
+-----------------------------------+
| starrocks                         |
+-----------------------------------+

MySQL > select substring("starrocks", -5, 5);
+-----------------------------+
| substring('starrocks', -5, 5) |
+-----------------------------+
| rocks                       |
+-----------------------------+

MySQL > select substring("apple", 1, 9);
+--------------------------+
| substring('apple', 1, 9) |
+--------------------------+
| apple                    |
+--------------------------+
```

keyword

substring,string,sub

trim

Description

Removes consecutive spaces or specified characters from the beginning and end of the `str` argument. Removing specified characters are supported from StarRocks 2.5.0.

Syntax

```
VARCHAR trim(VARCHAR str[, VARCHAR characters])
```

Parameters

`str`: required, the string to trim, which must evaluate to a VARCHAR value.

`characters`: optional, the characters to remove, which must be a VARCHAR value. If this parameter is not specified, spaces are removed from the string by default. If this parameter is set to an empty string, an error is returned.

Return value

Returns a VARCHAR value.

Examples

Example 1: Remove the five spaces from the beginning and end of the string.

```
MySQL > SELECT trim("   ab c  ");
+------------------+
| trim('   ab c  ') |
+------------------+
```

```
| ab c              |
+-------------------+
1 row in set (0.00 sec)
```

Example 2: Remove specified characters from the beginning and end of the string.

```
MySQL > SELECT trim("abcd", "ad");
+--------------------+
| trim('abcd', 'ad') |
+--------------------+
| bc                 |
+--------------------+

MySQL > SELECT trim("xxabcdxx", "x");
+----------------------+
| trim('xxabcdxx', 'x') |
+----------------------+
| abcd                 |
+----------------------+
```

References

- ltrim
- rtrim

ucase

Description

This function converts a string to upper-case. It is analogous to the function upper.

Syntax

```
VARCHAR ucase(VARCHAR str)
```

Examples

```
mysql> SELECT ucase("AbC123");
+-----------------+
|ucase('AbC123')  |
+-----------------+
|ABC123           |
+-----------------+
```

keyword

UCASE

unhex

Description

This function performs the opposite operation of hex().

It interprets each pair of hexadecimal digits in the input string as a number and converts it to the byte represented by the number. The return value is a binary string.

Syntax

```
UNHEX(str);
```

Parameters

`str`: the string to convert. The supported data type is VARCHAR. An empty string is returned if any of the following situations occur:

- The length of the string is 0 or the number of characters in the string is an odd number.
- The string contains characters other than `[0-9]`, `[a-z]`, and `[A-Z]`.

Return value

Returns a value of the VARCHAR type.

Examples

```
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

Description

Converts a string to upper-case.

Syntax

```
upper(str)
```

Parameters

- `str`: the string to convert. If `str` is not a string type, it will try implicit cast first.

Return value

Return an upper-case string.

Examples

```
MySQL [test]> select C_String, upper(C_String) from ex_iceberg_tbl;
+------------------+------------------+
| C_String         | upper(C_String)  |
+------------------+------------------+
| Hello, StarRocks! | HELLO, STARROCKS! |
| Hello, World!     | HELLO, WORLD!     |
+------------------+------------------+
```

Pattern Matching Functions

like

Description

regexp

Description

regexp_extract

Description

regexp_replace

Description

like

Description

Checks whether a given expression fuzzy matches the specified pattern. If yes, 1 is returned. Otherwise, 0 is returned. NULL is returned if any of the input parameter is NULL.

LIKE is usually used together with characters such as the percent sign (%) and underscore (_). `%` matches 0, 1, or more characters. `_` matches any single character.

Syntax

```
BOOLEAN like(VARCHAR expr, VARCHAR pattern);
```

Parameters

- `expr`: the string expression. The supported data type is VARCHAR.
- `pattern`: the pattern to match. The supported data type is VARCHAR.

Return value

Returns a BOOLEAN value.

Examples

```
mysql> select like("star","star");
+----------------------+
| like('star', 'star') |
+----------------------+
|                    1 |
```

```
+----------------------+

mysql> select like("starrocks","star%");
+----------------------+
| like('star', 'star') |
+----------------------+
|                    1 |
+----------------------+

mysql> select like("starrocks","star_");
+---------------------------+
| like('starrocks', 'star_') |
+---------------------------+
|                         0 |
+---------------------------+
```

regexp

Description

Checks whether a given expression matches the regular expression specified by `pattern`. If yes, 1 is returned. Otherwise, 0 is returned. NULL is returned if any of the input parameter is NULL.

regexp() supports more complex matching conditions than like().

Syntax

```
BOOLEAN regexp(VARCHAR expr, VARCHAR pattern);
```

Parameters

- `expr`: the string expression. The supported data type is VARCHAR.
- `pattern`: the pattern to match. The supported data type is VARCHAR.

Return value

Returns a BOOLEAN value.

Examples

```
mysql> select regexp("abc123","abc*");
+-------------------------+
| regexp('abc123', 'abc*') |
+-------------------------+
|                       1 |
```

```
+-------------------------+
1 row in set (0.06 sec)

select regexp("abc123","xyz*");
+-------------------------+
| regexp('abc123', 'xyz*') |
+-------------------------+
|                        0 |
+-------------------------+
```

Keywords

regexp, regular

regexp_extract

Description

This function returns the first matching substring in the target value which matches the regular expression pattern. It extracts the item in pos that matches the pattern. The pattern must completely match some parts of str so that the function can return parts needed to be matched in the pattern. If no matches are found, it will return an empty string.

Syntax

```
VARCHAR regexp_extract(VARCHAR str, VARCHAR pattern, int pos)
```

Examples

Given this data:

```
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)
```

Return the two sets of digits following the string `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)
```

Return only the second set of digits following the string `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)
```

TIP

The same result could be returned by not wrapping the first set of digits in a matching group `()` and returning the first group:

```
regexp_extract(HourlySkyConditions, 'OVC:\\d+ (\\d+)', 1)
```

keyword

REGEXP_EXTRACT,REGEXP,EXTRACT

regexp_replace

Description

This function uses repl to replace a sequence of characters in str that matches a regular expression pattern.

Syntax

```
VARCHAR regexp_replace(VARCHAR str, VARCHAR pattern, VARCHAR repl)
```

Examples

```
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

percentile_approx_raw

Description

percentile_empty

Description

percentile_hash

Description

percentile_union

Description

percentile_approx_raw

Description

Returns the value that corresponds to a specified percentile from `x`.

If `x` is a column, this function first sorts values in `x` in ascending order and returns the value that corresponds to percentile `y`.

Syntax

```
PERCENTILE_APPROX_RAW(x, y);
```

Parameters

- `x`: It can be a column or a set of values. It must evaluate to PERCENTILE.
- `y`: the percentile. The supported data type is DOUBLE. Value range: [0.0,1.0].

Return value

Returns a PERCENTILE value.

Examples

Create table `aggregate_tbl`, where the `percent` column is the input of 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`) BUCKETS 8
PROPERTIES ("replication_num" = "1");
```

Insert data into the table.

```
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));
```

Calculate the value corresponding to percentile 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_empty

Description

Constructs a PERCENTILE value, which is used to fill in null values for data loading using Stream Load or INSERT INTO.

Syntax

```
PERCENTILE_EMPTY();
```

Parameters

None

Return value

Returns a PERCENTILE value.

Examples

Create a table. The `percent` column is a PERCENTILE column.

```
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`) BUCKETS 8
PROPERTIES ("replication_num" = "3");
```

Insert data into the table.

```
INSERT INTO aggregate_tbl VALUES
(5, '2020-02-23', 'city_code', 555, percentile_empty());
```

percentile_hash

Description

Constructs DOUBLE values as PERCENTILE values.

Syntax

```
PERCENTILE_HASH(x);
```

Parameters

`x`: The supported data type is DOUBLE.

Return value

Returns a PERCENTILE value.

Examples

```
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

Description

Aggregates PERCENTILE data.

Syntax

```
percentile_union(expr);
```

Parameters

`expr`: The supported data type is PERCENTILE.

Return value

Returns a PERCENTILE value.

Examples

Example 1: Use percentile data in materialized views.

Create a table.

```
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");
```

Create a materialized view based on the `sale_amt` column of the table.

```
create materialized view mv as
select store_id, percentile_union(percentile_hash(sale_amt))
from sales_records
group by store_id;
```

Example 2: Load PERCENTILE data.

Create an aggregate table that contains a PERCENTILE column `sale_amt_per`.

```
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`) BUCKETS 3
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);
```

Query data from `sale_amt_per`.

```
select percentile_approx_raw(percentile_union(sale_amt_per), 0.99)
from sales_records;
```

Load data that contains PERCENTILE data into the `sales_records` table.

```
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
```

hll_cardinality

Description

hll_cardinality

Description

Calculates the cardinality of a single HLL type value.

Syntax

```
HLL_CARDINALITY(hll)
```

Examples

```
MySQL > select HLL_CARDINALITY(uv_set) from test_uv;
+--------------------------+
| hll_cardinality(`uv_set`) |
+--------------------------+
|                        3 |
+--------------------------+
```

keyword

HLL,HLL_CARDINALITY

catalog

Description

current_version

Description

database

Description

host_name

Description

isnull

Description

last_query_id

Description

sleep

Description

uuid

Description

uuid_numeric

Description

version

Description

catalog

Description

Returns the name of the current catalog. The catalog can be a StarRocks internal catalog or an external catalog that is mapped to an external data source. For more information about catalogs, see Catalog overview.

If no catalog is selected, the StarRocks internal catalog `default_catalog` is returned.

Syntax

```
catalog()
```

Parameters

This function does not require parameters.

Return value

Returns the name of the current catalog as a string.

Examples

Example 1: The current catalog is StarRocks internal catalog `default_catalog`.

```
select catalog();
+-----------------+
| CATALOG()       |
+-----------------+
| default_catalog |
```

```
+-----------------+
1 row in set (0.01 sec)
```

Example 2: The current catalog is an external catalog `hudi_catalog`.

```
-- Switch to an external catalog.
set catalog hudi_catalog;

-- Return the name of the current catalog.
select catalog();
+--------------+
| CATALOG()    |
+--------------+
| hudi_catalog |
+--------------+
```

current_version

Description

Returns the current version of StarRocks. Two syntaxes are provided for compatibility with different clients.

Syntax

```
current_version();

@@version_comment;
```

Parameters

None

Return value

Returns a value of the VARCHAR type.

Examples

```
mysql> select current_version();
+-------------------+
| current_version() |
+-------------------+
| 2.1.2 0782ad7     |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@version_comment;
```

```
+-------------------------+
| @@version_comment       |
+-------------------------+
| StarRocks version 2.1.2 |
+-------------------------+
1 row in set (0.01 sec)
```

database

Description

Returns the name of the current database. If no database is selected, an empty value is returned.

Syntax

```
database()
```

Parameters

This function does not require parameters.

Return value

Returns the name of the current database as a string.

Examples

```
-- Select a destination database.
use db_test

-- Query the name of the current database.
select database();
+------------+
| DATABASE() |
+------------+
| db_test    |
+------------+
```

See also

USE: Switches to a destination database.

host_name

Description

Obtains the hostname of the node on which the computation is performed.

Syntax

```
host_name();
```

Parameters

None

Return value

Returns a VARCHAR value.

Examples

```
select host_name();
+-------------+
| host_name() |
+-------------+
| sandbox-sql |
+-------------+
1 row in set (0.01 sec)
```

isnull

Description

Checks whether the value is `NULL`, returns `1` if it is `NULL`, and returns `0` if it is not `NULL`.

Syntax

```
ISNULL(v)
```

Parameters

- `v`: the value to check. All date types is supported.

Return value

Returns 1 if it is `NULL`, and returns 0 if it is not `NULL`.

Example

```
MYSQL > SELECT c1, isnull(c1) FROM t1;
+------+--------------+
| c1   | `c1` IS NULL |
+------+--------------+
| NULL |            1 |
|    1 |            0 |
+------+--------------+
```

last_query_id

Description

Obtains the ID of the most recently executed query in the current session.

Syntax

```
VARCHAR last_query_id();
```

Parameters

None

Return value

Returns a value of the VARCHAR type.

Examples

```
mysql> select last_query_id();
+--------------------------------------+
| last_query_id()                      |
+--------------------------------------+
| 7c1d8d68-bbec-11ec-af65-00163e1e238f |
+--------------------------------------+
1 row in set (0.00 sec)
```

Keywords

LAST_QUERY_ID

sleep

Description

Delays the execution of an operation for a specified period of time (in seconds) and returns a BOOLEAN value to indicate whether the sleep is completed without interruption. `1` is returned if the sleep is completed without interruption. Otherwise, `0` is returned.

Syntax

```
BOOLEAN sleep(INT x);
```

Parameters

`x`: the duration for which you want to delay the execution of an operation. It must be of the INT type. Unit: seconds. If the input is is NULL, NULL is returned immediately without sleeping.

Return value

Returns a value of the BOOLEAN type.

Examples

```
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)
```

Keywords

SLEEP, sleep

uuid

Description

Returns a random UUID of the VARCHAR type. Two calls to this function can generate two different numbers. The UUID is 36 characters in length. It contains 5 hexadecimal numbers which are connected with four hyphens in the aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee format.

Syntax

```
uuid();
```

Parameters

None

Return value

Returns a value of the VARCHAR type.

Examples

```
mysql> select uuid();
+--------------------------------------+
| uuid()                               |
+--------------------------------------+
| 74a2ed19-9d21-4a99-a67b-aa5545f26454 |
+--------------------------------------+
1 row in set (0.01 sec)
```

uuid_numeric

Description

Returns a random UUID of the LARGEINT type. This function has an execution performance 2 orders of magnitude better than the `uuid` function.

Syntax

```
uuid_numeric();
```

Parameters

None

Return value

Returns a value of the LARGEINT type.

Examples

```
MySQL > select uuid_numeric();
+--------------------------+
| uuid_numeric()           |
+--------------------------+
| 558712445286367898661205 |
+--------------------------+
1 row in set (0.00 sec)
```

version

Description

Returns the current version of the MySQL database.

You can use current_version to query StarRocks version.

Syntax

```
VARCHAR version();
```

Parameters

None

Return value

Returns a value of the VARCHAR type.

Examples

```
mysql> select version();
+-----------+
| version() |
+-----------+
| 5.1.0     |
+-----------+
1 row in set (0.00 sec)
```

References

current_version

CAST

Description

Converts an input into the specified type. For example, `cast (input as BIGINT)` converts the input into a BIGINT value.

From v2.4, StarRocks supports conversion to the ARRAY type.

Syntax

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

Parameters

`input`: the data you want to convert. `type`: the destination data type.

Return value

Returns a value whose data type is the same as `type`.

Examples

Example 1: common data conversions

```
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 |
+------------------+
```

Example 2: Convert the input into ARRAY.

```
-- Convert string to 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"]                                   |
+-------------------------------------------------+

-- Convert JSON array to 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"]                                               |
+-------------------------------------------------------------+
```

Example 3: Convert data during loading.

```
    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
```

> **Note**
>
> If the original value is a floating-point value (such as 12.0), it will be converted to NULL.
> If you want to forcibly convert this type into BIGINT, see the following example:

```
    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 |
    +----------------------------------------+
```

murmur_hash3_32

Description

Returns the 32-bit murmur3 hash value of the input string.

Syntax

```
INT MURMUR_HASH3_32(VARCHAR input, ...)
```

Examples

```
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 |
+-----------------------------------+
```

keyword

MURMUR_HASH3_32,HASH

System variables

StarRocks provides many system variables that can be set and modified to suit your requirements. This section describes the variables supported by StarRocks. You can view the settings of these variables by running the SHOW VARIABLES command on your MySQL client. You can also use the SET command to dynamically set or modify variables. You can make these variables take effect globally on the entire system, only in the current session, or only in a single query statement.

The variables in StarRocks refer to the variable sets in MySQL, but **some variables are only compatible with the MySQL client protocol and do not function on the MySQL database**.

> **NOTE**
>
> Any user has the privilege to run SHOW VARIABLES and make a variable take effect at session level. However, only users with the SYSTEM-level OPERATE privilege can make a variable take effect globally. Globally effective variables take effect on all future sessions (excluding the current session).
>
> If you want to make a setting change for the current session and also make that setting change apply to all future sessions, you can make the change twice, once without the `GLOBAL` modifier and once with it. For example:
>
> ```
> SET query_mem_limit = 137438953472; -- Apply to the current session.
> SET GLOBAL query_mem_limit = 137438953472; -- Apply to all future sessions.
> ```

Variable hierarchy and types

StarRocks supports three types (levels) of variables: global variables, session variables, and `SET_VAR` hints. Their hierarchical relationship is as follows:

- Global variables take effect on global level, and can be overridden by session variables and `SET_VAR` hints.
- Session variables take effect only on the current session, and can be overridden by `SET_VAR` hints.
- `SET_VAR` hints take effect only on the current query statement.

View variables

You can view all or some variables by using `SHOW VARIABLES [LIKE 'xxx']`. Example:

```
-- Show all variables in the system.
SHOW VARIABLES;

-- Show variables that match a certain pattern.
SHOW VARIABLES LIKE '%time_zone%';
```

Set variables

Set variables globally or for a single session

You can set variables to take effect **globally** or **only on the current session**. When set to global, the new value will be used for all future sessions, while the current session still uses the original value. When set to "current session only", the variable will only take effect on the current session.

A variable set by `SET <var_name> = xxx;` only takes effect for the current session. Example:

```
SET exec_mem_limit = 137438953472;

SET forward_to_master = true;

SET time_zone = "Asia/Shanghai";
```

A variable set by `SET GLOBAL <var_name> = xxx;` takes effect globally. Example:

```
SET GLOBAL exec_mem_limit = 137438953472;
```

The following variables only take effect globally. They cannot take effect for a single session, which means you must use `SET GLOBAL <var_name> = xxx;` for these variables. If you try to set such a variable for a single session (`SET <var_name> = xxx;`), an error is returned.

- 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

In addition, variable settings also support constant expressions, such as:

```
SET exec_mem_limit = 10 * 1024 * 1024 * 1024;
```

```
SET forward_to_master = concat('tr', 'u', 'e');
```

Set variables in a single query statement

In some scenarios, you may need to set variables specifically for certain queries. By using the `SET_VAR` hint, you can set session variables that will take effect only within a single statement. Example:

```
SELECT /*+ SET_VAR(exec_mem_limit = 8589934592) */ name FROM people
ORDER BY name;

SELECT /*+ SET_VAR(query_timeout = 1) */ sleep(3);
```

> **NOTE**
>
> `SET_VAR` can only be placed after the `SELECT` keyword and enclosed in `/*+...*/`.

You can also set multiple variables in a single statement. Example:

```
SELECT /*+ SET_VAR
  (
  exec_mem_limit = 515396075520,
  query_timeout=10000000,
  batch_size=4096,
  parallel_fragment_exec_instance_num=32
  )
  */ * FROM TABLE;
```

Descriptions of variables

The variables are described **in alphabetical order**. Variables with the `global` label can only take effect globally. Other variables can take effect either globally or for a single session.

auto_increment_increment

Used for MySQL client compatibility. No practical usage.

autocommit

Used for MySQL client compatibility. No practical usage.

chunk_size

Used to specify the number of rows of a single packet transmitted by each node during query execution. The default is 4096, i.e., every 4096 rows of data generated by the source node is packaged and sent to the destination node. A larger number of rows will improve the query throughput in large data volume scenarios, but may increase the query latency in small data volume scenarios. Also, it may increase the memory overhead of the query. We recommend to set `batch_size` between 1024 to 4096.

cbo_enable_low_cardinality_optimize

Whether to enable low cardinality optimization. After this feature is enabled, the performance of querying STRING columns improves by about three times. Default value: true.

cbo_eq_base_type (2.5.14 and later)

Specifies the data type used for data comparison between DECIMAL data and STRING data. The default value is `VARCHAR`, and DECIMAL is also a valid value.

enable_materialized_view_union_rewrite (2.5.20 and later)

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. The default value is `true`.

enable_materialized_view_plan_cache (2.5.13 and later)

Whether to enable materialized view plan cache, which can optimize the automatic rewrite performance of materialized views. The default value `true` indicates enabling it.

follower_query_forward_mode (2.5.20 and later)

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.

character_set_database (global)

The character set supported by StarRocks. Only UTF8 (`utf8`) is supported.

connector_io_tasks_per_scan_operator (2.5 and later)

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. Default value: 16.

Currently, StarRocks 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.

count_distinct_column_buckets (2.5 and later)

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`. Default value: 1024.

default_rowset_type (global)

Global variable. 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`.

disable_colocate_join

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.

disable_streaming_preaggregations

Used to enable the streaming pre-aggregations. The default value is `false`, meaning it is enabled.

div_precision_increment

Used for MySQL client compatibility. No practical usage.

enable_connector_adaptive_io_tasks (2.5 and later)

Whether to adaptively adjust the number of concurrent I/O tasks when querying external tables. Default value: 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`.

enable_distinct_column_bucketization (2.5 and later)

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.

Default value: false, which means this feature is disabled. 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;`.

enable_insert_strict

Used to enable the strict mode when loading data using the INSERT statement. The default value is `true`, indicating the strict mode is enabled by default. For more information, see Strict mode.

enable_materialized_view_for_insert

- Description: Whether to allow StarRocks to rewrite queries in INSERT INTO SELECT statements.
- Default: false, which means Query Rewrite in such scenarios is disabled by default.
- Introduced in: v2.5.18, v3.0.9, v3.1.7, v3.2.2

enable_materialized_view_union_rewrite (2.5 and later)

Boolean value to control whether to enable materialized view Union query rewrite. Default: `true`.

enable_rule_based_materialized_view_rewrite (2.5 and later)

Boolean value to control whether to enable rule-based materialized view query rewrite. This variable is mainly used in single-table query rewrite. Default: `true`.

enable_strict_order_by

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 v2.3 and earlier. When this variable is set to `FALSE`, a loose deduplication mechanism is used, which processes such queries as valid SQL queries.

This variable is supported from v2.5.18 and v3.1.7.

enable_profile

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.

enable_query_queue_load (global)

Boolean value to enable query queues for loading tasks. Default: `false`.

enable_query_queue_select (global)

Boolean value to enable query queues for SELECT queries. Default: `false`.

enable_query_queue_statistic (global)

Boolean value to enable query queues for statistics queries.

enable_scan_block_cache (2.5 and later)

Specifies whether to enable the Data Cache feature. After this feature is enabled, StarRocks caches hot data read from external storage systems into blocks, which accelerates queries and analysis. For more information, see Data Cache.

enable_populate_block_cache (2.5 and later)

Specifies whether to cache data blocks read from external storage systems in StarRocks. If you do not want to cache data blocks read from external storage systems, set this variable to `false`. Default value: `true`. This variable is supported from 2.5.

enable_tablet_internal_parallel (2.3 and later)

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. Default value: `true`.

enable_query_cache (2.5 and later)

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.

enable_adaptive_sink_dop (2.5 and later)

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 v2.5 StarRocks cluster, the value is `true` by default. For a v2.5 cluster upgraded from v2.4, the value is `false`.

enable_pipeline_engine

Specifies whether to enable the pipeline execution engine. `true` indicates enabled and `false` indicates the opposite. Default value: `true`.

enable_sort_aggregate (2.5 and later)

Specifies whether to enable sorted streaming. `true` indicates sorted streaming is enabled to sort data in data streams.

enable_query_tablet_affinity （2.5 and later)

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.

Default value: `false`, which means the system selects a replica for each query. This feature is supported since 2.5.6, 3.0.8, and 3.1.4.

tablet_internal_parallel_mode

- **Description**: 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.
- **Default**: auto
- **Data type**: String
- **Introduced in**: v2.5.0

enable_scan_datacache

- **Description**: Specifies whether to enable the Data Cache feature. After this feature is enabled, StarRocks 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`.
- **Default**: false
- **Introduced in**: v2.5

enable_populate_datacache

- **Description**: Specifies whether to cache data blocks read from external storage systems in StarRocks. If you do not want to cache data blocks read from external storage systems, set this variable to `false`. Default value: true. This variable is supported from 2.5. In versions prior to 3.2, this variable was named as `enable_scan_block_cache`.
- **Default**: true
- **Introduced in**: v2.5

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.

StarRocks 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.

event_scheduler

Used for MySQL client compatibility. No practical usage.

exec_mem_limit

Used to set the memory limit that can be used by a single query plan instance. The default value is 2GB, and the default unit is B. `B/K/KB/M/MB/G/GB/T/TB/P/PB` are supported.

There may be multiple instances of a query plan, and a BE node may execute one or more instances. Therefore, this parameter does not accurately limit the memory usage of a query across the cluster, nor does it accurately limit the memory usage of a query on a single BE node. It needs to be evaluated based on the generated query plan.

Usually, more memory is consumed on blocking nodes (e.g. Sort Node, Aggregate Node, Join Node). Unblocking nodes (e.g. Scan Node) transmit data in stream, which does not take up too much memory.

When there is a `Memory Exceed Limit` error, try to increase this parameter.

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_master

Used to specify whether some commands will be forwarded to the leader FE for execution. The default value is `false`, meaning not forwarding to the leader FE. There are multiple FEs in a StarRocks 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.

hash_join_push_down_right_table

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.

`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)

Used for MySQL client compatibility. No practical usage.

interactive_timeout

Used for MySQL client compatibility. No practical usage.

io_tasks_per_scan_operator (2.5 and later)

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.

The value is an integer. Default value: 4.

language (global)

Used for MySQL client compatibility. No practical usage.

license (global)

Displays the license of StarRocks.

load_mem_limit

Specifies the memory limit for the import operation. The default value is 0, meaning that this variable is not used and `exec_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.

lower_case_table_names (global)

Used for MySQL client compatibility. No practical usage. Table names in StarRocks are case-sensitive.

max_allowed_packet

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. Default value: 32 MB. Unit: Byte. You can raise this value if the client reports "PacketTooBigException".

max_scan_key_num

The maximum number of scan key segmented by each query. Default value: -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.

max_pushdown_conditions_per_column

The maximum number of predicates that can be pushed down for a column. Default value: -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

The maximum levels of nested materialized views that can be used for query rewrite. Type: INT. Range: [1, +∞). The value of `1` indicates that only materialized views created on base tables can be used for query rewrite. Default: `3`.

net_buffer_length

Used for MySQL client compatibility. No practical usage.

net_read_timeout

Used for MySQL client compatibility. No practical usage.

net_write_timeout

Used for MySQL client compatibility. No practical usage.

new_planner_optimize_timeout

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 StarRocks technical support for troubleshooting. A timeout often occurs when a query has too many joins.

Default value: 3000. Unit: 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.

performance_schema

Used for compatibility with MySQL JDBC versions 8.0.16 and above. No practical usage.

prefer_compute_node

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.

pipeline_dop

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. You can also set this variable to a value greater than 0. Generally, set the value to half the number of physical CPU cores.

query_cache_entry_max_bytes (2.5 and later)

The threshold for triggering the Passthrough mode. Valid values: 0 to 9223372036854775807. 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.

query_cache_entry_max_rows (2.5 and later)

The upper limit of rows that can be cached. See the description in `query_cache_entry_max_bytes`. Default value: 409600.

query_cache_agg_cardinality_limit (2.5 and later)

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.

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

Used to set the memory limit of a query on each BE node. Unit: Byte. 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.

query_queue_concurrency_limit (global)

The upper limit of concurrent queries on a BE. It takes effect only after being set greater than `0`. Default: `0`.

query_queue_cpu_used_permille_limit (global)

The upper limit of CPU usage permille (CPU usage * 1000) on a BE. It takes effect only after being set greater than `0`. Default: `0`. Range: [0, 1000]

query_queue_max_queued_queries (global)

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`. Default: `1024`.

query_queue_mem_used_pct_limit (global)

The upper limit of memory usage percentage on a BE. It takes effect only after being set greater than `0`. Default: `0`. Range: [0, 1]

query_queue_pending_timeout_second (global)

The maximum timeout of a pending query in a queue. When this threshold is reached, the corresponding query is rejected. Unit: second. Default: `300`.

query_timeout

Used to set the query timeout in "seconds". This variable will act on all query statements in the current connection, as well as INSERT statements. The default value is 300 seconds. Value range: [1, 259200].

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

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.

runtime_join_filter_push_down_limit

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.

The value is an integer. Default value: 1024000.

SQL_AUTO_IS_NULL

Used for compatibility with the JDBC connection pool C3P0. No practical usage.

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.

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

Used for MySQL client compatibility. No practical usage.

sql_select_limit

Used for MySQL client compatibility. No practical usage.

storage_engine

The types of engines supported by StarRocks:

- olap: StarRocks 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 v2.1.
- hudi: Hudi external tables, supported from v2.2.
- jdbc: external table for JDBC-compatible databases, supported from v2.3.

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.

tx_isolation

Used for MySQL client compatibility. No practical usage.

use_compute_nodes

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.

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 v2.4 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 v2.4 onwards and therefore, is deprecated.

version (global)

The MySQL server version returned to the client.

version_comment (global)

The StarRocks version. Cannot be changed.

wait_timeout

The number of seconds the server waits for activity on a noninteractive connection before closing it. If a client does not interact with StarRocks for this length of time, StarRocks will actively close the connection.

Unit: seconds. Default value: 28800 (8 hours).

User-defined variables

This topic describes how to declare and use user-defined variables.

StarRocks 2.5 and later versions support user-defined variables. These variables are used to store specific values that are referenced in later SQL statements, thereby simplifying the writing of SQL statements and avoiding duplicate computation.

Usage notes

- User-defined variables are variables, which can be created by the user and exist in the session. This means that no one can access user-defined variables that have been declared by another user, and when the session is closed these variables expire.
- StarRocks does not support using the SHOW statement to display existing user-defined variables.
- The following types of values cannot be declared as user-defined variables: BITMAP, HLL, PERCENTILE, and ARRAY. User-defined variables of the JSON type are converted to the STRING type for storage.

Declare user-defined variables

Syntax

```
SET @var_name = expr [, ...];
```

> **NOTE**
>
> - All variables must be preceded by a single at sign (@).
> - Multiple variables can be declared in the same SET statement and need to be separated with commas (`,`).
> - You can declare the same variable multiple times. The newly declared value overwrites the original value.

- If an undeclared variable is used, the value of the variable is `NULL` by default, and the NULL type is STRING.

Parameters

Parameter	Required	Description
var_name	Yes	The name of the user-defined variable. The naming conventions are as follows: • The name can contain letters, digits (0-9), and underscores (_). • The name cannot exceed 64 characters in length. You can declare a string as a user-defined variable, such as @'my-var', @"my-var", and @`my-var`. User-defined variables of the STRING type can contain characters other than letters, digits, and underscores (_), such as periods (.).
expr	Yes	The value of the user-defined variable. You can specify a number (such as 43) or a complex expression (such as the value returned by a SELECT statement) for this parameter. The data type of the variable is the same as the data type of the result returned by the expression.

Examples

Example 1: Declare a number as a user-defined variable.

```
SET @var = 43;
```

Example 2: Declare the value returned by a SELECT query as a user-defined variable.

```
SET @var = (SELECT SUM(v1) FROM test);
```

Example 3: Declare multiple user-defined variables in the same SET statement.

```
SET @v1=1, @v2=2;
```

Use user-defined variables in SQL

- Simplifies the writing of SQL statements. For example, when you execute the following SELECT statement, StarRocks parses `@var` as `1`.

  ```
  SET @var = 1;
  SELECT @var, v1 from test;
  ```

- Avoids duplicate computation. For example, when you execute the following SELECT statement, StarRocks parses `@var` as the result returned by the `select sum(c1) from tbl` command.

  ```
  SET @var = (select sum(c1) from tbl);
  SELECT @var, v1 from test;
  ```

Error Codes

This section describes the common error codes for query requests.

Serial number	Error code	Error message
1005	Table creation failed. Returns a specific reason.	
1007	Cannot create a database with the same name.	
1008	Cannot delete non-existent database.	
1044	Cannot access unauthorized database.	
1045	Username and password do not match and therefore cannot access the system.	
1046	The target database is not specified.	
1047	An invalid command is specified.	
1049	An invalid database is specified.	
1050	A table with the same name already exists.	
1051	An invalid table is specified.	
1052	The specified column name is ambiguous and therefore the corresponding column cannot be uniquely identified.	
1053	An illegal data column was specified for the Semi-Join/Anti-Join query.	
1054	The specified column does not exist in the table.	

Serial number	Error code	Error message
1058	The number of columns selected in the query statement does not match the number of columns in the query result.	
1060	There are duplicate column names.	
1064	There is no surviving BE node.	
1066	Duplicate table alias appears in the query statement.	
1094	Thread ID is invalid.	
1095	The non-owner of a thread cannot terminate the thread.	
1096	The query statement does not specify the table to be queried.	
1102	The database name is incorrect.	
1104	The table name is incorrect.	
1105	Other errors.	
1110	Duplicate columns were specified in the subquery.	
1111	Illegal use of aggregation function in `WHERE` clause	
1113	The set of columns in the new table cannot be empty.	
1115	Unsupported character sets are used.	
1130	An unauthorized IP address is used by the client.	
1132	No permission to change user password.	

Serial number	Error code	Error message
1141	Specified entries don't have privileges to revoke.	
1142	An unauthorized action was performed.	
1166	The data column name is incorrect.	
1193	System variable has invalid name	
1203	The number of active connections used exceeded the limit.	
1211	Not allowed to create new users.	
1227	The user has performed an out-of-authority operation.	
1228	Session variables cannot be modified by the `SET GLOBAL` command.	
1229	Global variables should be modified by the `SET GLOBAL` command.	
1230	Related system variables do not have default values.	
1231	An invalid value was set for a system variable.	
1232	A value of the wrong data type was set for a system variable.	
1248	No alias was set for an inline view.	
1251	The client does not support the user authentication protocol required by the server.	
1286	The storage engine is incorrectly configured.	

Serial number	Error code	Error message
1298	The time zone is incorrectly configured.	
1347	The object does not match the expected type.	
1353	The specified column number in the `Select` clause of the view is not equal to the defined column number.	
1364	No default value is set for columns that do not allow NULL values.	
1372	The password is not long enough.	
1396	The performed operation failed.	
1471	The specified table is not allowed to insert data.	
1507	Delete nonexistent partition, and no condition is specified to only delete existing partitions.	
1508	All partitions should be deleted by a delete table operation.	
1517	There are duplicate partition names.	
1524	The specified plugin has not been loaded.	
1567	The name of the partition is incorrect.	
1621	The specified system variable is read-only.	
1735	The specified partition name does not exist in the table.	
1748	Data cannot be inserted into a table that does not have a partition.	

Serial number	Error code	Error message
1749	The specified partition does not exist.	
5000	The specified table is not an OLAP table.	
5001	The specified storage path is invalid.	
5002	The name of the specified column should be displayed.	
5003	The dimension column should be preceded by the index column.	
5004	The table should contain at least 1 dimension column.	
5005	The cluster ID is invalid.	
5006	Invalid query plan.	
5007	Conflicting query plans.	
5008	The data insert is only available for data tables with partitions.	
5009	The `PARTITION` clause cannot be used to insert data into tables without partitions.	
5010	The number of columns in the table to be created is not equal to the number of columns in the `SELECT` clause.	
5011	Table reference could not be accessed.	
5012	The specified value is not a valid number.	
5013	The time unit is not supported.	
5014	The table status is not normal.	

Serial number	Error code	Error message
5015	The partition status is not normal.	
5016	A data import job exists in the partition.	
5017	The specified column is not a dimension column.	
5018	The format of the value is invalid.	
5019	The data replica does not match the version.	
5021	The BE node is offline.	
5022	The number of partitions in a non-partitioned table is not 1	
5023	No action was specified in the statement used to modify the table or data.	
5024	Job execution timed out.	
5025	Data insertion failed.	
5026	An unsupported data type was used when creating a table via the `SELECT` statement.	
5027	The specified parameter was not set.	
5028	The specified cluster was not found.	
5030	The user does not have permission to access the cluster.	
5031	No parameter specified or invalid parameter.	
5032	The number of cluster instances was not specified.	
5034	A cluster with the same name already exists.	

Serial number	Error code	Error message
5035	The number of cluster instances is incorrectly configured.	
5036	Insufficient BE nodes in the cluster.	
5037	All databases should be deleted before deleting the cluster.	
5038	The BE node with the specified ID does not exist in the cluster.	
5040	No cluster with the same name exists.	
5042	No permissions.	
5043	The number of instances should be greater than 0.	
5046	Source cluster does not exist.	
5047	Destination cluster does not exist.	
5048	Source database does not exist.	
5049	Destination database does not exist.	
5050	No cluster selected.	
5051	The source database should be associated with the destination database first.	
5052	Intra-cluster error: BE node information is incorrect.	
5053	There is no migration task from the source database to the destination database.	

Serial number	Error code	Error message
5054	The specified database has been associated with the destination database, or data is being migrated.	
5055	Database associations or data migrations cannot be performed within the same cluster.	
5056	Database cannot be deleted: it is associated with another database or data is being migrated.	
5056	Database cannot be renamed: it is associated with another database or data is being migrated.	
5056	Insufficient BE nodes in the cluster.	
5056	The specified number of BE nodes already exists in the cluster.	
5059	There are BE nodes in the cluster that are in the offline state.	
5063	The type name is incorrect.	
5064	Generic error message.	
5063	The Colocate feature has been disabled by the administrator.	
5063	A colocate data table with the same name does not exist.	
5063	The Colocate table must be an OLAP table.	
5063	Colocate tables should have the same number of replicas.	
5063	Colocate tables should have the same number of split buckets.	

Serial number	Error code	Error message
5063	Colocate tables should have the same number of partition columns.	
5063	Colocate tables should have the same data type of partitioned columns.	
5064	The specified table is not a colocate table.	
5065	The specified operation is invalid.	
5065	The specified time unit is illegal. The correct units are `DAY`, `WEEK`, and `MONTH`.	
5066	The start value of the dynamic partition should be less than 0.	
5066	The start value of the dynamic partition is not a valid number.	
5066	The end value of the dynamic partition should be greater than 0.	
5066	The end value of the dynamic partition is not a valid number.	
5066	The end value of the dynamic partition is null.	
5067	The bucket number of the dynamic partition should be greater than 0.	
5067	The bucket number of the dynamic partition is not a valid number.	
5066	The bucket number of the dynamic partition is empty.	

Serial number	Error code	Error message
5068	Whether to allow dynamic partition where the value is not a valid boolean: true or false.	
5069	The name of the specified dynamic partition has an illegal prefix.	
5070	The specified operation is disabled.	
5071	The number of replicas of the dynamic partition should be greater than 0.	
5072	The number of replicas of the dynamic partition is not a valid number.	

System limits

This topic describes the rules and limits that apply when you use StarRocks.

- StarRocks uses the MySQL protocol for communication. You can connect to the StarRocks cluster via a MySQL client or JDBC. We recommend that you use a MySQL client whose version is 5.1 or later. Versions earlier than 5.1 do not support usernames that are longer than 16 characters.

- Naming conventions for catalogs, databases, tables, views, partitions, columns, usernames, and roles:

  - The name can only consist of digits (0-9), letters (a-z or A-Z), and underscores (_). **Usernames can be all digits.**
  - The name can start with a letter or an underscore (_).
  - The name cannot exceed 64 characters in length, **among which:**
    - Database name cannot exceed 256 characters.
    - Table name and column name cannot exceed 1024 characters.
    - Username cannot exceed 128 characters.
  - Column name (column alias), partition name, and index name are **not** case-sensitive. Other names are **case-sensitive**.

- Naming conventions for labels:

  You can specify the label of a job when you load data. The label name can consist of digits (0-9), letters (a-z or A-Z), and underscores (_), and cannot exceed 128 characters in length. Label names can start with a letter or an underscore (_).

- When you create a table, the key column cannot be of the FLOAT or DOUBLE type. You can use the DECIMAL type to represent decimals.

- The maximum length of a VARCHAR value varies in different versions:

  - In versions earlier than StarRocks 2.1, the length ranges from 1 to 65533 bytes.
  - [Preview] In StarRocks 2.1 and later versions, the length ranges from 1 to 1048576 bytes. Maximum length of a VARCHAR value = Maximum row size (1048578 bytes)

- Length prefix (2 bytes). The length prefix indicates the number of bytes in the value.
    - The default length is 1 byte.

- StarRocks supports only UTF-8 encoding, not GBK.
- StarRocks does not support modifying column names in tables.
- StarRocks does not support modifying the table types of an existing table. For example, you cannot change a Duplicate Key table to a Primary Key table. You must create a new table.
- By default, a query can be nested with a maximum of 10,000 subqueries, which is controlled by the FE parameter `expr_children_limit`.

AWS IAM policies

A policy in AWS IAM declares a set of permissions on a specific AWS resource. After creating a policy, you need to attach it to an IAM role or user. As such, the IAM role or user is assigned the permissions declared in the policy to access the specified AWS resource.

Different operations in StarRocks require you to have access permissions on different AWS resources, and therefore you need to configure different policies.

This topic provides the IAM policies that you need to configure for StarRocks to integrate with different AWS resources in various business scenarios when you choose the instance profile, assumed role, or IAM user-based authentication method.

Batch load data from AWS S3

If you want to load data from your S3 bucket, configure the following IAM policy:

> **NOTICE**
>
> Replace `<bucket_name>` in the following JSON policy template with the name of your S3 bucket that stores your data files.

```
{
  "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>"
            ]
        }
    ]
}
```

Read/write AWS S3

If you want to query data from your S3 bucket, configure the following IAM policy:

> **NOTICE**
>
> Replace `<bucket_name>` in the following JSON policy template with the name of your S3 bucket that stores your data files.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "s3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket_name>/*"
            ]
        },
        {
            "Sid": "s3list",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket_name>"
            ]
```

```
      }
  ]
}
```

Integrate with AWS Glue

If you want to integrate with your AWS Glue Data Catalog, configure the following IAM policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "glue:GetDatabase",
              "glue:GetDatabases",
              "glue:GetPartition",
              "glue:GetPartitions",
              "glue:GetTable",
              "glue:GetTableVersions",
              "glue:GetTables",
              "glue:GetConnection",
              "glue:GetConnections",
              "glue:GetDevEndpoint",
              "glue:GetDevEndpoints",
              "glue:BatchGetPartition"
          ],
          "Resource": [
              "*"
            ]
        }
    ]
}
```

FAQs

Deployment

This topic provides answers to some frequently asked questions about deployment.

Data Migration

2 items

SQL

This topic provides answers to some frequently asked questions about SQL.

query_dump interface

This topic describes how to use the query_dump interface to obtain the details of an SQL query an...

Other FAQs

This topic provides answers to some general questions.

Deployment

This topic provides answers to some frequently asked questions about deployment.

How do I bind a fixed IP address with the `priority_networks` parameter in the `fe.conf` file?

Problem description

For example, if you have two IP addresses: 192.168.108.23 and 192.168.108.43. You might provide IP addresses as follows:

- If you specify the addresses as 192.168.108.23/24, StarRocks will recognize them as 192.168.108.43.
- If you specify the addresses as 192.168.108.23/32, StarRocks will recognize them as 127.0.0.1.

Solution

There are the following two ways to solve this problem:

- Do not add "32" at the end of an IP address or change "32" to "28".
- You can also upgrade to StarRocks 2.1 or later.

Why does the error "StarRocks BE http service did not start correctly, exiting" occur when I start a backend (BE) after installation?

When installing a BE, the system reports a startup error: StarRocks Be http service did not start correctly, exiting.

This error occurs because the web services port of the BE is occupied. Try to modify the ports in the `be.conf` file and restart the BE.

What do I do when the error occurs: ERROR 1064 (HY000): Could not initialize class com.starrocks.rpc.BackendServiceProxy?

This error occurs when you run programs in Java Runtime Environment (JRE). To solve this problem, replace JRE with Java Development Kit (JDK). We recommend that you use Oracle's JDK 1.8 or later.

Can FE and BE configuration items be modified and then take effect without restarting the cluster?

Yes. Perform the following steps to modify the configurations of FE and BE parameters:

- FE
  - SQL

```
ADMIN SET FRONTEND CONFIG ("key" = "value");
```

Example:

```
ADMIN SET FRONTEND CONFIG ("enable_statistic_collect" = "false");
```

  - Shell

```
curl --location-trusted -u username:password \
http://<ip>:<fe_http_port/api/_set_config?key=value>
```

Example:

```
curl --location-trusted -u <username>:<password> \
http://192.168.110.101:8030/api/_set_config?
enable_statistic_collect=true
```

- BE

```
curl -XPOST -u username:password \
http://<ip>:<be_http_port>/api/update_config?key=value
```

> Note: Make sure that the user has permission to log in remotely. If not, you can grant the permission to the user in the following way:

```
CREATE USER 'test'@'%' IDENTIFIED BY '123456';

GRANT SELECT_PRIV ON . TO 'test'@'%';
```

What do I do if the error "Failed to get scan range, no queryable replica found in tablet:xxxxx" occurs after I extend the BE disk space?

Problem description

This error may occur during data loading into Primary Key tables. During data loading, the destination BE does not have enough disk space for the loaded data and the BE crashes. New disks are then added to extend the disk space. However, Primary Key tables do not support disk space re-balancing and the data cannot be offloaded to other disks.

Solution

Patches to this bug (Primary Key tables do not support BE disk space re-balancing) is still under active development. Currently, you can fix it in either of the following two ways:

Manually distribute data among disks. For example, copy the directory from the disk with a high space usage to a disk with a larger space. If the data on these disks is not important, we recommend you delete the disks and modify the disk path. If this error persists, use TRUNCATE TABLE to clear data in the table to free up some space.

Why does the error "Fe type:unknown ,is ready :false." occur when I start an FE during the cluster restart?

Check if the leader FE is running. If not, restart the FE nodes in your cluster one by one.

Why does the error "failed to get service info err." occur when I deploy the cluster?

Check if OpenSSH Daemon (sshd) is enabled. If not, run the `/etc/init.d/sshd`` status` command to enable it.

Why does the error "Fail to get master client from `cache. ``host= port=0 code=THRIFT_RPC_ERROR`" occur when I start a BE?

Run the `netstat -anp |grep port` command to check whether the ports in the `be.conf` file are occupied. If so, replace the occupied port with a free port and then restart the BE.

Why does the error "Failed to transport upgrade files to agent host. src:..." occur when I upgrade a cluster of the Enterprise Edition?

This error occurs when the disk space specified in the deployment directory is insufficient. During the cluster upgrade, the StarRocks Manager distributes the binary file of the new version to each node. If the disk space specified in the deployment directory is insufficient, the file cannot be distributed to each node. To solve this problem, add data disks.

Why does the FE node log on the diagnostics page of StarRocks Manager display "Search log failed." for a newly deployed FE node that is running properly?

By default, StarRocks Manager obtains the path configuration of the newly deployed FE within 30 seconds. This error occurs when the FE starts slowly or does not respond within 30 seconds due to other reasons. Check the log of Manager Web via the path:

`/starrocks-manager-xxx/center/log/webcenter/log/web/``drms.INFO` (you can customize the path). Then find that whether the message "Failed to update FE configurations" display in the log. If so, restart the corresponding FE to obtain the new path configuration.

Why does the error "exceeds max permissable delta:5000ms." occur when I start an FE?

This error occurs when the time difference between two machines is more than 5s. To solve this problem, align the time of these two machines.

How do I set the `storage_root_path` parameter if there are multiple disks in a BE for data storage?

Configure the `storage_root_path` parameter in the `be.conf` file and separate values of this parameter with `;`. For example:

`storage_root_path=/the/path/to/storage1;/the/path/to/storage2;/the/path/to/storage3;`

Why does the error "invalid cluster id: 209721925." occur after an FE is added to my cluster?

If you do not add the `--helper` option for this FE when starting your cluster for the first time, the metadata between two machines is inconsistent, thus this error occurs. To solve this problem, you need to clear all metadata under the meta directory and then add an FE with the `--helper` option.

Why Alive is `false` when an FE is running and prints log `transfer: follower`?

This issue occurs when more than half of memory of Java Virtual Machine (JVM) is used and no checkpoint is marked. In general, a checkpoint will be marked after the system accumulates 50,000 pieces of log. We recommend that you modify the JVM's parameters of each FE and restarting these FEs when they are not heavily loaded.

Broker Load

1. Does Broker Load support re-running load jobs that have been run successfully and are in the FINISHED state?

Broker Load does not support re-running load jobs that have been run successfully and are in the FINISHED state. Also, to prevent data loss and duplication, Broker Load does not allow reusing the labels of successfully run load jobs. You can use SHOW LOAD to view the history of load jobs and find the load job that you want to re-run. Then, you can copy the information of that load job and use the job information, except the label, to create another load job.

2. When I load data from HDFS by using Broker Load, what do I do if the date and time values loaded into the destination StarRocks table are 8 hours later than the date and time values from the source data file?

Both the destination StarRocks table and the Broker Load job are compiled at creation to use a China Standard Time (CST) time zone (specified by using the `timezone` parameter). However, the server is set to run based on a Coordinated Universal Time (UTC) time zone. As a result, 8 extra hours are added to the date and time values from the source data file during data loading. To prevent this issue, do not specify the `timezone` parameter when you create the destination StarRocks table.

3. When I load ORC-formatted data by using Broker Load, what do I do if the `ErrorMsg:`

`type:ETL_RUN_FAIL; msg:Cannot cast '<slot 6>' from VARCHAR to ARRAY<VARCHAR(30)>` error occurs?

The source data file has different column names than the destination StarRocks table. In this situation, you must use the `SET` clause in the load statement to specify the column mapping between the file and the table. When executing the `SET` clause, StarRocks needs to perform a type inference, but it fails in invoking the cast function to transform the source data to the destination data types. To resolve this issue, make sure that the source data file has the same column names as the destination StarRocks table. As such, the `SET` clause is not needed and therefore StarRocks does not need to invoke the cast function to perform data type conversions. Then the Broker Load job can be run successfully.

4. The Broker Load job does not report errors, but why am I unable to query the loaded data?

Broker Load is an asynchronous loading method. The load job may still fail even if the load statement does not return errors. After you run a Broker Load job, you can use SHOW LOAD to view the result and `errmsg` of the load job. Then, you can modify the job configuration and retry.

5. What do I do if the "failed to send batch" or "TabletWriter add batch with unknown id" error occurs?

The amount of time taken to write the data exceeds the upper limit, causing a timeout error. To resolve this issue, modify the settings of the session variable `query_timeout` and the BE configuration item `streaming_load_rpc_max_alive_time_sec` based on your business requirements.

6. What do I do if the "LOAD-RUN-FAIL; msg:OrcScannerAdapter::init_include_columns. col name = xxx not found" error occurs?

If you are loading Parquet- or ORC-formatted data, check whether the column names held in the first row of the source data file are the same as the column names of the destination StarRocks table.

```
(tmp_c1,tmp_c2)
SET
(
   id=tmp_c2,
   name=tmp_c1
)
```

The preceding example maps the `tmp_c1` and `tmp_c2` columns of the source data file onto the `name` and `id` columns of the destination StarRocks table, respectively. If you do not specify the `SET` clause, the column names specified in the `column_list` parameter are used to declare the column mapping. For more information, see BROKER LOAD.

> **NOTICE**
>
> If the source data file is an ORC-formatted file generated by Apache Hive™ and the first row of the file holds `(_col0, _col1, _col2, ...)`, the "Invalid Column Name" error may occur. If this error occurs, you need to use the `SET` clause to specify the column mapping.

7. How do I handle errors such as the error that causes the Broker Load job to run for an excessively long period of time?

View the FE log file **fe.log** and search for the ID of the load job based on the job label. Then, view the BE log file **be.INFO** and retrieve the log records of the load job based on the job ID to locate the root cause of the error.

8. How do I configure an Apache HDFS cluster that runs in HA mode?

If an HDFS cluster runs in high availability (HA) mode, configure it as follows:

- `dfs.nameservices`: the name of the HDFS cluster, for example, `"dfs.nameservices" = "my_ha"`.
- `dfs.ha.namenodes.xxx`: the name of the NameNode in the HDFS cluster. If you specify multiple NameNode names, separate them with commas (`,`). `xxx` is the HDFS cluster name that you have specified in `dfs.nameservices`, for example, `"dfs.ha.namenodes.my_ha" = "my_nn"`.
- `dfs.namenode.rpc-address.xxx.nn`: the RPC address of the NameNode in the HDFS cluster. `nn` is the NameNode name that you have specified in `dfs.ha.namenodes.xxx`, for example, `"dfs.namenode.rpc-address.my_ha.my_nn" = "host:port"`.
- `dfs.client.failover.proxy.provider`: the provider of the NameNode to which the client will connect. Default value: `org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider`.

Example:

```
(
    "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.ConfiguredFailoverProxyProvi
)
```

The HA mode can be used with simple authentication or Kerberos authentication. For example, to use simple authentication to access an HDFS cluster that runs in HA mode, you need to specify the following configurations:

```
(
    "username"="user",
    "password"="passwd",
    "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.ConfiguredFailoverProxyProvi
)
```

You can add the configurations of the HDFS cluster to the **hdfs-site.xml** file. This way, you only need to specify the file path and authentication information when you use brokers to load data from the HDFS cluster.

9. How do I configure Hadoop ViewFS Federation?

Copy the ViewFs-related configuration files `core-site.xml` and `hdfs-site.xml` to the **broker/conf** directory.

If you have a custom file system, you also need to copy the file system-related **.jar** files to the **broker/lib** directory.

10. When I access an HDFS cluster that requires Kerberos authentication, what do I do if the "Can't get Kerberos realm" error occurs?

Check that the **/etc/krb5.conf** file is configured on all hosts on which brokers are deployed.

If the error persists, add `-Djava.security.krb5.conf:/etc/krb5.conf` to the end of the `JAVA_OPTS` variable in the broker startup script.

DataX

Why datax writer does not have writemode parameter? Why must the tables be created in the update mode?

Currently, only insert is supported.

How to process keywords when I synchronize data by using DataX Writer?

Enclose a keyword in back quotes (``). If you want to download the newest version of DataX, see DataX Writer.

Flink Connector

flink-connector-jdbc_2.11sink is 8 hours late in StarRocks

Issue description:

The time generated by localtimestap function is normal in Flink. But it became 8 hours late when sunk to StarRocks. Flink server and StarRocks server are located in the same timezone, namely Asia/Shanghai UTC/GMT+08:00. Flink version is 1.12. Driver: flink-connector-jdbc_2.11. Can I ask how to resolve this issue?

Solution:

Please try configure the time parameter 'server-time-zone' = 'Asia/Shanghai' in Flink sink table. You may also add &serverTimezone=Asia/Shanghai in jdbc url. An example is shown below:

```
CREATE TABLE sk (
    sid int,
    local_dtm TIMESTAMP,
    curr_dtm TIMESTAMP
)
WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://192.168.110.66:9030/sys_device?
characterEncoding=utf-8&serverTimezone=Asia/Shanghai',
    'table-name' = 'sink',
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'sr',
    'password' = 'sr123',
    'server-time-zone' = 'Asia/Shanghai'
);
```

In flink import, only the kafka clusters deployed in StarRocks clusters can be imported

Issue description:

```
failed to query wartermark offset, err: Local: Bad message format
```

Solution:

Kafka communication needs the hostname. Users need to configure the host name resolution /etc/hosts in StarRocks cluster nodes.

Can StarRocks export 'create table statements' in batches?

Solution:

You can use StarRocks Tools to export the statements.

Memory requested by BE is not released back to to the operation system

This is a normal phenomenon, as large blocks of memory allocated to the database from the operating system are reserved during allocation and deferred during release in order to reuse the memory and make memory allocation more convenient. It is recommended that users validate the test environment by monitoring memory usage over a longer period of time to see if the memory can be released.

Flink connector does not work after being downloaded

Issue description:

This package needs to be obtained through Aliyun mirror address.

Solution:

Please make sure that the mirror part of `/etc/maven/settings.xml` is all configured to be obtained through Aliyun mirror address.

If it is, change it to the following:

aliyunmaven centralaliyun public repohttps: //maven.aliyun.com/repository/public

The meaning of parameter sink.buffer-flush.interval-ms in Flink-connector-StarRocks

Issue description:

```
+---------------------+-----------------------------------------
-----------------+
|         Option        | Required |  Default   | Type   |
Description           |
+-------------------------------------------------------------------
-----------------+
|  sink.buffer-flush.  |  NO      |   300000   | String | the
flushing time interval, |
|  interval-ms         |          |            |        | range:
[1000ms, 3600000ms]  |
+---------------------+-----------------------------------------
-----------------+
```

If this parameter is set as 15s with checkpoint interval being equal to 5 mins, does this value still take effect?

Solution:

Whichever of the three thresholds is reached first, that one will take effect first. This is not affected by the checkpoint interval value which only works for exactly once. Interval-ms is used by at_least_once.

Insert Into

When performing data insert, each insert in SQL takes up 50 to 100ms. Is there any way to increase efficiency?

It is not recommended to insert data piece by piece to OLAP. It is usually inserted in batches. Both methods take up the same amount of time.

'Insert into select' task reports error: index channel has intoleralbe failure

You can solve this problem by changing the timeout duration for the Stream Load RPC. Change the following item in **be.conf** and restart the machines to allow the change to take effect:

`streaming_load_rpc_max_alive_time_sec`: The RPC timeout for Stream Load. Unit: Seconds. Default: `1200`.

Or you can set the query timeout using the following variable:

`query_timeout`: The timeout duration for queries. Its unit is seconds, and the default value is `300`.

The error "execute timeout" occurs when I run the INSER INTO SELECT command to load a large volume of data

By default, the query timeout duration is 300s. You can set the variable `query_timeout` to extend this duration. The unit is second.

Data loading

1. What do I do if the "close index channel failed" or "too many tablet versions" error occurs?

You were running load jobs too frequently, and the data was not compacted in a timely manner. As a result, the number of data versions generated during loading exceeds the maximum number (which defaults to 1000) of data versions that are allowed. Use one of the following methods to resolve this issue:

- Increase the amount of data loaded in each individual job, thereby reducing loading frequency.
- Modify the some configuration items in the BE configuration file **be.conf** of each BE to accelerate compactions:
  - For Duplicate Key tables, Aggregate tables, and Unique Key tables, you can appropriately increase the values of `cumulative_compaction_num_threads_per_disk`, `base_compaction_num_threads_per_disk`, and `cumulative_compaction_check_interval_seconds`. Example:

    ```
    cumulative_compaction_num_threads_per_disk = 4
    base_compaction_num_threads_per_disk = 2
    cumulative_compaction_check_interval_seconds = 2
    ```

  - For Primary Key tables, you can appropriately increase the value of `update_compaction_num_threads_per_disk` and decrease the value of `update_compaction_per_tablet_min_interval_seconds`.

  After you modify the settings of the preceding configuration items, you must observe the memory and I/O to ensure that they are normal.

2. What do I do if the "Label Already Exists" error occurs?

This error occurs because the load job has the same label as another load job, which has been successfully run or is being run, within the same StarRocks database.

Stream Load jobs are submitted according to HTTP. In general, request retry logic is embedded in HTTP clients of all programmatic languages. When the StarRocks cluster receives a load job request from an HTTP client, it immediately starts to process the request, but it does not return the job result to the HTTP client in a timely manner. As a result, the HTTP client sends the same load job request again. However, the StarRocks cluster is already processing the first request and therefore returns the `Label Already Exists` error for the second request.

Do as follows to check that load jobs submitted by using different loading methods do not have the same label and are not repeatedly submitted:

- View the FE log and check whether the label of the failed load job is recorded twice. If the label is recorded twice, the client has submitted the load job request twice.

  > **NOTE**
  >
  > The StarRocks cluster does not distinguish between the labels of load jobs based on loading methods. Therefore, load jobs submitted by using different loading methods may have the same label.

- Run SHOW LOAD WHERE LABEL = "xxx" to check for load jobs that have the same label and are in the **FINISHED** state.

  > **NOTE**
  >
  > `xxx` is the label that you want to check.

Before you submit a load job, we recommend that you calculate the approximate amount of time required to load the data and then adjust the client-side request timeout period accordingly. This way, you can prevent the client from submitting the load job request multiple times.

3. What do I do if the "ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel" error occurs?

Execute SHOW LOAD, and use the error URL in the returned execution result to view the error details.

Common data quality errors are as follows:

- "convert csv string to INT failed."

  Strings from a source column failed to be transformed into the data type of the matching destination column. For example, `abc` failed to be transformed into a numeric value.

- "the length of input is too long than schema."

  Values from a source column are in lengths that are not supported by the matching destination column. For example, the source column values of CHAR data type exceed the destination column's maximum length specified at table creation, or the source column values of INT data type exceed 4 bytes.

- "actual column number is less than schema column number."

  After a source row is parsed based on the specified column separator, the number of columns obtained is smaller than the number of columns in the destination table. A possible reason is that the column separator specified in the load command or statement differs from the column separator that is actually used in that row.

- "actual column number is more than schema column number."

  After a source row is parsed based on the specified column separator, the number of columns obtained is greater than the number of columns in the destination table. A possible reason is that the column separator specified in the load command or statement differs from the column separator that is actually used in that row.

- "the frac part length longer than schema scale."

  The decimal parts of values from a DECIMAL-type source column exceed the specified length.

- "the int part length longer than schema precision."

  The integer parts of values from a DECIMAL-type source column exceed the specified length.

- "there is no corresponding partition for this key."

  The value in the partition column for a source row is not within the partition range.

4. What do I do if RPC times out?

Check the setting of the `write_buffer_size` configuration item in the BE configuration file **be.conf** of each BE. This configuration item is used to control the maximum size per memory block on the BE. The default maximum size is 100 MB. If the maximum size is exceedingly large, Remote Procedure Call (RPC) may time out. To resolve this issue, adjust the settings of the `write_buffer_size` and `tablet_writer_rpc_timeout_sec` configuration items in the BE configuration file. For more information, see BE configurations.

5. What do I do if the "Value count does not match column count" error occurs?

After my load job failed, I used the error URL returned in the job result to retrieve the error details and found the "Value count does not match column count" error, which indicates a mismatch between the number of columns in the source data file and the number of columns in the destination StarRocks table:

```
Error: Value count does not match column count. Expect 3, but got 1.
Row: 2023-01-01T18:29:00Z,cpu0,80.99
Error: Value count does not match column count. Expect 3, but got 1.
Row: 2023-01-01T18:29:10Z,cpu1,75.23
Error: Value count does not match column count. Expect 3, but got 1.
Row: 2023-01-01T18:29:20Z,cpu2,59.44
```

The reason for this issue is as follows:

The column separator specified in the load command or statement differs from the column separator that is actually used in the source data file. In the preceding example, the CSV-

formatted data file consists of three columns, which are separated with commas (`,`). However, `\t` is specified as the column separator in the load command or statement. As a result, the three columns from the source data file are incorrectly parsed into one column.

Specify commas (`,`) as the column separator in the load command or statement. Then, submit the load job again.

Routine Load

How can I improve loading performance?

Method 1: Increase the actual load task parallelism by splitting a load job into as many parallel load tasks as possible.

> **NOTICE**
>
> This method may consume more CPU resources and cause too many tablet versions.

The actual load task parallelism is determined by the following formula composed of several parameters, with an upper limit of the number of BE nodes alive or the number of partitions to be consumed.

```
min(alive_be_number, partition_number, desired_concurrent_number,
max_routine_load_task_concurrent_num)
```

Parameter description:

- `alive_be_number`: the number of BE nodes alive.
- `partition_number`: the number of partitions to be consumed.
- `desired_concurrent_number`: the desired load task parallelism for a Routine Load job. The default value is `3`. You can set a higher value for this parameter to increase the actual load task parallelism.
  - If you have not created a Routine Load job, you need to set this parameter when using CREATE ROUTINE LOAD to create a Routine Load job.
  - If you have already created a Routine Load job, you need to use ALTER ROUTINE LOAD to modify this parameter.
- `max_routine_load_task_concurrent_num`: the default maximum task parallelism for a Routine Load job. The default value is `5`. This parameter is a an FE dynamic parameter. For more information and the configuration method, see Parameter configuration.

Therefore, when the number of partitions to be consumed and the number of BE nodes alive are greater than the other two parameters, you can increase the values of `desired_concurrent_number` and `max_routine_load_task_concurrent_num` parameters to increase the actual load task parallelism.

For example, the number of partitions to be consumed is `7`, the number of live BE nodes is `5`, and `max_routine_load_task_concurrent_num` is the default value `5`. At this time, if you need to increase the load task parallelism to the upper limit, you need to set `desired_concurrent_number` to `5` (the default value is `3`). Then, the actual task parallelism `min(5,7,5,5)` is computed to be `5`.

For more parameter descriptions, see CREATE ROUTINE LOAD.

Method 2: Increase the amount of data consumed by a Routine Load task from one or more partitions.

> **NOTICE**
>
> This method may cause delay in data loading.

The upper limit of the number of messages that a Routine Load task can consume is determined by either the parameter `max_routine_load_batch_size` which means the maximum number of messages that a load task can consume or the parameter `routine_load_task_consume_second` which means the maximum duration of message consumption. Once an load task consumes enough data that meets either requirement, the consumption is complete. These two parameters are FE dynamic parameters. For more information and the configuration method, see Parameter configuration.

You can analyze which parameter determines the upper limit of the amount of data consumed by a load task by viewing the log in **be/log/be.INFO**. By increasing that parameter, you can increase the amount of data consumed by a load task.

```
I0325 20:27:50.410579 15259 data_consumer_group.cpp:131] consumer
group done: 41448fb1a0ca59ad-30e34dabfa7e47a0. consume time(ms)=3261,
received rows=179190, received bytes=9855450, eos: 1, left_time:
-261, left_bytes: 514432550, blocking get time(us): 3065086, blocking
put time(us): 24855
```

Normally, the field `left_bytes` in the log is greater than or equal to `0`, indicating that the amount of data consumed by a load task has not exceeded

`max_routine_load_batch_size` within `routine_load_task_consume_second`. This means that a batch of scheduled load tasks can consume all data from Kafka without delay in consumption. In this scenario, you can set a larger value for `routine_load_task_consume_second` to increase the amount of data consumed by a load task from one or more partitions.

If the field `left_bytes` is less than `0`, it means that the amount of data consumed by a load task has reached `max_routine_load_batch_size` within `routine_load_task_consume_second`. Every time data from Kafka fills the batch of scheduled load tasks. Therefore, it is highly likely that there is remaining data that has not been consumed in Kafka, causing delay in consumption. In this case, you can set a larger value for `max_routine_load_batch_size`.

What do I do if the result of SHOW ROUTINE LOAD shows that the load job is in the `PAUSED` state?

- Check the field `ReasonOfStateChanged` and it reports the error message `Broker: Offset out of range`.

  Cause analysis: The consumer offset of the load job does not exist in the Kafka partition.

  Solution: You can execute SHOW ROUTINE LOAD and check the latest consumer offset of the load job in the parameter `Progress`. Then, you can verify if the corresponding message exists in the Kafka partition. If it does not exist, it may be because

  - The consumer offset specified when the load job is created is an offset in the future.
  - The message at the specified consumer offset in the Kafka partition has been removed before being consumed by the load job. It is recommended to set a reasonable Kafka log cleaning policy and parameters, such as `log.retention.hours and log.retention.bytes`, based on the loading speed.

- Check the field `ReasonOfStateChanged` and it doesn't report the error message `Broker: Offset out of range`.

  Cause analysis: The number of error rows in the load task exceeds the threshold `max_error_number`.

  Solution: You can troubleshoot and fix the issue by using error messages in the fields `ReasonOfStateChanged` and `ErrorLogUrls`.

  - If it is caused by incorrect data format in the data source, you need to check the data format and fix the issue. After successfully fixing the issue, you can use RESUME ROUTINE LOAD to resume the paused load job.
  - If it is because that StarRocks cannot parse the data format in the data source, you need to adjust the threshold `max_error_number`. You can first execute SHOW ROUTINE LOAD to view the value of `max_error_number`, and then use ALTER ROUTINE LOAD to increase the threshold. After modifying the threshold, you can use RESUME ROUTINE LOAD to resume the paused load job.

What do I do if the result of SHOW ROUTINE LOAD shows that the load job is in the `CANCELLED` state?

Cause analysis: The load job encountered an exception during loading, such as the table deleted.

Solution: When troubleshooting and fixing the issue, you can refer to the error messages in the fields `ReasonOfStateChanged` and `ErrorLogUrls`. However, after fixing the issue, you cannot resume the cancelled load job.

Can Routine Load guarantee consistency semantics when consuming from Kafka and writing to StarRocks?

Routine Load guarantees exactly-once semantics.

Each load task is a individual transaction. If an error occurs during the execution of the transaction, the transaction is aborted, and the FE does not update the consumption progress of the relevant partitions of the load tasks. When the FE schedules the load tasks from the task queue next time, the load tasks send the consumption request from the last saved consumption position of the partitions, thus ensuring exactly-once semantics.

Stream Load

1. Does Stream Load support identifying column names held in the first row of the source data file? Or, does Stream Load support skipping the first row during data reading?

Stream Load does not support identifying column names held in the first row of the source data file. Stream Load considers the first row to be normal data like the other rows. Additionally, Stream Load does not support skipping the first row during data reading. If the first row holds column names, take one of the following actions:

- Modify the settings of the tool that you use to export the data. Then, re-export the data as a source data file that does not hold column names in the first row.
- Use commands such as `sed -i '1d' filename` to delete the first row of the source data file.
- In the load command or statement, use `-H "where: <column_name> != '<column_name>'"` to filter out the first row of the source data file. `<column_name>` is any of the column names held in the first row. Note that StarRocks first transforms and then filters the source data. Therefore, if the column names in the first row fail to be transformed into their matching destination data types, `NULL` values are returned for them. This means the destination StarRocks table cannot contain columns that are set to `NOT NULL`.
- In the load command or statement, add `-H "max_filter_ratio:0.01"` to set a maximum error tolerance that is 1% or lower but can tolerate more than 1 error row, thereby allowing StarRocks to ignore the data transformation failures in the first row. In this case, the Stream Load job can still succeed even if `ErrorURL` is returned to indicate error rows. Do not set `max_filter_ratio` to a large value. If you set `max_filter_ratio` to a large value, some important data quality issues may be missed.

2. The data to be loaded into the partition column is not of standard DATE or INT type. For example, the data is in a format like 202106.00. How do I transform the data if I load it by using Stream Load?

StarRocks supports transforming data at loading. For more information, see Transform data at loading.

Suppose that you want to load a CSV-formatted source data file named `TEST` and the file consists of four columns, `NO`, `DATE`, `VERSION`, and `PRICE`, among which the data from the `DATE` column is in a non-standard format such as 202106.00. If you want to use `DATE` as the partition column in StarRocks, you need to first create a StarRocks table, for example, one that consists of the following four columns: `NO`, `VERSION`, `PRICE`, and `DATE`. Then, you need to specify the data type of the `DATE` column of the StarRocks table as DATE, DATETIME, or INT. Finally, when you create a Stream Load job, you need to specify the following setting in the load command or statement to transform data from the source `DATE` column's data type to the destination column's data type:

```
-H "columns: NO,DATE_1, VERSION, PRICE, DATE=LEFT(DATE_1,6)"
```

In the preceding example, `DATE_1` can be considered to be a temporarily named column mapping the destination `DATE` column, and the final results loaded into the destination `DATE` column are computed by the `left()` function. Note that you must first list the temporary names of the source columns and then use functions to transform data. The functions supported are scalar functions, including non-aggregate functions and window functions.

3. What do I do if my Stream Load job reports the "body exceed max size: 10737418240, limit: 10737418240" error?

The size of the source data file exceeds 10 GB, which is the maximum file size supported by Stream Load. Take one of the following actions:

- Use `seq -w 0 n` to split the source data file into smaller files.
- Use `curl -XPOST http://be_host:http_port/api/update_config?streaming_load_max_mb=<file_size>` to adjust the value of the BE configuration item `streaming_load_max_mb` to increase the maximum file size.

Synchronize data from MySQL in real time

What do I do if a Flink job reports an error?

A Flink job reports the error `Could not execute SQL statement. Reason:org.apache.flink.table.api.ValidationException: One or more required options are missing.`

A possible reason is that the required configuration information is missing in multiple sets of rules, such as `[table-rule.1]` and `[table-rule.2]`, in the SMT configuration file **config_prod.conf**.

You can check whether each set of rules, such as `[table-rule.1]` and `[table-rule.2]` is configured with the required database, table, and Flink connector information.

How can I make Flink automatically restart failed tasks?

Flink automatically restarts failed tasks through the checkpointing mechanism and restart strategy.

For example, if you need to enable the checkpointing mechanism and use the default restart strategy, which is the fixed delay restart strategy, you can configure the following information in the configuration file **flink-conf.yaml**:

```
execution.checkpointing.interval: 300000
state.backend: filesystem
state.checkpoints.dir: file:///tmp/flink-checkpoints-directory
```

Parameter description:

> **NOTE**

> For more detailed parameter descriptions in Flink documentation, see Checkpointing.

- `execution.checkpointing.interval`: the base time interval of checkpointing. Unit: millisecond. To enable the checkpointing mechanism, you need to set this parameter to a value greater than `0`.
- `state.backend`: specifies the state backend to determine how the state is represented internally, and how and where it is persisted upon checkpointing. Common values are `filesystem` or `rocksdb`. After the checkpointing mechanism is enabled, the state is persisted upon checkpoints to prevent data loss and ensure data consistency after recovery. For more information on state, see State Backends.
- `state.checkpoints.dir`: the directory to which checkpoints are written to.

How can I manually stop a Flink job and later restore it to the state before stopping?

You can manually trigger a savepoint when stopping a Flink job (a savepoint is a consistent image of the execution state of a streaming Flink job, and is created based on the checkpointing mechanism). Later, you can restore the Flink job from the specified savepoint.

1. Stop the Flink job with a savepoint. The following command automatically triggers a savepoint for the Flink job `jobId` and stops the Flink job. Additionally, you can specify a target file system directory to store the savepoint.

   ```
   bin/flink stop --type [native/canonical] --savepointPath
   [:targetDirectory] :jobId
   ```

   Parameter description:

   - `jobId`: You can view the Flink job ID from the Flink WebUI or by running `flink list -running` on the command line.
   - `targetDirectory`: You can specify `state.savepoints.dir` as the default directory for storing savepoints in the Flink configuration file **flink-conf.yml**. When a savepoint is triggered, the savepoint is stored in this default directory and you do not need to specify a directory .

   ```
   state.savepoints.dir: [file:// or
   ```

```
hdfs://]/home/user/savepoints_dir
```

2. Resubmit the Flink job with the preceding savepoint specified.

```
./flink run -c com.starrocks.connector.flink.tools.ExecuteSQL -s
savepoints_dir/savepoints-xxxxxxxx flink-connector-starrocks-
xxxx.jar -f flink-create.all.sql
```

Data Export

Alibaba cloud OSS backup and restore

StarRocks supports backing up data to alicloud OSS / AWS S3 (or object storage compatible with S3 protocol). Suppose there are two StarRocks clusters, namely DB1 cluster and DB2 cluster. We need to back up the data in DB1 to alicloud OSS and then restore it to DB2 when necessary. The general process of backup and recovery is as follows:

Create a cloud repository

Execute SQL in DB1 and DB2 respectively:

```
CREATE REPOSITORY `repository name`
WITH BROKER `broker_name`
ON LOCATION "oss://bucket name/path"
PROPERTIES
(
"fs.oss.accessKeyId" = "xxx",
"fs.oss.accessKeySecret" = "yyy",
"fs.oss.endpoint" = "oss-cn-beijing.aliyuncs.com"
);
```

a. Both DB1 and DB2 need to be created, and the created REPOSITORY name should be the same. View the repository:

```
SHOW REPOSITORIES;
```

b. broker_ name needs to fill in the broker name in a cluster. View BrokerName:

```
SHOW BROKER;
```

c. The path after fs.oss.endpoint does not need to have a bucket name.

Backup data table

BACKUP the tables to be backed up to the cloud repository in DB1. Execute SQL in DB1:

```
BACKUP SNAPSHOT [db_name].{snapshot_name}
TO `repository_name`
ON (
`table_name` [PARTITION (`p1`, ...)],
...
)
PROPERTIES ("key"="value", ...);
```

```
PROPERTIES currently supports the following properties:
"type" = "full": indicates that this is a full update (default).
"timeout" = "3600": task timeout. The default is one day. The unit is
seconds.
```

StarRocks does not support full database backup at present. We need to specify the tables or partitions to be backed up ON (...), and these tables or partitions will be backed up in parallel.

View the backup tasks in progress (note that only one backup task can be performed at the same time):

```
SHOW BACKUP FROM db_name;
```

After the backup is completed, you can check whether the backup data in the OSS already exists (unnecessary backups need to be deleted in the OSS):

```
SHOW SNAPSHOT ON OSS repository name;
```

Data restore

For data restore in DB2, there is no need to create a table structure to be restored in DB2. It will be created automatically during the Restore operation. Perform restore SQL:

```
RESTORE SNAPSHOT [db_name].{snapshot_name}
FROMrepository_name``
ON (
    'table_name' [PARTITION ('p1', ...)] [AS 'tbl_alias'],
    ...
)
PROPERTIES ("key"="value", ...);
```

View the restore progress:

```
SHOW RESTORE;
```

SQL

This topic provides answers to some frequently asked questions about SQL.

This error "fail to allocate memory." when I build a materialized view

To solve this problem, increase the value of the `memory_limitation_per_thread_for_schema_change` parameter in the **be.conf** file. This parameter refers to the maximum storage that can be allocated for a single task to change the scheme. The default value of the maximum storage is 2 GB.

Does StarRocks support caching query results?

StarRocks does not directly cache final query results. From v2.5 onwards, StarRocks uses the Query Cache feature to save the intermediate results of first-stage aggregation in the cache. New queries that are semantically equivalent to previous queries can reuse the cached computation results to accelerate computations. Query cache uses BE memory. For more information, see Query cache.

When a `Null` is included in the calculation, the calculation results of functions are false except for the ISNULL() function

In standard SQL, every calculation that includes an operand with a `NULL` value returns a `NULL`.

Does StarRocks support the DECODE function?

StarRocks does not support the DECODE function of the Oracle database. StarRocks is compatible with MySQL, so you can use the CASE WHEN statement.

Can the latest data be queried immediately after data is loaded into the Primary Key table of StarRocks?

Yes. StarRocks merges data in a way that references Google Mesa. In StarRocks, a BE triggers the data merge and it has two kinds of compaction to merge data. If the data merge is not completed, it is finished during your query. Therefore, you can read the latest data after data loading.

Do the utf8mb4 characters stored in StarRocks get truncated or appear garbled?

No.

This error "table's state is not normal" occurs when I run the `alter table` command

This error occurs because the previous alteration has not been completed. You can run the following code to check the status of the previous alteration:

```
show tablet from lineitem where State="ALTER";
```

The time spent on the alteration operation relates to the data volume. In general, the alteration can be completed in minutes. We recommend that you stop loading data into

StarRocks while you are altering tables because data loading lowers the speed at which alteration completes.

This error "get partition detail failed: org.apache.doris.common.DdlException: get hive partition meta data failed: java.net.UnknownHostException:hadooptest " occurs when I query the external tables of Apache Hive

This error occurs when the metadata of Apache Hive partitions cannot be obtained. To solve this problem, copy **core-sit.xml** and **hdfs-site.xml** to the **fe.conf** file and the **be.conf** file.

This error "planner use long time 3000 remaining task num 1" occurs when I query data

This error occurs usually due to a full garbage collection (full GC), which can be checked by using backend monitoring and the **fe.gc** log. To solve this problem, perform one of the following operations:

- Allows SQL's client to access multiple frontends (FEs) simultaneously to spread the load.
- Change the heap size of Java Virtual Machine (JVM) from 8 GB to 16 GB in the **fe.conf** file to increase memory and reduce the impact of full GC.

When cardinality of column A is small, the query results of `select B from tbl order by A limit 10` vary each time

SQL can only guarantee that column A is ordered, and it cannot guarantee that the order of column B is the same for each query. MySQL can guarantee the order of column A and column B because it is a standalone database.

StarRocks is a distributed database, of which data stored in the underlying table is in a sharding pattern. The data of column A is distributed across multiple machines, so the order of column B returned by multiple machines may be different for each query, resulting in inconsistent order of B each time. To solve this problem, change `select B from tbl order by A limit 10` to `select B from tbl order by A,B limit 10`.

Why is there a large gap in column efficiency between SELECT * and SELECT?

To solve this problem, check the profile and see MERGE details:

- Check whether the aggregation on the storage layer takes up too much time.
- Check whether there are too many indicator columns. If so, aggregate hundreds of columns of millions of rows.

```
MERGE:

    - aggr: 26s270ms

    - sort: 15s551ms
```

Does DELETE support nested functions?

Nested functions are not supported, such as `to_days(now())` in `DELETE from test_new WHERE to_days(now())-to_days(publish_time) >7;`.

How to improve the usage efficiency of a database when there are hundreds of tables in it?

To improve efficiency, add the `-A` parameter when you connect to MySQL's client server: `mysql -uroot -h127.0.0.1 -P8867 -A`. MySQL's client server does not pre-read database information.

How to reduce the disk space occupied by the BE log and the FE log?

Adjust the log level and corresponding parameters. For more information, see Parameter Configuration.

This error "table *** is colocate table, cannot change replicationNum" occurs when I modify the replication number

When you create colocated tables, you need to set the `group` property. Therefore, you cannot modify the replication number for a single table. You can perform the following steps to modify the replication number for all tables in a group:

1. Set `group_with` to `empty` for all tables in a group.
2. Set a proper `replication_num` for all tables in a group.
3. Set `group_with` back to its original value.

Does setting VARCHAR to the maximum value affect storage?

VARCHAR is a variable-length data type, which has a specified length that can be changed based on the actual data length. Specifying a different varchar length when you create a table has little impact on the query performance on the same data.

This error "create partititon timeout" occurs when I truncate a table

To truncate a table, you need to create the corresponding partitions and then swap them. If there are a larger number of partitions that need to be created, this error occurs. In addition, if there are many data load tasks, the lock will be held for a long time during the compaction process. Therefore, the lock cannot be acquired when you create tables. If there are too many data load tasks, set `tablet_map_shard_size` to `512` in the **be.conf** file to reduce the lock contention.

This error "Failed to specify server's Kerberos principal name" occurs when I access external tables of Apache Hive

Add the following information to **hdfs-site.xml** in the **fe.conf** file and the **be.conf** file:

```
<property>

<name>dfs.namenode.kerberos.principal.pattern</name>

<value>*</value>

</property>
```

Is "2021-10" a date format in StarRocks?

No.

Can "2021-10" be used as a partition field?

No, use functions to change "2021-10" to "2021-10-01" and then use "2021-10-01" as a partition field.

Where can I query the size of a StarRocks database or table?

You can use the SHOW DATA command.

`SHOW DATA;` displays the data size and replicas of all tables in the current database.

`SHOW DATA FROM <db_name>.<table_name>;` displays the data size, number of replicas, and number of rows in a specified table of a specified database.

query_dump interface

This topic describes how to use the query_dump interface to obtain the details of an SQL query and its related information.

If you encounter any of the following issues when executing SQL queries with StarRocks, you can use query_dump to obtain the SQL details and send the information to StarRocks technical support for troubleshooting:

- `Unknown Error` is returned when you execute an SQL query or EXPLAIN.
- An error message or exception is returned when you execute an SQL query.
- Executing an SQL query is not as efficient as expected, or the execution plan can be optimized (for example, partitions can be pruned or Join order can be adjusted).

Function overview

The query_dump interface returns the information that FE relies on when executing the SQL, including:

- Query statement
- Table creation statement
- Session variables
- Number of BEs
- Statistics information (Min, Max values in a column)
- Exception

Syntax

HTTP Post

```
 fe_host:fe_http_port/api/query_dump?db=${database}
post_data=${Query}
```

```
wget --user=${username} --password=${password} --post-file
${query_file} http://${fe_host}:${fe_http_port}/api/query_dump?
db=${database} -O ${dump_file}
```

Parameter description:

- query_file: the file containing the query
- dump_file: the output file
- db: the database where the SQL query is executed. The `db` parameter is optional if the query includes `use db`. Otherwise, it must be specified.

Example

```
wget --user=root --password=123 --post-file query_file
http://127.0.0.1:8030/api/query_dump?db=tpch -O dump_file
```

Return data

Data is returned in JSON format.


```
{
  "statement": "select\n    l_returnflag,\n    l_linestatus,\n    sum(l
  "table_meta": {
    "test.lineitem": "CREATE TABLE `lineitem` (\n  `L_ORDERKEY` int(11)
  },
  "table_row_count": {
    "test.lineitem": {
      "lineitem": 600000000
    }
  },
  "session_variables": "{\"runtime_join_filter_push_down_limit\":102400
READ\",\"wait_timeout\":28800,\"auto_increment_increment\":1,\"foreign_
  "column_statistics": {
    "test.lineitem": {
      "L_TAX": "[0.0, 0.08, 0.0, 8.0, 9.0]",
      "L_SHIPDATE": "[6.942816E8, 9.124416E8, 0.0, 4.0, 2526.0]",
      "L_EXTENDEDPRICE": "[901.0, 104949.5, 0.0, 8.0, 932377.0]",
      "L_DISCOUNT": "[0.0, 0.1, 0.0, 8.0, 11.0]",
      "L_RETURNFLAG": "[-Infinity, Infinity, 0.0, 1.0, 3.0]",
      "L_LINESTATUS": "[-Infinity, Infinity, 0.0, 1.0, 2.0]",
```

```
      "L_QUANTITY": "[1.0, 50.0, 0.0, 8.0, 50.0]"
    }
  },
  "be_number": 3,
  "exception": []
}
```

Other FAQs

This topic provides answers to some general questions.

Do VARCHAR (32) and STRING occupy the same storage space?

Both are variable-length data types. When you store data of the same length, VARCHAR (32) and STRING occupy the same storage space.

Do VARCHAR (32) and STRING perform the same for the data query?

Yes.

Why do TXT files imported from Oracle still appear garbled after I set the character set to UTF-8?

To solve this problem, perform the following steps:

1. For example, there is a file named **original**, whose text is garbled. The character set of this file is ISO-8859-1. Run the following code to obtain the character set of the file.

```
file --mime-encoding origin.txt
origin.txt: iso-8859-1
```

2. Run the `iconv` command to convert the character set of this file into UTF-8.

```
iconv -f iso-8859-1 -t utf-8 origin.txt > origin_utf-8.txt
```

3. After the conversion, the text of this file still appears garbled. You can then regrade the character set of this file as GBK and convert the character set into UTF-8 again.

```
iconv -f gbk -t utf-8 origin.txt > origin_utf-8.txt
```

Is the length of STRING defined by MySQL the same as that defined by StarRocks?

For VARCHAR(n), StarRocks defines "n" by bytes and MySQL defines "n" by characters. According to UTF-8, one Chinese character is equal to three bytes. When StarRocks and MySQL define "n" as the same number, MySQL saves three times as many characters as StarRocks.

Can the data type of partitioned fields of a table be FLOAT, DOUBLE, or DECIMAL?

No, only DATE, DATETIME, and INT are supported.

How to check the storage space that is occupied by the data in a table?

Execute the SHOW DATA statement to see the corresponding storage space. You can also see the data volume, the number of copies, and the number of rows.

Note: There is a time delay in data statistics.

How to request a quota increase for the StarRocks database?

To request a quota increase, run the following code:

```
ALTER DATABASE example_db SET DATA QUOTA 10T;
```

Does StarRocks support updating particular fields in a table by executing the UPSERT statement?

StarRocks 2.2 and later support updating specific fields in a table by using the Primary Key table. StarRocks 1.9 and later support updating all fields in a table by using the Primary Key table. For more information, see Primary Key table in StarRocks 2.2.

How to swap the data between two tables or two partitions?

Execute the SWAP WITH statement to swap the data between two tables or two partitions. The SWAP WITH statement is more secure than the INSERT OVERWRITE statement. Before you swap the data, check the data first and then see whether the data after the swapping is consistent with the data before the swapping.

- Swap two tables: For example, there is a table named table 1. If you want to replace table 1 with another one, perform the following steps:

  i. Create a new table named table 2.

  ```
  create table2 like table1;
  ```

  ii. Use Stream Load, Broker Load, or Insert Into to load data from table 1 into table 2.

  iii. Replace table 1 with table 2.

  ```
  ALTER TABLE table1 SWAP WITH table2;
  ```

  By doing so, the data is loaded accurately into table 1.

- Swap two partitions: For example, there is a table named table 1. If you want to replace the partition data in table 1, perform the following steps:

  i. Create a temporary partition.

```
ALTER TABLE table1

ADD TEMPORARY PARTITION tp1

VALUES LESS THAN("2020-02-01");
```

  ii. Load the partition data from table 1 into the temporary partition.

  iii. Replace the partition of table 1 with the temporary partition.

```
ALTER TABLE table1

REPLACE PARTITION (p1) WITH TEMPORARY PARTITION (tp1);
```

This error "error to open replicated environment, will exit" occurs when I restart a frontend (FE)

This error occurs due to BDBJE's bug. To solve this problem, update the BDBJE version to 1.17 or later.

This error "Broker list path exception" occurs when I query data from a new Apache Hive table

Problem description

```
msg:Broker list path exception
```

```
path=hdfs://172.31.3.136:9000/user/hive/warehouse/zltest.db/student_inf
broker=TNetworkAddress(hostname:172.31.4.233, port:8000)
```

Solution

Contact the StarRocks technical support and check whether the address and port of the namenode are correct and whether you have permission to access the address and port of the namenode.

This error "get hive partition metadata failed" occurs when I query data from a new Apache Hive table

Problem description

```
msg:get hive partition meta data failed:
java.net.UnknownHostException: emr-header-1.cluster-242
```

Solution

Ensure that the network is connected and upload the **host** file to each backend (BE) in your StarRocks cluster.

This error "do_open failed. reason = Invalid ORC postscript length" occurs when I access ORC external table in Apache Hive

Problem description

The metadata of the Apache Hive is cached in the FEs. But there is a two-hours time lag for StarRocks to update the metadata. Before StarRocks finishes the update, If you insert new data or update data in the Apache Hive table, the data in HDFS scanned by the BEs and the data obtained by the FEs are different. Therefore, this error occurs.

```
MySQL [bdp_dim]> select * from dim_page_func_s limit 1;

ERROR 1064 (HY000): HdfsOrcScanner::do_open failed. reason = Invalid
ORC postscript length
```

Solution

To solve this problem, perform one of the following operations:

- Upgrade your current version to StarRocks 2.2 or later.
- Manually refresh your Apache Hive table. For more information, see Metadata caching strategy.

This error "caching_sha2_password cannot be loaded" occurs when I connect external tables of MySQL

Problem description

The default authentication plugin of MySQL 8.0 is caching_sha2_password. The default authentication plugin of MySQL 5.7 is mysql_native_password. This error occurs because you use the wrong authentication plugin.

Solution

To solve this problem, perform one of the following operations:

- Connect to the StarRocks.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password
BY 'yourpassword';
```

- Modify the `my.cnf` file.

```
vim my.cnf
```

```
[mysqld]

default_authentication_plugin=mysql_native_password
```

How to release disk space immediately after deleting a table?

If you execute the DROP TABLE statement to delete a table, StarRocks takes a while to release the allocated disk space. To release the allocated disk space immediately, execute the DROP TABLE FORCE statement to delete a table. When you execute the DROP TABLE FORCE statement, the StarRocks deletes the table directly without checking whether there are unfinished events in it. We recommend that you execute the DROP TABLE FORCE statement with caution. Because once the table is deleted, you cannot restore it.

How to view the current version of StarRocks?

Run the `select current_version();` command or the CLI command `./bin/show_fe_version.sh` to view the current version.

How to set the memory size of an FE?

The metadata is stored in the memory used by the FE. You can set the memory size of the FE according to the number of tablets as shown in the table below. For example, if the number of tablets is below 1 million, you should allocate a minimum of 16 GB memory to the FE. You can configure the values of the parameters `-Xms` 和 `-Xmx` in the **JAVA_OPTS** configuration item in the **fe.conf** file, and the values of the parameters `-Xms` 和 `-Xmx` should be consistent. Note that the configuration should be same across all FEs because any of the FEs can be elected as a Leader.

Number of tablets	Memory size of each FE
below 1 million	16 GB

Number of tablets	Memory size of each FE
1 ～ 2 million	32 GB
2 ～ 5 million	64 GB
5 ～ 10 million	128 GB

How does StarRocks calculate its query time?

StarRocks supports querying data by using multiple threads. Query time refers to the time used by multiple threads to query data.

Does StarRocks support setting the path when I export data locally?

No.

What are the concurrency upper limits of StarRocks?

You can test the concurrency limitations based on the actual business scenarios or simulated business scenarios. According to the feedback of some users, maximum of 20,000 QPS or 30,000 QPS can be achieved.

Why is the first-time SSB test performance of StarRocks slower than that done the second time?

The speed to read disks for the first query relates to the performance of disks. After the first query, the page cache is generated for the subsequent queries, so the query is faster than

before.

How many BEs need to be configured at least for a cluster?

StarRocks supports single node deployment, so you need to configure at least one BE. BEs need to be run with AVX2, so we recommend that you deploy BEs on machines with 8-core and 16GB or higher configurations.

How to set data permissions when I use Apache Superset to visualize the data in StarRocks?

You can create a new user account and then set the data permission by granting permissions on the table query to the user.

Why does the profile fail to display after I set `enable_profile` to `true`?

The report is only submitted to the leader FE for access.

How to check field annotations in the tables of StarRocks?

Run the `show create table xxx` command.

When I create a table, how to specify the default value for the NOW() function?

Only StarRocks 2.1 or later version supports specifying the default value for a function. For versions earlier than StarRocks 2.1, you can only specify a constant for a function.

How can I release the storage space of BE nodes?

You can remove the directory `trash` using `rm -rf` command. If you have already restored your data from snapshot, you can remove the directory `snapshot`.

Can add extra disks to BE nodes?

Yes. You can add the disks to the directory specified by the BE configuration item `storage_root_path`.

Benchmarks

SSB Flat-table Benchmarking

Star schema benchmark (SSB) is designed to test basic performance metrics of OLAP database pr...

TPC-H Benchmarking

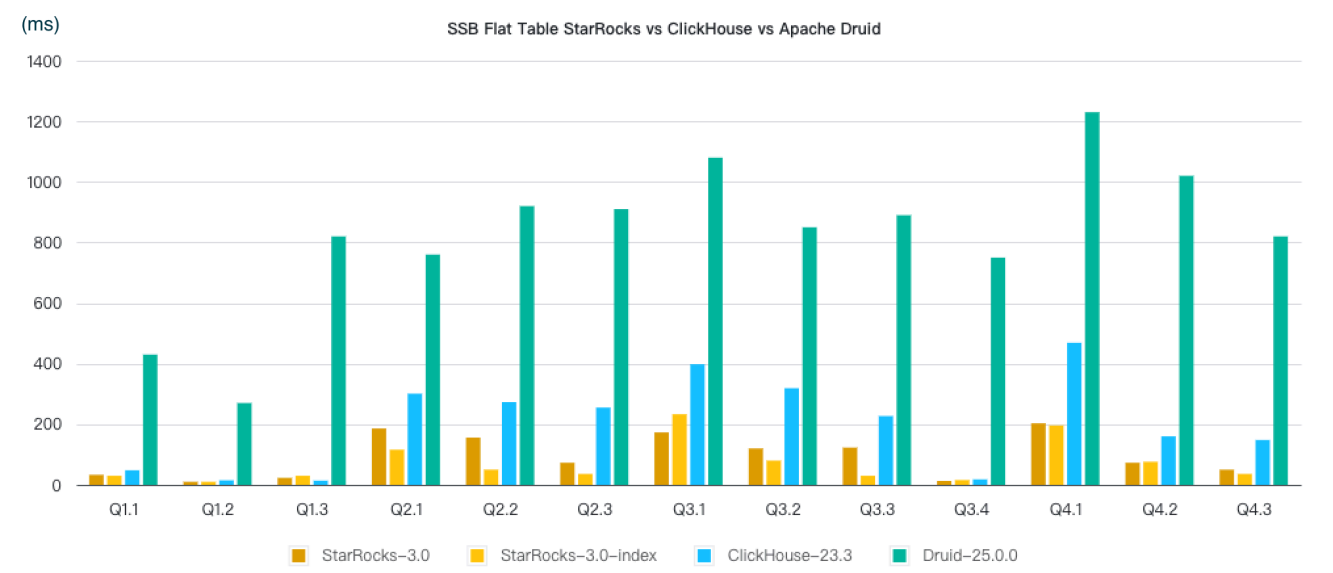
TPC-H is a decision support benchmark developed by the Transaction Processing Performance Co...

SSB Flat-table Benchmarking

Star schema benchmark (SSB) is designed to test basic performance metrics of OLAP database products. SSB uses a star schema test set that is widely applied in academia and industry. For more information, see the paper Star Schema Benchmark. ClickHouse flattens the star schema into a wide flat table and rewrites the SSB into a single-table benchmark. For more information, see Star schema benchmark of ClickHouse. This test compares the performance of StarRocks, Apache Druid, and ClickHouse against SSB single-table datasets.

Test conclusion

- Among the 13 queries performed on SSB standard datasets, StarRocks has an overall query performance **2.1x that of ClickHouse and 8.7x that of Apache Druid**.
- After Bitmap Indexing of StarRocks is enabled, the performance is 1.3x compared to when this feature is disabled. The overall performance of StarRocks is **2.8x that of ClickHouse and 11.4x that of Apache Druid**.



Test preparation

Hardware

Machine	3 cloud hosts
CPU	16-Core Intel (R) Xeon (R) Platinum 8269CY CPU @2.50GHz Cache size: 36608 KB
Memory	64 GB
Network bandwidth	5 Gbit/s
Disk	ESSD

Software

StarRocks, Apache Druid, and ClickHouse are deployed on hosts of the same configurations.

- StarRocks: one FE and three BEs. The FE can be separately or hybrid deployed with BEs.
- ClickHouse: three nodes with distributed tables
- Apache Druid: three nodes. One is deployed with Master Servers and Data Servers, one is deployed with Query Servers and Data Servers, and the third is deployed only with Data Servers.

Kernel version: Linux 3.10.0-1160.59.1.el7.x86_64

OS version: CentOS Linux release 7.9.2009

Software version: StarRocks Community Version 3.0, ClickHouse 23.3, Apache Druid 25.0.0

Test data and results

Test data

Table	Record	Description
lineorder	600 million	Lineorder fact table

Table	Record	Description
customer	3 million	Customer dimension table
part	1.4 million	Parts dimension table
supplier	200 thousand	Supplier dimension table
dates	2,556	Date dimension table
lineorder_flat	600 million	lineorder flat table

Test results

The following table shows the performance test results on thirteen queries. The unit of query latency is ms. `ClickHouse vs StarRocks` in the table header means using the query response time of ClickHouse to divide the query response time of StarRocks. A larger value indicates better performance of StarRocks.

	StarRocks-3.0	StarRocks-3.0-index	ClickHouse-23.3	ClickHouse vs StarRocks	Druid-25.0.0	Druid v StarRoc
Q1.1	33	30	48	1.45	430	13.03
Q1.2	10	10	15	1.50	270	27.00
Q1.3	23	30	14	0.61	820	35.65
Q2.1	186	116	301	1.62	760	4.09
Q2.2	156	50	273	1.75	920	5.90
Q2.3	73	36	255	3.49	910	12.47
Q3.1	173	233	398	2.30	1080	6.24

	StarRocks-3.0	StarRocks-3.0-index	ClickHouse-23.3	ClickHouse vs StarRocks	Druid-25.0.0	Druid v StarRoc
Q3.2	120	80	319	2.66	850	7.08
Q3.3	123	30	227	1.85	890	7.24
Q3.4	13	16	18	1.38	750	57.69
Q4.1	203	196	469	2.31	1230	6.06
Q4.2	73	76	160	2.19	1020	13.97
Q4.3	50	36	148	2.96	820	16.40
sum	1236	939	2645	2.14	10750	8.70

Test procedure

For more information about how to create a ClickHouse table and load data to the table, see ClickHouse official doc. The following sections describe data generation and data loading of StarRocks.

Generate data

Download the ssb-poc toolkit and compile it.

```
wget https://starrocks-public.oss-cn-zhangjiakou.aliyuncs.com/ssb-poc-1.0.zip
unzip ssb-poc-1.0.zip
cd ssb-poc-1.0/
make && make install
cd output/
```

After the compilation, all the related tools are installed to the `output` directories and the following operations are all performed under this directory.

First, generate data for SSB standard dataset `scale factor=100`.

```
sh bin/gen-ssb.sh 100 data_dir
```

Create table schema

1. Modify the configuration file `conf/starrocks.conf` and specify the cluster address. Pay special attention to `mysql_host` and `mysql_port`.
2. Run the following command to create a table:

   ```
   sh bin/create_db_table.sh ddl_100
   ```

Query data

```
sh bin/benchmark.sh ssb-flat
```

Enable Bitmap Indexing

StarRocks performs better with Bitmap Indexing enabled. If you want to test the performance of StarRocks with Bitmap Indexing enabled, especially on Q2.2, Q2.3, and Q3.3, you can create Bitmap Indexes for all STRING columns.

1. Create another `lineorder_flat` table and create Bitmap Indexes.

   ```
   sh bin/create_db_table.sh ddl_100_bitmap_index
   ```

2. Add the following configuration to the `be.conf` file of all BEs and restart the BEs for the configurations to take effect.

   ```
   bitmap_max_filter_ratio=1000
   ```

3. Run the data loading script.

```
sh bin/flat_insert.sh data_dir
```

After data is loaded, wait for data version compaction to complete and then perform 4.4 again to query the data after Bitmap Indexing is enabled.

You can view the progress of data version compaction by running `select CANDIDATES_NUM from information_schema.be_compactions`. For the three BE nodes, the following results show compaction is completed:

```
mysql> select CANDIDATES_NUM from information_schema.be_compactions;
+----------------+
| CANDIDATES_NUM |
+----------------+
|              0 |
|              0 |
|              0 |
+----------------+
3 rows in set (0.01 sec)
```

Test SQL and table creation statements

Test SQL

```
--Q1.1
SELECT sum(lo_extendedprice * lo_discount) AS `revenue`
FROM lineorder_flat
WHERE lo_orderdate >= '1993-01-01' and lo_orderdate <= '1993-12-31'
AND lo_discount BETWEEN 1 AND 3 AND lo_quantity < 25;

--Q1.2
SELECT sum(lo_extendedprice * lo_discount) AS revenue FROM
lineorder_flat
WHERE lo_orderdate >= '1994-01-01' and lo_orderdate <= '1994-01-31'
AND lo_discount BETWEEN 4 AND 6 AND lo_quantity BETWEEN 26 AND 35;

--Q1.3
SELECT sum(lo_extendedprice * lo_discount) AS revenue
FROM lineorder_flat
WHERE weekofyear(lo_orderdate) = 6
AND lo_orderdate >= '1994-01-01' and lo_orderdate <= '1994-12-31'
```

```
AND lo_discount BETWEEN 5 AND 7 AND lo_quantity BETWEEN 26 AND 35;

--Q2.1
SELECT sum(lo_revenue), year(lo_orderdate) AS year,  p_brand
FROM lineorder_flat
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY year, p_brand
ORDER BY year, p_brand;

--Q2.2
SELECT
sum(lo_revenue), year(lo_orderdate) AS year, p_brand
FROM lineorder_flat
WHERE p_brand >= 'MFGR#2221' AND p_brand <= 'MFGR#2228' AND s_region
= 'ASIA'
GROUP BY year, p_brand
ORDER BY year, p_brand;

--Q2.3
SELECT sum(lo_revenue), year(lo_orderdate) AS year, p_brand
FROM lineorder_flat
WHERE p_brand = 'MFGR#2239' AND s_region = 'EUROPE'
GROUP BY year, p_brand
ORDER BY year, p_brand;

--Q3.1
SELECT
    c_nation,
    s_nation,
    year(lo_orderdate) AS year,
    sum(lo_revenue) AS revenue FROM lineorder_flat
WHERE c_region = 'ASIA' AND s_region = 'ASIA' AND lo_orderdate >=
'1992-01-01'
AND lo_orderdate <= '1997-12-31'
GROUP BY c_nation, s_nation, year
ORDER BY  year ASC, revenue DESC;

--Q3.2
SELECT c_city, s_city, year(lo_orderdate) AS year, sum(lo_revenue) AS
revenue
FROM lineorder_flat
WHERE c_nation = 'UNITED STATES' AND s_nation = 'UNITED STATES'
AND lo_orderdate  >= '1992-01-01' AND lo_orderdate <= '1997-12-31'
GROUP BY c_city, s_city, year
ORDER BY year ASC, revenue DESC;
```

```
--Q3.3
SELECT c_city, s_city, year(lo_orderdate) AS year, sum(lo_revenue) AS
revenue
FROM lineorder_flat
WHERE c_city in ( 'UNITED KI1' ,'UNITED KI5') AND s_city in ('UNITED
KI1', 'UNITED KI5')
AND lo_orderdate  >= '1992-01-01' AND lo_orderdate <= '1997-12-31'
GROUP BY c_city, s_city, year
ORDER BY year ASC, revenue DESC;

--Q3.4
SELECT c_city, s_city, year(lo_orderdate) AS year, sum(lo_revenue) AS
revenue
FROM lineorder_flat
WHERE c_city in ('UNITED KI1', 'UNITED KI5') AND s_city in ('UNITED
KI1', 'UNITED KI5')
AND lo_orderdate  >= '1997-12-01' AND lo_orderdate <= '1997-12-31'
GROUP BY c_city, s_city, year
ORDER BY year ASC, revenue DESC;

--Q4.1
SELECT year(lo_orderdate) AS year, c_nation, sum(lo_revenue -
lo_supplycost) AS profit
FROM lineorder_flat
WHERE c_region = 'AMERICA' AND s_region = 'AMERICA' AND p_mfgr in
('MFGR#1', 'MFGR#2')
GROUP BY year, c_nation
ORDER BY year ASC, c_nation ASC;

--Q4.2
SELECT year(lo_orderdate) AS year,
    s_nation, p_category, sum(lo_revenue - lo_supplycost) AS profit
FROM lineorder_flat
WHERE c_region = 'AMERICA' AND s_region = 'AMERICA'
AND lo_orderdate >= '1997-01-01' and lo_orderdate <= '1998-12-31'
AND p_mfgr in ( 'MFGR#1' , 'MFGR#2')
GROUP BY year, s_nation, p_category
ORDER BY year ASC, s_nation ASC, p_category ASC;

--Q4.3
SELECT year(lo_orderdate) AS year, s_city, p_brand,
    sum(lo_revenue - lo_supplycost) AS profit
FROM lineorder_flat
WHERE s_nation = 'UNITED STATES'
AND lo_orderdate >= '1997-01-01' and lo_orderdate <= '1998-12-31'
AND p_category = 'MFGR#14'
```

```
GROUP BY year, s_city, p_brand
ORDER BY year ASC, s_city ASC, p_brand ASC;
```

Table creation statements

Default `lineorder_flat` table

The following statement matches the current cluster size and data size (three BEs, scale factor = 100). If your cluster has more BE nodes or larger data size, you can adjust the number of buckets, create the table again, and load data again to achieve better test results.

```
CREATE TABLE `lineorder_flat` (
  `LO_ORDERDATE` date NOT NULL COMMENT "",
  `LO_ORDERKEY` int(11) NOT NULL COMMENT "",
  `LO_LINENUMBER` tinyint(4) 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_ORDERPRIORITY` varchar(100) NOT NULL COMMENT "",
  `LO_SHIPPRIORITY` tinyint(4) NOT NULL COMMENT "",
  `LO_QUANTITY` tinyint(4) NOT NULL COMMENT "",
  `LO_EXTENDEDPRICE` int(11) NOT NULL COMMENT "",
  `LO_ORDTOTALPRICE` int(11) NOT NULL COMMENT "",
  `LO_DISCOUNT` tinyint(4) NOT NULL COMMENT "",
  `LO_REVENUE` int(11) NOT NULL COMMENT "",
  `LO_SUPPLYCOST` int(11) NOT NULL COMMENT "",
  `LO_TAX` tinyint(4) NOT NULL COMMENT "",
  `LO_COMMITDATE` date NOT NULL COMMENT "",
  `LO_SHIPMODE` varchar(100) NOT NULL COMMENT "",
  `C_NAME` varchar(100) NOT NULL COMMENT "",
  `C_ADDRESS` varchar(100) NOT NULL COMMENT "",
  `C_CITY` varchar(100) NOT NULL COMMENT "",
  `C_NATION` varchar(100) NOT NULL COMMENT "",
  `C_REGION` varchar(100) NOT NULL COMMENT "",
  `C_PHONE` varchar(100) NOT NULL COMMENT "",
  `C_MKTSEGMENT` varchar(100) NOT NULL COMMENT "",
  `S_NAME` varchar(100) NOT NULL COMMENT "",
  `S_ADDRESS` varchar(100) NOT NULL COMMENT "",
  `S_CITY` varchar(100) NOT NULL COMMENT "",
  `S_NATION` varchar(100) NOT NULL COMMENT "",
  `S_REGION` varchar(100) NOT NULL COMMENT "",
  `S_PHONE` varchar(100) NOT NULL COMMENT "",
```

```
  `P_NAME` varchar(100) NOT NULL COMMENT "",
  `P_MFGR` varchar(100) NOT NULL COMMENT "",
  `P_CATEGORY` varchar(100) NOT NULL COMMENT "",
  `P_BRAND` varchar(100) NOT NULL COMMENT "",
  `P_COLOR` varchar(100) NOT NULL COMMENT "",
  `P_TYPE` varchar(100) NOT NULL COMMENT "",
  `P_SIZE` tinyint(4) NOT NULL COMMENT "",
  `P_CONTAINER` varchar(100) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`LO_ORDERDATE`, `LO_ORDERKEY`)
COMMENT "OLAP"
PARTITION BY date_trunc('year', `LO_ORDERDATE`)
DISTRIBUTED BY HASH(`LO_ORDERKEY`) BUCKETS 48
PROPERTIES ("replication_num" = "1");
```

`lineorder_flat` table with Bitmap Indexes

```
CREATE TABLE `lineorder_flat` (
  `LO_ORDERDATE` date NOT NULL COMMENT "",
  `LO_ORDERKEY` int(11) NOT NULL COMMENT "",
  `LO_LINENUMBER` tinyint(4) 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_ORDERPRIORITY` varchar(100) NOT NULL COMMENT "",
  `LO_SHIPPRIORITY` tinyint(4) NOT NULL COMMENT "",
  `LO_QUANTITY` tinyint(4) NOT NULL COMMENT "",
  `LO_EXTENDEDPRICE` int(11) NOT NULL COMMENT "",
  `LO_ORDTOTALPRICE` int(11) NOT NULL COMMENT "",
  `LO_DISCOUNT` tinyint(4) NOT NULL COMMENT "",
  `LO_REVENUE` int(11) NOT NULL COMMENT "",
  `LO_SUPPLYCOST` int(11) NOT NULL COMMENT "",
  `LO_TAX` tinyint(4) NOT NULL COMMENT "",
  `LO_COMMITDATE` date NOT NULL COMMENT "",
  `LO_SHIPMODE` varchar(100) NOT NULL COMMENT "",
  `C_NAME` varchar(100) NOT NULL COMMENT "",
  `C_ADDRESS` varchar(100) NOT NULL COMMENT "",
  `C_CITY` varchar(100) NOT NULL COMMENT "",
  `C_NATION` varchar(100) NOT NULL COMMENT "",
  `C_REGION` varchar(100) NOT NULL COMMENT "",
  `C_PHONE` varchar(100) NOT NULL COMMENT "",
  `C_MKTSEGMENT` varchar(100) NOT NULL COMMENT "",
  `S_NAME` varchar(100) NOT NULL COMMENT "",
  `S_ADDRESS` varchar(100) NOT NULL COMMENT "",
  `S_CITY` varchar(100) NOT NULL COMMENT "",
```

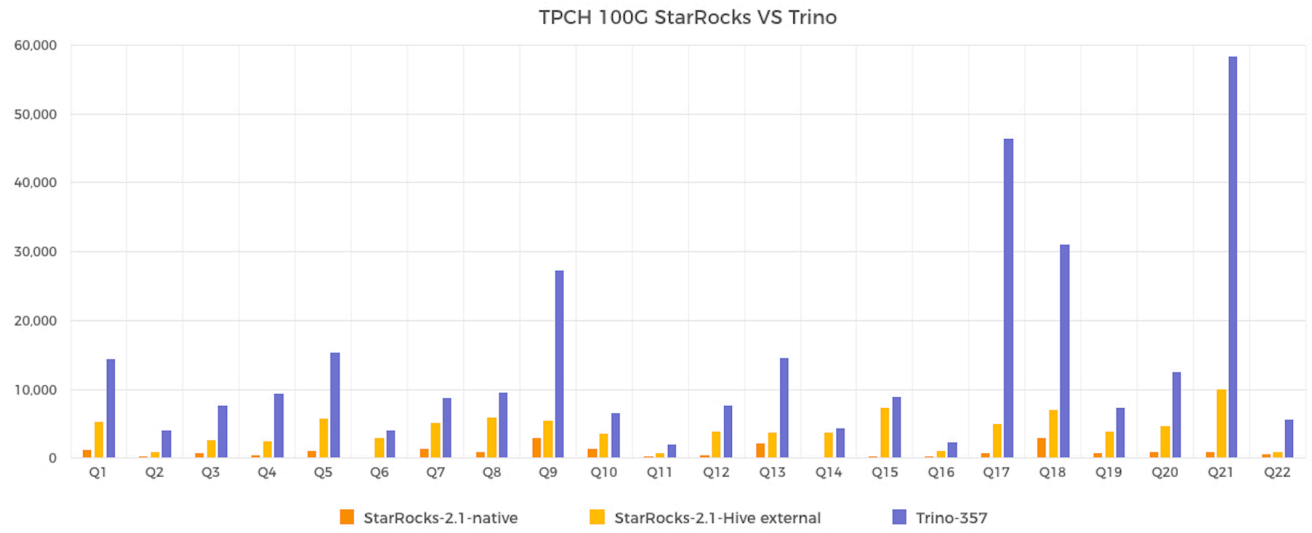
```
  `S_NATION` varchar(100) NOT NULL COMMENT "",
  `S_REGION` varchar(100) NOT NULL COMMENT "",
  `S_PHONE` varchar(100) NOT NULL COMMENT "",
  `P_NAME` varchar(100) NOT NULL COMMENT "",
  `P_MFGR` varchar(100) NOT NULL COMMENT "",
  `P_CATEGORY` varchar(100) NOT NULL COMMENT "",
  `P_BRAND` varchar(100) NOT NULL COMMENT "",
  `P_COLOR` varchar(100) NOT NULL COMMENT "",
  `P_TYPE` varchar(100) NOT NULL COMMENT "",
  `P_SIZE` tinyint(4) NOT NULL COMMENT "",
  `P_CONTAINER` varchar(100) NOT NULL COMMENT "",
  index bitmap_lo_orderpriority (lo_orderpriority) USING BITMAP,
  index bitmap_lo_shipmode (lo_shipmode) USING BITMAP,
  index bitmap_c_name (c_name) USING BITMAP,
  index bitmap_c_address (c_address) USING BITMAP,
  index bitmap_c_city (c_city) USING BITMAP,
  index bitmap_c_nation (c_nation) USING BITMAP,
  index bitmap_c_region (c_region) USING BITMAP,
  index bitmap_c_phone (c_phone) USING BITMAP,
  index bitmap_c_mktsegment (c_mktsegment) USING BITMAP,
  index bitmap_s_region (s_region) USING BITMAP,
  index bitmap_s_nation (s_nation) USING BITMAP,
  index bitmap_s_city (s_city) USING BITMAP,
  index bitmap_s_name (s_name) USING BITMAP,
  index bitmap_s_address (s_address) USING BITMAP,
  index bitmap_s_phone (s_phone) USING BITMAP,
  index bitmap_p_name (p_name) USING BITMAP,
  index bitmap_p_mfgr (p_mfgr) USING BITMAP,
  index bitmap_p_category (p_category) USING BITMAP,
  index bitmap_p_brand (p_brand) USING BITMAP,
  index bitmap_p_color (p_color) USING BITMAP,
  index bitmap_p_type (p_type) USING BITMAP,
  index bitmap_p_container (p_container) USING BITMAP
) ENGINE=OLAP
DUPLICATE KEY(`LO_ORDERDATE`, `LO_ORDERKEY`)
COMMENT "OLAP"
PARTITION BY date_trunc('year', `LO_ORDERDATE`)
DISTRIBUTED BY HASH(`LO_ORDERKEY`) BUCKETS 48
PROPERTIES ("replication_num" = "1");
```

TPC-H Benchmarking

TPC-H is a decision support benchmark developed by the Transaction Processing Performance Council (TPC). It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. TPC-H can be used to build models based on real production environments to simulate the data warehouse of a sales system. This test uses eight tables with a data size ranging from 1 GB to 3 TB. A total of 22 queries are tested and the main performance metrics are the response time of each query, which is the duration between the time a query is submitted to the time the result is returned.

1. Test Conclusion

We perform a test on 22 queries against a TPC-H 100 GB dataset. The following figure shows the test result.



StarRocks native storage and StarRocks Hive external tables are used for queries. We use StarRocks Hive external tables and Trino to query the same copy of data that is stored in the ORC format and compressed in the ZLIB format. StarRocks native storage has a query response time of 21s, that for StarRocks Hive external tables is 92s, and for Trino is as long as 307s.

2. Test Preparation

2.1 Hardware Environment

Machine	3 cloud hosts
CPU	16-core
Memory	64 GB
Network bandwidth	5 Gbit/s
Disk	System disk: ESSD 40 GB Data disk: ESSD 100 GB

2.2 Software Environment

- Kernel version: Linux 3.10.0-1127.13.1.el7.x86_64
- OS version: CentOS Linux released 7.8.2003
- Software version: StarRocks 2.1, Trino-357, Hive-3.1.2

> StarRocks FE can be separately deployed or hybrid deployed with BEs, which does not affect the test results.

3. Test Data and Results

3.1 Test Data

The test uses a TPC-H 100 GB dataset.

Table	Records	Data size after compression on StarRocks
customer	15 million	2.4 GB
lineitem	600 million	75 GB
nation	25	2.2 KB
orders	150 million	16 GB

Table	Records	Data size after compression on StarRocks
part	200 million	2.4 GB
partsupp	800 million	11.6 GB
region	5	389 B
supplier	1 million	0.14 GB

3.2 Test SQL

```
--Q1
select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as
sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from lineitem
where  l_shipdate <= date '1998-12-01'
group by  l_returnflag,  l_linestatus
order by  l_returnflag,  l_linestatus;

--Q2
select  s_acctbal, s_name,  n_name,  p_partkey,  p_mfgr,  s_address,
s_phone,  s_comment
from  part,  partsupp,  supplier,  nation,  region
where
  p_partkey = ps_partkey
  and s_suppkey = ps_suppkey
  and p_size = 15
  and p_type like '%BRASS'
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'EUROPE'
```

```
  and ps_supplycost = (
    select  min(ps_supplycost)
    from  partsupp,  supplier,  nation,  region
    where
      p_partkey = ps_partkey
      and s_suppkey = ps_suppkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'EUROPE'
  )
order by  s_acctbal desc, n_name,  s_name,  p_partkey;

--Q3
select
  l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
o_orderdate, o_shippriority
from
  customer, orders, lineitem
where
  c_mktsegment = 'BUILDING'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < date '1995-03-15'
  and l_shipdate > date '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate;

--Q4
select  o_orderpriority,  count(*) as order_count
from  orders
where
  o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and exists (
    select  * from  lineitem
    where  l_orderkey = o_orderkey and l_commitdate < l_receiptdate
  )
group by o_orderpriority
order by o_orderpriority;

--Q5
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem, supplier, nation, region
where
  c_custkey = o_custkey
  and l_orderkey = o_orderkey
```

```sql
  and l_suppkey = s_suppkey
  and c_nationkey = s_nationkey
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'ASIA'
  and o_orderdate >= date '1994-01-01'
  and o_orderdate < date '1994-01-01' + interval '1' year
group by n_name
order by revenue desc;

--Q6
select
  sum(l_extendedprice * l_discount) as revenue
from
  lineitem
where
  l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval '1' year
  and l_discount between .06 - 0.01 and .06 + 0.01
  and l_quantity < 24;

--Q7
select supp_nation, cust_nation, l_year, sum(volume) as revenue
from (
    select
      n1.n_name as supp_nation,
      n2.n_name as cust_nation,
      extract(year from l_shipdate) as l_year,
      l_extendedprice * (1 - l_discount) as volume
    from supplier, lineitem, orders, customer, nation n1, nation n2
    where
      s_suppkey = l_suppkey
      and o_orderkey = l_orderkey
      and c_custkey = o_custkey
      and s_nationkey = n1.n_nationkey
      and c_nationkey = n2.n_nationkey
      and (
        (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
        or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
      )
      and l_shipdate between date '1995-01-01' and date '1996-12-31'
  ) as shipping
group by supp_nation, cust_nation, l_year
order by supp_nation, cust_nation, l_year;

--Q8
```

```sql
select
  o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) /
sum(volume) as mkt_share
from
  (
    select
      extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) as volume,
      n2.n_name as nation
    from part, supplier, lineitem, orders, customer, nation n1,
nation n2, region
    where
      p_partkey = l_partkey
      and s_suppkey = l_suppkey
      and l_orderkey = o_orderkey
      and o_custkey = c_custkey
      and c_nationkey = n1.n_nationkey
      and n1.n_regionkey = r_regionkey
      and r_name = 'AMERICA'
      and s_nationkey = n2.n_nationkey
      and o_orderdate between date '1995-01-01' and date '1996-12-31'
      and p_type = 'ECONOMY ANODIZED STEEL'
  ) as all_nations
group by o_year
order by o_year;

--Q9
select nation, o_year, sum(amount) as sum_profit
from
  (
    select
      n_name as nation,
      extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity
as amount
    from part, supplier, lineitem, partsupp, orders, nation
    where
      s_suppkey = l_suppkey
      and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey
      and p_partkey = l_partkey
      and o_orderkey = l_orderkey
      and s_nationkey = n_nationkey
      and p_name like '%green%'
  ) as profit
group by nation, o_year
```

```sql
order by nation, o_year desc;

--Q10
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as
revenue, c_acctbal, n_name, c_address, c_phone, c_comment
    from customer, orders, lineitem, nation
    where
      c_custkey = o_custkey
      and l_orderkey = o_orderkey
      and o_orderdate >= date '1993-10-01'
      and o_orderdate < date '1993-12-01'
      and l_returnflag = 'R'
      and c_nationkey = n_nationkey
    group by c_custkey, c_name, c_acctbal, c_phone, n_name,
c_address, c_comment
    order by revenue desc;

--Q11
select ps_partkey, sum(ps_supplycost * ps_availqty) as value
from partsupp, supplier, nation
where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'GERMANY'
group by ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
      select sum(ps_supplycost * ps_availqty) * 0.0001000000
      from partsupp, supplier, nation
      where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'GERMANY'
    )
order by value desc;

--Q12
select
  l_shipmode,
  sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-
HIGH' then 1 else 0 end) as high_line_count,
  sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <>
'2-HIGH' then 1 else 0 end) as low_line_count
from orders, lineitem
where
  o_orderkey = l_orderkey
  and l_shipmode in ('MAIL', 'SHIP')
```

```
  and l_commitdate < l_receiptdate
  and l_shipdate < l_commitdate
  and l_receiptdate >= date '1994-01-01'
  and l_receiptdate < date '1994-01-01' + interval '1' year
group by l_shipmode
order by l_shipmode;

--Q13
select  c_count,  count(*) as custdist
from (select c_custkey,count(o_orderkey) as c_count
    from customer left outer join orders on c_custkey = o_custkey and
o_comment not like '%special%requests%'
    group by  c_custkey
) as c_orders
group by  c_count
order by custdist desc, c_count desc;

--Q14
select
  100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice *
(1 - l_discount) else 0 end) / sum(l_extendedprice * (1 -
l_discount)) as promo_revenue
from lineitem, part
where
  l_partkey = p_partkey
  and l_shipdate >= date '1995-09-01'
  and l_shipdate < date '1995-09-01' + interval '1' month;

--Q15
select s_suppkey, s_name, s_address, s_phone, total_revenue
from supplier, revenue0
where
  s_suppkey = supplier_no
  and total_revenue = (
    select max(total_revenue) from revenue0
  )
order by s_suppkey;

--Q16
select p_brand, p_type, p_size, count(distinct ps_suppkey) as
supplier_cnt
from partsupp, part
where
  p_partkey = ps_partkey
  and p_brand <> 'Brand#45'
  and p_type not like 'MEDIUM POLISHED%'
```

```
  and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
  and ps_suppkey not in (
    select s_suppkey from supplier where s_comment like
'%Customer%Complaints%'
  )
group by p_brand, p_type, p_size
order by supplier_cnt desc, p_brand, p_type, p_size;

--Q17
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where
  p_partkey = l_partkey
  and p_brand = 'Brand#23'
  and p_container = 'MED BOX'
  and l_quantity < (
    select 0.2 * avg(l_quantity) from lineitem where l_partkey =
p_partkey
  );

--Q18
select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
sum(l_quantity)
from customer, orders, lineitem
where
  o_orderkey in (
    select l_orderkey from lineitem
    group by l_orderkey having
        sum(l_quantity) > 300
  )
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc, o_orderdate;

--Q19
select sum(l_extendedprice* (1 - l_discount)) as revenue
from lineitem, part
where
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#12'
    and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    and l_quantity >= 1 and l_quantity <= 1 + 10
    and p_size between 1 and 5
    and l_shipmode in ('AIR', 'AIR REG')
```

```
    and l_shipinstruct = 'DELIVER IN PERSON'
  )
  or
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    and l_quantity >= 10 and l_quantity <= 10 + 10
    and p_size between 1 and 10
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  )
  or
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#34'
    and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    and l_quantity >= 20 and l_quantity <= 20 + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  );

--Q20
select s_name, s_address
from supplier, nation
where
  s_suppkey in (
    select ps_suppkey
    from partsupp
    where
      ps_partkey in (
        select p_partkey from part
        where p_name like 'forest%'
      )
      and ps_availqty > (
        select 0.5 * sum(l_quantity)
        from lineitem
        where
          l_partkey = ps_partkey
          and l_suppkey = ps_suppkey
          and l_shipdate >= date '1994-01-01'
          and l_shipdate < date '1994-01-01' + interval '1' year
      )
  )
  and s_nationkey = n_nationkey
```

```
  and n_name = 'CANADA'
order by s_name;

--Q21
select s_name, count(*) as numwait
from  supplier,  lineitem l1,  orders,  nation
where
  s_suppkey = l1.l_suppkey
  and o_orderkey = l1.l_orderkey
  and o_orderstatus = 'F'
  and l1.l_receiptdate > l1.l_commitdate
  and exists (
    select * from  lineitem l2
    where
      l2.l_orderkey = l1.l_orderkey
      and l2.l_suppkey <> l1.l_suppkey
  )
  and not exists (
    select * from  lineitem l3
    where
      l3.l_orderkey = l1.l_orderkey
      and l3.l_suppkey <> l1.l_suppkey
      and l3.l_receiptdate > l3.l_commitdate
  )
  and s_nationkey = n_nationkey
  and n_name = 'SAUDI ARABIA'
group by s_name
order by  numwait desc, s_name;


--Q22
select
  cntrycode,
  count(*) as numcust,
  sum(c_acctbal) as totacctbal
from
  (
    select
      substr(c_phone, 1, 2) as cntrycode,
      c_acctbal
    from
      customer
    where
      substr(c_phone, 1, 2) in
        ('13', '31', '23', '29', '30', '18', '17')
      and c_acctbal > (
```

```
        select
          avg(c_acctbal)
        from
          customer
        where
          c_acctbal > 0.00
          and substr(c_phone, 1, 2) in
            ('13', '31', '23', '29', '30', '18', '17')
      )
      and not exists (
        select
          *
        from
          orders
        where
          o_custkey = c_custkey
      )
  ) as custsale
group by
  cntrycode
order by
  cntrycode;
```

3.3 Test Results

Query	StarRocks-native (ms)	StarRocks-Hive external (ms)	Trino (ms)
q1	1291	5349	14350
q2	257	940	4075
q3	690	2585	7615
q4	529	2514	9385
q5	1142	5844	15350
q6	115	2936	4045
q7	1435	5128	8760

Query	StarRocks-native (ms)	StarRocks-Hive external (ms)	Trino (ms)
q8	855	5989	9535
q9	3028	5507	27310
q10	1381	3654	6560
q11	265	691	2090
q12	387	3827	7635
q13	2165	3733	14530
q14	184	3824	4355
q15	354	7321	8975
q16	240	1029	2265
q17	753	5011	46405
q18	2909	7032	30975
q19	702	3905	7315
q20	963	4623	12500
q21	870	10016	58340
q22	545	991	5605
sum	21060	92449	307975

4. Test Procedure

4.1 StarRocks Internal Tables

4.1.1 Generate Data

1. Download the tpch-poc toolkit and compile it.

```
wget https://starrocks-public.oss-cn-
zhangjiakou.aliyuncs.com/tpch-poc-0.1.2.zip
unzip tpch-poc-0.1.2.zip
cd tpch-poc-0.1.2
cd benchmark
```

   The related tools are stored in the `bin` directory and you can initiate operations in the home directory.

2. Generate data.

```
# Generate 100 GB data in the `data_100` directory.
./bin/gen_data/gen-tpch.sh 100 data_100
```

4.1.2 Create Table Schema

1. Modify the configuration file conf/starrocks.conf and specify the cluster address.

```
# StarRocks configuration
mysql_host: 192.168.1.1
mysql_port: 9030
mysql_user: root
mysql_password:
database: tpch

# Cluster ports
http_port: 8030
be_heartbeat_port: 9050
broker_port: 8000

# parallel_fragment_exec_instance_num specifies the parallelism.
# The recommended parallelism is half the CPU cores of a cluster.
The following example uses 8.
parallel_num: 8
```

2. Run the following script to create a table:

```
# create tables for 100GB data
./bin/create_db_table.sh ddl_100
```

The table has been created and the default bucket number has been specified. You can delete the previous table and re-plan the bucket number based on your cluster size and node configuration, which helps achieve better test results.

```
#Create a table named customer.
drop table if exists customer;
CREATE TABLE customer (
    c_custkey     int NOT NULL,
    c_name        VARCHAR(25) NOT NULL,
    c_address     VARCHAR(40) NOT NULL,
    c_nationkey   int NOT NULL,
    c_phone       VARCHAR(15) NOT NULL,
    c_acctbal     decimal(15, 2)   NOT NULL,
    c_mktsegment  VARCHAR(10) NOT NULL,
    c_comment     VARCHAR(117) NOT NULL
)ENGINE=OLAP
DUPLICATE KEY(`c_custkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`c_custkey`) BUCKETS 24
PROPERTIES (
    "replication_num" = "1",
    "in_memory" = "false",
    "storage_format" = "DEFAULT"
);
#Create a table named lineitem.
drop table if exists lineitem;
CREATE TABLE lineitem (
    l_shipdate    DATE NOT NULL,
    l_orderkey    int NOT NULL,
    l_linenumber  int not null,
    l_partkey     int NOT NULL,
    l_suppkey     int not null,
    l_quantity    decimal(15, 2) NOT NULL,
    l_extendedprice  decimal(15, 2) NOT NULL,
    l_discount    decimal(15, 2) NOT NULL,
    l_tax         decimal(15, 2) NOT NULL,
    l_returnflag  VARCHAR(1) NOT NULL,
    l_linestatus  VARCHAR(1) NOT NULL,
    l_commitdate  DATE NOT NULL,
    l_receiptdate DATE NOT NULL,
    l_shipinstruct VARCHAR(25) NOT NULL,
```

```
    l_shipmode     VARCHAR(10) NOT NULL,
    l_comment      VARCHAR(44) NOT NULL
)ENGINE=OLAP
DUPLICATE KEY(`l_shipdate`, `l_orderkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`l_orderkey`) BUCKETS 96
PROPERTIES (
    "replication_num" = "1",
    "in_memory" = "false",
    "storage_format" = "DEFAULT",
    "colocate_with" = "tpch2"
);
#Create a table named nation.
drop table if exists nation;
CREATE TABLE `nation` (
`n_nationkey` int(11) NOT NULL,
`n_name`      varchar(25) NOT NULL,
`n_regionkey` int(11) NOT NULL,
`n_comment`   varchar(152) NULL
) ENGINE=OLAP
DUPLICATE KEY(`N_NATIONKEY`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`N_NATIONKEY`) BUCKETS 1
PROPERTIES (
    "replication_num" = "3",
    "in_memory" = "false",
    "storage_format" = "DEFAULT"
);
#Create a talbe named orders.
drop table if exists orders;
CREATE TABLE orders  (
    o_orderkey       int NOT NULL,
    o_orderdate      DATE NOT NULL,
    o_custkey        int NOT NULL,
    o_orderstatus    VARCHAR(1) NOT NULL,
    o_totalprice     decimal(15, 2) NOT NULL,
    o_orderpriority  VARCHAR(15) NOT NULL,
    o_clerk          VARCHAR(15) NOT NULL,
    o_shippriority   int NOT NULL,
    o_comment        VARCHAR(79) NOT NULL
)ENGINE=OLAP
DUPLICATE KEY(`o_orderkey`, `o_orderdate`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`o_orderkey`) BUCKETS 96
PROPERTIES (
    "replication_num" = "1",
```

```
    "in_memory" = "false",
    "storage_format" = "DEFAULT",
    "colocate_with" = "tpch2"
);
#Create a table named part.
drop table if exists part;
CREATE TABLE part (
    p_partkey          int NOT NULL,
    p_name        VARCHAR(55) NOT NULL,
    p_mfgr        VARCHAR(25) NOT NULL,
    p_brand       VARCHAR(10) NOT NULL,
    p_type        VARCHAR(25) NOT NULL,
    p_size        int NOT NULL,
    p_container   VARCHAR(10) NOT NULL,
    p_retailprice decimal(15, 2) NOT NULL,
    p_comment     VARCHAR(23) NOT NULL
)ENGINE=OLAP
DUPLICATE KEY(`p_partkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`p_partkey`) BUCKETS 24
PROPERTIES (
    "replication_num" = "1",
    "in_memory" = "false",
    "storage_format" = "DEFAULT",
    "colocate_with" = "tpch2p"
);
#Create a table named partsupp.
drop table if exists partsupp;
CREATE TABLE partsupp (
    ps_partkey          int NOT NULL,
    ps_suppkey     int NOT NULL,
    ps_availqty    int NOT NULL,
    ps_supplycost  decimal(15, 2)  NOT NULL,
    ps_comment     VARCHAR(199) NOT NULL
)ENGINE=OLAP
DUPLICATE KEY(`ps_partkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`ps_partkey`) BUCKETS 24
PROPERTIES (
    "replication_num" = "1",
    "in_memory" = "false",
    "storage_format" = "DEFAULT",
    "colocate_with" = "tpch2p"
);
#Create a table named region.
drop table if exists region;
```

```
CREATE TABLE region  (
    r_regionkey      int NOT NULL,
    r_name       VARCHAR(25) NOT NULL,
    r_comment    VARCHAR(152)
)ENGINE=OLAP
DUPLICATE KEY(`r_regionkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`r_regionkey`) BUCKETS 1
PROPERTIES (
    "replication_num" = "3",
    "in_memory" = "false",
    "storage_format" = "DEFAULT"
);
#Create a table named supplier.
drop table if exists supplier;
CREATE TABLE supplier (
    s_suppkey       int NOT NULL,
    s_name        VARCHAR(25) NOT NULL,
    s_address     VARCHAR(40) NOT NULL,
    s_nationkey   int NOT NULL,
    s_phone       VARCHAR(15) NOT NULL,
    s_acctbal     decimal(15, 2) NOT NULL,
    s_comment     VARCHAR(101) NOT NULL
)ENGINE=OLAP
DUPLICATE KEY(`s_suppkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`s_suppkey`) BUCKETS 12
PROPERTIES (
    "replication_num" = "1",
    "in_memory" = "false",
    "storage_format" = "DEFAULT"
);
drop view if exists revenue0;
create view revenue0 (supplier_no, total_revenue) as
select
    l_suppkey,
    sum(l_extendedprice * (1 - l_discount))
from
    lineitem
where
    l_shipdate >= date '1996-01-01'
    and l_shipdate < date '1996-01-01' + interval '3' month
group by
    l_suppkey;
```

4.1.3 Import Data

```
./bin/stream_load.sh data_100
```

4.1.4 Collect Statistics (Optional)

This step is optional. We recommend that full statistical information be collected to generate better test results. Run the following commands after you connect to the StarRocks cluster:

```
ANALYZE FULL TABLE customer;
ANALYZE FULL TABLE lineitem;
ANALYZE FULL TABLE nation;
ANALYZE FULL TABLE orders;
ANALYZE FULL TABLE part;
ANALYZE FULL TABLE partsupp;
ANALYZE FULL TABLE region;
ANALYZE FULL TABLE supplier;
```

4.1.5 Query Data

```
./bin/benchmark.sh -p -d tpch
```

4.2 StarRocks Hive External Tables

4.2.1 Create Table Schema

Create table schema in Hive. The storage format is ORC.

```
create database tpch_hive_orc;
use tpch_hive_orc;
--Create a table named customer.
CREATE TABLE `customer`(
  `c_custkey` int,
  `c_name` varchar(25),
  `c_address` varchar(40),
  `c_nationkey` int,
  `c_phone` varchar(15),
  `c_acctbal` decimal(15,2),
  `c_mktsegment` varchar(10),
```

```
  `c_comment` varchar(117))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/customer';
--Create a table named lineitem.
  CREATE TABLE `lineitem`(
  `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` date,
  `l_commitdate` date,
  `l_receiptdate` date,
  `l_shipinstruct` varchar(25),
  `l_shipmode` varchar(10),
  `l_comment` varchar(44))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/lineitem';
--Create a table named nation.
CREATE TABLE `nation`(
  `n_nationkey` int,
```

```
  `n_name` varchar(25),
  `n_regionkey` int,
  `n_comment` varchar(152))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/nation';
--Create a table named orders.
CREATE TABLE `orders`(
  `o_orderkey` bigint,
  `o_custkey` int,
  `o_orderstatus` varchar(1),
  `o_totalprice` decimal(15,2),
  `o_orderdate` date,
  `o_orderpriority` varchar(15),
  `o_clerk` varchar(15),
  `o_shippriority` int,
  `o_comment` varchar(79))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/orders';
--Create a table named part.
CREATE TABLE `part`(
  `p_partkey` int,
  `p_name` varchar(55),
  `p_mfgr` varchar(25),
  `p_brand` varchar(10),
  `p_type` varchar(25),
  `p_size` int,
```

```
  `p_container` varchar(10),
  `p_retailprice` decimal(15,2),
  `p_comment` varchar(23))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/part';
--Create a table named partsupp.
CREATE TABLE `partsupp`(
  `ps_partkey` int,
  `ps_suppkey` int,
  `ps_availqty` int,
  `ps_supplycost` decimal(15,2),
  `ps_comment` varchar(199))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/partsupp';
--Create a table named region.
CREATE TABLE `region`(
  `r_regionkey` int,
  `r_name` varchar(25),
  `r_comment` varchar(152))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
```

```
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/region';
--Create a table named supplier.
CREATE TABLE `supplier`(
  `s_suppkey` int,
  `s_name` varchar(25),
  `s_address` varchar(40),
  `s_nationkey` int,
  `s_phone` varchar(15),
  `s_acctbal` decimal(15,2),
  `s_comment` varchar(101))
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'='|',
  'serialization.format'='|')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'hdfs://emr-header-1.cluster-
49146:9000/user/hive/warehouse/tpch_hive_orc.db/supplier';
```

Create external tables in StarRocks.

```
create database if not exists tpch_sr;
use tpch_sr;
CREATE EXTERNAL RESOURCE "hive_test" PROPERTIES (
  "type" = "hive",
  "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
CREATE EXTERNAL TABLE IF NOT EXISTS nation (
    n_nationkey int,
    n_name VARCHAR(25),
    n_regionkey int,
    n_comment varchar(152))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "nation",
    "database" = "tpch_hive_orc"
```

```
);
CREATE EXTERNAL TABLE IF NOT EXISTS customer (
    c_custkey     INTEGER,
    c_name        VARCHAR(25),
    c_address     VARCHAR(40),
    c_nationkey   INTEGER,
    c_phone       VARCHAR(15),
    c_acctbal     DECIMAL(15,2),
    c_mktsegment  VARCHAR(10),
    c_comment     VARCHAR(117))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "customer",
    "database" = "tpch_hive_orc"
);
CREATE EXTERNAL TABLE IF NOT EXISTS lineitem (
    l_orderkey    BIGINT,
    l_partkey     INTEGER,
    l_suppkey     INTEGER,
    l_linenumber  INTEGER,
    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    DATE,
    l_commitdate  DATE,
    l_receiptdate DATE,
    l_shipinstruct VARCHAR(25),
    l_shipmode     VARCHAR(10),
    l_comment      VARCHAR(44))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "lineitem",
    "database" = "tpch_hive_orc"
);
CREATE EXTERNAL TABLE IF NOT EXISTS orders  (
    o_orderkey       BIGINT,
    o_custkey        INTEGER,
    o_orderstatus    VARCHAR(1),
    o_totalprice     DECIMAL(15,2),
    o_orderdate      DATE,
    o_orderpriority  VARCHAR(15),
```

```
    o_clerk          VARCHAR(15),
    o_shippriority   INTEGER,
    o_comment        VARCHAR(79))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "orders",
    "database" = "tpch_hive_orc"
);
CREATE EXTERNAL TABLE IF NOT EXISTS part  (
    p_partkey     INTEGER,
    p_name        VARCHAR(55),
    p_mfgr        VARCHAR(25),
    p_brand       VARCHAR(10),
    p_type        VARCHAR(25),
    p_size        INTEGER,
    p_container   VARCHAR(10),
    p_retailprice DECIMAL(15,2),
    p_comment     VARCHAR(23))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "part",
    "database" = "tpch_hive_orc"
);
CREATE EXTERNAL TABLE IF NOT EXISTS partsupp (
    ps_partkey     INTEGER,
    ps_suppkey     INTEGER,
    ps_availqty    INTEGER,
    ps_supplycost  DECIMAL(15,2),
    ps_comment     VARCHAR(199))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "partsupp",
    "database" = "tpch_hive_orc"
);
CREATE EXTERNAL TABLE IF NOT EXISTS region  (
    r_regionkey  INTEGER,
    r_name       VARCHAR(25),
    r_comment    VARCHAR(152))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "region",
    "database" = "tpch_hive_orc"
```

```
);
CREATE EXTERNAL TABLE IF NOT EXISTS supplier (
    s_suppkey     INTEGER,
    s_name        VARCHAR(25),
    s_address     VARCHAR(40),
    s_nationkey   INTEGER,
    s_phone       VARCHAR(15),
    s_acctbal     DECIMAL(15,2),
    s_comment     VARCHAR(101))
ENGINE=hive
properties (
    "resource" = "hive_test",
    "table" = "supplier",
    "database" = "tpch_hive_orc"
);
create view revenue0 (supplier_no, total_revenue) as
select
    l_suppkey,
    sum(l_extendedprice * (1 - l_discount))
from
    lineitem
where
    l_shipdate >= date '1996-01-01'
    and l_shipdate < date '1996-01-01' + interval '3' month
group by
    l_suppkey;
```

4.2.2 Import Data

1. Create Hive external tables in Hive. The table format is CSV. Upload CSV test data to the data storage directory on HDFS. In this example, the HDFS data storage path of the Hive external table is /user/tmp/csv/.

```
create database tpch_hive_csv;
use tpch_hive_csv;

--Create an external table named customer.
CREATE EXTERNAL TABLE `customer`(
`c_custkey` int,
`c_name` varchar(25),
`c_address` varchar(40),
`c_nationkey` int,
`c_phone` varchar(15),
`c_acctbal` double,
```

```
`c_mktsegment` varchar(10),
`c_comment` varchar(117))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-
49146:9000/user/tmp/csv/customer_csv';
--Create an external table named lineitem.
CREATE EXTERNAL TABLE `lineitem`(
`l_orderkey` int,
`l_partkey` int,
`l_suppkey` int,
`l_linenumber` int,
`l_quantity` double,
`l_extendedprice` double,
`l_discount` double,
`l_tax` double,
`l_returnflag` varchar(1),
`l_linestatus` varchar(1),
`l_shipdate` date,
`l_commitdate` date,
`l_receiptdate` date,
`l_shipinstruct` varchar(25),
`l_shipmode` varchar(10),
`l_comment` varchar(44))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-
49146:9000/user/tmp/csv/lineitem_csv';
--Create an external table named nation.
CREATE EXTERNAL TABLE `nation`(
```

```
`n_nationkey` int,
`n_name` varchar(25),
`n_regionkey` int,
`n_comment` varchar(152))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-49146:9000/user/tmp/csv/nation_csv';
--Create an external table named orders.
CREATE EXTERNAL TABLE `orders`(
`o_orderkey` int,
`o_custkey` int,
`o_orderstatus` varchar(1),
`o_totalprice` double,
`o_orderdate` date,
`o_orderpriority` varchar(15),
`o_clerk` varchar(15),
`o_shippriority` int,
`o_comment` varchar(79))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-49146:9000/user/tmp/csv/orders_csv';
--Create an external table named part.
CREATE EXTERNAL TABLE `part`(
`p_partkey` int,
`p_name` varchar(55),
`p_mfgr` varchar(25),
`p_brand` varchar(10),
`p_type` varchar(25),
`p_size` int,
`p_container` varchar(10),
```

```
`p_retailprice` double,
`p_comment` varchar(23))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-49146:9000/user/tmp/csv/part_csv';
--Create an external table named partsupp.
CREATE EXTERNAL TABLE `partsupp`(
`ps_partkey` int,
`ps_suppkey` int,
`ps_availqty` int,
`ps_supplycost` double,
`ps_comment` varchar(199))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-
49146:9000/user/tmp/csv/partsupp_csv';
--Create an external table named region.
CREATE EXTERNAL TABLE `region`(
`r_regionkey` int,
`r_name` varchar(25),
`r_comment` varchar(152))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
```

```
LOCATION
'hdfs://emr-header-1.cluster-49146:9000/user/tmp/csv/region_csv';
--Create an external table named supplier.
CREATE EXTERNAL TABLE `supplier`(
`s_suppkey` int,
`s_name` varchar(25),
`s_address` varchar(40),
`s_nationkey` int,
`s_phone` varchar(15),
`s_acctbal` double,
`s_comment` varchar(101))
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'field.delim'='|',
'serialization.format'='|')
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://emr-header-1.cluster-
49146:9000/user/tmp/csv/supplier_csv';
```

2. Transfer CSV data in each Hive external table to a Hive table of the ORC format. This helps create Hive external tables of the ORC format in StarRocks and Trino for subsequent testing.

```
use tpch_hive_csv;
insert into tpch_hive_orc.customer  select * from customer;
insert into tpch_hive_orc.lineitem  select * from lineitem;
insert into tpch_hive_orc.nation  select * from nation;
insert into tpch_hive_orc.orders  select * from orders;
insert into tpch_hive_orc.part  select * from part;
insert into tpch_hive_orc.partsupp  select * from partsupp;
insert into tpch_hive_orc.region  select * from region;
insert into tpch_hive_orc.supplier  select * from supplier;
```

4.2.3 Query Data

```
use tpch_sr;
--Query system variables of current session.
show variables;
```

```
--Set the parallelism to 8.
set parallel_fragment_exec_instance_num = 8;

--Q1
select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as
sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from lineitem
where  l_shipdate <= date '1998-12-01'
group by  l_returnflag,  l_linestatus
order by  l_returnflag,  l_linestatus;

--Q2
select  s_acctbal, s_name,  n_name,  p_partkey,  p_mfgr,  s_address,
s_phone,  s_comment
from  part,  partsupp,  supplier,  nation,  region
where
  p_partkey = ps_partkey
  and s_suppkey = ps_suppkey
  and p_size = 15
  and p_type like '%BRASS'
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'EUROPE'
  and ps_supplycost = (
    select  min(ps_supplycost)
    from  partsupp,  supplier,  nation,  region
    where
      p_partkey = ps_partkey
      and s_suppkey = ps_suppkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'EUROPE'
  )
order by  s_acctbal desc, n_name,  s_name,  p_partkey;

--Q3
```

```sql
select
  l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
o_orderdate, o_shippriority
from
  customer, orders, lineitem
where
  c_mktsegment = 'BUILDING'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < date '1995-03-15'
  and l_shipdate > date '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate;

--Q4
select  o_orderpriority,  count(*) as order_count
from  orders
where
  o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and exists (
    select  * from  lineitem
    where  l_orderkey = o_orderkey and l_commitdate < l_receiptdate
  )
group by o_orderpriority
order by o_orderpriority;

--Q5
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem, supplier, nation, region
where
  c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and l_suppkey = s_suppkey
  and c_nationkey = s_nationkey
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'ASIA'
  and o_orderdate >= date '1994-01-01'
  and o_orderdate < date '1994-01-01' + interval '1' year
group by n_name
order by revenue desc;

--Q6
select
  sum(l_extendedprice * l_discount) as revenue
```

```
from
  lineitem
where
  l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval '1' year
  and l_discount between .06 - 0.01 and .06 + 0.01
  and l_quantity < 24;

--Q7
select supp_nation, cust_nation, l_year, sum(volume) as revenue
from (
    select
      n1.n_name as supp_nation,
      n2.n_name as cust_nation,
      extract(year from l_shipdate) as l_year,
      l_extendedprice * (1 - l_discount) as volume
    from supplier, lineitem, orders, customer, nation n1, nation n2
    where
      s_suppkey = l_suppkey
      and o_orderkey = l_orderkey
      and c_custkey = o_custkey
      and s_nationkey = n1.n_nationkey
      and c_nationkey = n2.n_nationkey
      and (
        (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
        or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
      )
      and l_shipdate between date '1995-01-01' and date '1996-12-31'
  ) as shipping
group by supp_nation, cust_nation, l_year
order by supp_nation, cust_nation, l_year;

--Q8
select
  o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) /
sum(volume) as mkt_share
from
  (
    select
      extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) as volume,
      n2.n_name as nation
    from part, supplier, lineitem, orders, customer, nation n1,
nation n2, region
    where
      p_partkey = l_partkey
```

```
      and s_suppkey = l_suppkey
      and l_orderkey = o_orderkey
      and o_custkey = c_custkey
      and c_nationkey = n1.n_nationkey
      and n1.n_regionkey = r_regionkey
      and r_name = 'AMERICA'
      and s_nationkey = n2.n_nationkey
      and o_orderdate between date '1995-01-01' and date '1996-12-31'
      and p_type = 'ECONOMY ANODIZED STEEL'
  ) as all_nations
group by o_year
order by o_year;

--Q9
select nation, o_year, sum(amount) as sum_profit
from
  (
    select
      n_name as nation,
      extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity
as amount
    from part, supplier, lineitem, partsupp, orders, nation
    where
      s_suppkey = l_suppkey
      and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey
      and p_partkey = l_partkey
      and o_orderkey = l_orderkey
      and s_nationkey = n_nationkey
      and p_name like '%green%'
  ) as profit
group by nation, o_year
order by nation, o_year desc;

--Q10
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as
revenue, c_acctbal, n_name, c_address, c_phone, c_comment
    from customer, orders, lineitem, nation
    where
      c_custkey = o_custkey
      and l_orderkey = o_orderkey
      and o_orderdate >= date '1993-10-01'
      and o_orderdate < date '1993-12-01'
      and l_returnflag = 'R'
      and c_nationkey = n_nationkey
```

```
    group by c_custkey, c_name, c_acctbal, c_phone, n_name,
c_address, c_comment
    order by revenue desc;

--Q11
select ps_partkey, sum(ps_supplycost * ps_availqty) as value
from partsupp, supplier, nation
where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'GERMANY'
group by ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
      select sum(ps_supplycost * ps_availqty) * 0.0001000000
      from partsupp, supplier, nation
      where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'GERMANY'
    )
order by value desc;

--Q12
select
  l_shipmode,
  sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-
HIGH' then 1 else 0 end) as high_line_count,
  sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <>
'2-HIGH' then 1 else 0 end) as low_line_count
from orders, lineitem
where
  o_orderkey = l_orderkey
  and l_shipmode in ('MAIL', 'SHIP')
  and l_commitdate < l_receiptdate
  and l_shipdate < l_commitdate
  and l_receiptdate >= date '1994-01-01'
  and l_receiptdate < date '1994-01-01' + interval '1' year
group by l_shipmode
order by l_shipmode;

--Q13
select  c_count,  count(*) as custdist
from (select c_custkey,count(o_orderkey) as c_count
    from customer left outer join orders on c_custkey = o_custkey and
o_comment not like '%special%requests%'
    group by  c_custkey
```

```
) as c_orders
group by  c_count
order by custdist desc, c_count desc;

--Q14
select
  100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice *
(1 - l_discount) else 0 end) / sum(l_extendedprice * (1 -
l_discount)) as promo_revenue
from lineitem, part
where
  l_partkey = p_partkey
  and l_shipdate >= date '1995-09-01'
  and l_shipdate < date '1995-09-01' + interval '1' month;

--Q15
select s_suppkey, s_name, s_address, s_phone, total_revenue
from supplier, revenue0
where
  s_suppkey = supplier_no
  and total_revenue = (
    select max(total_revenue) from revenue0
  )
order by s_suppkey;

--Q16
select p_brand, p_type, p_size, count(distinct ps_suppkey) as
supplier_cnt
from partsupp, part
where
  p_partkey = ps_partkey
  and p_brand <> 'Brand#45'
  and p_type not like 'MEDIUM POLISHED%'
  and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
  and ps_suppkey not in (
    select s_suppkey from supplier where s_comment like
'%Customer%Complaints%'
  )
group by p_brand, p_type, p_size
order by supplier_cnt desc, p_brand, p_type, p_size;

--Q17
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where
  p_partkey = l_partkey
```

```
  and p_brand = 'Brand#23'
  and p_container = 'MED BOX'
  and l_quantity < (
    select 0.2 * avg(l_quantity) from lineitem where l_partkey =
p_partkey
  );

--Q18
select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
sum(l_quantity)
from customer, orders, lineitem
where
  o_orderkey in (
    select l_orderkey from lineitem
    group by l_orderkey having
        sum(l_quantity) > 300
  )
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc, o_orderdate;

--Q19
select sum(l_extendedprice* (1 - l_discount)) as revenue
from lineitem, part
where
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#12'
    and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    and l_quantity >= 1 and l_quantity <= 1 + 10
    and p_size between 1 and 5
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  )
  or
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    and l_quantity >= 10 and l_quantity <= 10 + 10
    and p_size between 1 and 10
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  )
  or
```

```
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#34'
    and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    and l_quantity >= 20 and l_quantity <= 20 + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  );

--Q20
select s_name, s_address
from supplier, nation
where
  s_suppkey in (
    select ps_suppkey
    from partsupp
    where
      ps_partkey in (
        select p_partkey from part
        where p_name like 'forest%'
      )
      and ps_availqty > (
        select 0.5 * sum(l_quantity)
        from lineitem
        where
          l_partkey = ps_partkey
          and l_suppkey = ps_suppkey
          and l_shipdate >= date '1994-01-01'
          and l_shipdate < date '1994-01-01' + interval '1' year
      )
  )
  and s_nationkey = n_nationkey
  and n_name = 'CANADA'
order by s_name;

--Q21
select s_name, count(*) as numwait
from  supplier,  lineitem l1,  orders,  nation
where
  s_suppkey = l1.l_suppkey
  and o_orderkey = l1.l_orderkey
  and o_orderstatus = 'F'
  and l1.l_receiptdate > l1.l_commitdate
  and exists (
    select      *     from    lineitem l2
```

```
    where
      l2.l_orderkey = l1.l_orderkey
      and l2.l_suppkey <> l1.l_suppkey
  )
  and not exists (
    select      *       from    lineitem l3
    where
      l3.l_orderkey = l1.l_orderkey
      and l3.l_suppkey <> l1.l_suppkey
      and l3.l_receiptdate > l3.l_commitdate
  )
  and s_nationkey = n_nationkey
  and n_name = 'SAUDI ARABIA'
group by s_name
order by  numwait desc, s_name;

--Q22
select
  cntrycode,
  count(*) as numcust,
  sum(c_acctbal) as totacctbal
from
  (
    select
      substr(c_phone, 1, 2) as cntrycode,
      c_acctbal
    from
      customer
    where
      substr(c_phone, 1, 2) in
        ('13', '31', '23', '29', '30', '18', '17')
      and c_acctbal > (
        select
          avg(c_acctbal)
        from
          customer
        where
          c_acctbal > 0.00
          and substr(c_phone, 1, 2) in
            ('13', '31', '23', '29', '30', '18', '17')
      )
      and not exists (
        select
          *
        from
          orders
```

```
        where
          o_custkey = c_custkey
      )
  ) as custsale
group by
  cntrycode
order by
  cntrycode;
```

Developers

Contribute to StarRocks

Contributing to StarRocks is cordially welcome from everyone. Contributing to StarRocks is not limi...

build-starrocks

3 items

Use the debuginfo file for debugging

Change description

development-environment

1 item

trace-tools

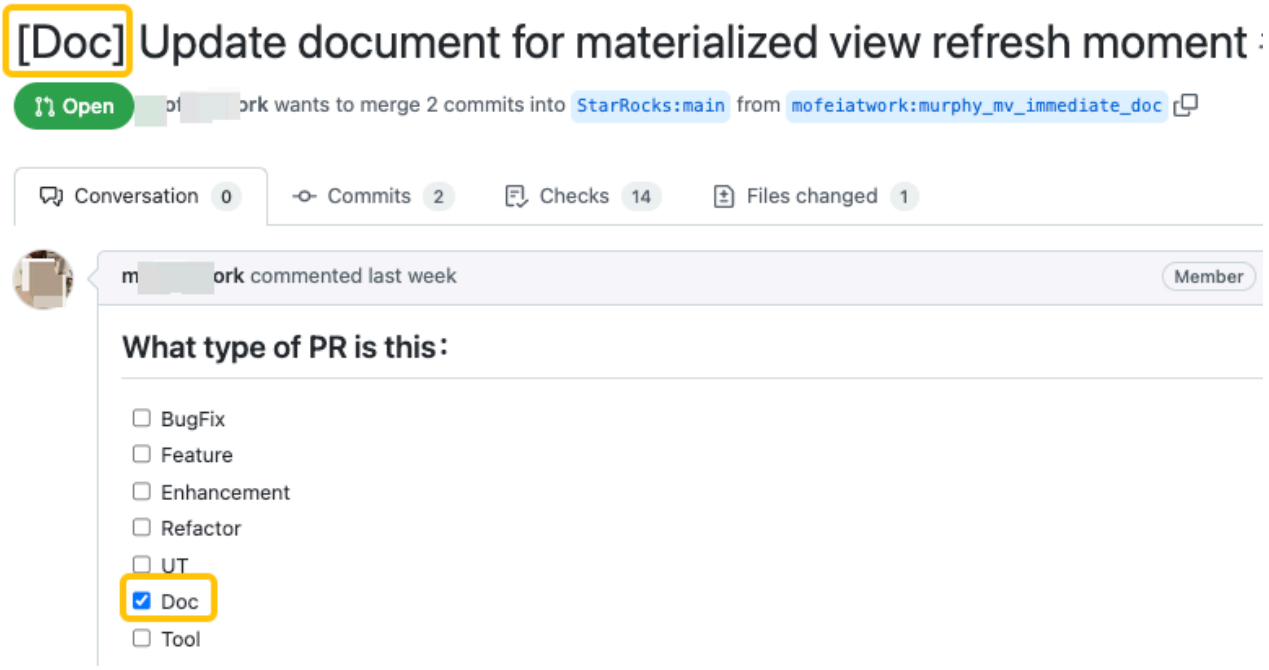
1 item

Contribute to StarRocks

Contributing to StarRocks is cordially welcome from everyone. Contributing to StarRocks is not limited to contributing code. Below, we list different approaches to contributing to our community.

Report a bug	**You can file an issue to report a bug with StarRocks. You can also click `Feedback` in the upper-right corner of the page you are reading in the Documentation Site to report a bug.**
Contribute code	You can contribute your code by fixing a bug or implementing a feature.
Contribute test case	You can contribute your test cases.
Help review code	If you are an active contributor or committer of StarRocks, you can help us review the pull requests (PRs).
Contribute documentation	StarRocks community maintains a tremendous amount of documentation both in Chinese and English. You can contribute documentation changes by fixing a documentation bug or proposing a new piece of content.
Help StarRocks users	You can help newcomers who meet difficulties in our community.
Spread the word about StarRocks	You can author an article or give a talk about us to help spread our technology to the world.

> **NOTE**
>
> To contribute documentation, remember to **sign off** your commit using `git commit -s`. Otherwise, the Developer Certificate of Origin (DCO) check cannot pass. In addition,

> prefix `[Doc]` to your PR title and select the **Doc** check box.
>
> 
>

Community resources

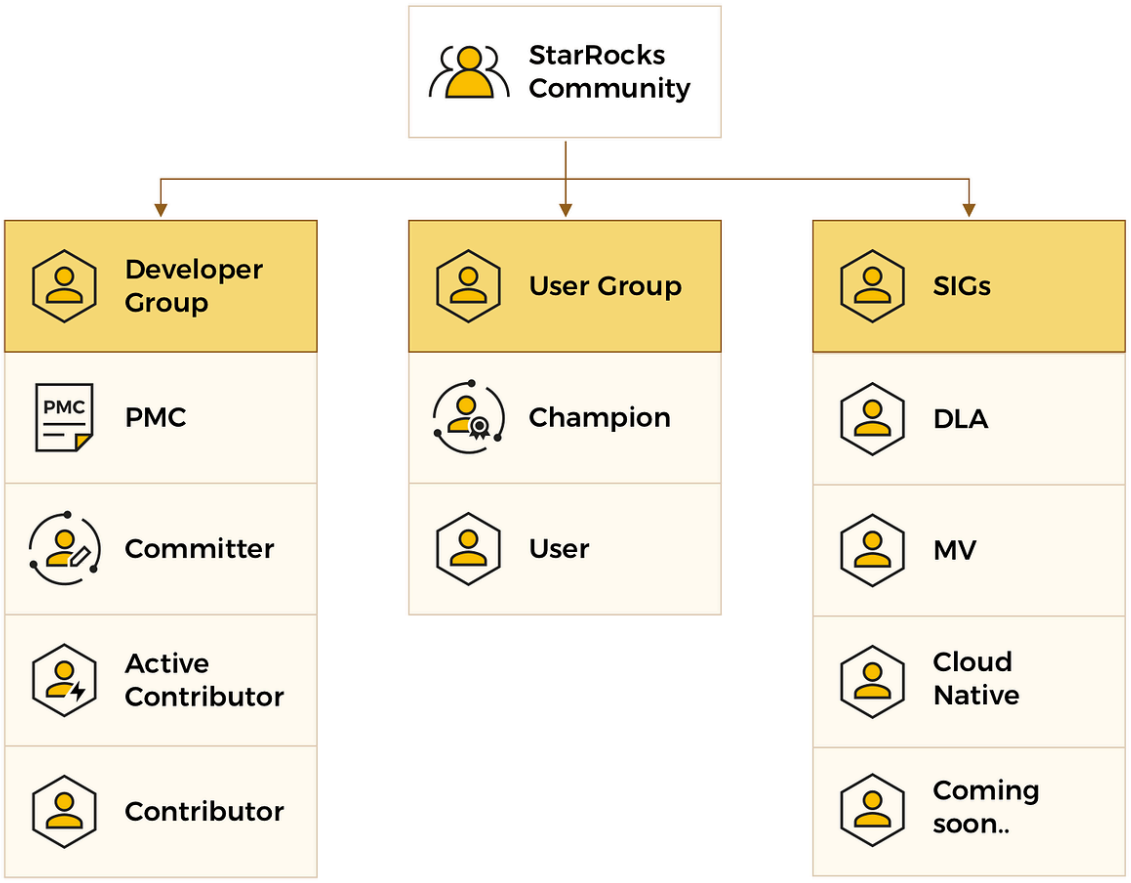
The best place to get a wide variety of help about StarRocks is via StarRocks's Slack Channel For contribution-related discussions, please go to the **#contributing-to-starrocks** channel.

You can also report issues and problems, or suggest new features, on GitHub.

Code of conduct

Our community strictly adheres to the code of conduct.

Community roles



Everyone is encouraged to participate in the StarRocks project. Anyone can make an impact by simply being involved in the discussions about new features, project roadmap, architecture, and even reporting issues you are facing.

The roles listed below are a few possible ways to get involved in the community, it also defines what is expected from each role.

User group

Participants

Participants are actively involved in the community and work to make StarRocks better for everyone.

As a participant, you can submit issue reports on GitHub, contribute test cases, translate/modify documentation, help answer user questions in forums or communities, participate in events, share your experience with StarRocks, star StarRocks on GitHub, and more.

Expectations and responsibilities:

- Follow the community's code of conduct.
- Be involved in discussions and community events.
- Share feedback with the community so everyone else knows what is/isn't working.
- Suggest improvements.

Champions

StarRocks Community Champions are a group of passionate community evangelists and pioneers who are well-versed in StarRocks technology. They enjoy sharing the latest developments and products in the community, and their enthusiasm drives the progress and development of the community.

Benefits

Product

- Early access to new products/features.
- Participate in exclusive meetings with the project core team.

Community

- Listed as a StarRocks champion on the website.
- Exclusive prizes and badges.

Events

- Free access to StarRocks' events.
- Expense reimbursement for travel and accommodations for events.
- VIP seats and VIP events.

Requirements

1. Have published at least 3 StarRocks-related original and quality technical content (including articles, tutorials, videos, etc.).

2. Participate in at least 2 online/offline technical sharing sessions as a speaker.
3. Help answer user questions on Slack and other channels.

Other Requirements

1. StarRocks Champions must abide by the StarRocks Community Code of Conduct.
2. StarRocks Champions are eligible for one year, and must reapply each year based on their contributions in the past year.

Developer Group

Contributor

Everyone who contributes can become a StarRocks contributor. The members will provide mentorship and guidance when new contributors need assistance.

How to become a Contributor?

- 1 merged PR in any StarRocks' public repos.

As a Contributor, we expect you to

- Actively participate in StarRocks' project development.
- Participate in community events (meetups, hackathons, etc.).
- Learn and help others learn StarRocks-related technologies.

Privileges

- Be listed as a StarRocks contributor.
- Be awarded a StarRocks Contributor e-certificate.

Active Contributor

Active contributors are contributors who have made outstanding contributions and sustained commitment to StarRocks. They actively participate in the community by contributing code, improving docs, and helping others.

How to become an Active Contributor?

- Have 5 merged PRs or fixed major bugs.
- Participate in more than 5 code reviews.
- Actively participate in community events such as online/offline meetups and community discussions.

Responsibilities and privileges

- Join the community meeting and discussion.
- Mentor and guide new contributors.
- Be listed as a StarRocks Active Contributor.
- Be awarded a StarRocks Active Contributor e-certificate.

Committer

Committers are promoted from Active Contributors.

They have the authority to merge PRs into master branches and are responsible for the planning and maintenance of StarRocks. They also are active members in sharing their knowledge with the community.

How to become a Committer?

- Have a deep understanding of StarRocks' principles and future plans.
- Have the ability to deal with various issues that arise in the project promptly.
- Lead a major development, write and revise related documents.
- Receive at least two PMC nominations and pass voting.

Responsibilities and privileges

- Mentor and guide other memberships in the community.
- Ensure continued health of subproject.

- Be granted write access to StarRocks repos (to be specified).
- Be listed as a StarRocks Committer.
- Be awarded a StarRocks Committer e-certificate.

PMC

PMCs are promoted from Committers. They have the authority to merge merge PRs into master branches and are responsible for the planning and maintenance of StarRocks. They also are active members in sharing their knowledge with the community.

How to become a PMC?

- In-depth understanding of StarRocks principles and a clear understanding of StarRocks' future plans.
- Have the ability to deal with project issues promptly.
- Lead project development and iterations, and steer the overall direction of the project.
- Receive at least two PMC nominations and pass voting.

Responsibilities and privileges

- Mentor and guide other memberships in the community.
- Ensure continued health of the project, such as code quality and test coverage.
- Make and approve technical design decisions.
- Define milestones and releases.
- Vote and promote new committers and PMCs.
- Be listed as a StarRocks PMC.
- Be awarded a StarRocks PMC e-certificate.

Contribution process

If you don't know how to get started, this is the process we suggest for contributions. This process is designed to help reduce your learning curve and get your pull requests merged more efficiently!

1. Sign the Contributor License Agreement (CLA).

2. Start a discussion by creating a Github issue, or asking on Slack channel unless the change is trivial).

   Before getting your hands on codes, you should comment in the issue body, and inform the maintainer to assign you the issue that you wish to solve. It is recommended to share your plan on how to solve this problem in the issue body as well.

   - This step helps you identify possible collaborators and reviewers.
   - Will the change conflict with another change in progress? If so, work with others to minimize impact.
   - Is this change major? If so, work with others to break the change into smaller steps.

3. Implement the change.

   - If the change is major, split it into smaller PRs.

   - Include tests and documentation as necessary.

4. Create a Github pull request:

   In StarRocks community, we follow the fork-and-merge GitHub workflow when contributing code.

   - Create a fork of StarRocks in your GitHub account.
   - Clone this forked repository to your computer.
   - Check out a new branch based on the branch you expect to contribute to.
   - Commit your code changes to the new branch.
   - Push the branch with code changes to GitHub.
   - Create a PR to submit your code changes. It is recommended to submit ONE commit in ONE PR. You can follow the PR templated when submitting a PR.
   - Make sure the pull request passes the tests in CI.

5. Review is required by at least 2 reviewers:

    - For StarRocks project, we require at least 2 lgtm from reviewers (committers) to merge the pull request.
    - This normally happens within a few days, but may take longer if the change is major, complex, or if a critical reviewer is unavailable. (feel free to ping the reviewer on the pull request).
6. Maintainers merge the pull request after the final changes are accepted.

Compile StarRocks with Docker

This topic describes how to compile StarRocks using Docker.

Overview

StarRocks provides development environment images for both Ubuntu 22.04 and CentOS 7.9. With the image, you can launch a Docker container and compile StarRocks in the container.

StarRocks version and DEV ENV image

Different branches of StarRocks correspond to different development environment images provided on StarRocks Docker Hub.

- For Ubuntu 22.04:

Branch name	Image name
main	starrocks/dev-env-ubuntu:latest
branch-3.0	starrocks/dev-env-ubuntu:3.0-latest
branch-2.5	starrocks/dev-env-ubuntu:2.5-latest

- For CentOS 7.9:

Branch name	Image name
main	starrocks/dev-env-centos7:latest

Branch name	Image name
branch-3.0	starrocks/dev-env-centos7:3.0-latest
branch-2.5	starrocks/dev-env-centos7:2.5-latest

Prerequisites

Before compiling StarRocks, make sure the following requirements are satisfied:

- **Hardware**

  Your machine must have at least 8 GB RAM.

- **Software**

  - Your machine must be running on Ubuntu 22.04 or CentOS 7.9.
  - You must have Docker installed on your machine.

Step 1: Download the image

Download the development environment image by running the following command:

```
# Replace <image_name> with the name of the image that you want to
download,
# for example, `starrocks/dev-env-ubuntu:latest`.
# Make sure you have choose the correct image for your OS.
docker pull <image_name>
```

Docker automatically identifies the CPU architecture of your machine and pulls the corresponding image that suits your machine. The `linux/amd64` images are for the x86-based CPUs, and `linux/arm64` images are for the ARM-based CPUs.

Step 2: Compile StarRocks in a Docker container

You can launch the development environment Docker container with or without the local host path mounted. We recommend you launch the container with the local host path mounted, so that you can avoid re-downloading the Java dependency during the next compilation, and you do not need to manually copy the binary files from the container to your local host.

- **Launch the container with the local host path mounted**:

  i. Clone the StarRocks source code to your local host.

  ```
  git clone https://github.com/StarRocks/starrocks.git
  ```

  ii. Launch the container.

  ```
  # Replace <code_dir> with the parent directory of the
  StarRocks source code directory.
  # Replace <branch_name> with the name of the branch that
  corresponds to the image name.
  # Replace <image_name> with the name of the image that you
  downloaded.
  docker run -it -v <code_dir>/.m2:/root/.m2 \
      -v <code_dir>/starrocks:/root/starrocks \
      --name <branch_name> -d <image_name>
  ```

  iii. Launch a bash shell inside the container you have launched.

  ```
  # Replace <branch_name> with the name of the branch that
  corresponds to the image name.
  docker exec -it <branch_name> /bin/bash
  ```

  iv. Compile StarRocks in the container.

  ```
  cd /root/starrocks && ./build.sh
  ```

- **Launch the container without the local host path mounted**:

  i. Launch the container.

```
# Replace <branch_name> with the name of the branch that
corresponds to the image name.
# Replace <image_name> with the name of the image that you
downloaded.
docker run -it --name <branch_name> -d <image_name>
```

ii. Launch a bash shell inside the container.

```
# Replace <branch_name> with the name of the branch that
corresponds to the image name.
docker exec -it <branch_name> /bin/bash
```

iii. Clone the StarRocks source code to the container.

```
git clone https://github.com/StarRocks/starrocks.git
```

iv. Compile StarRocks in the container.

```
cd starrocks && ./build.sh
```

Troubleshooting

Q: The StarRocks BE building fails, and the following error message has been returned:

```
g++: fatal error: Killed signal terminated program cc1plus
compilation terminated.
```

What should I do?

A: This error message indicates a lack of memory in the Docker container. You need to allocate at least 8 GB of memory resources to the container.

Compile StarRocks on Ubuntu

This topic describes how to compile StarRocks on the Ubuntu operating system. StarRocks supports compilation on both x86_64 and AArch64 architectures.

Prerequisites

Install Dependencies

Run the following commands to install necessary dependencies:

```
sudo apt-get update
```

```
sudo apt-get install automake binutils-dev bison byacc ccache flex
libiberty-dev libtool maven zip python3 python-is-python3 -y
```

Install Compiler

If you are using Ubuntu 22.04 or later, run the following command to install the tools and compilers:

```
sudo apt-get install cmake gcc g++ default-jdk -y
```

If you are using an Ubuntu version earlier than 22.04, run the following commands to check the versions of tools and compilers:

1. Check GCC/G++ versions:

   ```
   gcc --version
   g++ --version
   ```

GCC/G++ versions must be 10.3 or later. If you are using earlier versions, click here to install GCC/G++.

2. Check JDK version:

```
java --version
```

OpenJDK version must be 8 or later. If you are using an earlier version, click here to install OpenJDK.

3. Check CMake version:

```
cmake --version
```

CMake version must be 3.20.1 or later. If you are using an earlier version, click here to install CMake.

Compile StarRocks

Run the following command to start the compilation:

```
./build.sh
```

The default compilation parallelism is equal to **CPU core count/4**. Assuming you have 32 CPU cores, the default parallelism is 8.

If you want to adjust the parallelism, you can specify the number of CPU cores to be used for compilation via `-j` in the command line.

The following example uses 24 CPU cores for compilation:

```
./build.sh -j 24
```

FAQ

Q: Building `aws_cpp_sdk` fails on Ubuntu 20.04 with the error "Error: undefined reference to pthread_create". How can I resolve this?

A: This error occurs due to a lower version of CMake. Please upgrade CMake to version 3.20.1 or above.

How to build StarRocks

In general, you can build StarRocks by just executing

```
./build.sh
```

This command will check if all the thirdpary dependencies are ready at first. If all dependencies are ready, it will build StarRocks `Backend` and `Frontend`.

After this command executes succefully, the generated binary will be in `output` directory.

build FE/BE separately

You don't need to build both FE and BE each time, you can build them separately. For example, you can only build BE by

```
./build.sh --be
```

and, only build FE by

```
./build.sh --fe
```

How to run unit test

Unit tests of BE and FE are separted. In general, you can run BE test by

```
./run-be-ut.sh
```

run FE test by

```
./run-fe-ut.sh
```

How to run BE UT in command line

Now, BE UT needs some dependency to run, and `./run-be-ut.sh` helps it. But it is not flexible enough. When you want to run UT in the command-line, you can execute

```
UDF_RUNTIME_DIR=./ STARROCKS_HOME=./
LD_LIBRARY_PATH=/usr/lib/jvm/java-18-openjdk-amd64/lib/server
./be/ut_build_ASAN/test/starrocks_test
```

StarRocks Backend UT is built on top of google-test, so you can pass filter to run some of the UT, For example, you want to test only MapColumn related tests, you can execute

```
UDF_RUNTIME_DIR=./ STARROCKS_HOME=./
LD_LIBRARY_PATH=/usr/lib/jvm/java-18-openjdk-amd64/lib/server
./be/ut_build_ASAN/test/starrocks_test --gtest_filter="*MapColumn*"
```

Build options

build with clang

You can build StarRocks by `clang` too

```
CC=clang CXX=clang++ ./build.sh --be
```

Then you can see the following similar message in the build message

```
-- compiler Clang version 14.0.0
```

build with different linker

The default linker is slow, developer can specify different linker to speed up linking. For example, you can use `lld`, the LLVM-based linker.

You need to install `lld` firstly.

```
sudo apt install lld
```

Then you set the environment variable STARROCKS_LINKER with the linker you want to use. For example:

```
STARROCKS_LINKER=lld ./build.sh --be
```

build different type

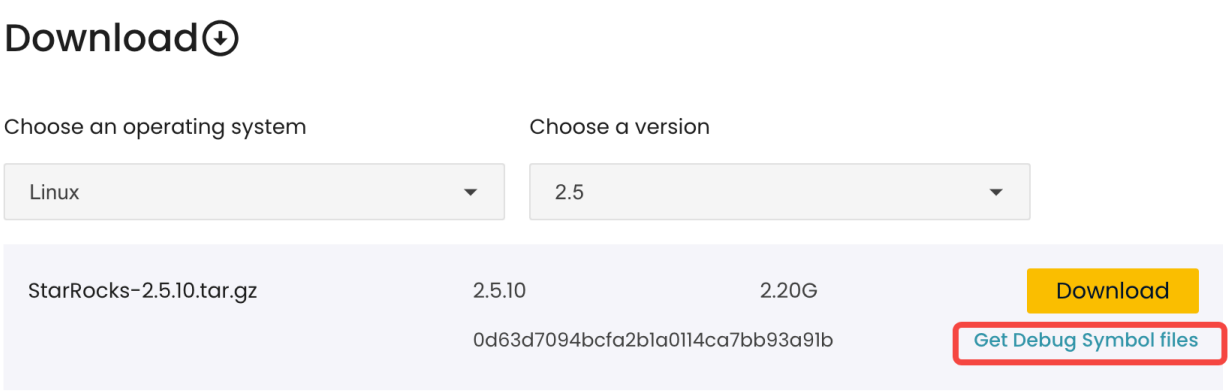
You can build StarRocks with different types with different BUILD_TYPE variable, the default BUILD_TYPE is `RELEASE`. For example, you can build StarRocks with `ASAN` type by

```
BUILD_TYPE=ASAN ./build.sh --be
```

Use the debuginfo file for debugging

Change description

From v2.5 onwards, the debuginfo file of BE is stripped from the StarRocks installation package to reduce the size and space usage of the installation package. You can see two packages at StarRocks website.



In this figure, you can click `Get Debug Symbol files` to download the debuginfo package. `StarRocks-2.5.10.tar.gz` is the installation package and you can click **Download** to download this package.

This change does not affect your download behavior or use of StarRocks. You can download only the installation package for cluster deployment and upgrade. The debuginfo package is only for developers to debug programs using GDB.

Precautions

GDB 12.1 or later is recommended for debugging.

How to use the debuginfo file

1. Download and decompress the debuginfo package.

```
wget https://releases.starrocks.io/starrocks/StarRocks-<sr_ver>.debuginfo.tar.gz

tar -xzvf StarRocks-<sr_ver>.debuginfo.tar.gz
```

> **NOTE**
>
> Replace `<sr_ver>` with the version number of the StarRocks installation package you want to download.

2. Load the debuginfo file when you perform GDB debugging.

   - **Method 1**

```
objcopy --add-gnu-debuglink=starrocks_be.debug starrocks_be
```

   This operation associates the debug info file with your executable file.

   - **Method 2**

```
gdb -s starrocks_be.debug -e starrocks_be -c `core_file`
```

The debuginfo file works well with perf and pstack. You can directly use perf and pstack without additional operations.

Setup StarRocks FE development environment on IDEA

This tutorial is based on macOS and has been tested on Apple Chip(M1, M2). Even if you are not using macOS, you can also refer to this tutorial.

Requirements

Thrift 0.13

If you installed Thrift directly with brew, you will find that there is no version 0.13, you can use the following command:

```
brew tap-new $USER/local-tap
brew extract --version='0.13.0' thrift $USER/local-tap
brew install thrift@0.13.0
```

After installing Thrift successfully, you can check by executing following command:

```
$ thrift -version
Thrift version 0.13.0
```

Protobuf

Just use the latest version v3, because the latest version of Protobuf is compatible with the v2 version of the Protobuf used in StarRocks.

```
brew install protobuf
```

Maven

```
brew install maven
```

Openjdk 1.8 or 11

```
brew install openjdk@11
```

Python3

MacOS is already installed by default.

Everyone's Thrift and Protobuf installation directories may be different, you can use the brew list command to inspect:

```
brew list thrift@0.13.0
brew list protobuf
```

Configure the StarRocks

Download the StarRocks

```
git clone https://github.com/StarRocks/starrocks.git
```

Setup thirdparty directory

Create `installed/bin` directory in `thirdparty`.

```
cd starrocks && mkdir -p starrocks/thirdparty/installed/bin
```

Then create soft link for Thrift and Protobuf respectively.

```
ln -s /opt/homebrew/bin/thrift thirdparty/installed/bin/thrift
ln -s /opt/homebrew/bin/protoc thirdparty/installed/bin/protoc
```

Setting environment variables

```
export JAVA_HOME="/opt/homebrew/Cellar/openjdk@11/11.0.15" (Caution:
The jdk stable version number may be changed)
export PYTHON=/usr/bin/python3
export STARROCKS_THIRDPARTY=/Users/smith/Code/starrocks/thirdparty
```

Generate source code

Many source files in FE need to be generated manually, otherwise IDEA will report an error due to missing files. Execute the following command to automatically generate:

```
cd gensrc
make clean
make
```

Compile FE

Enter `fe` directory and use Maven to compile:

```
cd fe
mvn install -DskipTests
```

Open StarRocks in IDEA

1. Open the `StarRocks` directory in IDEA.

2. Add Coding style setting To standardize the coding style, you should import the `fe/starrocks_intellij_style.xml` code style file in IDEA.

Run StarRocks FE in MacOS

Use IDEA to open the `fe` directory.

If you execute the Main function directly in `StarRocksFE.java`, some errors will be reported. You only need to do some simple settings to run it smoothly.

NOTICE: `StarRocksFE.java` is in the `fe/fe-core/src/main/java/com/starrocks` directory.

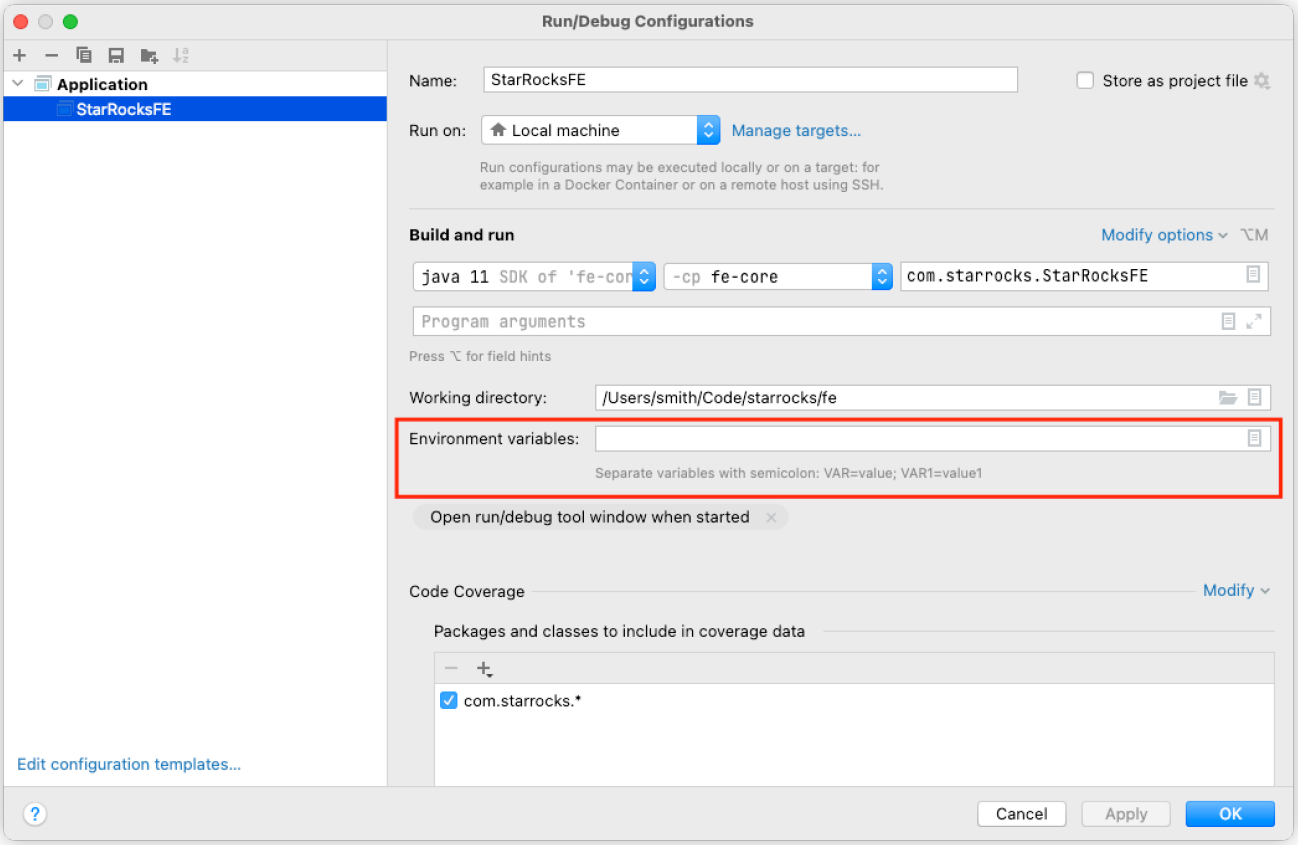
1. Copy the conf, bin and webroot directories from the StarRocks directory to `fe` directory:

```
cp -r conf fe/conf
cp -r bin fe/bin
cp -r webroot fe/webroot
```

2. Enter the `fe` directory and create the log and meta folders under the `fe` directory:

```
cd fe
mkdir log
mkdir meta
```

3. Set the environment variable, as shown in the following figure:

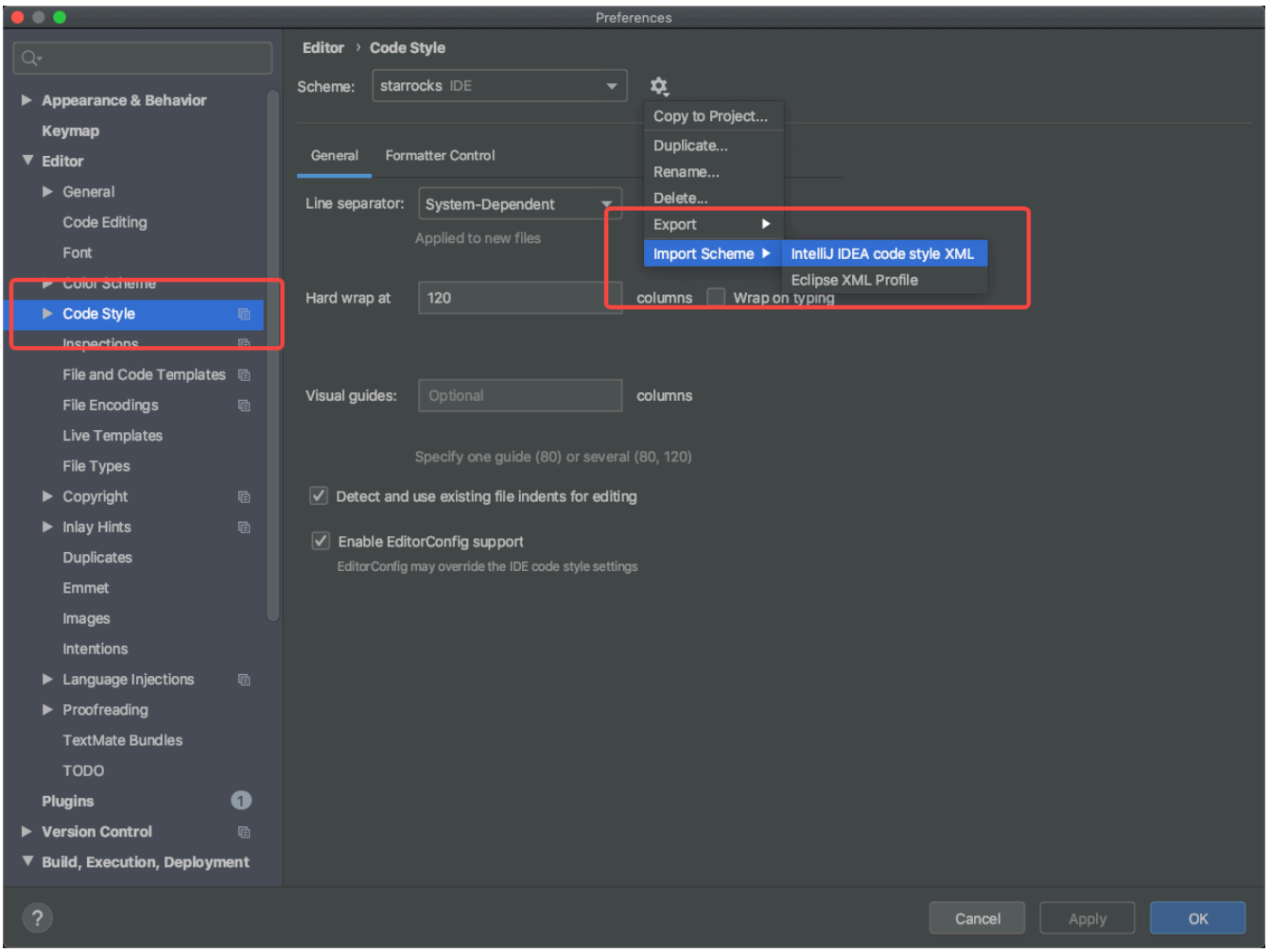



```
export PID_DIR=/Users/smith/Code/starrocks/fe/bin
export STARROCKS_HOME=/Users/smith/Code/starrocks/fe
```

```
export LOG_DIR=/Users/smith/Code/starrocks/fe/log
```

4. Modify the priority_networks in `fe/conf/fe.conf` to `127.0.0.1/24` to prevent FE from using the current computer's LAN IP and cause the port fail to bind.

5. Then you have run StarRocks FE successfully.



DEBUG StarRocks FE in MacOS

If you started the FE with the debug option, you can then attach the IDEA debugger to the FE process.

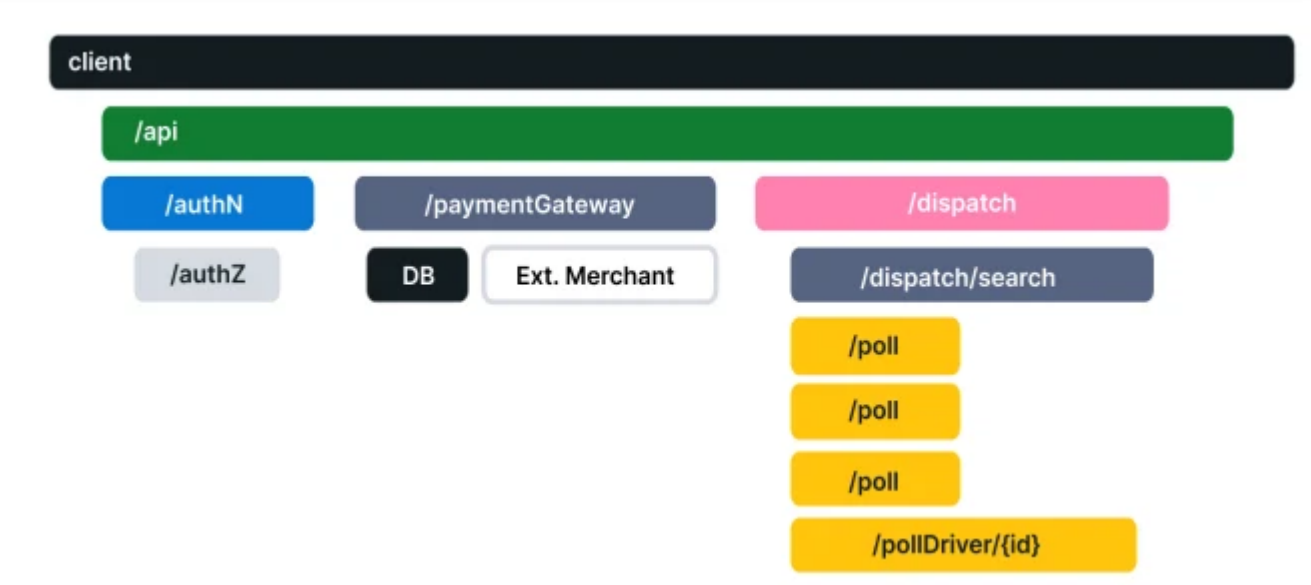
```
./start_fe.sh --debug
```

See https://www.jetbrains.com/help/idea/attaching-to-local-process.html#attach-to-local.

Trace

Background

A Distributed Trace, more commonly known as a Trace, records the paths taken by requests (made by an application or end-user) as they propagate through multi-service architectures, like microservice and serverless applications. Without tracing, it is challenging to pinpoint the cause of performance problems in a distributed system. It improves the visibility of our application or system's health and lets us debug behavior that is difficult to reproduce locally. Tracing is essential for distributed systems, which commonly have nondeterministic problems or are too complicated to reproduce locally. Tracing makes debugging and understanding distributed systems less daunting by breaking down what happens within a request as it flows through a distributed system. A Trace is made of one or more Spans. The first Span represents the Root Span. Each Root Span represents a request from start to finish. The Spans underneath the parent provide a more in-depth context of what occurs during a request (or what steps make up a request). Many Observability back-ends visualize Traces as waterfall diagrams that may look something like this picture.



Waterfall diagrams show the parent-child relationship between a Root Span and its child Spans. When a Span encapsulates another Span, this also represents a nested relationship.

Recently SR added a tracing framework. It leverages opentelemetry and jaeger to trace distributed events in the system.

- Opentelemetry is an instrumentation/tracing SDK. Developers can use it to instrument code and emit tracing data to an observability backend. It supports many languages. We use java and CPP SDK in SR.
- Currently, Jaeger is used as the observability backend.

Basic Usage

Steps to enable tracing in SR:

1. Install Jaeger The guide above uses docker. For simplicity, you can also just download binary package and run locally, `https://github.com/jaegertracing/jaeger/releases`

```
    decster@decster-MS-7C94:~/soft/jaeger-1.31.0-linux-amd64$ ll
    total 215836
    drwxr-xr-x  2 decster decster     4096 02-05 05:01:30 ./
    drwxrwxr-x 28 decster decster     4096 05-18 18:24:07 ../
    -rwxr-xr-x  1 decster decster 19323884 02-05 05:01:31 example-
hotrod*
    -rwxr-xr-x  1 decster decster 23430444 02-05 05:01:29 jaeger-
agent*
    -rwxr-xr-x  1 decster decster 51694774 02-05 05:01:29 jaeger-all-
in-one*
    -rwxr-xr-x  1 decster decster 41273869 02-05 05:01:30 jaeger-
collector*
    -rwxr-xr-x  1 decster decster 37576660 02-05 05:01:30 jaeger-
ingester*
    -rwxr-xr-x  1 decster decster 47698843 02-05 05:01:30 jaeger-
query*

    decster@decster-MS-7C94:~/soft/jaeger-1.31.0-linux-amd64$
./jaeger-all-in-one
```

2. Config FE&FE to enable tracing Currently, opentelemetry java & cpp sdk use different protocols, java uses grpc proto, while cpp uses thrift&UDP, so the endpoint ports are different.

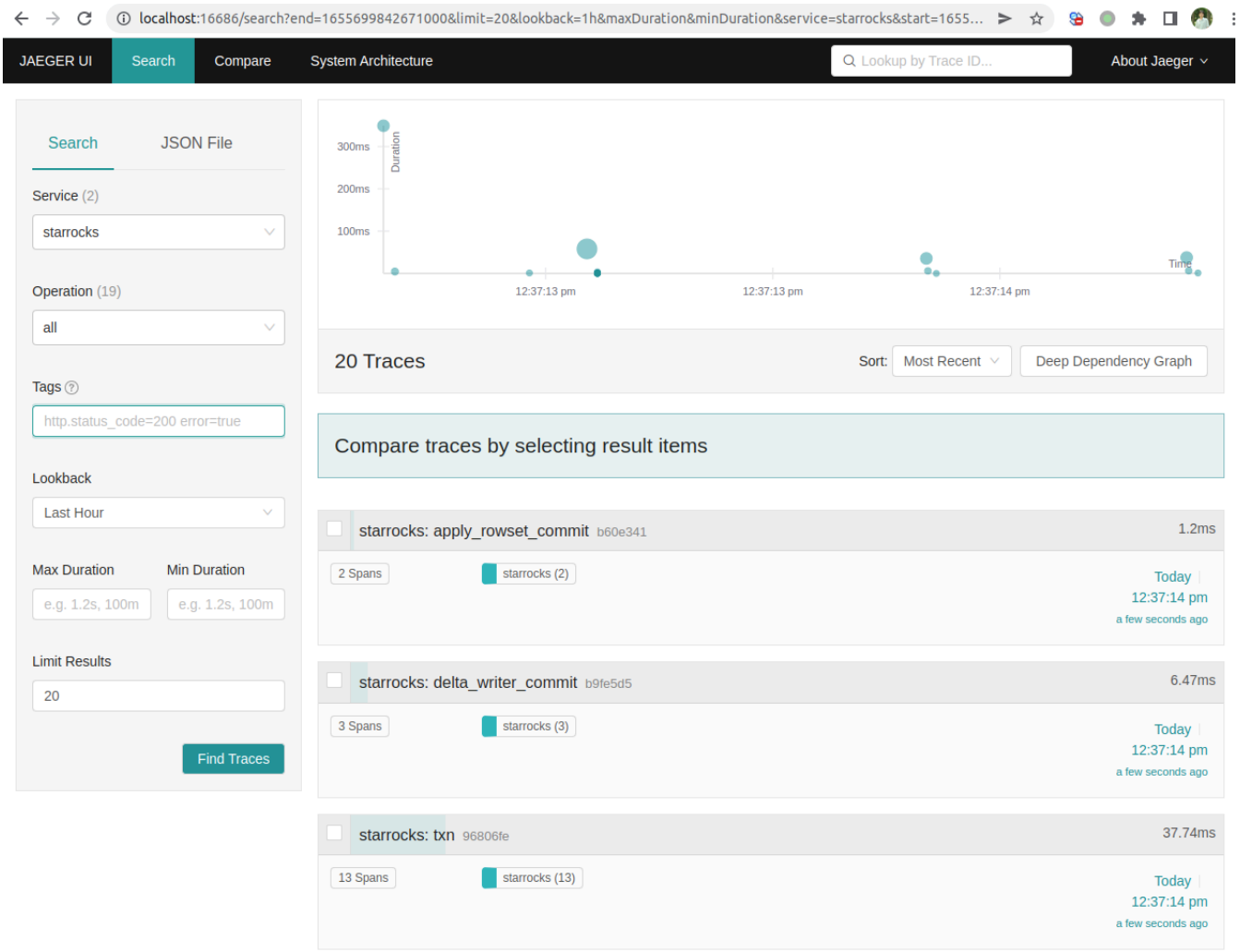
```
    fe.conf

    # Enable jaeger tracing by setting jaeger_grpc_endpoint
    # jaeger_grpc_endpoint = http://localhost:14250
```

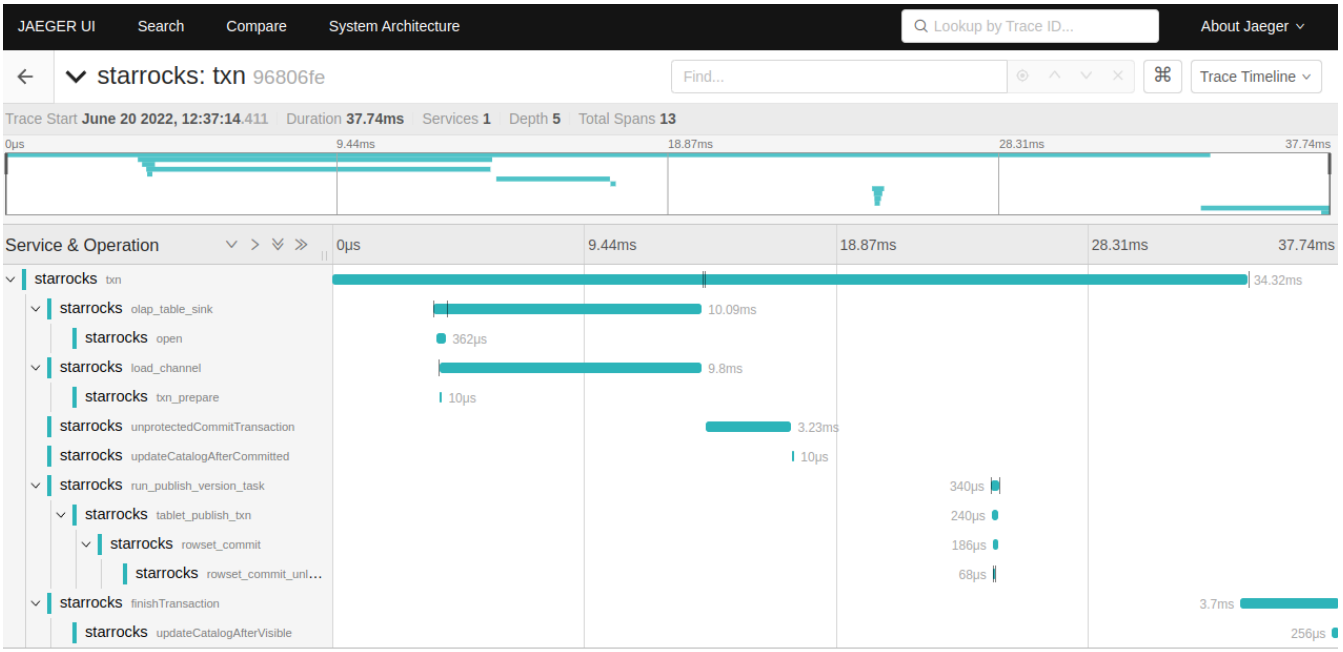
```
be.conf

# Enable jaeger tracing by setting jaeger_endpoint
# jaeger_endpoint = localhost:6831
```

3. Open jaeger web UI, usually in `http://localhost:16686/search`
4. Do some data ingestion (streamload/insert into) and search TXN traces on web UI



(trace_pic2.png)



(trace_pic3.png)

Adding traces

- To add trace, first get familiar with basic concepts like tracer, span, trace propagation read the observability primer.
- Read utility class and it's usages in SR: TraceManager.java(java) `common/tracer.h/cpp (cpp)`, it's current usage(like write txn(load/insert/update/delete) trace, and its propagation to BE).
- Add your own trace