Alibaba Cloud Realtime Compute（StreamCompute）

Flink SQL Development Guide

Issue: 20190722

MORE THAN JUST CLOUD | Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades , adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults " and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity , applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	It is used for commands.	Run the `cd / d C :/ windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*

Style	Description	Example
{} or {a|b}	**It indicates that it is a required value, and only one item can be selected.**	`swich` *{stand | slave}*

Contents

1 Data storage

1.1 Data storage overview

Realtime Compute provides a management page for various data storage systems, such as ApsaraDB for RDS and Table Store. It offers you an end-to-end cloud-based data storage management solution.

Data storage in Realtime Compute has two meanings. First, it refers to the data storage systems or database tables (hereinafter referred to as "storage resources") at the input and output nodes of Realtime Compute. Second, it refers to how Realtime Compute manages the input and output storage resources (hereinafter referred to as "the data storage feature"). Realtime Compute supports two methods for using input and output storage resources: plaintext referencing and data storage resource registration.

Note:

To allow Realtime Compute to access the data storage resources, you must grant Realtime Compute the permission to access these resources in advance. For more information about how to verify whether Realtime Compute already has the permission, and about how to grant the permission to Realtime Compute, see Create a role for Realtime Compute exclusive mode and grant permissions to the role.

Plaintext referencing

You can explicitly reference the input and output storage resources by using AccessKey ID or AccessKey Secret in the WITH function of the DDL statement of your Realtime Compute job. For more information, see EN-US_TP_40871.dita#concept_62515_zh. Plaintext referencing allows you to authorize Realtime Compute of both the same and different accounts (including the Alibaba Cloud account and the RAM user). Assume that user A (including RAM users created under the Alibaba Cloud account of user A) wants to use storage resources of user B in Realtime Compute. User A can reference storage resources of user B in plaintext mode by defining the reference in the DDL statement as follows:

```
CREATE   TABLE   in_stream (
  a   varchar ,
  b   varchar ,
```

```
    c    timeStamp
) with (
 type =' datahub ',
   endPoint =' http :// dh - cn - hangzhou . aliyuncs . com ',
   project =' blink_test ',
   topic =' ip_count02 ',
   accessId ='< yourAccess  Secret >', --  AccessKey   ID   granted
 by   user   B .
   accessKey ='< yourProjec  tName > --  AccessKey   Secret   granted
   by   user   B .
);
```

Data storage resource registration

To help you manage input and output storage resources, Realtime Compute offers the data storage management feature. This feature offers many conveniences, such as data preview, data sampling, and auto DDL generation, to help you manage your cloud-based storage resources. All you have to do is to register these resources on the Realtime Compute development platform in advance.

> **Note:**
> The data storage resource registration feature of Realtime Compute currently supports only storage resources under the same Alibaba Cloud account. In other words, user A (including RAM users created under the Alibaba Cloud account of user A) can only register storage resources purchased by user A. This feature does not support registration of storage resources under different Alibaba Cloud accounts. To use storage resources of a different Alibaba Cloud account, use the plaintext referencing method.

- Data storage resource registration

  In the left-side navigation pane of the Development Platform page, click Data Storage. Then, click Registration and Connection in the upper-right corner to go to the data storage resource registration page.

  Realtime Compute allows you to register three types of storage resources. For more information about the registration methods, click the following links:

  - Register Table Store resources
  - Register ApsaraDB for RDS resources
  - Register Log Service resources
- Data preview

  Realtime Compute offers a data preview feature for registered storage resources. Click Data Storage and select a data storage type to preview related data.

- Data sampling

  Realtime Compute offers a data sampling feature for registered storage resources. Click Data Storage, select a data storage type, and click Data Sampling to enter the data sampling page. You can click Download Data in the upper-right corner to download the sampled data.

- Auto DDL generation

  Realtime Compute offers an auto DDL generation feature for registered storage resources. On the job edit page, click Data Storage, select a data storage type, and click Reference as Source Table, Reference as Result Table, or Reference as Dimension Table to enable auto DDL generation.

  The automatically generated DDL statement contains only the basic WITH parameters to ensure the smooth connection between Realtime Compute and the storage resources. You can add other WITH parameters to the DDL statement as needed.

- Network detection

  The data storage feature of Realtime Compute also provides a network detection feature. This feature is used to detect the network connectivity between Realtime Compute and the target storage resource. You can choose Data Storage > Registration and Connection, and enable the network detection feature.

1.2 Data storage resource registration

1.2.1 Register Table Store resources

This topic describes how to connect to Table Store by registering Table Store resources in Realtime Compute.

Register Table Store resources

Table Store is a NoSQL data storage service built on the Apsara system of Alibaba Cloud. It enables you to store and access large amounts of structured data in real time. Realtime Compute has high requirements on access delay, and has low requirements on the relational algebra. Therefore, Table Store dimension tables and result tables are very suitable for Realtime Compute.

> Note:

If you encounter errors that are similar to the `No Permission` error when you use Table Store, grant Realtime Compute the permission to access Table Store. For more information, see Create a role for Realtime Compute exclusive mode and grant permissions to the role.

- Endpoint
  - Enter the endpoint of Table Store. View the endpoint information of Table Store in the Table Store console, and enter the internal IP address of Table Store. For more information, see Endpoint.
  - Set Accessed By to Any Network. Log on to the Table Store console. Choose Instance Details > Network > Change > Any Network.
- Instance name

  Enter the name of the Table Store instance.

1.2.2 Register ApsaraDB for RDS resources

This topic describes how to access ApsaraDB for RDS by registering ApsaraDB for RDS resources in Realtime Compute.

ApsaraDB for RDS (RDS for short) is a stable, reliable, and scalable online database service. Realtime Compute supports several RDS engines such as MySQL.

Note:

- When you use ApsaraDB for RDS on Alibaba Cloud, you need to grant Realtime Compute the permission to access ApsaraDB for RDS. For more information, see Create a role for Realtime Compute exclusive mode and grant permissions to the role. Otherwise, a `No Permission` error may be returned.
- Realtime Compute uses relational databases such as MySQL to store result data. The relational databases use Taobao Distributed Data Layer (TDDL) and RDS connectors. When Realtime Compute frequently writes data into a table or resource file, a deadlock may occur. In scenarios that require high queries per second (QPS), high transactions per second (TPS), or highly concurrent write operations, we recommend that you avoid using TDDL or RDS result tables. To prevent deadlocks, we recommend that you use Table Store result tables.

Register ApsaraDB for RDS resources

Note:
During the registration process, the system automatically configures the IP address whitelist for ApsaraDB for RDS.

Specify related information:

- Region

  Select the region where your ApsaraDB for RDS instance is located.
- Instance

  Enter the ID of the ApsaraDB for RDS instance.

  Note:
  Enter the instance ID, rather than the instance name.
- DBName

  Enter the name of the database corresponding to the ApsaraDB for RDS instance.

  Note:
  The database name is not the instance name.
- Username

  Enter the username that you use to log on to the database.
- Password

  Enter the password that you use to log on to the database.
- Whitelist Authorization

  ApsaraDB for RDS uses the whitelist feature to ensure data security. When you register ApsaraDB for RDS resources in Realtime Compute, the system automatica lly configures a whitelist for ApsaraDB for RDS.

1.2.3 Register Log Service resources

This topic describes how to connect to Log Service by registering Log Service resources in Realtime Compute, and lists FAQs during the registration process.

Register Log Service resources

Log Service (LOG for short) is formerly known as SLS. Log Service is an end-to-end log management solution. Log Service offers many features such as data collection , subscription, dumping, and query of large amounts of logs. Log Service is the log management platform of Alibaba Cloud. After you implement the management of ECS logs by using Log Service, Realtime Compute can directly connect to LogHub of Log Service. You do not need to migrate your data.

Note:
When you use Log Service on Alibaba Cloud, you need to grant Realtime Compute the permission to access Log Service. For more information, see Create a role for Realtime Compute exclusive mode and grant permissions to the role. Otherwise, you may receive a `No Permission` error.

- Enter the endpoint

  Enter the endpoint of Log Service. Log Service has different endpoints in different regions. For more information, see Service endpoint.

  Note:
  - The endpoint must start with `http ://` and cannot end with `/`, such as `http :// cn - hangzhou - intranet . log . aliyuncs . com` .
  - If your Realtime Compute and Log Service are both deployed in the intranet of Alibaba Cloud, we recommend that you enter a service endpoint of a classic network or VPC. To avoid consuming large amounts of Internet bandwidth and causing possible performance issues, we do not recommend that you enter an Internet endpoint.
  - For more information about how to enter the endpoint of Log Service on Apsara Stack, contact your Apsara Stack system administrator.

- Enter the project name

  Enter the Log Service project name.

  > **Note:**
  > You can only register data storage resources under the same Alibaba Cloud account. For example, user A owns Log Service project A, and user B wants to use project A in Realtime Compute. Realtime Compute does not allow user B to register project A, but user B can explicitly reference project A in the code. For more information, see Plaintext referencing.

FAQ

Question: Why does an error (error message XXX) occur when I register data storage resources?

Answer: Realtime Compute uses a storage software development kit (SDK) to access different data storage systems. The Data Storage tab on the Realtime Compute development platform helps you manage data from different data storage systems by using this SDK. Therefore, in most cases, the errors are caused during your registrati on process. Please perform the following operations to troubleshoot the problem.

- Check whether you have created the Log Service project and own the project. Log on to the Log Service console with your Alibaba Cloud account, and check whether you can access the project.
- Make sure that you are the owner of the Log Service project. You can only register data storage resources under the same Alibaba Cloud account.
- Check whether you have entered the correct Log Service endpoint and project name. The endpoint must start with `http` and cannot end with `/`. For example, `http :// cn - hangzhou . log . aliyuncs . com` is correct, but `http :// cn - hangzhou . log . aliyuncs . com /` is incorrect.
- Do not register the same data storage resource. Realtime Compute provides a registration check mechanism that prevents repeated registration of the same data storage resource.

Question: Why is only time-based data sampling supported?

Answer: Log Service is designed for streaming data storage, and APIs provided by it only support time parameters. Therefore, Realtime Compute only supports time-based data sampling. If you want to use the search feature of Log Service, log on to the Log Service console and make sure that you have created and own a Log Service project.

1.3 Configure a data storage whitelist

This topic uses ApsaraDB for RDS as an example to describe how to configure a data storage whitelist.

What is a whitelist

Some data storage services provide the whitelist feature to guarantee security. After you configure a whitelist, only IP addresses in the whitelist are allowed to access your data storage resources. Note that this feature also blocks data writing requests from other Alibaba Cloud products whose IP addresses are not in the whitelist.

Configure a whitelist

The whitelist configuration method may vary depending on different data storage services. You need to configure the whitelist in the console of the corresponding data storage service. For more information, see the help document of the correspond ing data storage service. For example, a new ApsaraDB for RDS instance blocks all external requests by default. You must add a whitelist to allow the specified IP addresses to access the ApsaraDB for RDS instance. When Realtime Compute uses ApsaraDB for RDS tables as dimension tables and result tables, it needs to read data from and write data to the ApsaraDB for RDS instance frequently. You must include the IP addresses of WebConsole and Worker of Realtime Compute to the whitelist to allow Realtime Compute to access your ApsaraDB for RDS instance.

Log on to the ApsaraDB for RDS console, and add the IP address of your Realtime Compute cluster to the whitelist, refer to Configure a whitelist. For more information about the IP address, see the following section of this topic.

IP address

If you are using the exclusive mode, you only need to add the elastic network interface (ENI) IP address of your exclusive ECS cluster to the whitelist. To view the ENI IP address of your Realtime Compute cluster, choose Project Management > Clusters > Cluster Name > ENI.

2 Data development

2.1 Development

This topic describes SQL code assistance, SQL version management, engine version change, and data storage management during the development phase of Realtime Compute.

Users of Realtime Compute mainly use Flink SQL for data development. For more information, see Flink SQL. In the data development phase, Realtime Compute provides a complete set of integrated development environment (IDE) tools, and offers the following functions to assist you with your business development.

SQL code assistance

- Flink SQL syntax check

  Realtime Compute supports AutoSave after you modify any text in the IDE. The saving operation triggers SQL syntax check. If a syntax error is detected, the IDE interface displays the row and column where the error is located, and the description of the error.
- Flink SQL intelligent code completion

  When you enter SQL statements on the Development page of Realtime Compute, the IDE provides the auto-suggestions of keywords, built-in functions, tables, and fields.
- Flink SQL syntax highlighting

  Flink SQL keywords are highlighted in different colors to differentiate data structures.

SQL code version management

On this platform, you can manage SQL code versions. A code version is generated each time a user publishes an SQL file for a job. The code management feature allows you to track code changes and roll back to an earlier version if necessary.

- Version management

  A snapshot of a code version is created after you submit an SQL file for publishing a job every time. This allows you to track code changes. Choose Development

> Versions. You can view all version information of this job. You can use the comparison feature to view the differences between the new code and the code of the specified version. You can also use the rollback feature to roll back to a specific version.

- Version clearance

  Realtime Compute has a limit on the maximum number of versions. The default limit of the maximum number of versions supported by Alibaba Cloud is 20. For more information about the limits in other environments, contact your Realtime Compute administrator. If the maximum number of job versions is exceeded, the system does not allow you to submit new versions of the job, and suggests you delete some earlier versions. You can delete some versions from Versions on the right sidebar of the Development Platform.

  Note:
  You can submit the job again after the number of job versions is less than or equal to the limit.

Engine version change

Realtime Compute supports multiple engine versions, and you can choose the right version that suits your business needs. To change the engine version, follow these steps:

- Upgrade
  1. In the lower-right corner of the page, click Recommended Version.
  2. Click OK.
- Downgrade
  1. In the lower-right corner of the page, click Downgrade.
  2. Click OK.

  Note:
  You can only change the engine version when your Realtime Compute job is terminated.

Data storage management

The Realtime Compute development platform provides a complete set of convenient tools for data storage management. You can register your data storage resources on the platform to use these tools.

- Data preview

  The Realtime Compute development platform allows you to preview the data from multiple data storage systems. Data preview helps you efficiently analyze the input and output data, identify key business logic, and complete development tasks.
- Auto DDL generation

  In most cases, the DDL statements for data storage systems are manually translated into the DDL statements for stream processing. Therefore, the DDL generation process includes a large number of repetitive tasks. Realtime Compute provides an auto DDL generation feature. This feature simplifies the way that you edit SQL statements for stream processing jobs, and reduces the errors when you manually enter SQL statements.

2.2 Publish

This topic describes how to publish a Realtime Compute job.

After you complete the development and debugging process, and verify that the Flink SQL code is correct, you can click Publish to publish the data to the production system.

Procedure

1. Configure resources.

   Select Automatic CU Configuration. If it is the first time that you publish a job, use the default number of CUs.
2. Check the data.

   After you verify that no error exists, click Next.
3. Publish a job.

   Click Publish.

4. Start a job.

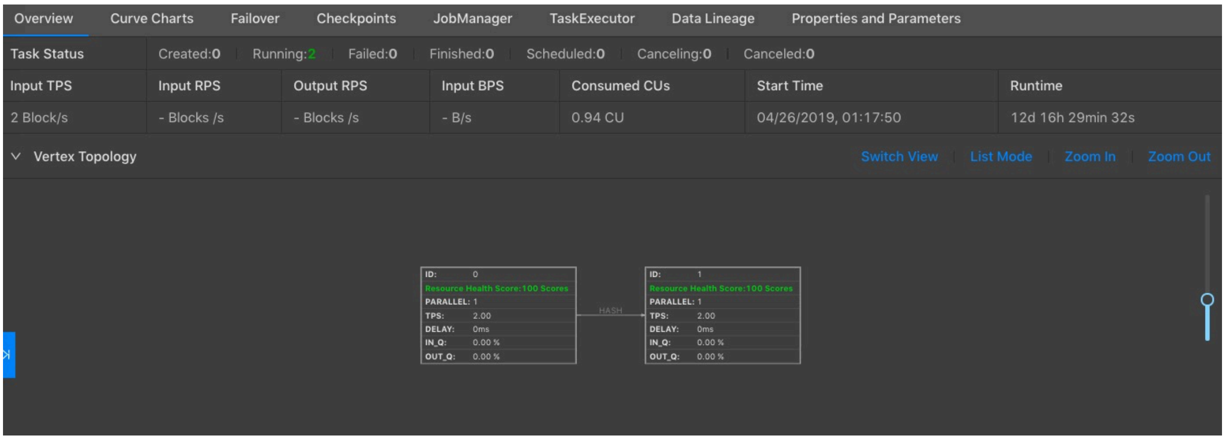
   Enter the Administration page, and click the job.

   > **Note:**
   > Realtime Compute completely isolates the development and production environments. All operations performed on the Realtime Compute development platform, such as modification and debugging, do not affect the production and debugging jobs on the data administration page.

3 Data administration

3.1 Overview

The Overview page shows the real-time running status and instantaneous values of a job. Based on the analysis of the job status, you can determine whether the job is healthy, and whether it meets your expectations.



Health score

To help you quickly locate job performance issues, Realtime Compute offers a health check feature.

A health score of less than 60 points indicates that the current node has stacked up some data, and its data processing capacity is insufficient. You can solve this issue by using either of the two methods:Automatic configuration optimization and Manual configuration optimization. You can optimize the performance based on your business requirements.

Task status

A task can be in any of the following statuses: created, running, failed, completed , scheduled, canceling, and canceled. You can determine whether a job is running properly based on the task status.

Job instantaneous values

Name	Description
Computing duration	Indicates the computing performance of the job.

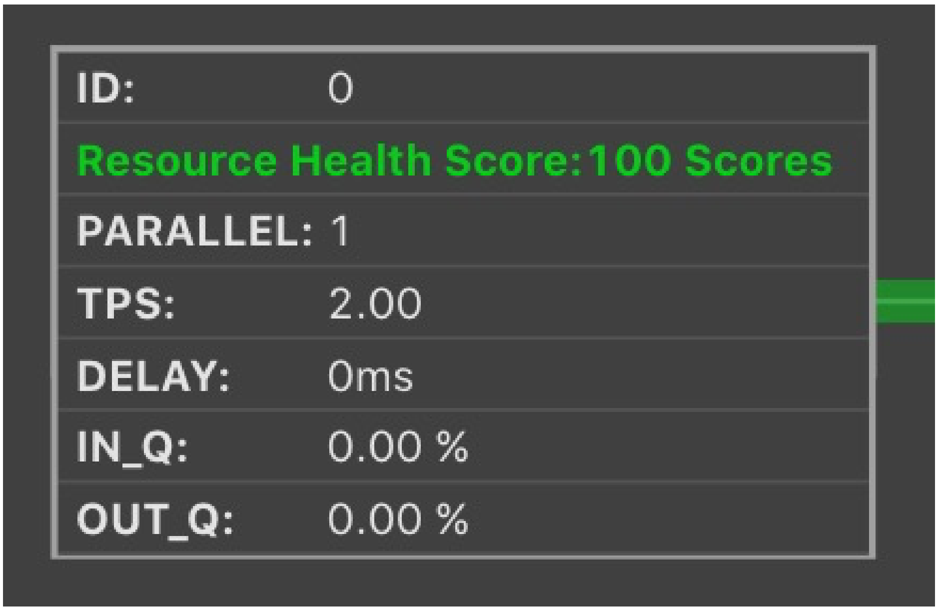
Name	Description
Input TPS	The number of blocks read from the source table every second. If you use Log Service, it can include multiple data records into a LogGroup for Realtime Compute to read. The number of blocks reflects the number of LogGroups that are read by Realtime Compute from the source table every second.
Input RPS	The number of data records that are read from the source table every second. The unit is record/s.
Output RPS	The number of data records that are written into result tables every second. The unit is record/s.
CPU usage	The current CPU usage of the job.
Start time	The start time of the job.
Running duration	The duration that the job has been running since it was started.

Running topology

A running topology shows how the underlying computational logic of Realtime Compute works. Each component corresponds to a task. Each data stream starts from one or more data sources and ends in one or more result tables. The flow of data streams is similar to a Directed Acyclic Graph (DAG). For more efficient distributed execution, Realtime Compute chains operator subtasks together into tasks if possible. Every task is run in a thread. Combining operators into a task reduces inter-thread switching, serialized or deserialized messages, and data exchange in the cache, shortening latency and increasing overall throughput. An operator indicates a computational logic operator, and a task is a collection of multiple operators.

- View mode

  To help you better understand the abstract underlying computational logic of Realtime Compute, the Realtime Compute platform offers the following view.



The detailed information about a task is as follows. When you move the pointer over a task, the detailed information appears.

Name	Description
ID	The task ID in the running topology.
PARALLEL	The number of parallel subtasks.
TPS	The amount of data read from the input tables, which are measured in blocks per second.
LATENCY	The computing duration of the task node.
DELAY	The processing delay at the task node.
IN_Q	The percentage of input queues for the task node.
OUT_Q	The percentage of output queues for the task node.

You can click a task node to enter its details page, and view its Subtask List.

Realtime Compute also provides Curve Charts for all metrics of each task. Click a task node to view the Curve Charts.

- List mode

  Realtime Compute also allows you to view each task in the list mode.

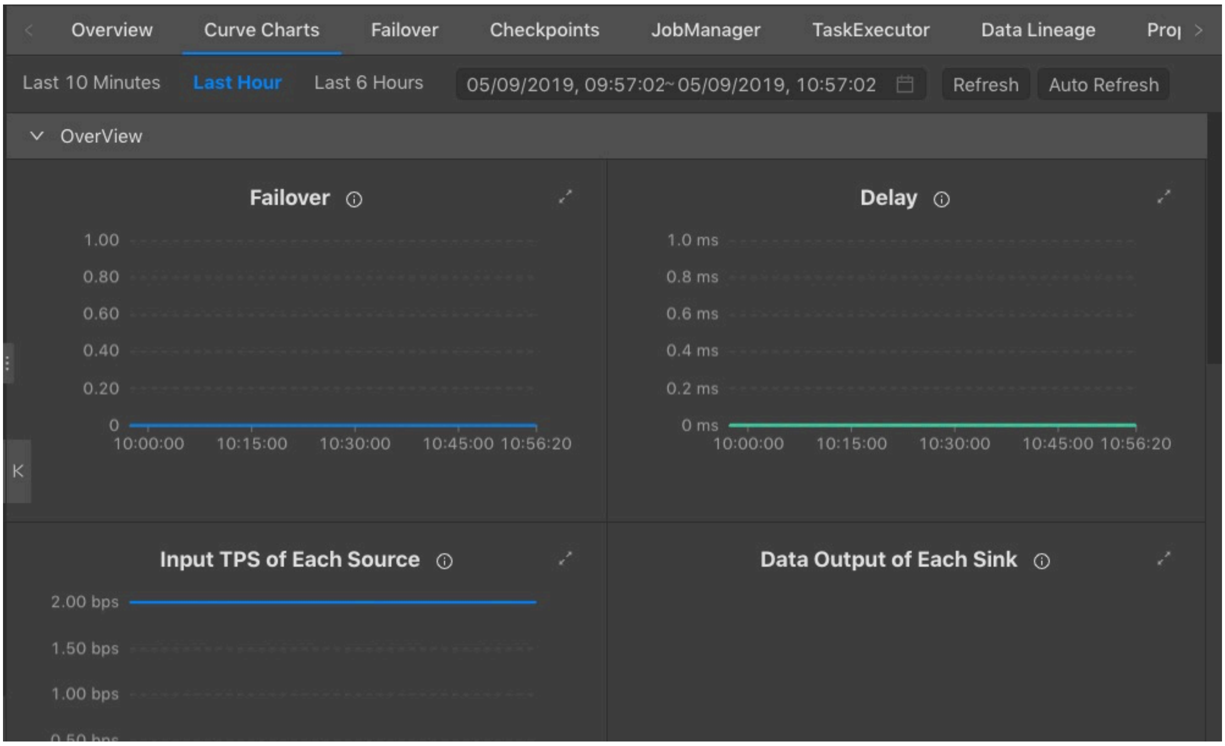
  You can click a task node to enter its details page, and view its Subtask List.

Name	Description
ID	The task ID in the running topology.
Name	The name of the task.
Status	The status of the task.
InQ max	The maximum percentage of input queues for the task node.
OutQ max	The maximum percentage of output queues for the task node.
RecvCnt sum	The total amount of data that is received by the task node.
SendCnt sum	The total amount of data that is sent from the task node.
TPS sum	The total amount of data that is read from the input source every second.
Delay max	The maximum processing delay at the task node.
StartTime	The start time of the task node.
Durations(s)	The running duration of the task node, in seconds.
Task	The running status of the parallel subtasks under the task node.

3.2 Curve Charts

Realtime Compute provides an overview page of core metrics of the current job. You can diagnose the running status of the current job with one click based on curve charts. In the future, Realtime Compute will provide more deep analysis algorithms based on the current job status to assist you with smart and automatic diagnostics of errors.

The following figure shows the curve charts of the job diagnostic metrics.



Note:

The metrics shown in this figure are displayed only when the Realtime Compute job is in the running state. Realtime Compute asynchronously collects job metrics in the background, which results in delays. The metrics can be displayed properly only after a job has been running for more than 1 minute.

Overview

- Failover rate

  The failover rate refers to the failover (caused by errors or exceptions) frequency of the current job. Calculation method: Divide the accumulated failover count in the last minute by 60. For example, if one failover occurred in the last minute, the failover rate would be 1/60 = 0.01667.

  The trend of the failover rate allows you to better analyze problems of the job.

- Delay
  - fetched_delay: Data pending time (fetched_delay) = Time when data enters Flink - Data event time (event time). This metric indicates the actual processing capability of Realtime Compute.
  - no_data_delay: Data arrival interval (no_data_delay) = Current system time - Time when the last data record arrives at Flink. This metric indicates the time required by a data record to flow from the data source to Flink.
  - delay: Processing delay (delay) = Current system time - Current data event time (event time). This metric indicates the data processing progress.
- Input TPS of Each Source

  Realtime Compute collects statistics about all streaming data input of a Realtime Compute job, and records the number of data blocks read from the source table every second. This allows you to intuitively view the transaction per second (TPS) information of the data storage system. Unlike TPS, records per second (RPS) indicates the number of data records that are parsed from data blocks of the source table. The unit is record/s. For example, Log Service reads N LogGroups and parses M log records every second. The number of log records parsed per second is the RPS of the data input system.
- Data Output of Each Sink

  Realtime Compute collects statistics about all data output (instead of only the streaming data output) of a Realtime Compute job. This allows you to intuitively view the RPS information of the data storage system. Typically, if you cannot detect data output during system O&M, check both the data input and output systems.
- Input RPS of Each Source

  Realtime Compute collects statistics about the streaming data input of a Realtime Compute job. This allows you to intuitively view the RPS information of the data storage system. Typically, if you cannot detect data output during system maintenance, check whether there is data input from the data source.
- Input BPS of Each Source

  Realtime Compute collects statistics about the streaming data input of a Realtime Compute job, and records the traffic read from the source table every second. This allows you to intuitively view the byte per second (BPS) information of the data traffic.

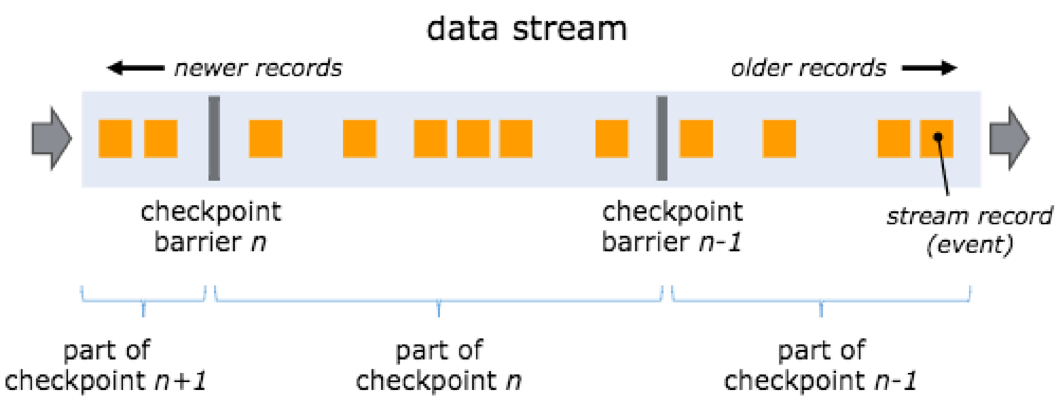
- Dirty data from Each Source

  This metric indicates whether dirty data exists in the source section of Realtime Compute.

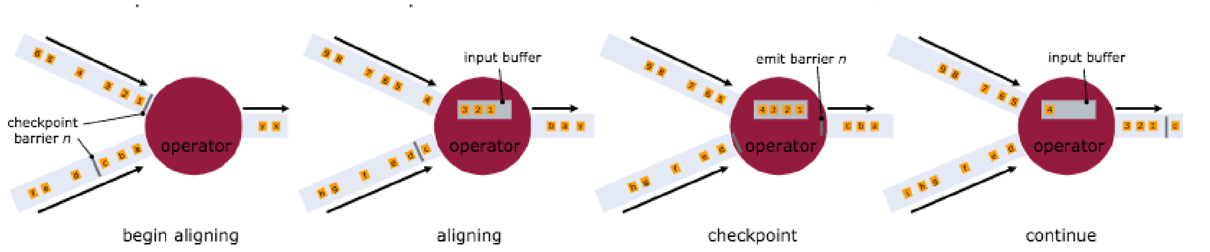
Advanced view

Realtime Compute provides a fault tolerance mechanism that allows you to restore data streams. This mechanism ensures that the data streams are consistent with the application. The core of this fault tolerance mechanism is to continuously create consistent snapshots for distributed data streams and their statuses. These snapshots act as consistency checkpoints for rollback in case of system failure.

One of the core concepts of distributed snapshots is the barrier. Barriers are inserted into data streams and flow together with the data streams to the output system. Barriers will not interfere with normal data. Instead, data streams are strictly ordered. A barrier cuts a data stream into two parts, one entering the current snapshot and the other the next snapshot. Each barrier has a snapshot ID. Data that flows before a barrier is included in the snapshot corresponding to this barrier. Barriers are lightweight, and do not interfere with the processing of data streams. Multiple barriers of different snapshots can simultaneously exist in the same data stream. This means that multiple snapshots may be created concurrently.

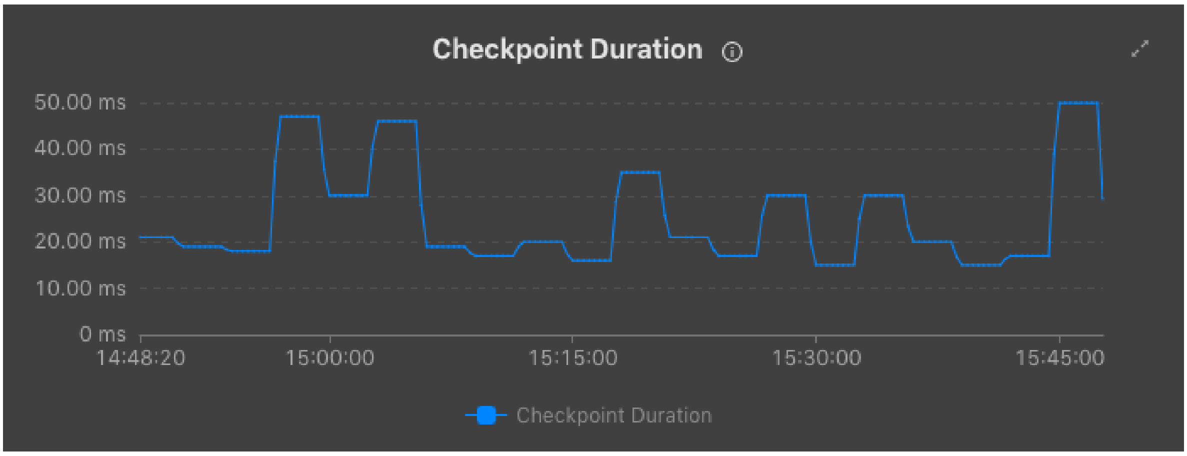


Barriers are inserted at the data source. After barriers of snapshot n are inserted into all input data streams, the system records the current snapshot as Sn (n indicates the snapshot position). Then, the barriers keep flowing down. When operator A receives all barriers marked with snapshot n (Sn barriers) from its input streams, it inserts an Sn barrier into each of its output streams. The flow of data streams is similar to a Directed Acyclic Graph (DAG). Therefore, these streams are also known as DAG streams. When a sink operator (operator B), the destination of a DAG stream, receives all Sn barriers from its input streams, it reports to the checkpoint coordinator that the Sn snapshot is created. After all sink operators have reported that the Sn snapshot is created, this snapshot is marked created.



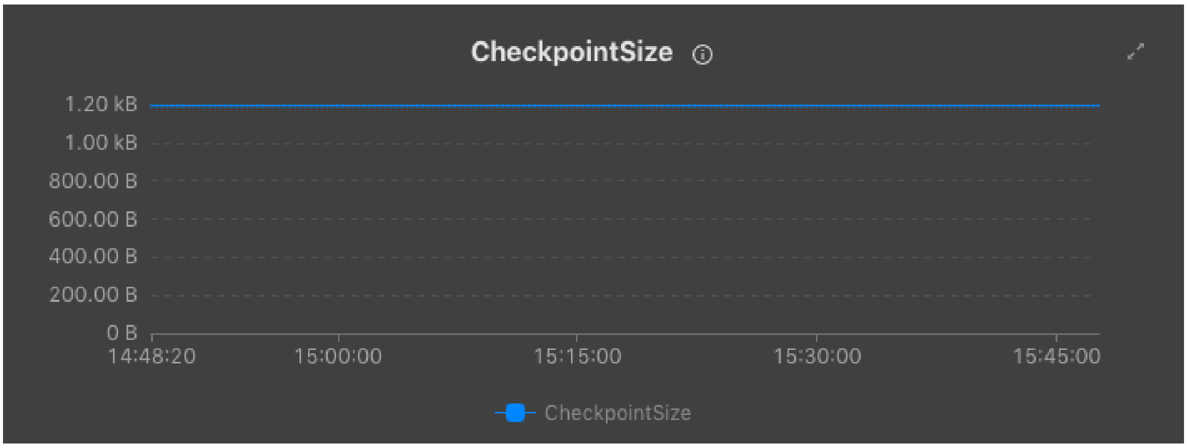
The curve charts of various checkpoint metrics are as follows.

- Checkpoint Duration



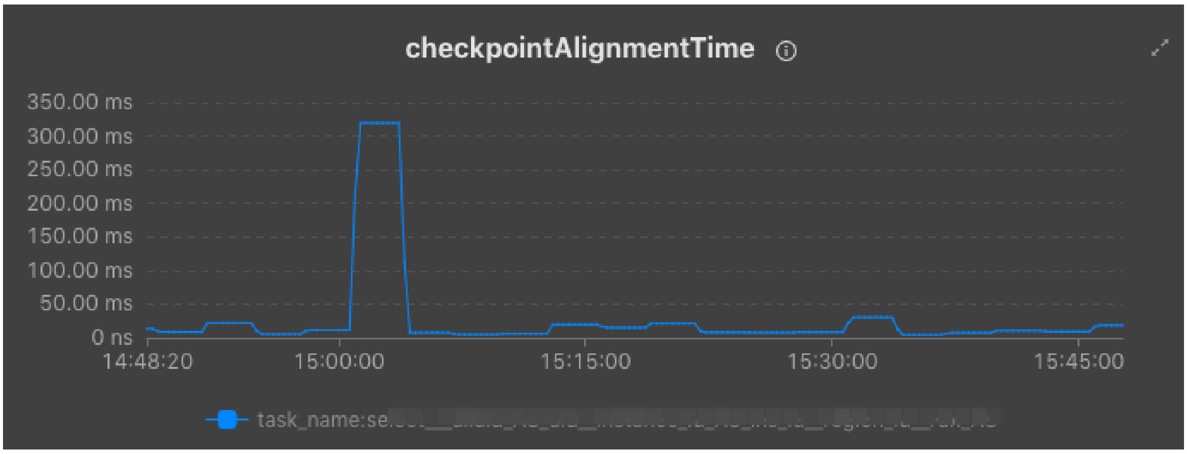
This metric indicates the duration for creating a checkpoint, in milliseconds.

- CheckpointSize



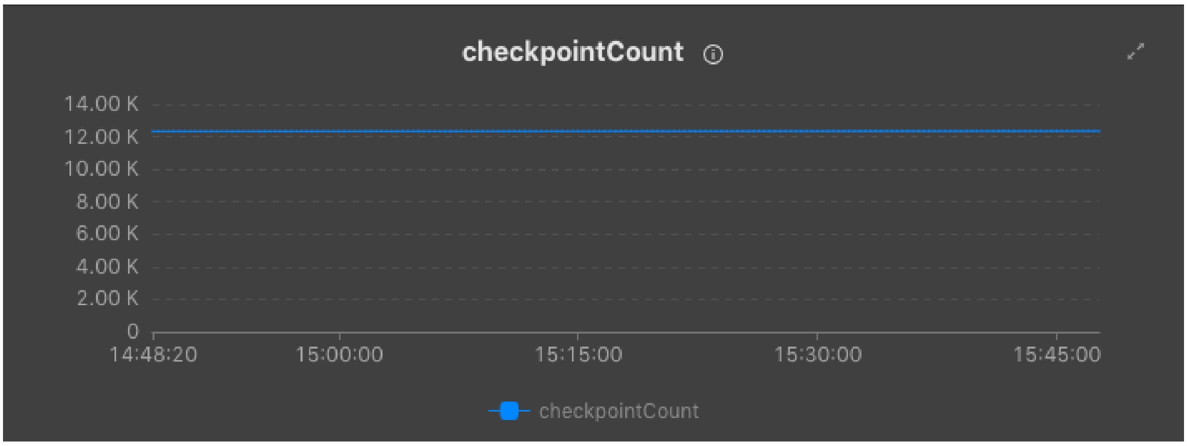
This metric indicates the size of each checkpoint.

- checkpointAlignmentTime

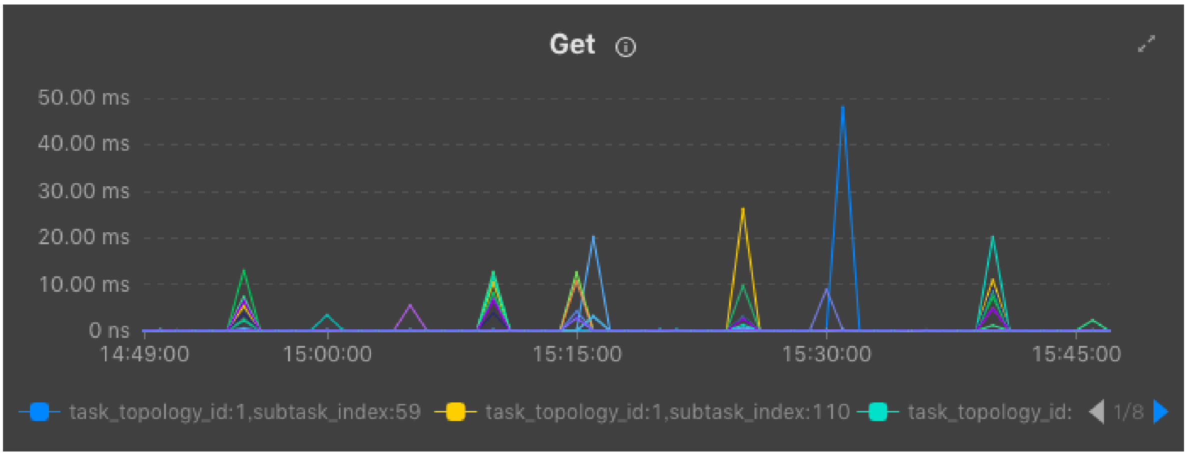


This metric indicates the duration required for all data streams to flow from the upstream nodes to the node at which you create a checkpoint. In other words, when a sink operator (destination of a DAG stream) receives all Sn barriers from its input streams, it reports to the checkpoint coordinator that the Sn snapshot is created. After all sink operators have reported that the Sn snapshot is created, this snapshot is marked created. This duration is known as the checkpoint alignment time.

- checkpointCount

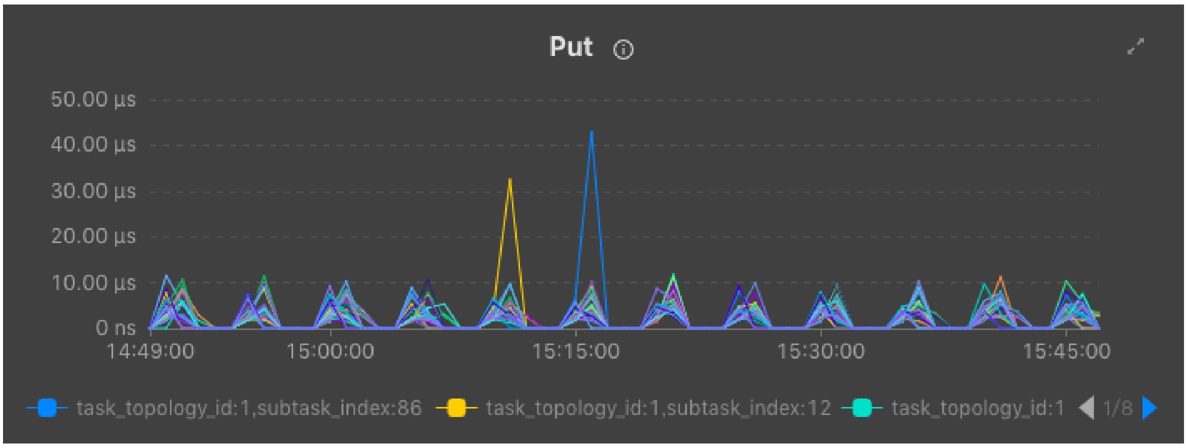


- Get



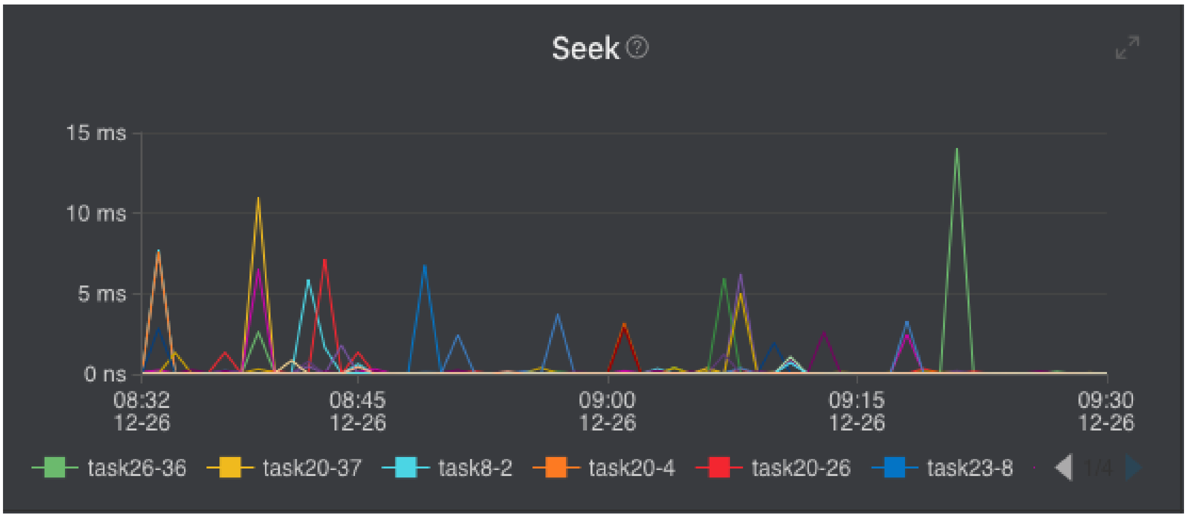
This metric indicates the longest time that a subtask spends on performing a GET operation on the RocksDB within a specific period.

- Put



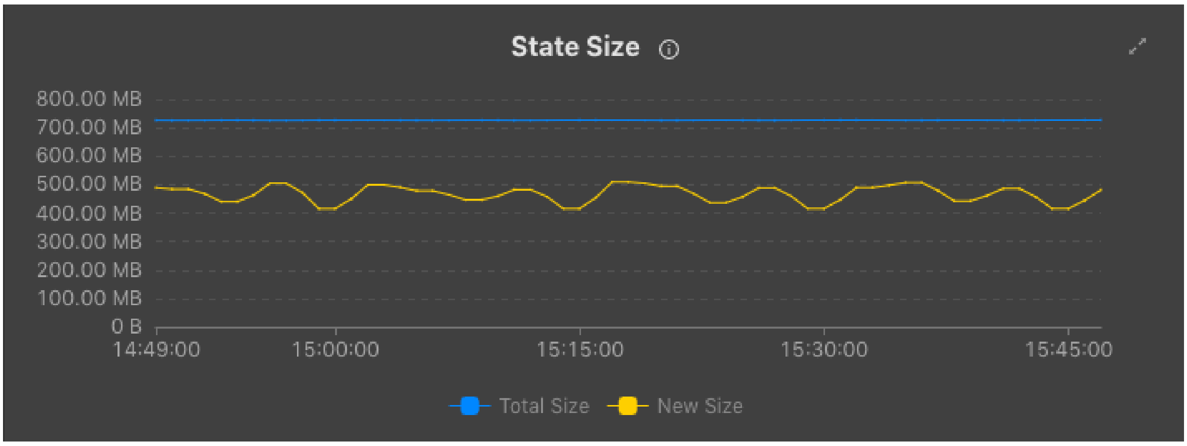
This metric indicates the longest time that a subtask spends on performing a PUT operation on the RocksDB within a specific period.

- Seek



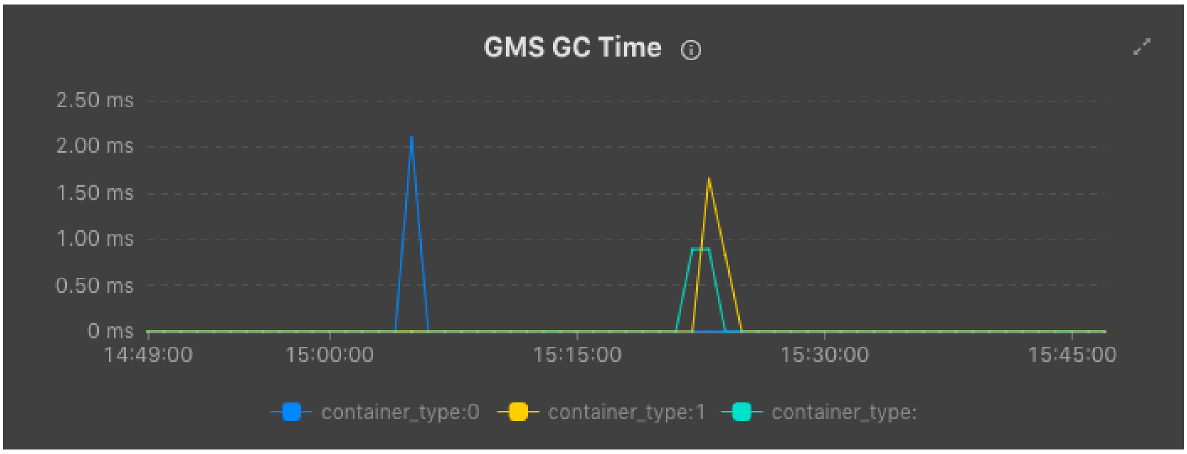
This metric indicates the longest time that a subtask spends on performing a SEEK operation on the RocksDB within a specific period.

- State Size



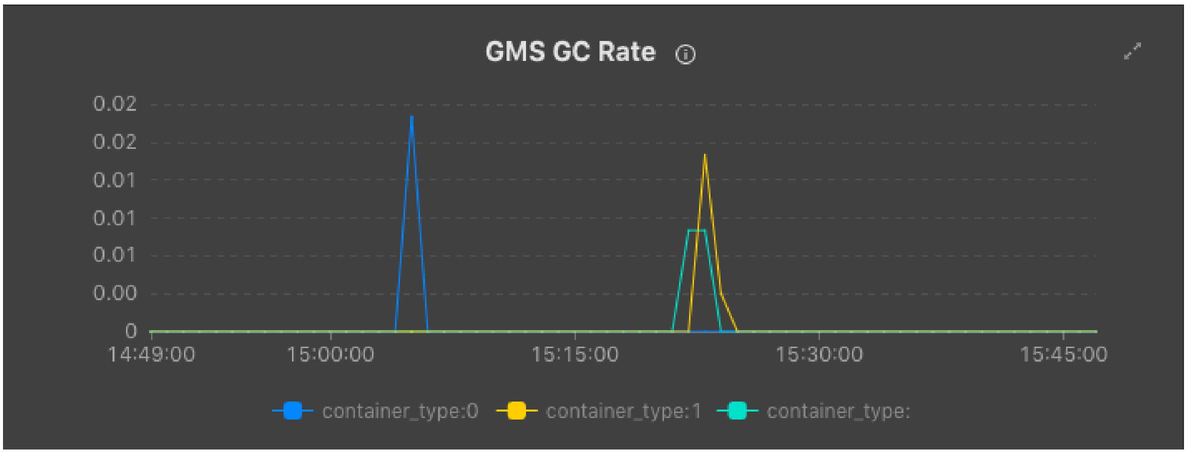
This metric indicates the state size of a job. If the size increases too fast, the job is abnormal.

- CMS GC Time



This metric indicates the time that the underlying container of a job spends on garbage collections (GC).

- CMS GC Rate



This metric indicates the frequency that the underlying container of a job performs GC.

WaterMark

- WaterMark Delay

  This metric indicates the difference between the watermark time and the system time.

- Dropped Records per Second

  When the time at which a data record reaches the window is later than the watermark time, the data record will be discarded. This metric indicates the number of late data records discarded per second.

- Dropped Records

  When the time at which a data record reaches the window is later than the watermark time, the data record will be discarded. This metric indicates the accumulated number of discarded late data records at a specific time point.

Delay

Top 15 Source Subtasks with the Longest Processing Delay

This metric indicates the processing delay of each source subtask.

Throughput

- Task Input TPS

  This metric indicates the data input of all tasks under the same Realtime Compute job.
- Task Output TPS

  This metric indicates the data output of all tasks under the same Realtime Compute job.

Queue

- Input Queue Usage

  This metric indicates the data input queue of all tasks under the same Realtime Compute job.
- Output Queue Usage

  This metric indicates the data output queue of all tasks under the same Realtime Compute job.

Tracing

Advanced metrics are as follows:

- Time Used In Processing Per Second

  This metric indicates the time that a task spends on processing data per second.
- Time Used In Waiting Output Per Second

  This metric indicates the time that a task spends on waiting for the output per second.
- TaskLatency Histogram Mean

  This metric indicates the computing latency curve of each task under the same job.

- WaitOutput Histogram Mean

  This metric indicates the curve of the time that a task spends on waiting for the output.
- WaitInput Histogram Mean

  This metric indicates the curve of the time that a task spends on waiting for the input.
- PartitionLatency Mean

  This metric indicates the latency curve of each parallel subtask in a partition.

Process

- Process Memory RSS

  This metric indicates the memory usage curve of each process.
- CPU Usage

  This metric indicates the CPU usage curve of each process.

JVM

- Memory Heap Used

  This metric indicates the Java Virtual Machine (JVM) heap memory usage of the job .
- Memory Non-Heap Used

  This metric indicates the JVM non-heap memory usage of the job.
- Threads Count

  This metric indicates the number of threads for the job.
- GC (CMS)

  This metric indicates the number of garbage collections (GC) that have been performed for the job.

3.3 Checkpoint

Realtime Compute provides a fault tolerance mechanism that allows you to restore data streams. This mechanism ensures that the data streams are consistent with the application. The core of this fault tolerance mechanism is to continuously create consistent snapshots for distributed data streams and their statuses. These snapshots act as consistency checkpoints for rollback in case of system failure.

Completed Checkpoints

On this tab, you can view checkpoints that have been completed. The following table describes the parameters on this tab.

Name	Description
ID	The ID of the checkpoint.
StartTime	The time when the checkpoint is created.
Durations(ms)	The time that is spent on creating the checkpoint.

Task Latest Completed Checkpoint

On this tab, you can view the detailed information about the latest checkpoint. The following table describes the parameters on this tab.

Name	Description
SubTask ID	The ID of the subtask.
State Size	The size of the checkpoint.
Durations(ms)	The time that is spent on creating the checkpoint.

3.4 JobManager

This topic describes the usage of JobManager and its role in the startup process of a Realtime Compute cluster.

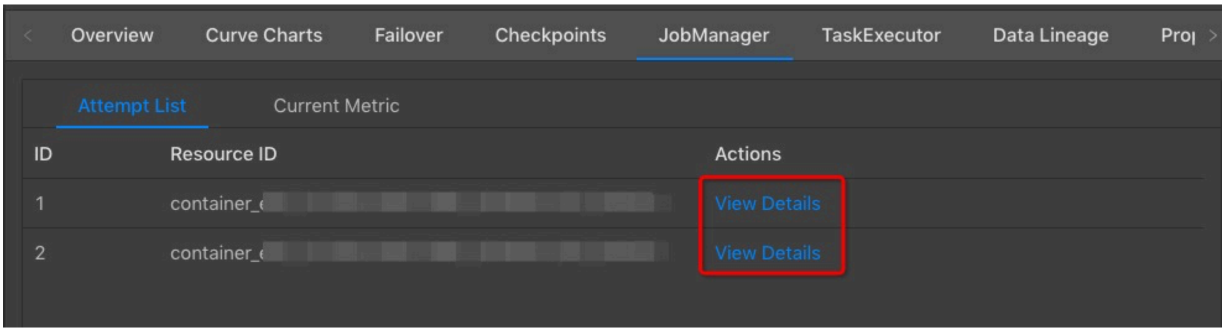
JobManager is essential to the startup process of a Realtime Compute cluster. The startup process of a Realtime Compute cluster is described as follows:

1. The Realtime Compute cluster starts one JobManager and one or more TaskExecutors.
2. A client submits jobs to the JobManager.
3. The JobManager assigns tasks of the jobs to TaskExecutors.
4. TaskExecutors report the heartbeats and statistics to the JobManager.

Usage of JobManager

Similar to Storm Nimbus, a JobManager receives jobs, and arranges for TaskExecutors to create checkpoints. The JobManager receives jobs and resources, such as JAR packages, from a client. Then, the JobManager generates an optimized execution plan, and assigns tasks to TaskExecutors.

JobManager parameters



3.5 TaskExecutor

This topic describes the role of TaskExecutor in the startup process of a Realtime Compute cluster, as well as its user interface.

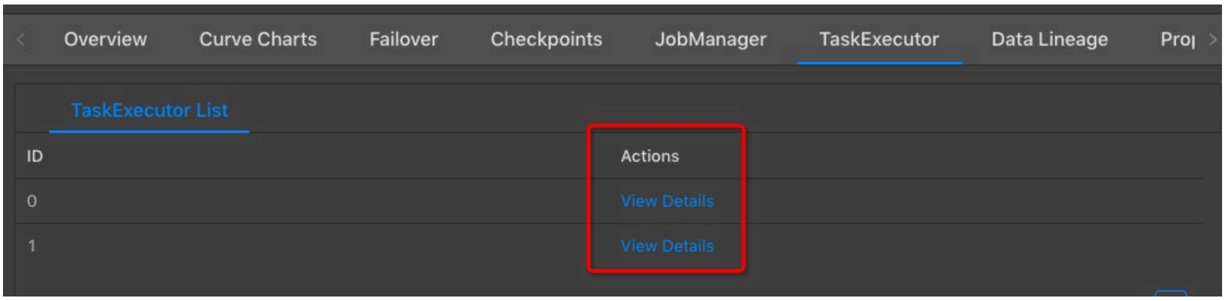
Background

After a Realtime Compute cluster is started, one JobManger and one or more TaskExecutors are started. A client submits jobs to the JobManager, and the JobManager assigns tasks of the jobs to TaskExecutors. During task execution, TaskExecutors report the heartbeats and statistics to the JobManager. TaskExecutors transmit data to one another through data streams.

The number of slots is specified before a TaskExecutor is started. A TaskExecut or executes each task in each slot, and each task can be considered as a thread. A TaskExecutor receives tasks from the JobManager, and then establishes a Netty connection with its upstream to receive and process data.

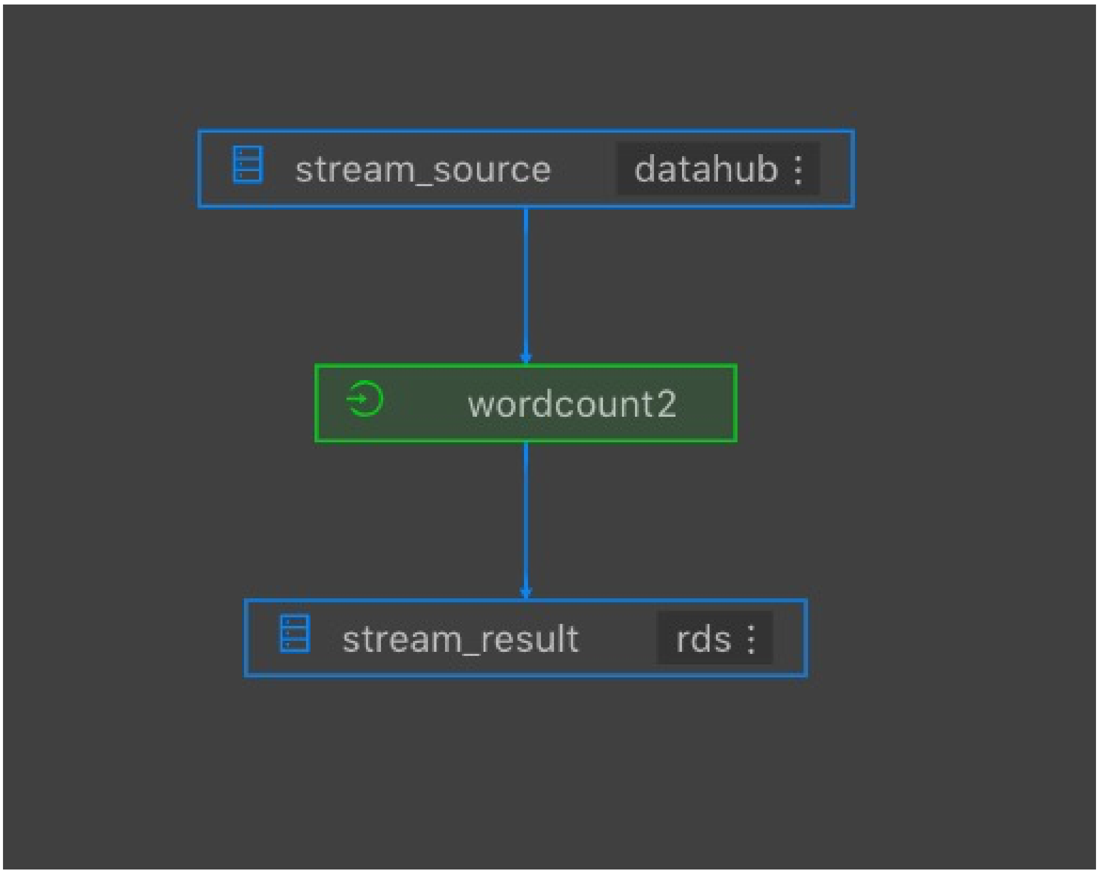
User interface of TaskExecutor

The TaskExecutor page provides you with a list of tasks and entries to their details.



3.6 Data Lineage

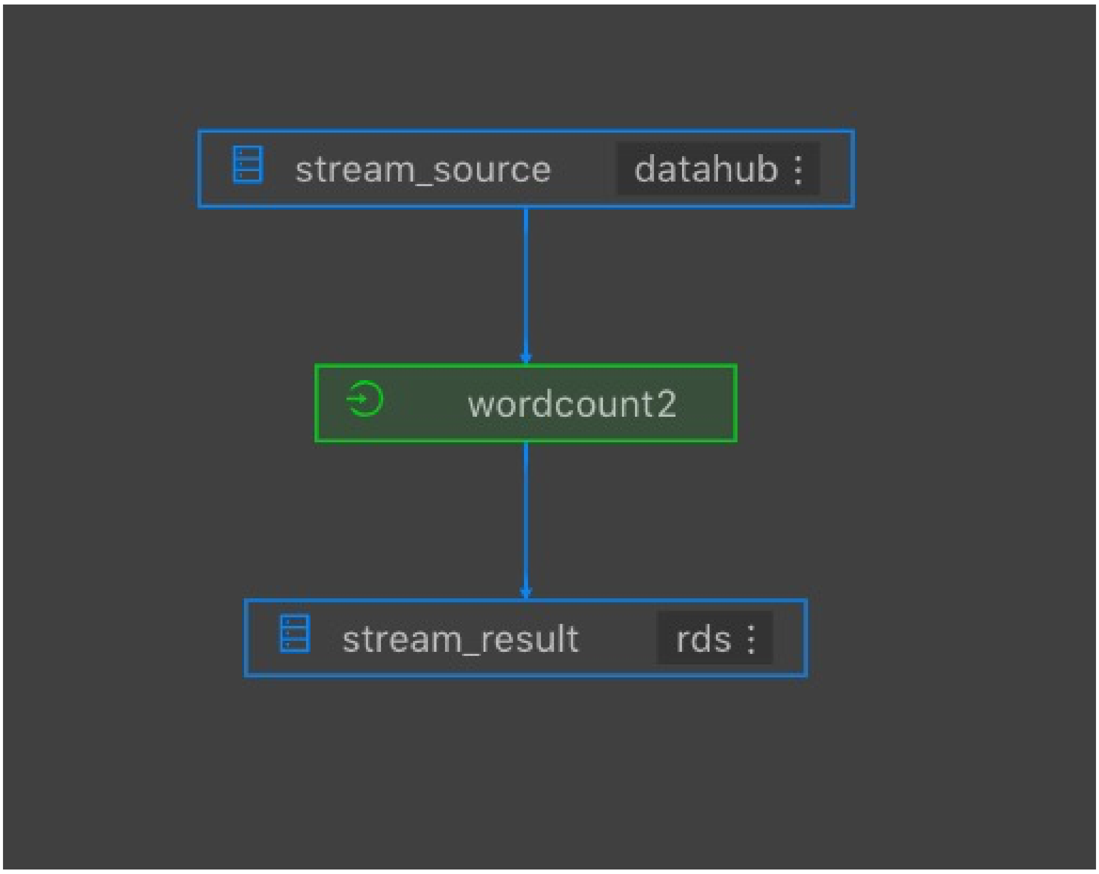
The data lineage of a Realtime Compute job reflects the dependency between the input and output data of the job. In scenarios where the business dependency between the input and output data of a job is complex, Realtime Compute provides a data topology on the Data Lineage page to clearly show the dependency.



Data Sampling

The Data Lineage page provides the data sampling feature for the source and result tables of jobs. This feature is consistent with that on the development platform. It allows you to detect data and locate problems on the data administration page at any time.

To enable the data sampling feature, click the source or result table name.



3.7 Properties and Parameters

The Properties and Parameters page provides detailed information about the current job, including the current running information and running history.

Code

On this tab, you can preview the entire SQL job. You can also click Edit Job to redirect to the development platform.

Resource Configuration

On this tab, you can configure the resources for the current job, including the CPU, memory, and parallelism.

Properties

On this tab, you can view the basic running information of all jobs.

No.	Description
1	Job Name: the name of the job.

No.	Description
2	Job ID: the ID of the job.
3	Referenced Resource: the resources that are referenced by the job.
4	Engine: the engine of the job.
5	Last Operated By: the user who last operates the job.
6	Action: the action that is last performed.
7	Created By: the user who creates the job.
8.	Created At: the time when the job is created.
9	Last Modified By: the user who last modifies the job.
10	Last Modified At: the time when the job is last modified.

Runtime Parameters

On this tab, you can view the underlying checkpoints, start time, and runtime parameters of the job.

History

On this tab, you can view the detailed information about all versions of the job, including the start time, end time, and the user who operates the job.

Parameters

On this tab, you can view other job parameters that can be customized, such as delimiters for debugging.

4 Monitoring and alerting

This topic describes monitoring and alerting in Realtime Compute, and how to create and start alert rules.

Background

Realtime Compute provides data processing capabilities for CloudMonitor, so that you can monitor the health of your job in real time. CloudMonitor collects monitoring metrics of Alibaba Cloud resources and your custom metrics. It can be used to detect the availability of your services and allows you to set alerts for specific metrics. It allows you to be fully aware of resource usage, service status, and service health on Alibaba Cloud. It also enables you to promptly respond to error alerts and ensure smooth running of your application.

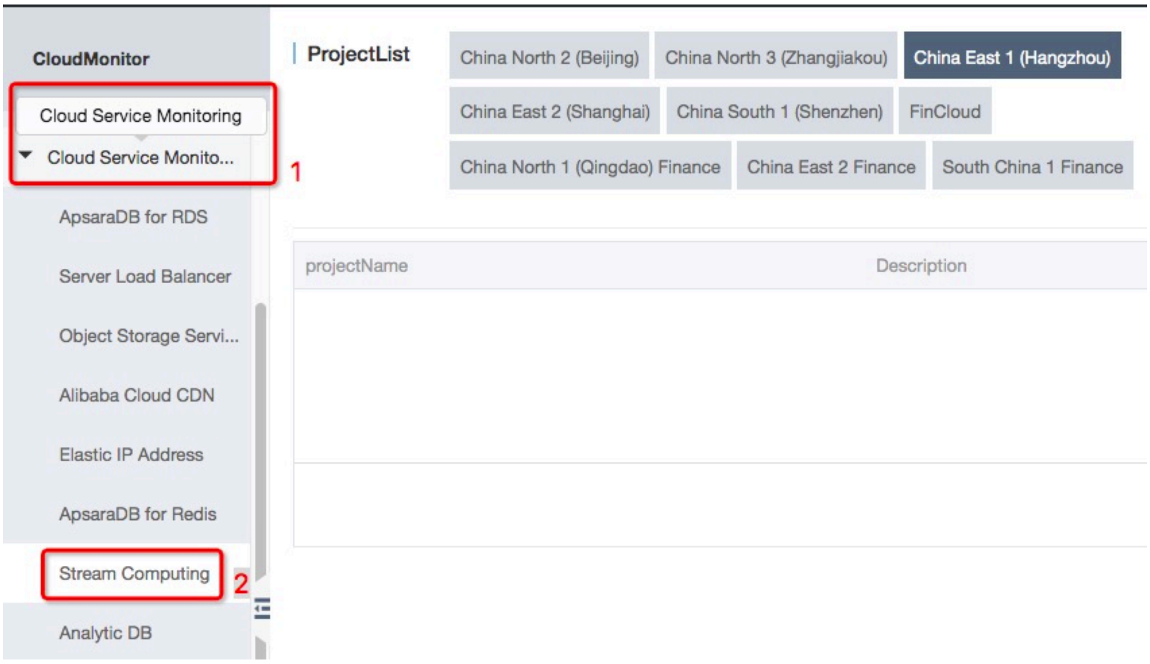
With Realtime Compute, you can specify alerts for the following performance metrics:

- Processing delay
- Input RPS
- Output RPS
- Failover rate
- Data pending time

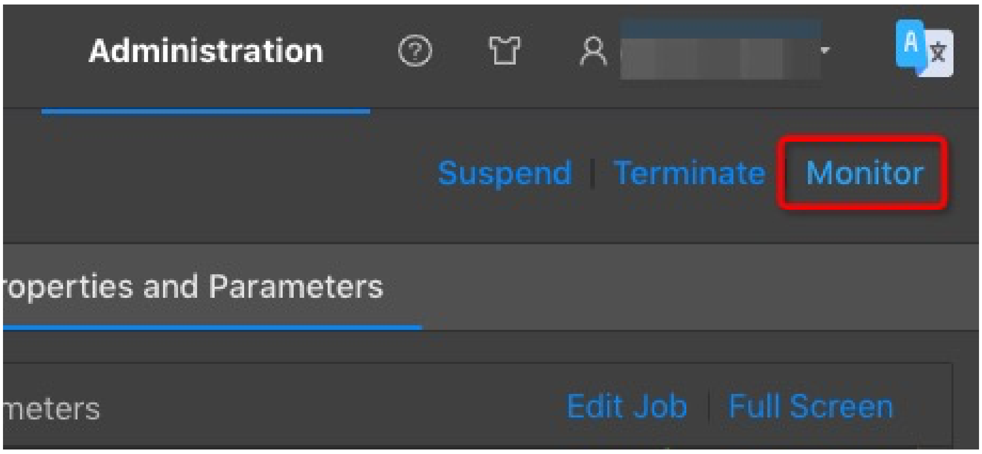
Operations

1. Log on to CloudMonitor.

   - Option 1: Log on to the Alibaba Cloud official website, and go to the CloudMonitor console.



   - Option 2: On the Administration page of Realtime Compute, click Monitor to redirect to CloudMonitor.



2. View Realtime Compute monitoring alerts.

   a. Select a project that you want to monitor, and click ViewJob.

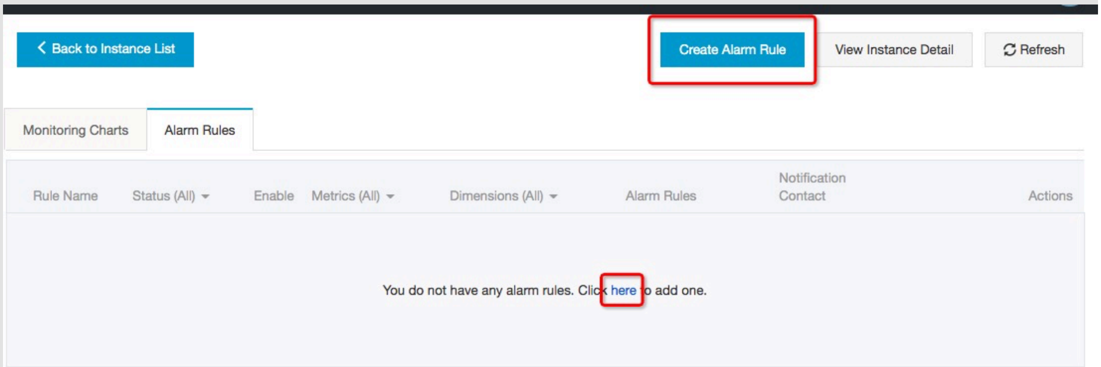
   b. On the page that appears, click Monitoring Charts in the Actions column of a job.

   Note:

If you have not set any alert rules, use either of the following methods to enter the alert rule creation page:

A. Select a job that you want to set alerts for, and click Set Alert Rules.

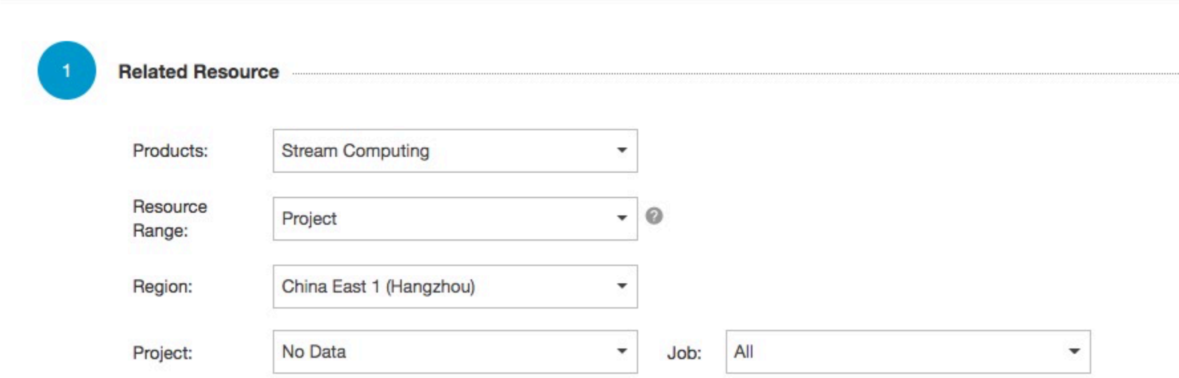
B. On the Alert Rules tab, click here or Create Alert Rule.



Create alert rules

To create alert rules, follow these steps:

1. In the Related Resource section, set Products to Stream Computing (which is the old version of Realtime Compute) and Resource Range to Project, select a project, and select one or more jobs from the Job drop-down list.



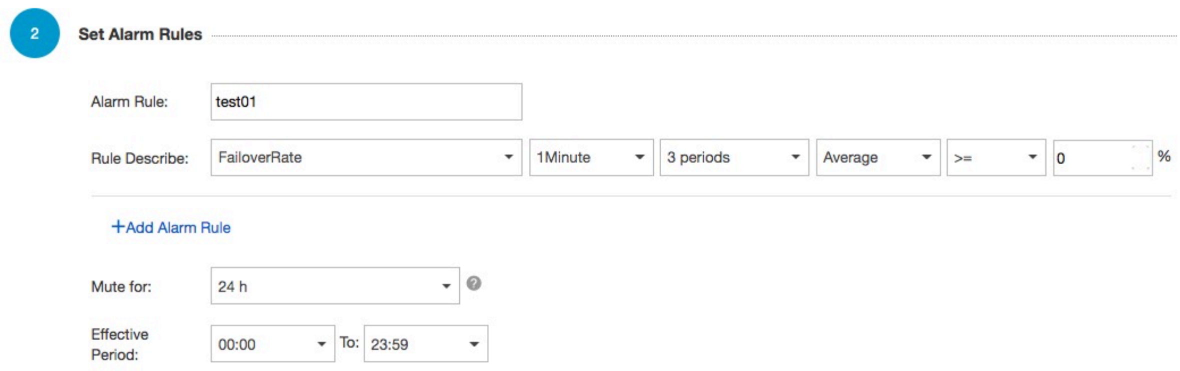
2. Set alert rules.

   The following table lists the supported types of alert rules.

Rule	Unit
Processing delay	Seconds
Input RPS	Entries
Output RPS	Entries

Rule	Unit
Failover rate	Failover times per second Note: · The failover rate indicates the average number of failover times per second in a period of time. Assume that a failover occurred in the last minute. The failover rate in the last minute is 1/60 = 0.01667 = 1.667%. · When you specify the failover rate threshold, enter a percentage value.
Data pending time	Seconds

The following figure shows the recommended parameter settings.



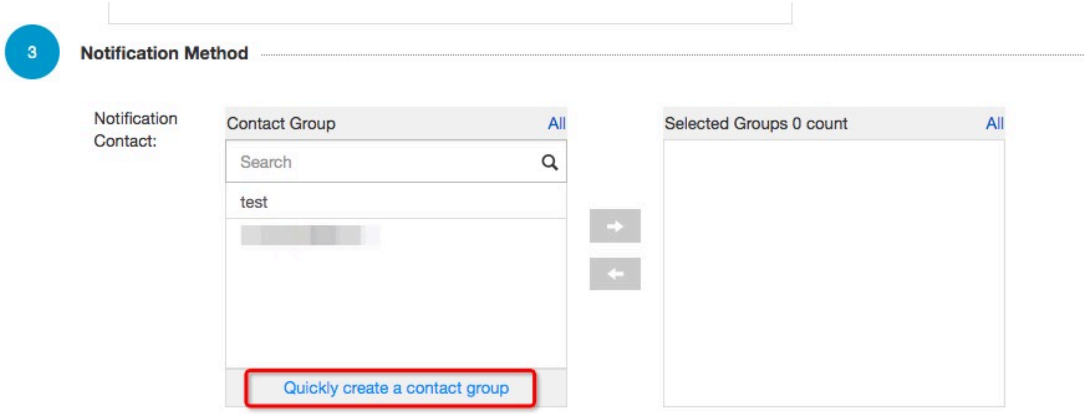
3. Configure notification methods.

- Notification contacts

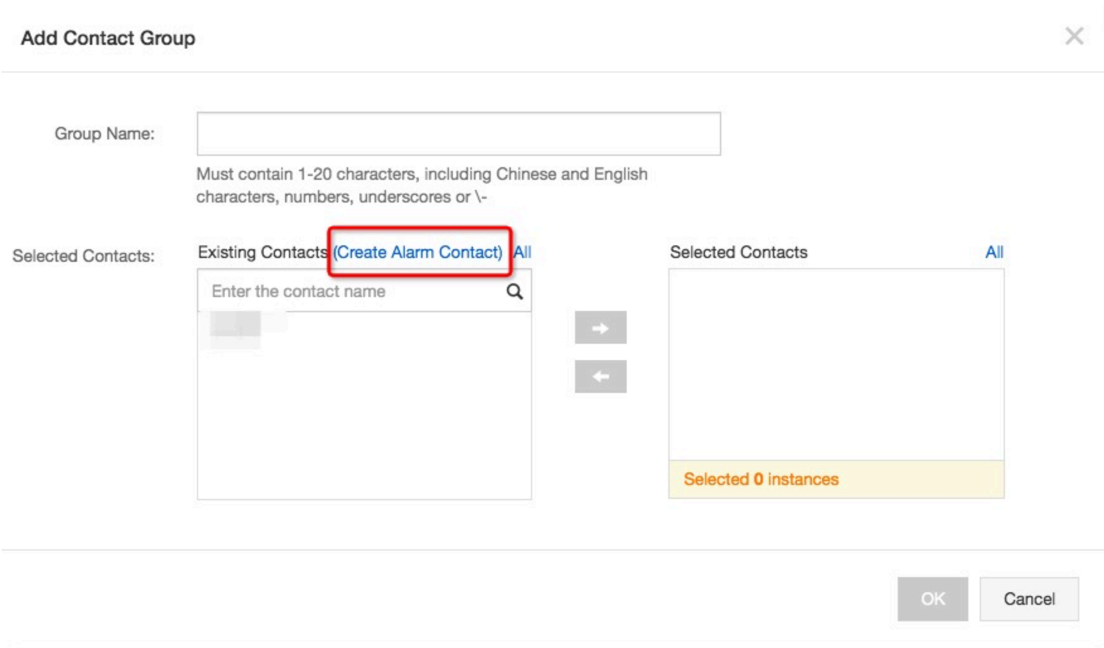
  You can click Quickly create a contact group in the Notification Contact area, or add other people to an existing contact group.

  To add new alert contacts, follow these steps:

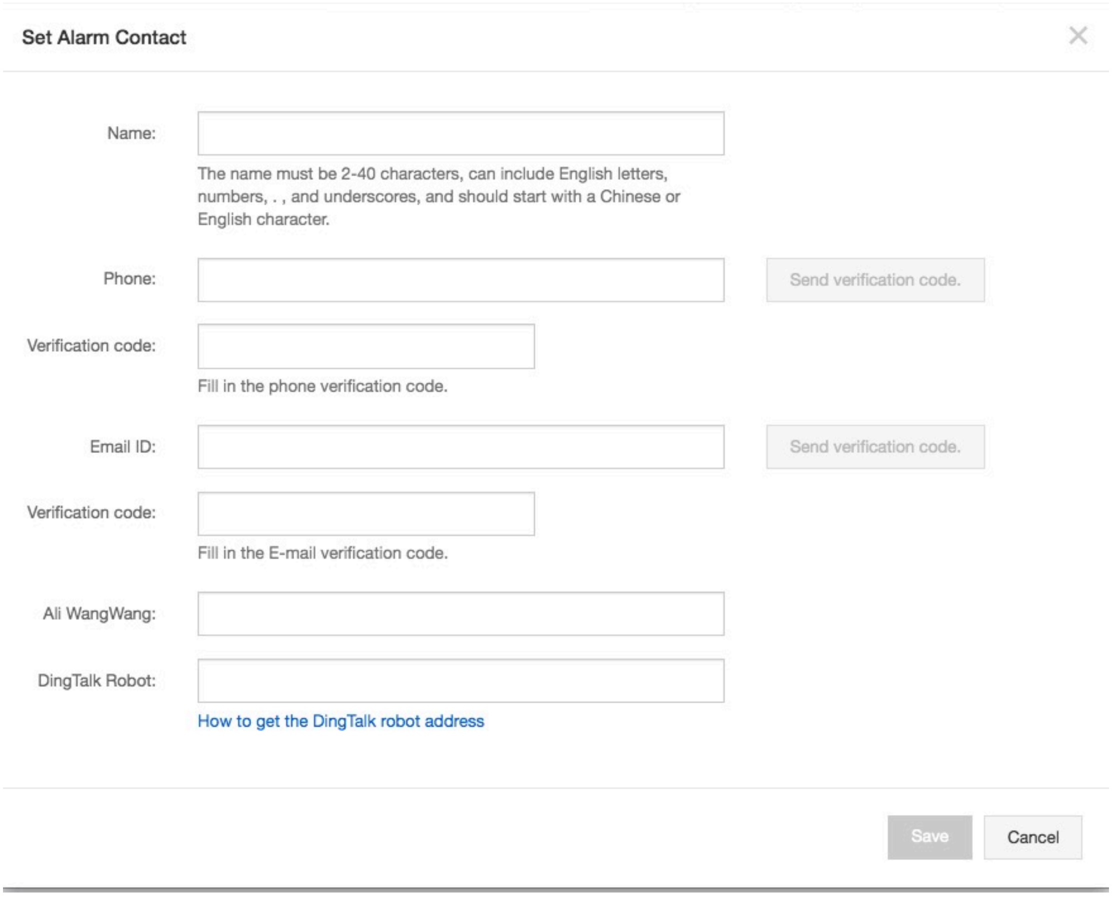
  a. In the Notification Method section, click Quickly create a contact group.



  b. In the Add Contact Group dialog box that appears, click Create Alert Contact.

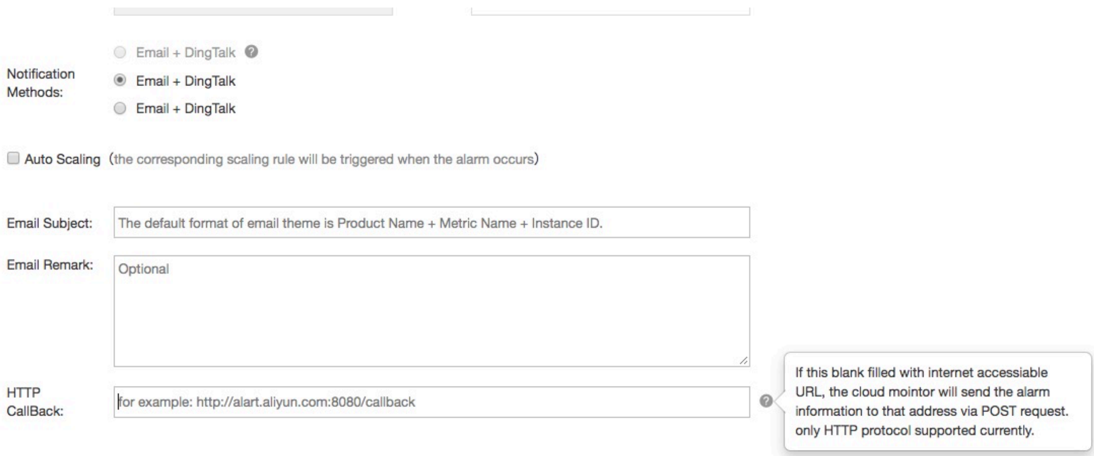


  c. Configure the alert contact information.



- Notification methods
  - Mobile phone + email + TradeManager + DingTalk Chatbot
  - Email + TradeManager + DingTalk Chatbot
  - Alert callback

  You can configure the notification methods based on your actual business requirements.



5 Configuration optimization

5.1 Flink SQL skills for improving job performance

This topic describes the recommended Flink SQL statements, configuration, and functions that are helpful to significantly improve the job performance.

Skills for improving Group Aggregate operations

- Enable microBatch or miniBatch to improve the throughput

  Both microBatch and miniBatch are mini-batch processing methods. The only difference lies in their triggering mechanisms. In principle, they cache a specific amount of data before they trigger the processing. This reduces the frequency that Realtime Compute has to access the state, thus significantly improving the throughput and reducing the amount of data output.

  miniBatch triggers mini-batch processing by using the timer threads that are registered with each task. This involves some thread scheduling overheads. microBatch is an upgraded version of miniBatch. It triggers mini-batch processing by using event messages, which are inserted into the data sources at a specific interval. microBatch outperforms mini-batch in terms of data accumulation efficiency, back pressure control, throughput, and low latency.

  - Scenarios

    Mini-batch processing is a policy that increases some latency to achieve greater throughput. We recommend that you disable mini-batch processing if you have high requirements on low latency. Generally, we recommend that you enable mini-batch processing, because it significantly improves the job performance in aggregation scenarios.

    Note:

The microBatch mode is also helpful to solve the pain point of two-level aggregation data jitter.

- Enable mini-batch processing

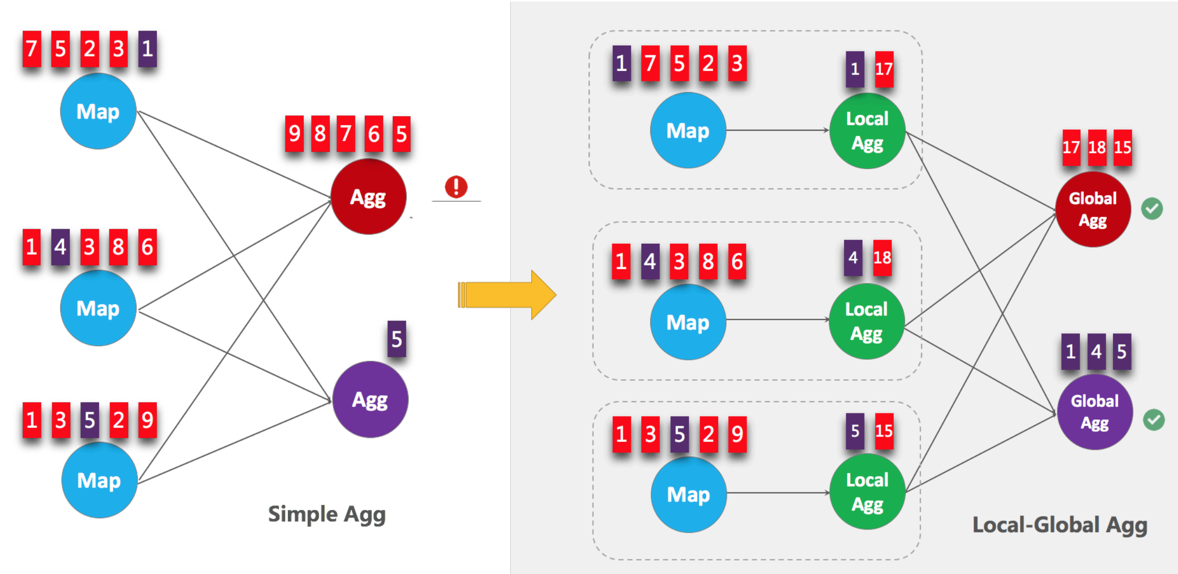
  microBatch and miniBatch are disabled by default. To enable them, specify the following parameters:

```
# This parameter specifies the interval at which
 data is accumulate d . It must be specified when
   you use the microBatch policy . We recommend that
   you set this parameter to the same value as
 that of blink . miniBatch . allowLaten cyMs .
 blink . microBatch . allowLaten cyMs = 5000
# When you use the microBatch policy , you need to
   reserve the following miniBatch settings :
 blink . miniBatch . allowLaten cyMs = 5000
# This parameter specifies the maximum number of
 data records that can be cached for each batch .
 The purpose is to avoid the out of memory ( OOM
 ) error .
 blink . miniBatch . size = 20000
```

· Enable LocalGlobal to solve common data hotspot problems

  The LocalGlobal optimization method divides the conventional aggregation process into two stages: local aggregation and global aggregation. This is similar to the Combine + Reduce processing method that is commonly used in the MapReduce model. In the first stage, Realtime Compute aggregates the first batch of data that is buffered locally at the input node (LocalAgg), and generates accumulators for this micro-batch. In the second stage, it merges the accumulators to obtain the final result (globalAgg).

Essentially, LocalGlobal can eliminate data skew through LocalAgg and solve data hotspot problems during globalAgg to improve the job performance. The following diagram can help you understand how LocalGlobal solves data skew.



- Scenarios

  LocalGlobal is suitable for improving the performance of general aggregation operations such as SUM, COUNT, MAX, MIN, and AVG. It is also helpful to solve data hotspot problems when you perform such operations.

  > Note:
  > To enable LocalGlobal, you need to implement the merge method by using User Defined Aggregation Functions (UDAFs).

- Enable LocalGlobal

  Starting from Realtime Compute `V2.0`, LocalGlobal is enabled by default. When the value of the `blink.localAgg.enabled` parameter is set to true, LocalGlobal is enabled. However, this setting takes effect only when microBatch or miniBatch is enabled.

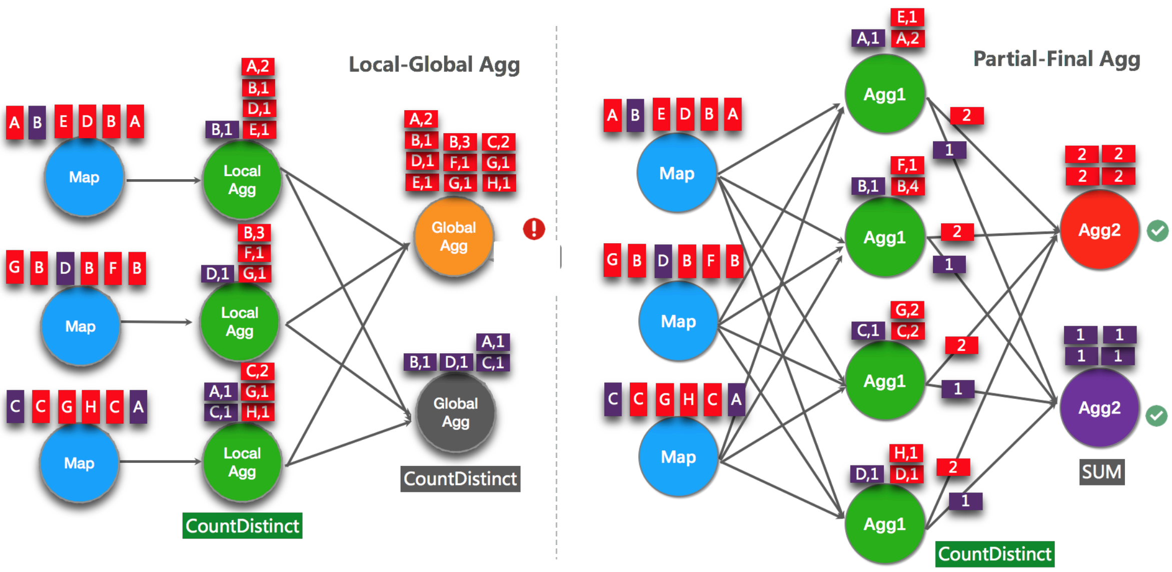
- Check whether the setting takes effect

  You can check whether the GlobalGroupAggregate or LocalGroupAggregate node exists in the final topology.

- Enable PartialFinal to solve data hotspot problems when you run the CountDistinct function

  The LocalGlobal method can effectively improve the performance of general aggregation functions, such as SUM, COUNT, MAX, MIN, and AVG. However, its performance improvement effect for the CountDistinct function is limited. The reason is that the duplicate removal function of LocalAgg does not work well with distinct keys. As a result, a large amount of data is still stacked up at the global node.

  Typically, users who use earlier versions of Realtime Compute need to manually divide the aggregation process into two stages by adding a layer that scatters data by distinct key. This was a workaround to solve data hotspot problems when they run the CountDistinct function. Starting from `V2.2.0`, Realtime Compute offers PartialFinal, an automatic data scattering feature that saves your effort of manually dividing the aggregation process. The following diagram can help you understand the difference between LocalGlobal and PartialFinal.



  - Scenarios

    PartialFinal applies to scenarios where the CountDistinct function is used and the performance at the aggregation node cannot meet your requirements.

    Note:

    - The PartialFinal feature cannot be used in a Flink SQL statement that contains UDAFs.

■ We recommend that you use PartialFinal when data volume is large. The PartialFinal feature automatically scatters data into two aggregation layers, and introduces additional network shuffling. When data volume is not large, it will be a waste of resources.

- Enable PartialFinal

  PartialFinal is disabled by default. You can explicitly enable it by setting the value of the `blink . partialAgg . enabled` parameter to true.

- Check whether the setting takes effect

  You can check whether the Expand node exists in the final topology, or whether the number of the aggregation layer changes from one to two.

· Use the agg with filter syntax to significantly improve the job performance when you run the CountDistinct function

Note:
This method is supported by Realtime Compute `V2 . 2 . 2` and later.

Some statistical jobs may record unique visitors (UVs) of different dimensions, such as UVs of all channels, UVs of the Taobao app, and UVs of the PC client. Many users choose to use the CASE WHEN statement to implement multi-dimensional statistical analysis. However, we recommend that you use the standard agg with filter syntax. The reason is that Realtime Compute has an SQL optimizer that can analyze the filter parameter. The SQL optimizer allows Realtime Compute to run the CountDisatinct function on the same field under different conditions by sharing the state. This reduces the read and write operations on the state. This syntax improves the job performance by one time based on the performance test.

- Scenarios

  We recommend that you replace the agg with CASE WHEN syntax with the agg with filter syntax. This is particularly helpful to improve the job performance when you run the CountDiscount function on the same field under different conditions.

- Original statement

```
COUNT ( distinct   visitor_id ) as   UV1 , COUNT ( distinctca
sewhen   is_wireles  s =' y ' then   visitor_id   elsenullen  d )
as   UV2
```

```

```

- **Optimized statement**

```
COUNT ( distinct   visitor_id ) as   UV1 , COUNT ( distinct
visitor_id )  filter  ( where   is_wireles  s =' y ') as   UV2
```

TopN optimization skills

When the input streams of TopN are non-update streams (such as source), TopN supports only one algorithm: AppendRank. When the input streams of TopN are update streams (that have undergone Agg or Join operations), TopN supports three algorithms. Ranked in a descending order of performance, these algorithms are UpdateFastRank, UnaryUpdateRank, and RetractRank (). The algorithm names will be displayed on the node names of the topology.

- RetractRank is the algorithm with the lowest performance. We do not recommend that you use this algorithm in production. If you have to use this algorithm, check whether the input streams have the primary key (PK) information, and whether you can optimize the job performance.
- UpdateFastRank is the optimal algorithm. The following conditions must be met if you want to use this algorithm: 1. The input streams must have the PK informatio n. 2. The update of the ORDERBY field is monotonic, and the monotonic direction is opposite to the sorting order. For example, order by count, count_distinct, sum (positive) desc.
- The performance of the UnaryUpdateRank algorithm is only second to UpdateFast Rank. One condition must be met if you want to use this algorithm: The input streams must have the PK information. No monotonic information is required. For example, order by avg.
- In the case of order by sum (positive) desc, a positive filter condition must be added.

  In addition, the parameter value of sum must not be negative, and you need to inform the optimizer of such information by adding a positive filter. Then, you can use the UpdateFastRank algorithm. This algorithm is supported by Realtime Compute `V2 . 2 . 2` and later. See the following statements for reference (pay attention to sum(total_fee) filter ...))

```
SELECT   cate_id ,  seller_id ,  stat_date ,  pay_ord_am  t   #
The   rownum   field   is   not   included   in   the   output
.  This   reduces   the   amount   of   output   data   to   be
written   into   the   result   table .
```

```
FROM ( SELECT *
      ROW_NUMBER () OVER ( PARTITIONB Y  cate_id , stat_date
# Be  sure  to  specify  the  stat_date  field . Otherwise
, the  data  will  become  disordered  when  the  state
expires .
ORDERBY  pay_ord_am  t  DESC ## Sort  by  the  sum  of  the
  input  data ) AS  rownum
  FROM ( SELECT  cate_id , seller_id , stat_date ,# Important
! Parameters  that  are  used  to  declare  sum  are  all
  positive , so  results  of  the  sum () function  are
monotonica lly  increasing . That ' s  why  TopN  supports
optimizati on  algorithms . sum ( total_fee ) filter ( where
total_fee >= 0 ) as  pay_ord_am t
    FROMWHERE  total_fee >= 0GROUPBY  cate_name , seller_id ,
stat_date ) WHERE  rownum <= 100 ))
```

- No-ranking optimization

  Do not include rownum in the output of TopN. We recommend that you sort the results when they are finally displayed in the front end. This can significantly reduce the amount of data that is to be written into the result table. For more information, seeTopN statement .

- Increase the cache size of TopN

  TopN has a state cache layer to improve the performance. The cache layer can improve the state access efficiency. The calculation formula of the TopN cache hit rate is:

```
cache_hit  =  cache_size * parallelis  m / top_n / partition_
key_num
```

  Taking Top100 for example. Assume that the cache size is 10,000 and the parallelism is 50. If the number of keys for the partitionBy field is 100,000, the cache hit rate will be 10,000 × 50/100/100,000 = 5%. This value is very low, and large amounts of requests will access the state (disk), and the state seek metric would not be smooth. This also significantly affects the performance. Therefore, if the size of the partitionKey is very large, you may increase the cache size and heap memory of the TopN node. For more information, see Manual configuration optimization.

```
In  this  case , if  you  increase  the  TopN  cache  from
  the  default  value  10 , 000  to  200 , 000 , the  cache
  hit  rate  may  reach  200 , 000  ×  50 / 100 / 100 , 000  =
100 %.
```

```
blink . topn . cache . size = 200 , 000
```

- A time field must be included in the partitionBy field.

  For example, you need to include the day field in your statement for a daily ranking. Otherwise, the TopN result may become disordered due to the state time to live (TTL).

Efficient deduplication solution

- Use the FirstRow method to replace the first_value function.

  Note:
  The FirstRow function is supported by Realtime Compute `V2 . 2 . 2` and later.

  The FirstRow method is used to perform deduplication, where it keeps the first occurrence of duplicate records under the specified primary key, and discards the rest duplicate records. After you replace the first_value function with the FirstRow function, the state of the FirstRow function only stores keys, and the state access efficiency is significantly increased. This improves the Realtime Compute job performance by one time.

  Note:
  Difference between FirstRow and first_value: The FirstRow method applies to an entire row, and reads data of the first row of the key, regardless of whether the field in this row is null. The first_value function applies to the field, and reads the first non-null data record of the key.

  - Original statement (using first_value to remove duplicates):

    ```
    select   biz_order_  id ,  first_valu  e ( seller_id ),
    first_valu  e ( buyer_id ),  first_valu  e ( total_fee ) from
    tt_source
    groupby   biz_order_  id ;
    ```

  - Optimized statement (by using the FirstRow method): You need to add the PK property to the source table, and add the `fetchFirst  Row =' true '` configuration.

    ```
    CREATETABL  E   tt_source   (
    biz_order_  id   varchar ,
    seller_id   varchar ,
    buyer_id   varchar ,
    total_fee   doublePRIM  ARYKEY ( biz_order_  id  #  1 .  Declare
      the   primary   key   that   you   want   to   remove
    duplicates   with ,  which   can   be   a   composite   key .
    ) WITH (
    ```

```
type =' tt ', fetchFirst  Row =' true ' #  2 .  Set   the
value   to   true   to   only   keep   the   first   row .  The
default   value   is   false ,  which   means   to   keep   the
last   row .)
```

- Use the LastRow function to replace the last_value function

  The LastRow function is used to perform deduplication, and it only keeps the last data record under the specified primary key. Its performance is slightly better than that of the last_value function.

  > Note:
  > Difference between LastRow and last_value: The LastRow function applies to an entire row, and reads data of the last row of the key, regardless of whether the field in this row is null. The last_value function applies to the field, and reads the last non-null data record of the key.

  - Original statement (using last_value to remove duplicates):

    ```
    select   biz_order_  id ,
     last_value ( seller_id ),
     last_value ( buyer_id ),
     last_value ( total_fee )
    from   tt_source
    group   by   biz_order_  id ;
    ```

  - Optimized statement (using the LastRow function): You need to add the Primary Key property to the source table, and add the `fetchFirst  Row =' false '` configuration.

    ```
    CREATETABL  E   tt_source   (
    biz_order_  id   varchar ,
    seller_id   varchar ,
    buyer_id   varchar ,
    total_fee   doublePRIM  ARYKEY ( biz_order_  id )#  1 .  Declare
      the   primary   key   that   you   want   to   remove
    duplicates   with ,  which   can   be   a   composite   key .
    ) WITH ( type =' tt ',
    fetchFirst  Row =' false ',#  2 .  The   default   value   is
    false ,  which   means   to   keep   the   last   row .  Use
    the   default   value .)
    ```

Efficient built-in functions

- Use built-in functions to replace UDXs

  Use built-in functions whenever possible. This is very important. Built-in functions of earlier Realtime Compute versions are incomplete. Many users had to use third-party User Defined Extensions (UDXs). In Realtime Compute V2.X, the built-in functions are greatly improved (reduces message serialization or deserialization, and allows direct operations on Bytes). However, UDXs cannot benefit from such improvements.

- The KEY VALUE function uses single-character separators.

  The signature of the KEY VALUE function is: `KEYVALUE ( content , keyValueSp lit , keySplit , keyName )`. When keyValueSplit and KeySplit are single-character separators, such as `:` and `,`, Realtime Compute will use an optimized algorithm. Instead of segmenting the entire content, Realtime Compute directly looks for the required KeyName values among the binary data. This improves the job performance by approximately 30%.
- MULTI_KEYVALUE is used for scenarios with multiple key values.

  > Note:
  > This is supported by Realtime Compute `V2 . 2 . 2` and later.

  Sometimes, a query may involve multiple key value operations on the same content. For example, a content contains 10 key-value pairs, and you hope to extract all these 10 values to use them as fields. You may write 10 key value functions to parse the content 10 times. In this case, we recommend that you use the MULTI_KEYVALUE function, which is a table-valued function. This function only requires one SPLIT parsing on the content. The performance is improved by 50%-100%.
- Notes on LIKE operations
  - If you want to perform startWith operations, use `LIKE ' xxx %'`.
  - If you want to perform endWith operations, use `LIKE '% xxx '`.
  - If you want to perform contains operations, use `LIKE '% xxx %'`.
  - If you want to perform equals operations, use `LIKE ' xxx '`, which is equal to `str = ' xxx '`.
  - If you want to match the `_` character, be sure to use `LIKE '% seller / id %' ESCAPE '/'`. Because `_` is a single-character wildcard in SQL, it can match any characters. If you declare it as `LIKE '% seller_id %'`, it matches a lot of characters such as `Seller_id` , `Seller # id` , `Sellerxid` , and `Seller1id` . The results may be unsatisfactory, and the efficiency may be rather low when regular expressions are used.

- Use the regular expression functions sparingly

  Regular expression operations can be very time consuming, and may require a hundred more times of computing resources in comparison with other operations such as plus, minus, multiplication, and division. If you run regular expressions under some particular circumstances, your job may be stuck in an infinite loop. Therefore, use LIKE whenever possible. For more information, see Notes on LIKE operations. Common regular expression functions include: REGEXP, REGEXP_EXTRACT, and REGEXP_REPLACE.

Network transmission optimization

Commonly used Partitioner policies are:

- KeyGroup/Hash: distributes data based on specified keys.
- Rebalance: distributes data to each channel through round-robin scheduling.
- Dynamic-Rebalance: dynamically distributes data to channels with lower load based on load status of output channels.
- Forward: similar to Rebalance when unchained. When it is chained, it does one-to-one data distribution.
- Rescale: distributes data in a one-to-many or many-to-one mode between input and output systems.

- Use Dynamic-Rebalance to replace Rebalance

  Dynamic Rebalance can write data into subpartitions with lower load based on the amount of buffered data in each subpartition to achieve dynamic load balancing. In comparison with the static rebalance policy, when computing capacity of output computing nodes is unbalanced, Dynamic Rebalance can balance the load and improve the overall job performance. For example, if you find the load of your output nodes is unbalanced when you use rebalance, you may consider to use Dynamic-Rebalance. Parameter: `task . dynamic . rebalance . enabled = true`. Default value: false.

- Use Rescale to replace Rebalance

  > Note:
  >
  > Rescale is supported by Realtime Compute `V2 . 2 . 2` and later.

  Assume that you have five parallel input nodes and 10 parallel output nodes. When you use Rebalance, each input node distributes data to all 10 output nodes through round-robin scheduling. When you use Rescale, each input node only needs to distribute data to two output nodes through round-robin scheduling. This reduces the number of channels, increases the buffering speed of each subpartition, and thus improves the network efficiency. When input data is even and the number of parallel input and output nodes are in proportion, you can use Rescale to replace Rebalance. Parameter: `enable . rescale . shuffling = true`. Default value: false.

Recommended configuration

To sum up, we recommend you use the following job configuration:

```
#  excatly - once   semantics
 blink . checkpoint . mode = EXACTLY_ON  CE
#  The   checkpoint   interval ,  in   millisecon  ds .
 blink . checkpoint . interval . ms = 180000
 blink . checkpoint . timeout . ms = 600000
#  Realtime   Compute   V2 . X   uses   Niagara   as   the   state
 back - end ,  and   uses   it   to   set   the   lifecycle   of   the
   state   data ,  in   millisecon  ds .
 state . backend . type = niagara
 state . backend . niagara . ttl . ms = 129600000
#  Realtime   Compute   V2 . X   enables   a   5 - second   micro -
 batch  ( You   cannot   set   this   parameter   when   you   use   a
   window   function .)
 blink . microBatch . allowLaten  cyMs = 5000
#  The   allowed   latency   for   a   job .
 blink . miniBatch . allowLaten  cyMs = 5000
#  The   size   of   a   batch .
 blink . miniBatch . size = 20000
#  Local   optimizati  on .  This   feature   is   enabled   by
 default   in   Realtime   Compute   V2 . X ,  but   you   need   to
 enable   it   manually   if   you   use   Realtime   Compute   V1 . 6
 . 4 .
 blink . localAgg . enabled = true
#  Realtime   Compute   V2 . X   allows   you   to   enable
 PartialFin  al   to   solve   data   hotspot   problems   when   you
   run   the   CountDisti  nct   function .
 blink . partialAgg . enabled = true
#  Union   all   optimizati  on
 blink . forbid . unionall . as . breakpoint . in . subsection .
 optimizati  on = true
#  Object   reuse   optimizati  on .  Enabled   by   default .
# blink . object . reuse = true
#  GC   optimizati  on  ( You   cannot   set   this   parameter   when
   you   use   a   Log   Service   source   table .)
 blink . job . option =- yD   heartbeat . timeout = 180000  - yD
 env . java . opts ='- verbose : gc  - XX : NewRatio = 3  - XX :+
 PrintGCDet  ails  - XX :+ PrintGCDat  eStamps  - XX : ParallelGC
 Threads = 4 '
#  Time   zone   setting
```

```
blink . job . timeZone = Asia / Shanghai
```

5.2 Automatic configuration optimization

To improve user experience, the Realtime Compute team offers the automatic configuration optimization (AutoConf) feature.

Background and scope

To improve user experience, the Realtime Compute team offers the automatic configuration optimization (AutoConf) feature.

When all operators and input and output systems of your Realtime Compute job meet the performance requirements and are stable, AutoConf can help you properly adjust the job configuration, such as operator resources and parallelism. It also helps you address performance issues of your job throughout the entire process, such as low throughput and back pressure of data hotspot.

In the following scenarios, AutoConf can optimize job performance but cannot address the performance bottlenecks. To address the performance bottlenecks, manually configure your job or contact the technical support team of Realtime Compute.

- Performance issues exist in the input or output systems of a Realtime Compute job.
  - Performance issues in the data source, such as insufficient DataHub partitions and MQ throughput. In this case, you need to increase the partitions of the corresponding source table.
  - Performance issues in the output sink, such as ApsaraDB for RDS deadlock.
- Performance issues of User Defined Extensions (UDXs) (such as UDFs, UDAFs, and UDTFs) of your Realtime Compute job.

Operations

- New jobs

1. Publish a job.

   a. After you complete the SQL development and syntax check, click Publish. The Publish New Version page appears.

   b. Select Automatic CU Configuration for Resource Configuration. Use the default value for the first time.

      - Automatic CU Configuration: AutoConf can generate an optimized resource configuration and assign a CU value based on the default configuration . If you run AutoConf for the first time, AutoConf generates an initial configuration based on empirical data. We recommend that you use AutoConf after your job has been properly running for 5 to 10 minutes and your job metrics, such as source RPS, have been stable for 2 to 3 minutes. Repeat this three to five times to obtain the optimal configuration.
      - Use Latest Manually Configured Resources: The latest saved manual resource configuration is used. If the latest resource configuration was generated by AutoConf, the AutoConf configuration is used. If the latest resource configuration was done manually, the manual configuration is used.

2. Use the default configuration to start the job.

   a. Use the default configuration to start the job.

   b. Start the job.

   Note:

Use automatic CU configuration after your job has been properly running for more than 10 minutes, and your job metrics, such as source RPS, have been stable for 2 to 3 minutes.

3. Use the custom configuration to start the job.

   a. Fine-tune resource parameters.

      You can start the job in AutoConf mode after you manually set the number of CUs (example: 40). You can increase or decrease the number of CUs based on the actual status of the job to optimize the performance.

      - Minimum number of CUs

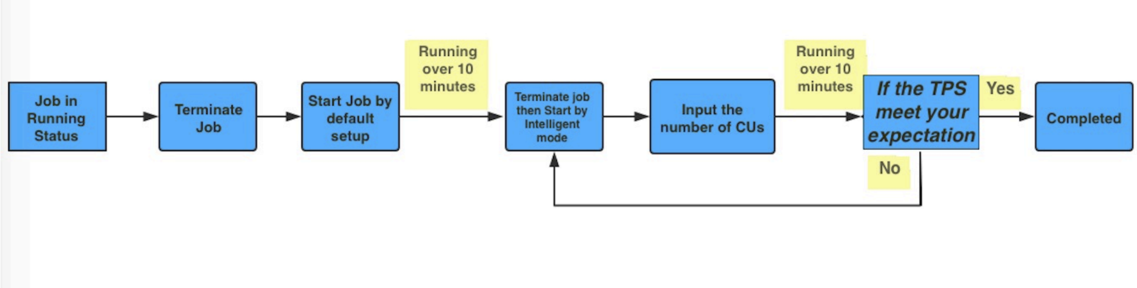
        We recommend that you set the number of CUs to a value that is greater than or equal to 50% of the default value. The minimum number of CUs is 1. If the default value assigned by the automatic configuration is 71, we recommend that you set the minimum number of CUs to 36. 71 CUs × 50% = 35.5 CUs.
      - Increase the number of CUs

        If the throughput of your Realtime Compute job is lower than your expectation, try to increase the number of CUs. We recommend that you increase the number of CUs by at least 30% of the existing value. For example, if the existing value is 10, you can increase the number to 13.
      - Repeat the optimization process

        If one optimization attempt cannot achieve the desired performance, try it more times. You can increase or decrease the number of resources based on the job status after each optimization attempt.
   b. View the result of the optimization inAdministration > Overview > Consumed CUs .

      Note:
      If you are performing automatic configuration on a new job, do not select Use Latest Manually Configured Resources. Otherwise, an error message is displayed.

- Existing jobs
  - Schematic diagram of the optimization process



Note:

1. Before performing automatic configuration on an existing job, check whether stateful operations are involved. This is because the saved state information of a job may be cleared during the automatic configuration process.
2. If your job is changed, for example, an SQL statement is modified or the Realtime Compute version is changed, the automatic configuration may fail. The reason is that these changes may lead to topology changes, which further results in certain issues. For example, curve charts do not display the latest data, or the state cannot be used for fault tolerance. In this case , resource configuration cannot be optimized based on the job running history. An error will be returned when you perform automatic configurat ion. You need to take the changed job as a new job, and repeat the previous operations.

  - Procedure

1. Suspend the job.
2. Repeat the optimization procedure for a new job on this job, and use the latest configuration to start the job.

FAQ

The result of automatic configuration may be compromised in the following scenarios :

- The job runs only for a short period. Only limited useful information can be collected during data sampling. This reduces the accuracy of the results that are computed based on the AutoConf algorithm. We recommend that you prolong the running duration of the job and wait until your job metrics, such as source RPS, are stable for 2 to 3 minutes.

- The job has encountered a failover. This reduces the accuracy of the results. We recommend that you check and handle failovers before performing automatic configuration.
- Only a small amount of data is available for the job. This reduces the accuracy of the results. We recommend that you trace more historical data.
- Affected by many factors, the configuration obtained by using the automatic configuration feature is not always better than the one that was generated the last time. If the automatic configuration feature cannot meet your needs for improving the job performance, manually optimize the configuration. For more information, see Manual configuration optimization.

Recommendations

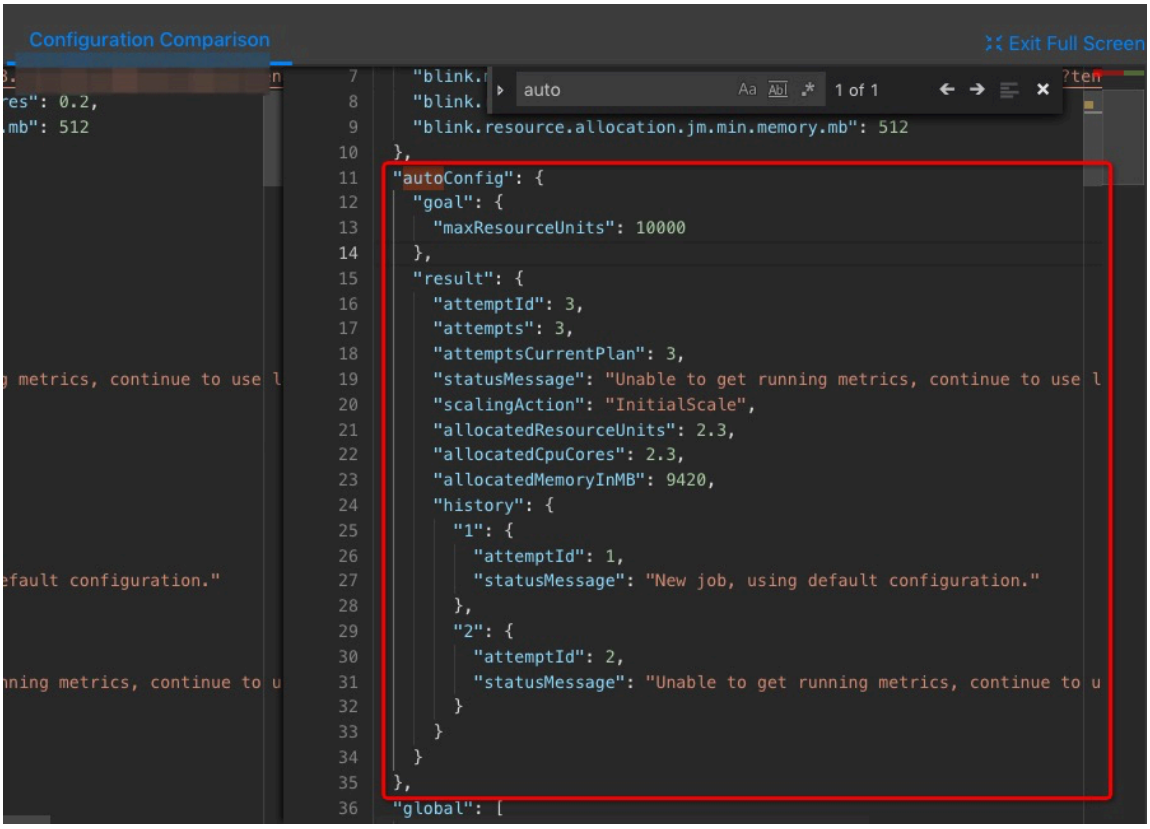
- Before performing automatic configuration on a job, ensure that the job has been running stably and properly for more than 10 minutes. This is helpful to collect accurate job running information for performing automatic configuration.
- You may need to perform automatic configuration for three to five times before the job performance is significantly improved. Therefore, you need to repeat the publishing and O&M operations multiple times.
- When you use AutoConf, you can specify the start offset to read data from the past or even stack up large amounts of data for a job. This allows you to easily and quickly view the configuration optimization results.

Method for determining the effectiveness of automatic configuration

The AutoConf feature for Realtime Compute is enabled based on a JSON configuration file. After performing automatic configuration optimization, you can view the JSON configuration file to check whether this feature is running properly.

- You can view the JSON configuration file by using either of the following methods:

1. View the file on the Development by click Versions > More > Compare > Configuration Comparison.



2. View the file on the job Administration by click Properties and Parameters > Resource Configuration > .

- JSON configuration description

```
" autoConfig " : {
    " goal ": {  //  This  indicates  the  goal  of  automatic
   configurat  ion .
        " maxResourc  eUnits ":  10000 . 0 ,  //  This  indicates
   the  maximum  number  of  CUs  for  a  Blink  job .  The
   value  cannot  be  modified ,  and  you  can  ignore  this
   item  when  checking  whether  the  feature  is  running
 properly .
        " targetReso  ureUnits ":  20 . 0   //  This  indicates
 the  number  of  CUs  that  you  have  specified .
    },
    " result " : {  //  This  indicates  the  results  of
 automatic  configurat  ion .  This  is  very  important .
      " scalingAct  ion " : " ScaleToTar  getResourc  e ",  //  This
   indicates  the  action  of  automatic  configurat  ion . *
      " allocatedR  esourceUni  ts " :  18 . 5 , //  This
 indicates  the  total  resources .
      " allocatedC  puCores " :  18 . 5 ,      //  This  indicates
   the  total  CPU  cores .
      " allocatedM  emoryInMB " :  40960     //  This  indicates
 the  total  memory  size .
      " messages " : " xxxx "  //  We  recommend  that  you  pay
   special  attention  to  the  displayed  messages . *
    }
}
```

- scalingAction: The value of `InitialSca le` indicates that automatic configuration is performed for the first time. The value of `ScaleToTar` `getResourc e` indicates that automatic configuration is not performed for the first time.

- If no message is displayed, the AutoConf feature is running properly. If some messages are displayed, you need to analyze the messages and handle the issues. Messages are categorized into the following two types:
  - Warning: Messages of this type indicate that the feature is running properly , but you need to pay attention to potential issues, such as insufficient partitions of source tables.
  - Error or exception: The `Previous job statistics and configurat ion will be used` error message appears, indicating that automatic configuration optimization fails. The automatic configuration for a job fails in either of the following two cases:
    - The job or Blink version has been modified. In this case, the previous running information cannot be used for automatic configuration.
    - The AutoConf algorithm encounters problems that need to be comprehens ively analyzed based on relevant information and logs. This is often indicated by XxxException. If you do not have enough information to find out the cause of the failure, submit a ticket.

Error messages

IllegalStateException

If the following error messages are displayed, the state cannot be used. To resolve this issue, terminate the job, clear its state, and then specify the start offset to re-read the data.

If you cannot migrate the target job to a backup node and you are concerned that online business may be interrupted, click Properties on the right side of the Development Platform, roll back the target job to the previous version, and then specify the start offset to re-read the data during off-peak hours.

```
java . lang . IllegalSta teExceptio n : Could not initialize
keyed state backend .
    at org . apache . flink . streaming . api . operators
. AbstractSt reamOperat or . initKeyedS tate ( AbstractSt
reamOperat or . java : 687 )
    at org . apache . flink . streaming . api . operators .
AbstractSt reamOperat or . initialize State ( AbstractSt
reamOperat or . java : 275 )
    at org . apache . flink . streaming . runtime . tasks .
StreamTask . initialize Operators ( StreamTask . java : 870 )
```

```
    at   org . apache . flink . streaming . runtime . tasks .
StreamTask . initialize  State ( StreamTask . java : 856 )
    at   org . apache . flink . streaming . runtime . tasks .
StreamTask . invoke ( StreamTask . java : 292 )
    at   org . apache . flink . runtime . taskmanage  r . Task . run
( Task . java : 762 )
    at   java . lang . Thread . run ( Thread . java : 834 )
Caused   by :  org . apache . flink . api . common . typeutils .
Serializat  ionExcepti  on :  Cannot   serialize / deserializ  e
the   object .
    at   com . alibaba . blink . contrib . streaming . state .
AbstractRo  cksDBRawSe  condarySta  te . deserializ  eStateEntr  y (
AbstractRo  cksDBRawSe  condarySta  te . java : 167 )
    at   com . alibaba . blink . contrib . streaming . state .
RocksDBInc  rementalRe  storeOpera  tion . restoreRaw  StateData (
RocksDBInc  rementalRe  storeOpera  tion . java : 425 )
    at   com . alibaba . blink . contrib . streaming . state .
RocksDBInc  rementalRe  storeOpera  tion . restore ( RocksDBInc
rementalRe  storeOpera  tion . java : 119 )
    at   com . alibaba . blink . contrib . streaming . state .
RocksDBKey  edStateBac  kend . restore ( RocksDBKey  edStateBac  kend
. java : 216 )
    at   org . apache . flink . streaming . api . operators
. AbstractSt  reamOperat  or . createKeye  dStateBack  end (
AbstractSt  reamOperat  or . java : 986 )
    at   org . apache . flink . streaming . api . operators
. AbstractSt  reamOperat  or . initKeyedS  tate ( AbstractSt
reamOperat  or . java : 675 )
   ...  6   more
Caused   by :  java . io . EOFExcepti  on
    at   java . io . DataInputS  tream . readUnsign  edByte (
DataInputS  tream . java : 290 )
    at   org . apache . flink . types . StringValu  e . readString (
StringValu  e . java : 770 )
    at   org . apache . flink . api . common . typeutils . base .
StringSeri  alizer . deserializ  e ( StringSeri  alizer . java : 69 )
    at   org . apache . flink . api . common . typeutils . base .
StringSeri  alizer . deserializ  e ( StringSeri  alizer . java : 28 )
    at   org . apache . flink . api . java . typeutils . runtime .
RowSeriali  zer . deserializ  e ( RowSeriali  zer . java : 169 )
    at   org . apache . flink . api . java . typeutils . runtime .
RowSeriali  zer . deserializ  e ( RowSeriali  zer . java : 38 )
    at   com . alibaba . blink . contrib . streaming . state .
AbstractRo  cksDBRawSe  condarySta  te . deserializ  eStateEntr  y (
AbstractRo  cksDBRawSe  condarySta  te . java : 162 )
   ...  11   more
```

5.3 Manual configuration optimization

This topic describes how to manually optimize the configuration of a Realtime Compute job.

Manual configuration optimization

You can manually optimize the configuration of a Realtime Compute job by using one or all of the following methods:

- Fine-tune job parameters such as miniBatch.
- Fine-tune resource parameters, such as parallelism, core, and heap_memory, for operators.
- Fine-tune input and output storage parameters of the job.

More details about these three methods are described in the following sections. After you fine-tune the specific parameters, Realtime Compute generates a new configurat ion. You need to republish the job or resume the job (if it is suspended) to apply the new configuration. The detailed process is provided in the last section of this topic.

Fine-tune job parameters

The miniBatch parameter can be used to optimize only GROUP BY operators. During the streaming data processing of Flink SQL, the state is read each time a data record arrives for processing, which consumes large amounts of I/O resources. After you set the miniBatch parameter, Realtime Compute reads the state only once for data records with the same key, and the output contains only the latest data record. This reduces the frequency of reading the state and minimizes the data output updates. When you add miniBatch as a new parameter for your job, we recommend that you terminate the job before you set the parameter, and then restart the job. If you want to change the value of this parameter, you can suspend the job beforehand, and then resume the job.

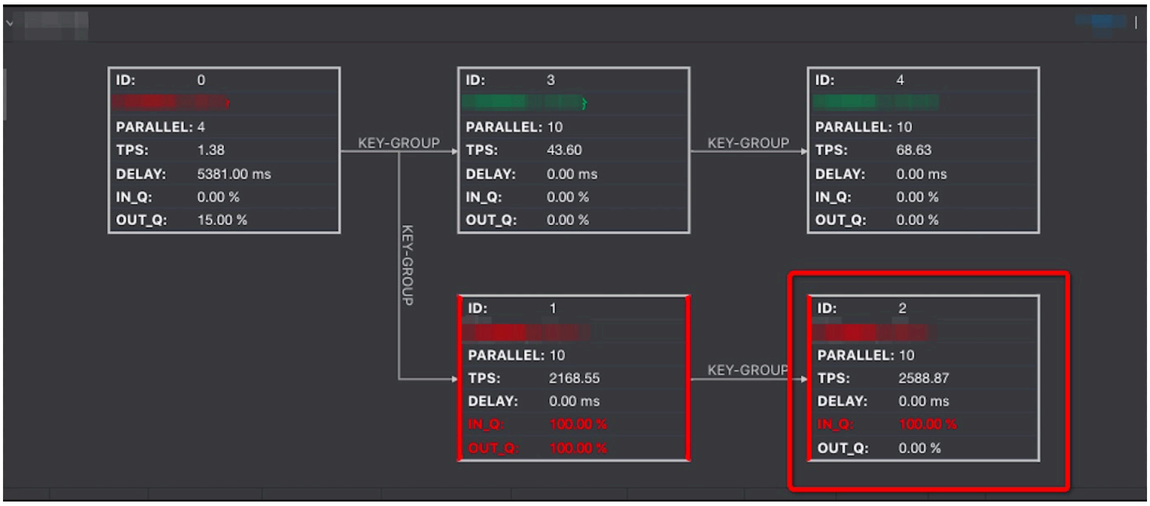
```
# excatly - once semantics
 blink . checkpoint . mode = EXACTLY_ON CE
# The checkpoint interval , in millisecon ds .
 blink . checkpoint . interval . ms = 180000
 blink . checkpoint . timeout . ms = 600000
# Realtime Compute V2 . X uses Niagara as the state
 back - end , and uses it to set the lifecycle of the
   state data , in millisecon ds .
 state . backend . type = niagara
 state . backend . niagara . ttl . ms = 129600000
# Realtime Compute V2 . X enables a 5 - second micro -
 batch ( You cannot set this parameter when you use a
   window function .)
 blink . microBatch . allowLaten cyMs = 5000
# The allowed latency for a job .
 blink . miniBatch . allowLaten cyMs = 5000
# The size of a batch .
 blink . miniBatch . size = 20000
# Local optimizati on . This feature is enabled by
 default in Realtime Compute V2 . X , but you need to
 enable it manually if you use Realtime Compute V1 . 6
 . 4 .
 blink . localAgg . enabled = true
# Realtime Compute V2 . X allows you to enable
 PartialFin al to solve data hotspot problems when you
   run the CountDisti nct function .
 blink . partialAgg . enabled = true
```

```
# union  all  optimizati on
 blink . forbid . unionall . as . breakpoint . in . subsection .
 optimizati on = true
# GC  optimizati on ( You  cannot  set  this  parameter  when
   you  use  a  Log  Service  source  table .)
 blink . job . option =- yD  heartbeat . timeout = 180000  - yD
 env . java . opts ='- verbose : gc  - XX : NewRatio = 3  - XX :+
 PrintGCDet ails  - XX :+ PrintGCDat eStamps  - XX : ParallelGC
 Threads = 4 '
# Time  zone  setting
 blink . job . timeZone = Asia / Shanghai
```

Fine-tune resource parameters

1. Problem analysis

   a. As shown in the following topology, the percentage of input queues at task node 2 has reached 100%. The data of task node 2 is stacked up and puts pressure back on task node 1, at which the percentage of output queues has reached 100%.



   b. You can click task node 2 and locate the subtask in SubTask List where the percentage of InQueue has reached 100%. Then, click View Logs to view the detailed information.

   c. Check the CPU and memory usage in TaskExecutor > Metrics Graph. Increase the CPU capacity and memory size based on the actual use.

2. Performance optimization

   a. On Development , click Basic Properties > Configure Resources .
   b. Locate the group (if any) or operator that corresponds to task node 2. You can modify the parameters of one or multiple operators in one group at a time.

      - Modify parameters of multiple operators in a group:

        A. Hover your mouse on the target GROUP box.
        B. Click the Pencil icon .
        C. Modify Operator parameters in Modify Operator Data.
      - Modify parameters of a single operator:

        A. Click# at the group box where the target operator belongs to.
        B. Hover your mouse on the target operator box.
        C. Click the Pencil icon .
        D. Modify Operator parameters in Modify Operator Data.

   c. After you modify the parameters, click Apply and Close in the upper-right corner of the page.

> Note:
>
> During optimization, if the resource configuration of a group has been optimized but the performance still does not improve, you need to verify whether data skew exists in the current node. If data skew is detected, fix it immediately. Then, separate operators that involve complex computation (such as GROUP BY, WINDOW, and JOIN) from the group, and locate the abnormal operator. Fine-tune the abnormal operator. To separate an operator from a group, click the operator to be modified and change the value of its chainingStrategy parameter to HEAD . If the value is already HEAD, click the next operator and modify the value of its chainingStrategy parameter to HEAD. The options for the chainingStrategy parameter are as follows:
>
> - ALWAYS: combines the operator with others to form a group.
> - NEVER: retains the status of the operator.
> - HEAD: separates the operator from a group.

3. Principles and recommendations

- Adjustable parameters

  - parallelism

    - Sources

      Note:
      The number of parallel subtasks in the source must not be greater than the number of shards in the source table.

      a. Set the parallelism parameter based on the number of source table partitions.
      b. For example, if the number of source table partitions is 16, set the parallelism parameter to 16, 8, or 4. Note that the maximum value is 16.

    - Intermediate processing nodes

      a. Set the parallelism parameter based on the estimated queries per second (QPS).
      b. For tasks with low QPS, set the parallelism parameter for the intermedia te processing nodes to the same value as that for the sources.
      c. For tasks with high QPS, set the parallelism parameter to a larger value, such as 64, 128, or 256.

    - Sinks

      a. Set the parallelism parameter for sinks to a value that is two or three times the number of result table partitions.
      b. However, if the specified parallelism limit is exceeded, a write timeout or failure occurs. For example, if the number of output sinks is 16, the recommended maximum value of the parallelism parameter for sinks is 48.

  - core

    This parameter indicates the number of CPU cores. The default value is 0.1 . Set this parameter based on CPU usage. We recommend that you set this parameter to a value whose reciprocal is an integer. The recommended value is 0.25.

- heap_memory

  This parameter indicates the heap memory size, whose default value is 256 MB. The value is determined based on the actual memory usage. You can click GROUP on the resource editing page to modify the preceding parameters.

· Adjustable parameters at task nodes with GROUP BY operators

  state_size: specifies the state size. The default value is 0. If the operator state is used, set the state_size parameter to `1`. In this case, the corresponding job requests extra memory for this operator. The extra memory is used to store the state. If the state_size parameter is not set to `1`, the corresponding job may be killed by YARN. Operators whose state_size needs to be set to 1: GROUP BY, JOIN, OVER, and WINDOW. In the face of so many configuration items, you can focus on these parameters: core, parallelism, and heap_memory. For each job, we recommend that you assign 4 GB memory for each core.

> Note:
>
> Rules to follow when you adjust the parallelism and memory size:
>
> Total number of compute units (CUs) of an operator = Value of parallelism × Number of cores. Total memory size of an operator = Value of parallelism × heap_mem. The CPU to MEM ratio must be 1:4, where CPU is the maximum number of CPU cores within a group, and MEM is the total memory of each operator within the group. For example, if you have 1 CU and 3 GB memory, the final configuration would be 1 CU and 4 GB memory. If you have 1 CU and 5 GB memory, the final configuration would be 1.25 CUs and 5 GB memory.

Fine-tune input and output storage parameters

Realtime Compute enables real-time computing, which means that each data record can trigger read and write operations on source and result tables. This brings considerable challenges for data storage performance. To address these challenges , you can set batch size parameters to specify the number of data records that are read from a source table or written into a result table at a time. The following table describes the available batch size parameters.

Table	Parameter	Description	Value
DataHub source table	batchReadSize	The number of data records that are read at a time.	Optional. Default value: 10.
DataHub result table	batchSize	The number of data records that are written at a time.	Optional. Default value: 300.
Log Service source table	batchGetSize	The number of log groups that are read at a time.	Optional. Default value: 10.
AnalyticDB result table	batchSize	The number of data records that are written at a time.	Optional. Default value: 1000.
ApsaraDB for RDS result table	batchSize	The number of data records that are written at a time.	Optional. Default value: 50.
HybridDB for MySQL result table	batchSize	The number of data records that are written at a time.	Optional. Default value: 1000. We recommend that you set a value smaller than 4096.
	bufferSize	The buffer size after deduplication. This parameter takes effect only when the primary key is specified.	Optional. You must set bufferSize before you can set batchSize. We recommend that you set bufferSize to a value smaller than 4096.

Apply the new configuration

After completing parameter settings in the preceding sections, you need to restart your job or resume your job (if it is suspended) to apply the new configuration.

1. Republish the job. In the Publish New Version dialog box, select Use Latest Manually Configured Resources for Resource Configuration.
2. Suspend the job.
3. Resume the job.

4. In the Resume Job dialog box, select Resume with Latest Configuration. Otherwise, the new configuration cannot take effect.
5. After you resume the job, you can choose Administration > Overview > Vertex Topology to check whether the new configuration has taken effect.

Note:

We do not recommend that you terminate and restart a job to apply the new configuration. After a job is terminated, its status is cleared. In this case, the computing result may be inconsistent with the result that is obtained if you suspend and resume the job.

Glossary

- global
  - isChainingEnabled: indicates whether chaining is enabled. Default value: true. Use the default value.

- nodes
  - id: specifies the unique ID of a node. The ID is automatically generated and does not need to be changed.
  - uid: specifies the UID of a node, which is used to calculate the operator ID. If this parameter is not specified, the ID is used.
  - pact: specifies the type of a node. Example values: Data Source, Operator, and Data Sink. Use the default value.
  - name: specifies the name of a node, which can be customized.
  - slotSharingGroup: `default`. Use the default value.
  - chainingStrategy: specifies the chaining strategy. Valid values: HEAD, ALWAYS, and NEVER. You can change the value as needed.
  - parallelism: specifies the number of parallel subtasks. Default value: `1`. You can increase the value based on the data volume.
  - core: specifies the number of CPU cores. Default value: `0.1`. The value is configured based on the CPU usage. We recommend that you set this parameter to a value whose reciprocal is an integer. The recommended value is `0.25`.
  - heap_memory: specifies the heap memory size. Default value: 256 MB. Set this parameter based on the memory usage.
  - direct_memory: specifies the JVM non-heap memory size. Default value: `0`. Use the default value.
  - native_memory: specifies the JVM non-heap memory size for the Java Native Interface (JNI). Default value: `0`. The recommended value is 10 MB.
- chain
  - A Flink SQL job resembles a Directed Acyclic Graph (DAG) that contains many nodes, which are also known as operators. Some input and output operators can be combined to form a chain when they are running. The CPU capacity of a chain is set to the maximum CPU capacity among operators in the chain. The memory size of a chain is set to the total memory size of operators in the chain. For example, node 1 (256 MB, 0.2 cores), node 2 (128 MB, 0.5 cores), and node 3 (128 MB, 0.25 cores) are combined to form a chain. The CPU capacity of the chain is 0.5 cores and the memory is 512 MB. The prerequisite for chaining operators is that the operators to be chained must have the same parallelism settings. However, some operators cannot be chained, such as GROUP BY operators.

We recommend that you chain operators to improve the efficiency of network transmission.

6 Flink SQL

6.1 Flink SQL overview

As a development language that conforms to standard SQL semantics, Flink SQL is designed to simplify the computational model, making it easy for users to use Realtime Compute.

This topic describes how to use Flink SQL in Realtime Compute from the following perspectives:

- Basic concepts
- Keywords
- Data types
- DDL statements
- DML statements
- Query statements
- Data views
- Window functions
- Logical functions
- Built-in functions
- UDFs

6.2 Keywords

This topic describes the reserved keywords in Realtime Compute and how to use these keywords.

Common keyword types

Common type	Keyword
Data type	VARCHAR, INT, BIGINT, DOUBLE, DATE, BOOLEAN, TINYINT, SMALLINT, FLOAT, DECIMAL, and VARBINARY
DDL	CREATE TABLE, CREATE FUNCTION, and CREATE VIEW
DML	INSERT INTO

Common type	Keyword
SELECT clause	SELECT FROM, WHERE, GROUP BY, and JOIN

Naming conventions

Names of source tables, result tables, views, and aliases must follow the standard database naming conventions. The names must start with a letter, and can contain only letters, numbers, and underscores (_).

Reserved keywords

The following combinations of characters are reserved as keywords in Realtime Compute for future use. If you want to use any of the following keywords as a field name, enclose the keyword in backticks (`` ` ``), for example, `` ` value ` ``.

`A , ABS , ABSOLUTE , ACTION , ADA , ADD , ADMIN , AFTER , ALL , ALLOCATE , ALLOW , ALTER , ALWAYS , AND , ANY , ARE , ARRAY , AS , ASC , ASENSITIVE , ASSERTION , ASSIGNMENT , ASYMMETRIC , AT , ATOMIC , ATTRIBUTE , ATTRIBUTES , AUTHORIZAT ION , AVG , BEFORE , BEGIN , BERNOULLI , BETWEEN , BIGINT , BINARY , BIT , BLOB , BOOLEAN , BOTH , BREADTH , BY , C , CALL , CALLED , CARDINALIT Y , CASCADE , CASCADED , CASE , CAST , CATALOG , CATALOG_NA ME , CEIL , CEILING , CENTURY , CHAIN , CHAR , CHARACTER , CHARACTERI STICTS , CHARACTERS , CHARACTER_ LENGTH , CHARACTER_ SET_CATALO G , CHARACTER_ SET_NAME , CHARACTER_ SET_SCHEMA , CHAR_LENGT H , CHECK , CLASS_ORIG IN , CLOB , CLOSE , COALESCE , COBOL , COLLATE , COLLATION , COLLATION_ CATALOG , COLLATION_ NAME , COLLATION_ SCHEMA , COLLECT , COLUMN , COLUMN_NAM E , COMMAND_FU NCTION , COMMAND_FU NCTION_COD E , COMMIT , COMMITTED , CONDITION , CONDITION_ NUMBER , CONNECT , CONNECTION , CONNECTION _NAME , CONSTRAINT , CONSTRAINT S , CONSTRAINT _CATALOG , CONSTRAINT _NAME , CONSTRAINT _SCHEMA , CONSTRUCTO R , CONTAINS , CONTINUE , CONVERT , CORR , CORRESPOND ING , COUNT , COVAR_POP , COVAR_SAMP , CREATE , CROSS , CUBE , CUME_DIST , CURRENT , CURRENT_CA TALOG , CURRENT_DA TE , CURRENT_DE FAULT_TRAN SFORM_GROU P , CURRENT_PA TH , CURRENT_RO LE , CURRENT_SC HEMA , CURRENT_TI ME , CURRENT_TI MESTAMP , CURRENT_TR ANSFORM_GR OUP_FOR_TY PE ,`

CURRENT_US ER , CURSOR , CURSOR_NAM E , CYCLE , DATA , DATABASE ,
DATE , DATETIME_I NTERVAL_CO DE , DATETIME_I NTERVAL_PR ECISION
, DAY , DEALLOCATE , DEC , DECADE , DECIMAL , DECLARE , DEFAULT
, DEFAULTS , DEFERRABLE , DEFERRED , DEFINED , DEFINER , DEGREE ,
DELETE , DENSE_RANK , DEPTH , DEREF , DERIVED , DESC , DESCRIBE ,
DESCRIPTIO N , DESCRIPTOR , DETERMINIS TIC , DIAGNOSTIC S , DISALLOW
, DISCONNECT , DISPATCH , DISTINCT , DOMAIN , DOUBLE , DOW , DOY ,
DROP , DYNAMIC , DYNAMIC_FU NCTION , DYNAMIC_FU NCTION_COD E ,
EACH , ELEMENT , ELSE , END , END - EXEC , EPOCH , EQUALS , ESCAPE ,
EVERY , EXCEPT , EXCEPTION , EXCLUDE , EXCLUDING , EXEC , EXECUTE ,
EXISTS , EXP , EXPLAIN , EXTEND , EXTERNAL , EXTRACT , FALSE , FETCH
, FILTER , FINAL , FIRST , FIRST_VALU E , FLOAT , FLOOR , FOLLOWING
, FOR , FOREIGN , FORTRAN , FOUND , FRAC_SECON D , FREE , FROM ,
FULL , FUNCTION , FUSION , G , GENERAL , GENERATED , GET , GLOBAL ,
GO , GOTO , GRANT , GRANTED , GROUP , GROUPING , HAVING , HIERARCHY ,
HOLD , HOUR , IDENTITY , IMMEDIATE , IMPLEMENTA TION , IMPORT , IN ,
INCLUDING , INCREMENT , INDICATOR , INITIALLY , INNER , INOUT , INPUT
, INSENSITIV E , INSERT , INSTANCE , INSTANTIAB LE , INT , INTEGER ,
INTERSECT , INTERSECTI ON , INTERVAL , INTO , INVOKER , IS , ISOLATION
, JAVA , JOIN , K , KEY , KEY_MEMBER , KEY_TYPE , LABEL , LANGUAGE ,
LARGE , LAST , LAST_VALUE , LATERAL , LEADING , LEFT , LENGTH , LEVEL
, LIBRARY , LIKE , LIMIT , LN , LOCAL , LOCALTIME , LOCALTIMES TAMP ,
LOCATOR , LOWER , M , MAP , MATCH , MATCHED , MAX , MAXVALUE , MEMBER
, MERGE , MESSAGE_LE NGTH , MESSAGE_OC TET_LENGTH , MESSAGE_TE XT
, METHOD , MICROSECON D , MILLENNIUM , MIN , MINUTE , MINVALUE , MOD
, MODIFIES , MODULE , MONTH , MORE , MULTISET , MUMPS , NAME , NAMES ,
NATIONAL , NATURAL , NCHAR , NCLOB , NESTING , NEW , NEXT , NO , NONE
, NORMALIZE , NORMALIZED , NOT , NULL , NULLABLE , NULLIF , NULLS ,
NUMBER , NUMERIC , OBJECT , OCTETS , OCTET_LENG TH , OF , OFFSET ,
OLD , ON , ONLY , OPEN , OPTION , OPTIONS , OR , ORDER , ORDERING ,
ORDINALITY , OTHERS , OUT , OUTER , OUTPUT , OVER , OVERLAPS , OVERLAY
, OVERRIDING , PAD , PARAMETER , PARAMETER_ MODE , PARAMETER_ NAME
, PARAMETER_ ORDINAL_PO SITION , PARAMETER_ SPECIFIC_C ATALOG ,

PARAMETER_ SPECIFIC_N AME , PARAMETER_ SPECIFIC_S CHEMA , PARTIAL
, PARTITION , PASCAL , PASSTHROUG H , PATH , PERCENTILE _CONT ,
PERCENTILE _DISC , PERCENT_RA NK , PLACING , PLAN , PLI , POSITION
, POWER , PRECEDING , PRECISION , PREPARE , PRESERVE , PRIMARY , PRIOR
, PRIVILEGES , PROCEDURE , PUBLIC , QUARTER , RANGE , RANK , READ ,
READS , REAL , RECURSIVE , REF , REFERENCES , REFERENCIN G , REGR_AVGX
, REGR_AVGY , REGR_COUNT , REGR_INTER CEPT , REGR_R2 , REGR_SLOPE
, REGR_SXX , REGR_SXY , REGR_SYY , RELATIVE , RELEASE , REPEATABLE ,
RESET , RESTART , RESTRICT , RESULT , RETURN , RETURNED_C ARDINALITY
, RETURNED_L ENGTH , RETURNED_O CTET_LENGT H , RETURNED_S QLSTATE
, RETURNS , REVOKE , RIGHT , ROLE , ROLLBACK , ROLLUP , ROUTINE ,
ROUTINE_CA TALOG , ROUTINE_NA ME , ROUTINE_SC HEMA , ROW , ROWS
, ROW_COUNT , ROW_NUMBER , SAVEPOINT , SCALE , SCHEMA , SCHEMA_NAM
E , SCOPE , SCOPE_CATA LOGS , SCOPE_NAME , SCOPE_SCHE MA , SCROLL
, SEARCH , SECOND , SECTION , SECURITY , SELECT , SELF , SENSITIVE
, SEQUENCE , SERIALIZAB LE , SERVER , SERVER_NAM E , SESSION ,
SESSION_US ER , SET , SETS , SIMILAR , SIMPLE , SIZE , SMALLINT
, SOME , SOURCE , SPACE , SPECIFIC , SPECIFICTY PE , SPECIFIC_N
AME , SQL , SQLEXCEPTI ON , SQLSTATE , SQLWARNING , SQL_TSI_DA Y
, SQL_TSI_FR AC_SECOND , SQL_TSI_HO UR , SQL_TSI_MI CROSECOND ,
SQL_TSI_MI NUTE , SQL_TSI_MO NTH , SQL_TSI_QU ARTER , SQL_TSI_SE
COND , SQL_TSI_WE EK , SQL_TSI_YE AR , SQRT , START , STATE ,
STATEMENT , STATIC , STDDEV_POP , STDDEV_SAM P , STREAM , STRUCTURE
, STYLE , SUBCLASS_O RIGIN , SUBMULTISE T , SUBSTITUTE , SUBSTRING
, SUM , SYMMETRIC , SYSTEM , SYSTEM_USE R , TABLE , TABLESAMPL E ,
TABLE_NAME , TEMPORARY , THEN , TIES , TIME , TIMESTAMP , TIMESTAMPA
DD , TIMESTAMPD IFF , TIMEZONE_H OUR , TIMEZONE_M INUTE , TINYINT
, TO , TOP_LEVEL_ COUNT , TRAILING , TRANSACTIO N , TRANSACTIO
NS_ACTIVE , TRANSACTIO NS_COMMITT ED , TRANSACTIO NS_ROLLED_
BACK , TRANSFORM , TRANSFORMS , TRANSLATE , TRANSLATIO N , TREAT ,
TRIGGER , TRIGGER_CA TALOG , TRIGGER_NA ME , TRIGGER_SC HEMA , TRIM
, TRUE , TYPE , UESCAPE , UNBOUNDED , UNCOMMITTE D , UNDER , UNION ,
UNIQUE , UNKNOWN , UNNAMED , UNNEST , UPDATE , UPPER , UPSERT , USAGE

, USER , USER_DEFIN ED_TYPE_CA TALOG , USER_DEFIN ED_TYPE_CO DE , USER_DEFIN ED_TYPE_NA ME , USER_DEFIN ED_TYPE_SC HEMA , USING , VALUE , VALUES , VARBINARY , VARCHAR , VARYING , VAR_POP , VAR_SAMP , VERSION , VIEW , WEEK , WHEN , WHENEVER , WHERE , WIDTH_BUCK ET , WINDOW , WITH , WITHIN , WITHOUT , WORK , WRAPPER , WRITE , XML , YEAR , ZONE

6.3 Basic concepts

6.3.1 Time zone

This topic describes how to set the time zone for a job in Realtime Compute.

Note:
This topic applies to Realtime Compute `V1 . 6 . 0` and later.

Introduction

Realtime Compute allows you to set the time zone for an entire job. The default time zone is UTC+8. Examples of valid time zones are `Asia / Shanghai`, `America / New_York`, and `UTC`. For the list of supported time zones, see the last part of this topic.

You can also set the time zone for a source or sink table independently. Assume that you want to read data from or write data into a MySQL database where the Time, Date, and Timestamp columns use the `America / New_York` time zone. However, the `Asia / Shanghai` time zone needs to be used in the computation of a job. In this scenario, you can set the time zone for a source or sink table independently as follows:

```
 CREATE   TABLE   mysql_sour  ce_my_tabl  e  (
 -- ...
)  WITH  (
 timeZone =' America / New_York '
 -- ...
```

```
)
```

Examples

In Realtime Compute `V1 . 6 . 0` and later, all time zone-related functions compute data based on custom time zones in terms of semantics. The following uses the custom time zone `Asia / Shanghai` as an example to describe the functions:

- Functions for converting a string to a timestamp (TO_TIMESTAMP, TIMESTAMP, and UNIX_TIMESTAMP)

```
--  Scalar   function
 TO_TIMESTA  MP (" 2018 - 03 - 14   19 : 01 : 02 . 123 ")
--  SQL   Literal

 TIMESTAMP  ' 2018 - 03 - 14   19 : 01 : 02 . 123 '
--  Output :
--  Realtime   Compute   V1 . 6 . 0   and   later : ` 1521025262
 123 `.
--  Realtime   Compute   V1 . 5 . x : ` 1520996462  123 `.
--  The   UNIX_TIMES  TAMP   function   can   be   used   for   a
   similar   purpose .  The   difference   lies   in   that   its
 output   is   measured   in   seconds .
```

- Functions for converting a timestamp to a string (FROM_UNIXTIME and DATE_FORMAT)

> **Note:**
> If the input parameter is of the TIMESTAMP type, the output depends on your custom time zone.

```
 SELECT   DATE_FORMA  T ( TO_TIMESTA  MP ( 1520960523  000 ), ' yyyy
 - MM - dd   HH : mm : ss ')
--  Output :
--  Realtime   Compute   V1 . 6 . 0   and   later : ` 2018 - 03 - 14
   01 : 02 : 03 `.
--  Realtime   Compute   V1 . 5 . x : ` 2018 - 03 - 13   15 : 02 :
 03 `.

 SELECT   DATE_FORMA  T ( TO_TIMESTA  MP ( 1520960523  000 ), ' yyyy
 - MM - dd   HH : mm : ss ')
--  Output :
--  Realtime   Compute   V1 . 6 . 0   and   later : ` 2018 - 03 - 14
   01 : 02 : 03 `.
--  Realtime   Compute   V1 . 5 . x : ` 2018 - 03 - 13   15 : 02 :
 03 `.


--  Note   that   in   the   following   example ,  the   output
   in   Realtime   Compute   V1 . 6 . 0   is   consistent   with
 that   in   Realtime   Compute   V1 . 5 . x .  This   is   because
   the   input   and   output   time   strings   are   computed
 based   on   the   same   time   zone .
```

```
 DATE_FORMA  T (' 2018 - 03 - 14   01 : 02 : 03 ', ' yyyy - MM - dd
   HH : mm : ss ', ' yyyy / MM / dd   HH : mm : ss ')
 FROM_UNIXT  IME ( 1521025200  000 / 1000 )
--  Output :
--  Realtime   Compute   V1 . 6 . 0   and   later : ` 2018 - 03 - 14
   19 : 00 : 00 `.
--  Realtime   Compute   V1 . 5 . x : ` 2018 - 03 - 14   11 : 00 :
 00 `.
```

- Time-related computation functions

If the input parameter is of the TIMESTAMP type, the output of the EXTRACT, FLOOR, CEIL, or DATEDIFF function depends on your custom time zone. If the input parameter is a string, the output in Realtime Compute `V1 . 6 . 0` is consistent with that in Realtime Compute `V1 . 5 . x`. This is because the input and output time strings are computed based on the same time zone.

```
--  1521503999  000   2018 - 03 - 19T23 : 59 : 59 + 0000 ,  2018 -
 03 - 20T07 : 59 : 59 + 0800
  EXTRACT ( DAY   FROM   TO_TIMESTA  MP ( 1521503999  000 ))
--  Output :
--  Realtime   Compute   V1 . 6 . 0   and   later : ` 20 `,  which
 indicates   the   20th   day   of   the   month   in   UTC + 8 .
--  Realtime   Compute   V1 . 5 . x : ` 19 `.
```

- Functions for computing the current time (`LOCALTIMES  TAMP ()`, `CURRENT_TI  MESTAMP ()`, `NOW ()`, and `UNIX_TIMES  TAMP ()`)

In Realtime Compute `V1 . 6 . 0`, the semantics of the `LOCALTIMES  TAMP` function are changed to return the current timestamp. In contrast, the `DATE_FORMA  T` function does not involve a time zone in Realtime Compute `V1 . 5 . x`. To make sure that the output of `DATE_FORMA  T ( CURRENT_TI  MESTAMP )` is correct, the default time zone `offset` is added to the `LOCALTIMES  TAMP` function, which is incorrect.

```
--  The   current   time   is   2018 - 04 - 03   16 : 56 : 10   in
 the   Asia / Shanghai   time   zone .
 SELECT   DATE_FORMA  T ( CURRENT_TI  MESTAMP , ‘ yyyy - MM - dd
 HH : mm : ss ’);
--  Output :
--  Realtime   Compute   V1 . 6 . 0   and   later : ` 2018 - 04 - 03
   16 : 56 : 10 `.
--  Realtime   Compute   V1 . 5 . x : ` 2018 - 04 - 03   08 : 56 :
 10 `.

 SELECT   DATE_FORMA  T ( LOCALTIMES  TAMP , ‘ yyyy - MM - dd   HH :
 mm : ss ’);
--  Output :
--  Realtime   Compute   V1 . 6 . 0 : ` 2018 - 04 - 03   16 : 56 :
 10 `.
--  Realtime   Compute   V1 . 5 . x : ` 2018 - 04 - 03   16 : 56 :
 10 `.
```

```
-- The same output is returned in Realtime Compute
   V1 . 6 . 0 and V1 . 5 . x . However , the output
 timestamps of the LOCALTIMES TAMP function are
 actually different in these versions .
 SELECT FROM_UNIXT IME ( NOW ()); SELECT FROM_UNIXT IME (
 UNIX_TIMES TAMP ());
-- Output :
-- Realtime Compute V1 . 6 . 0 and later : ` 2018 - 04 - 03
   16 : 56 : 10 `.
-- Realtime Compute V1 . 5 . x : ` 2018 - 04 - 03 08 : 56 :
 10 `.
-- The semantics of the NOW () and UNIX_TIMES TAMP ()
 functions remain unchanged in Realtime Compute V1 . 6
 . 0 and V1 . 5 . x to return the current timestamp ,
 in seconds . The output results are different because
   the time zone is considered in the semantics of
 the FROM_UNIXT IME function in Realtime Compute V1 . 6
 . 0 , but not in earlier versions .
```

- Date and Time functions

  The date and time data is expressed and computed as integers within Flink SQL. Date refers to the number of days that have elapsed after 00:00:00 Thursday, 1 January 1970. Time refers to the number of milliseconds that have elapsed after 00:00:00 in the current day of your time zone. If you compute the date and time data in UDFs, note that a time zone offset has been added to the Java object when the internal data is converted to the `java . sql . Date` and `java . sql . Time` types.

List of supported time zones

- Africa/Abidjan
- Africa/Accra
- Africa/Addis_Ababa
- Africa/Algiers
- Africa/Asmara
- Africa/Asmera
- Africa/Bamako
- Africa/Bangui
- Africa/Banjul
- Africa/Bissau
- Africa/Blantyre
- Africa/Brazzaville
- Africa/Bujumbura
- Africa/Cairo

- **Africa/Casablanca**
- **Africa/Ceuta**
- **Africa/Conakry**
- **Africa/Dakar**
- **Africa/Dar_es_Salaam**
- **Africa/Djibouti**
- **Africa/Douala**
- **Africa/El_Aaiun**
- **Africa/Freetown**
- **Africa/Gaborone**
- **Africa/Harare**
- **Africa/Johannesburg**
- **Africa/Juba**
- **Africa/Kampala**
- **Africa/Khartoum**
- **Africa/Kigali**
- **Africa/Kinshasa**
- **Africa/Lagos**
- **Africa/Libreville**
- **Africa/Lome**
- **Africa/Luanda**
- **Africa/Lubumbashi**
- **Africa/Lusaka**
- **Africa/Malabo**
- **Africa/Maputo**
- **Africa/Maseru**
- **Africa/Mbabane**
- **Africa/Mogadishu**
- **Africa/Monrovia**
- **Africa/Nairobi**
- **Africa/Ndjamena**
- **Africa/Niamey**
- **Africa/Nouakchott**

- Africa/Ouagadougou
- Africa/Porto-Novo
- Africa/Sao_Tome
- Africa/Timbuktu
- Africa/Tripoli
- Africa/Tunis
- Africa/Windhoek
- America/Adak
- America/Anchorage
- America/Anguilla
- America/Antigua
- America/Araguaina
- America/Argentina/Buenos_Aires
- America/Argentina/Catamarca
- America/Argentina/ComodRivadavia
- America/Argentina/Cordoba
- America/Argentina/Jujuy
- America/Argentina/La_Rioja
- America/Argentina/Mendoza
- America/Argentina/Rio_Gallegos
- America/Argentina/Salta
- America/Argentina/San_Juan
- America/Argentina/San_Luis
- America/Argentina/Tucuman
- America/Argentina/Ushuaia
- America/Aruba
- America/Asuncion
- America/Atikokan
- America/Atka
- America/Bahia
- America/Bahia_Banderas
- America/Barbados
- America/Belem

- America/Belize
- America/Blanc-Sablon
- America/Boa_Vista
- America/Bogota
- America/Boise
- America/Buenos_Aires
- America/Cambridge_Bay
- America/Campo_Grande
- America/Cancun
- America/Caracas
- America/Catamarca
- America/Cayenne
- America/Cayman
- America/Chicago
- America/Chihuahua
- America/Coral_Harbour
- America/Cordoba
- America/Costa_Rica
- America/Creston
- America/Cuiaba
- America/Curacao
- America/Danmarkshavn
- America/Dawson
- America/Dawson_Creek
- America/Denver
- America/Detroit
- America/Dominica
- America/Edmonton
- America/Eirunepe
- America/El_Salvador
- America/Ensenada
- America/Fort_Nelson
- America/Fort_Wayne

- **America/Fortaleza**
- **America/Glace_Bay**
- **America/Godthab**
- **America/Goose_Bay**
- **America/Grand_Turk**
- **America/Grenada**
- **America/Guadeloupe**
- **America/Guatemala**
- **America/Guayaquil**
- **America/Guyana**
- **America/Halifax**
- **America/Havana**
- **America/Hermosillo**
- **America/Indiana/Indianapolis**
- **America/Indiana/Knox**
- **America/Indiana/Marengo**
- **America/Indiana/Petersburg**
- **America/Indiana/Tell_City**
- **America/Indiana/Vevay**
- **America/Indiana/Vincennes**
- **America/Indiana/Winamac**
- **America/Indianapolis**
- **America/Inuvik**
- **America/Iqaluit**
- **America/Jamaica**
- **America/Jujuy**
- **America/Juneau**
- **America/Kentucky/Louisville**
- **America/Kentucky/Monticello**
- **America/Knox_IN**
- **America/Kralendijk**
- **America/La_Paz**
- **America/Lima**

- **America/Los_Angeles**
- **America/Louisville**
- **America/Lower_Princes**
- **America/Maceio**
- **America/Managua**
- **America/Manaus**
- **America/Marigot**
- **America/Martinique**
- **America/Matamoros**
- **America/Mazatlan**
- **America/Mendoza**
- **America/Menominee**
- **America/Merida**
- **America/Metlakatla**
- **America/Mexico_City**
- **America/Miquelon**
- **America/Moncton**
- **America/Monterrey**
- **America/Montevideo**
- **America/Montreal**
- **America/Montserrat**
- **America/Nassau**
- **America/New_York**
- **America/Nipigon**
- **America/Nome**
- **America/Noronha**
- **America/North_Dakota/Beulah**
- **America/North_Dakota/Center**
- **America/North_Dakota/New_Salem**
- **America/Ojinaga**
- **America/Panama**
- **America/Pangnirtung**
- **America/Paramaribo**

- America/Phoenix
- America/Port-au-Prince
- America/Port_of_Spain
- America/Porto_Acre
- America/Porto_Velho
- America/Puerto_Rico
- America/Punta_Arenas
- America/Rainy_River
- America/Rankin_Inlet
- America/Recife
- America/Regina
- America/Resolute
- America/Rio_Branco
- America/Rosario
- America/Santa_Isabel
- America/Santarem
- America/Santiago
- America/Santo_Domingo
- America/Sao_Paulo
- America/Scoresbysund
- America/Shiprock
- America/Sitka
- America/St_Barthelemy
- America/St_Johns
- America/St_Kitts
- America/St_Lucia
- America/St_Thomas
- America/St_Vincent
- America/Swift_Current
- America/Tegucigalpa
- America/Thule
- America/Thunder_Bay
- America/Tijuana

- **America/Toronto**
- **America/Tortola**
- **America/Vancouver**
- **America/Virgin**
- **America/Whitehorse**
- **America/Winnipeg**
- **America/Yakutat**
- **America/Yellowknife**
- **Antarctica/Casey**
- **Antarctica/Davis**
- **Antarctica/DumontDUrville**
- **Antarctica/Macquarie**
- **Antarctica/Mawson**
- **Antarctica/McMurdo**
- **Antarctica/Palmer**
- **Antarctica/Rothera**
- **Antarctica/South_Pole**
- **Antarctica/Syowa**
- **Antarctica/Troll**
- **Antarctica/Vostok**
- **Arctic/Longyearbyen**
- **Asia/Aden**
- **Asia/Almaty**
- **Asia/Amman**
- **Asia/Anadyr**
- **Asia/Aqtau**
- **Asia/Aqtobe**
- **Asia/Ashgabat**
- **Asia/Ashkhabad**
- **Asia/Atyrau**
- **Asia/Baghdad**
- **Asia/Bahrain**
- **Asia/Baku**

- **Asia/Bangkok**
- **Asia/Barnaul**
- **Asia/Beirut**
- **Asia/Bishkek**
- **Asia/Brunei**
- **Asia/Calcutta**
- **Asia/Chita**
- **Asia/Choibalsan**
- **Asia/Chongqing**
- **Asia/Chungking**
- **Asia/Colombo**
- **Asia/Dacca**
- **Asia/Damascus**
- **Asia/Dhaka**
- **Asia/Dili**
- **Asia/Dubai**
- **Asia/Dushanbe**
- **Asia/Famagusta**
- **Asia/Gaza**
- **Asia/Harbin**
- **Asia/Hebron**
- **Asia/Ho_Chi_Minh**
- **Asia/Hong_Kong**
- **Asia/Hovd**
- **Asia/Irkutsk**
- **Asia/Istanbul**
- **Asia/Jakarta**
- **Asia/Jayapura**
- **Asia/Jerusalem**
- **Asia/Kabul**
- **Asia/Kamchatka**
- **Asia/Karachi**
- **Asia/Kashgar**

- Asia/Kathmandu
- Asia/Katmandu
- Asia/Khandyga
- Asia/Kolkata
- Asia/Krasnoyarsk
- Asia/Kuala_Lumpur
- Asia/Kuching
- Asia/Kuwait
- Asia/Macao
- Asia/Macau
- Asia/Magadan
- Asia/Makassar
- Asia/Manila
- Asia/Muscat
- Asia/Nicosia
- Asia/Novokuznetsk
- Asia/Novosibirsk
- Asia/Omsk
- Asia/Oral
- Asia/Phnom_Penh
- Asia/Pontianak
- Asia/Pyongyang
- Asia/Qatar
- Asia/Qyzylorda
- Asia/Rangoon
- Asia/Riyadh
- Asia/Saigon
- Asia/Sakhalin
- Asia/Samarkand
- Asia/Seoul
- Asia/Shanghai
- Asia/Singapore
- Asia/Srednekolymsk

- **Asia/Taipei**
- **Asia/Tashkent**
- **Asia/Tbilisi**
- **Asia/Tehran**
- **Asia/Tel_Aviv**
- **Asia/Thimbu**
- **Asia/Thimphu**
- **Asia/Tokyo**
- **Asia/Tomsk**
- **Asia/Ujung_Pandang**
- **Asia/Ulaanbaatar**
- **Asia/Ulan_Bator**
- **Asia/Urumqi**
- **Asia/Ust-Nera**
- **Asia/Vientiane**
- **Asia/Vladivostok**
- **Asia/Yakutsk**
- **Asia/Yangon**
- **Asia/Yekaterinburg**
- **Asia/Yerevan**
- **Atlantic/Azores**
- **Atlantic/Bermuda**
- **Atlantic/Canary**
- **Atlantic/Cape_Verde**
- **Atlantic/Faeroe**
- **Atlantic/Faroe**
- **Atlantic/Jan_Mayen**
- **Atlantic/Madeira**
- **Atlantic/Reykjavik**
- **Atlantic/South_Georgia**
- **Atlantic/St_Helena**
- **Atlantic/Stanley**
- **Australia/ACT**

- Australia/Adelaide
- Australia/Brisbane
- Australia/Broken_Hill
- Australia/Canberra
- Australia/Currie
- Australia/Darwin
- Australia/Eucla
- Australia/Hobart
- Australia/LHI
- Australia/Lindeman
- Australia/Lord_Howe
- Australia/Melbourne
- Australia/NSW
- Australia/North
- Australia/Perth
- Australia/Queensland
- Australia/South
- Australia/Sydney
- Australia/Tasmania
- Australia/Victoria
- Australia/West
- Australia/Yancowinna
- Brazil/Acre
- Brazil/DeNoronha
- Brazil/East
- Brazil/West
- CET
- CST6CDT
- Canada/Atlantic
- Canada/Central
- Canada/Eastern
- Canada/Mountain
- Canada/Newfoundland

- Canada/Pacific
- Canada/Saskatchewan
- Canada/Yukon
- Chile/Continental
- Chile/EasterIsland
- Cuba
- EET
- EST5EDT
- Egypt
- Eire
- Etc/GMT
- Etc/GMT+0
- Etc/GMT+1
- Etc/GMT+10
- Etc/GMT+11
- Etc/GMT+12
- Etc/GMT+2
- Etc/GMT+3
- Etc/GMT+4
- Etc/GMT+5
- Etc/GMT+6
- Etc/GMT+7
- Etc/GMT+8
- Etc/GMT+9
- Etc/GMT-0
- Etc/GMT-1
- Etc/GMT-10
- Etc/GMT-11
- Etc/GMT-12
- Etc/GMT-13
- Etc/GMT-14
- Etc/GMT-2
- Etc/GMT-3

- Etc/GMT-4
- Etc/GMT-5
- Etc/GMT-6
- Etc/GMT-7
- Etc/GMT-8
- Etc/GMT-9
- Etc/GMT0
- Etc/Greenwich
- Etc/UCT
- Etc/UTC
- Etc/Universal
- Etc/Zulu
- Europe/Amsterdam
- Europe/Andorra
- Europe/Astrakhan
- Europe/Athens
- Europe/Belfast
- Europe/Belgrade
- Europe/Berlin
- Europe/Bratislava
- Europe/Brussels
- Europe/Bucharest
- Europe/Budapest
- Europe/Busingen
- Europe/Chisinau
- Europe/Copenhagen
- Europe/Dublin
- Europe/Gibraltar
- Europe/Guernsey
- Europe/Helsinki
- Europe/Isle_of_Man
- Europe/Istanbul
- Europe/Jersey

- Europe/Kaliningrad
- Europe/Kiev
- Europe/Kirov
- Europe/Lisbon
- Europe/Ljubljana
- Europe/London
- Europe/Luxembourg
- Europe/Madrid
- Europe/Malta
- Europe/Mariehamn
- Europe/Minsk
- Europe/Monaco
- Europe/Moscow
- Europe/Nicosia
- Europe/Oslo
- Europe/Paris
- Europe/Podgorica
- Europe/Prague
- Europe/Riga
- Europe/Rome
- Europe/Samara
- Europe/San_Marino
- Europe/Sarajevo
- Europe/Saratov
- Europe/Simferopol
- Europe/Skopje
- Europe/Sofia
- Europe/Stockholm
- Europe/Tallinn
- Europe/Tirane
- Europe/Tiraspol
- Europe/Ulyanovsk
- Europe/Uzhgorod

- Europe/Vaduz
- Europe/Vatican
- Europe/Vienna
- Europe/Vilnius
- Europe/Volgograd
- Europe/Warsaw
- Europe/Zagreb
- Europe/Zaporozhye
- Europe/Zurich
- GB
- GB-Eire
- GMT
- GMT0
- Greenwich
- Hongkong
- Iceland
- Indian/Antananarivo
- Indian/Chagos
- Indian/Christmas
- Indian/Cocos
- Indian/Comoro
- Indian/Kerguelen
- Indian/Mahe
- Indian/Maldives
- Indian/Mauritius
- Indian/Mayotte
- Indian/Reunion
- Iran
- Israel
- Jamaica
- Japan
- Kwajalein
- Libya

- MET
- MST7MDT
- Mexico/BajaNorte
- Mexico/BajaSur
- Mexico/General
- NZ
- NZ-CHAT
- Navajo
- PRC
- PST8PDT
- Pacific/Apia
- Pacific/Auckland
- Pacific/Bougainville
- Pacific/Chatham
- Pacific/Chuuk
- Pacific/Easter
- Pacific/Efate
- Pacific/Enderbury
- Pacific/Fakaofo
- Pacific/Fiji
- Pacific/Funafuti
- Pacific/Galapagos
- Pacific/Gambier
- Pacific/Guadalcanal
- Pacific/Guam
- Pacific/Honolulu
- Pacific/Johnston
- Pacific/Kiritimati
- Pacific/Kosrae
- Pacific/Kwajalein
- Pacific/Majuro
- Pacific/Marquesas
- Pacific/Midway

- Pacific/Nauru
- Pacific/Niue
- Pacific/Norfolk
- Pacific/Noumea
- Pacific/Pago_Pago
- Pacific/Palau
- Pacific/Pitcairn
- Pacific/Pohnpei
- Pacific/Ponape
- Pacific/Port_Moresby
- Pacific/Rarotonga
- Pacific/Saipan
- Pacific/Samoa
- Pacific/Tahiti
- Pacific/Tarawa
- Pacific/Tongatapu
- Pacific/Truk
- Pacific/Wake
- Pacific/Wallis
- Pacific/Yap
- Poland
- Portugal
- ROK
- Singapore
- SystemV/AST4
- SystemV/AST4ADT
- SystemV/CST6
- SystemV/CST6CDT
- SystemV/EST5
- SystemV/EST5EDT
- SystemV/HST10
- SystemV/MST7
- SystemV/MST7MDT

- SystemV/PST8
- SystemV/PST8PDT
- SystemV/YST9
- SystemV/YST9YDT
- Turkey
- UCT
- US/Alaska
- US/Aleutian
- US/Arizona
- US/Central
- US/East-Indiana
- US/Eastern
- US/Hawaii
- US/Indiana-Starke
- US/Michigan
- US/Mountain
- US/Pacific
- US/Pacific-New
- US/Samoa
- UTC
- Universal
- W-SU
- WET
- Zulu

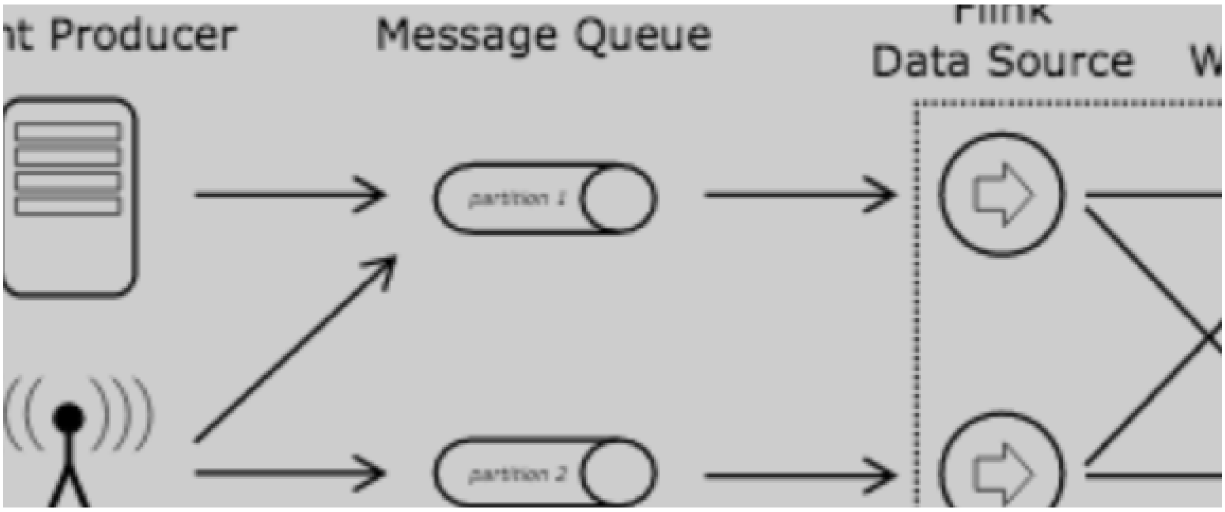
6.3.2 Time attributes

This topic describes the time attributes Event Time and Processing Time supported by Flink SQL.

Flink SQL supports the following two time attributes:

- Event Time: the event time that you provide in the table schema, which is generally the original creation time of data.
- Processing Time: the local system time at which the system processes an event.

The following figure shows the positions of different time attributes in the workflow of Realtime Compute.



As shown in the preceding figure, Ingestion Time and Processing Time are time attributes that are automatically generated by the system for a data record. You do not have control over them. Event Time is a time attribute that comes with a data record . Because of data out-of-order, network jitter, or other reasons, a data record whose Event Time is t1 (corresponding to partition 1) may be processed by Flink later than another data record whose Event Time is t2 (corresponding to partition 2). t2 is later than t1.

Event Time

Event Time is also known as rowtime. The Event Time attribute must be declared in the source table DDL. You can declare a certain field in the source table as an Event Time (rowtime) field. Currently, you can only declare a field of the TIMESTAMP type as a rowtime field. The LONG type will be supported in the future. If you do not have a TIMESTAMP column available, you need to use a computed column to build a TIMESTAMP column based on an existing column.

Because of data out-of-order, network jitter, or other reasons, the order in which data records are received may be different from the order in which they are processed. Therefore, to define a rowtime field, you need to explicitly define a watermark computation method.

The following is an example of Event Time-based window aggregation:

```
CREATE   TABLE   tt_stream (
  a   VARCHAR ,
  b   VARCHAR ,
  c   TIMESTAMP ,
```

```
    WATERMARK   wk1   FOR   c   as   withOffset ( c ,  1000 )  -- The
 watermark   computatio  n   method .
)  WITH  (
    type  = ' sls ',
    topic  = ' yourTopicN  ame ',
    accessId  = ' yourAccess  Id ',
    accessKey  = ' yourAccess  Secret '
);

 CREATE   TABLE   rds_output (
    id   VARCHAR ,
    c   TIMESTAMP ,
    f   TIMESTAMP ,
    cnt   BIGINT
)  WITH  (
    type  = ' rds ',
    url  = ' jdbc : mysql ://**** 3306 / test ',
    tableName  = ' yourTableN  ame ',
    userName  = ' yourUserNa  me ',
    password  = ' yourPasswo  rd '
);

 INSERT   INTO   rds_output
 SELECT   a   AS   id ,
      SESSION_ST  ART ( c ,  INTERVAL  ' 1 '  SECOND )  AS   c ,
      CAST ( SESSION_EN  D ( c ,  INTERVAL  ' 1 '  SECOND )  AS
 TIMESTAMP )  AS   f ,
      COUNT ( a )  AS   cnt
 FROM   tt_stream
 GROUP   BY   SESSION ( c ,  INTERVAL  ' 1 '  SECOND ),  a
```

Processing Time

Processing Time is generated by the system and is not included in your raw data. Therefore, you need to explicitly define a Processing Time column in the declaration of the source table.

```
 filedName   as   PROCTIME ()
```

The following is an example of Processing Time-based window aggregation:

```
 CREATE   TABLE   mq_stream  (
    a   VARCHAR ,
    b   VARCHAR ,
    c   BIGINT ,
    d   AS   PROCTIME ()   --  Explicitly   define   a   Processing
 Time   column   in   the   declaratio  n   of   the   source   table
 .)  WITH  (
    type  = ' mq ',
    topic  = ' yourTopic ',
    accessId  = ' yourAccess  Id ',
    accessKey  = ' yourAccess  Secret '
);

 CREATE   TABLE   rds_output  (
    id   VARCHAR ,
    c   TIMESTAMP ,
    f   TIMESTAMP ,
    cnt   BIGINT )  with  (
    type  = ' rds ',
```

```
    url  = ' yourDateba  seURL ',
    tableName  = ' yourDataba  sTableName ',
    userName  = ' yourUserNa  me ',
    password  = ' yourPasswo  rd '
);

 INSERT   INTO   rds_output
 SELECT   a   AS   id ,
      SESSION_ST  ART ( d ,  INTERVAL  ' 1 '  SECOND )  AS   c ,
      SESSION_EN  D ( d ,  INTERVAL  ' 1 '  SECOND )  AS   f ,
      COUNT ( a )  AS   cnt
 FROM   mq_stream
 GROUP   BY   SESSION ( d ,  INTERVAL  ' 1 '  SECOND ),  a
```

6.3.3 Watermark

Realtime Compute supports window aggregation over data based on time attributes. For a job that runs an Event Time-based window function, the watermark method must be used in the declaration of the source table.

Watermark is a mechanism that is used to measure the Event Time progress. It is a hidden data attribute. Watermark definition is a part of the source table DDL definition. Flink provides the following syntax to define a watermark:

Note:

For more information about time attributes in Realtime Compute, see Time attributes.

```
 WATERMARK  [ watermarkN  ame ]  FOR  < rowtime_fi  eld >  AS
 withOffset (< rowtime_fi  eld >,  offset )
```

Name	Description
watermarkName	The name of the watermark. This parameter is optional.
<rowtime_field>	The column used to generate the watermark. This column is identified as the Event Time column and can be used to define a window in a subsequent query . The parameter value must be a column defined in the table. Only columns of the TIMESTAMP type are supported.

Name	Description
withOffset	The watermark generation policy. The watermark value is generated based on the following formula: `< rowtime_fi eld > - offset`. The first parameter in `withOffset` must be `< rowtime_fi eld >`.
offset	The offset between the watermark value and Event Time value. Unit: millisecond.

Generally, a field in a data record indicates the time when the record is generated. For example, a table has a `rowtime` field of the TIMESTAMP type, and one field value is `1501750584 000 ( 2017 - 08 - 03 08 : 56 : 24 . 000 )`. If you want to define a watermark based on the `rowtime` field and configure a 4-second offset in the watermark policy, add the following definition:

```
WATERMARK FOR rowtime AS withOffset ( rowtime , 4000 )
```

In this example, the watermark time of the data record is `1501750584 000 - 4000 = 1501750580 000 ( 2017 - 08 - 03 08 : 56 : 20 . 000 )`. It means that all data whose timestamp is earlier than `1501750580 000 ( 2017 - 08 - 03 08 : 56 : 20 . 000 )` has arrived.

Note:

- When you use an Event Time-based watermark, note that the `rowtime` field must be of the TIMESTAMP type. Currently, Realtime Compute supports a 13-digit UNIX timestamp in milliseconds. If the rowtime field is of another type or the UNIX timestamp is not 13 digits in length, we recommend that you use a computed column to convert the time.
- The Event Time and Processing Time can only be declared in the source table.

`Summary`

- A watermark indicates that all the events whose timestamp (t') is earlier than the watermark time t (`t '< t`) have occurred. After the watermark time `t` has taken effect, all subsequently received records whose Event Time is earlier than `t` will be discarded. This is the current mechanism used in Realtime Compute.

In the future, Realtime Compute will allow you to change configurations to ensure that later data can also be updated.

- The watermark is particularly important for out-of-order data streams because it maximizes the chance that a window is correctly computed even when the arrival of some events is delayed.
- When an operator has multiple input data streams for parallel processing, the Event Time of the data stream with the shortest time is used as the Event Time at the operator.

6.3.4 Computed column

A computed column can use data from other columns to compute a value for the column to which it belongs. If your source table does not have any columns of the TIMESTAMP type, you can use a computed column to convert a field of another type to the TIMESTAMP type.

Concept

A computed column is a virtual column that is not physically stored in the table. By using expressions, built-in functions, or UDFs, a computed column can use data from other columns to compute a value for the column to which it belongs. The computed column can be used as a common field in Flink SQL.

Usage

Currently, the Event Time column (also known as rowtime column) of the watermark only supports the TIMESTAMP type. The LONG type will be supported in the future. The watermark can only be defined in the DDL of the source table. If your source table does not have any columns of the TIMESTAMP type, you can use a computed column to convert a field of another type to the TIMESTAMP type.

Syntax

```
column_nam e  AS  computed_c olumn_expr ession
```

Examples

The rowtime column of the watermark must be of the TIMESTAMP type. Currently, Realtime Compute supports a 13-digit UNIX timestamp in milliseconds. If the value of the TIME field of DataHub is in microseconds (that is, a 16-digit UNIX timestamp

), you can use a computed column to convert it to a 13-digit UNIX timestamp. The sample code is as follows:

```
CREATE   TABLE   test_strea  m (
   a   INT ,
   b   BIGINT ,
  ` TIME `  BIGINT ,
   ts   AS   TO_TIMESTA  MP ( TIME / 1000 ), --  Use   a   computed
   column   to   convert   a   16 - digit   timestamp   to   a   13 -
 digit   timestamp .
   WATERMARK   FOR   ts   AS   WITHOFFSET ( ts ,  1000 )
)  WITH  (
   type  = ' datahub ',
  ...
);
```

As shown in the preceding example, the `TIME` field in the source table is of the BIGINT type. With the computed column used, the `TIME` field is converted to the `ts` field of the TIMESTAMP type. The `ts` field is then used as the rowtime field of the watermark.

6.4 Data types

6.4.1 Overview

This topic describes data types supported by Realtime Compute and how to convert between different data types.

Data types supported by Realtime Compute

Data type	Description	Value range
VARCHAR	Character string with a changeable length	The maximum storage size is 4 MB.
BOOLEAN	Logical value	Valid values: TRUE, FALSE, and UNKNOWN.
TINYINT	Tiny integer with a length of one byte	Valid values: –128 to 127.
SMALLINT	Small integer with a length of two bytes	Valid values: –32768 to 32767.
INT	Integer with a length of four bytes	Valid values: –2147483648 to 2147483647.

Data type	Description	Value range
BIGINT	Big integer with a length of eight bytes	Valid values: –9223372036854775808 to 9223372036854775807.
FLOAT	Four-byte floating point number	The FLOAT type provides six digits of precision.
DECIMAL	Decimal number	Example: The value for `DECIMAL ( 5 , 2 )` is `123 . 45` .
DOUBLE	Eight-byte floating point number	The DOUBLE type provides 15 decimal digits of precision.
DATE	Date	Example: `DATE ' 1969 - 07 - 20 '`.
TIME	Time	Example: `TIME ' 20 : 17 : 40 '`.
TIMESTAMP	Timestamp representing the date and time	Example: `TIMESTAMP ' 1969 - 07 - 20 20 : 17 : 40 '`.
VARBINARY	Binary data	This type corresponds to the `byte []` type in Java.

Data type conversion

Examples

- Test data

var1(VARCHAR)	big1(BIGINT)
1000	323

- Test statements

```
cast ( var1 as bigint ) as AA ;
cast ( big1 as varchar ) as BB ;
```

- Test results

AA (BIGINT)	BB (VARCHAR)
1000	323

6.4.2 Computational relationships between data types

This topic describes mathematical and logical operations between different data types in Realtime Compute.

Mathematical and logical operations

Operation statement	Description	Data type supported by numeric1 and numeric2	Example
numeric1 + numeric2	Addition	INT, DOUBLE, DECIMAL, and BIGINT	`2 + 4.2`
numeric1 - numeric2	Subtraction		`3 - 5.3`
numeric1 × numeric2	Multiplication		`2 × 4`
numeric1/numeric2	Division		2.4/5
numeric1 > numeric2	Checks whether the first numeric is greater than the second numeric.		`2.4 > 5`
numeric1 < numeric2	Checks whether the first numeric is less than the second numeric.		`2.4 < 5`
numeric1 >= numeric2	Checks whether the first numeric is greater than or equal to the second numeric.		`2.4 >= 5`
numeric1 <= numeric2	Checks whether the first numeric is less than or equal to the second numeric.		`2.4 <= 5`
numeric1 = numeric2	Checks whether the first numeric is equal to the second numeric.	INT, DOUBLE, DECIMAL, BIGINT, and VARCHAR	`‘iphone’ = 5`
numeric1 <> numeric2	Checks whether the first numeric is not equal to the second numeric.		`'iphone' <> 5`

Note:

Data types of numeric1 and numeric2 in an operation statement must be the same.

6.5 Create a data view

You can create a data view in Realtime Compute to simplify the development process.

Syntax

If the computational logic is complex, you can create a data view by defining it in Realtime Compute to simplify the development process.

Note:

The data view only helps describe the computational logic and does not lead to physical storage of data.

```
CREATE  VIEW  viewName [ ( columnName [ ,  columnName ]*
) ]  AS  queryState  ment ;
```

Example 1

```
CREATE  VIEW  LargeOrder  s  ( r ,  t ,  c ,  u )  AS
SELECT
    rowtime ,
    productId ,
    c ,
    units
FROM
    orders ;
INSERT  INTO
    rds_output
SELECT
    r ,
    t ,
    c ,
    u
FROM
    LargeOrder  s ;
```

Example 2

- Test data

a(VARCHAR)	b (BIGINT)	c (TIMESTAMP)
test1	1	1506823820000
test2	1	1506823850000
test1	1	1506823810000

a(VARCHAR)	b (BIGINT)	c (TIMESTAMP)
test2	1	1506823840000
test2	1	1506823870000
test1	1	1506823830000
test2	1	1506823860000

- **Test statements**

```
 CREATE   TABLE   datahub_st  ream  (
    a   VARCHAR ,
    b   BIGINT ,
    c   TIMESTAMP ,
    d   AS   PROCTIME ()
)  WITH  (
   TYPE =' datahub ',
  ...
);
 CREATE   TABLE   rds_output  (
    a   VARCHAR ,
    b   TIMESTAMP ,
    cnt   BIGINT ,
    PRIMARY   KEY ( a )
) WITH (
   TYPE  = ' rds ',
  ...
);
 CREATE   VIEW   rds_view   AS
 SELECT   a ,
    CAST (
       HOP_START ( d ,  INTERVAL  ' 5 '  SECOND ,  INTERVAL  ' 30 '
 SECOND )  AS   TIMESTAMP
   )  AS   cc ,
    SUM ( b )  AS   cnt
 FROM
    datahub_st  ream
 GROUP   BY
     HOP ( d ,  INTERVAL  ' 5 '  SECOND ,  INTERVAL  ' 30 '  SECOND
 ), a ;
 INSERT   INTO
    rds_output
 SELECT
    a ,
    cc ,
    cnt
 FROM
    rds_view
 WHERE   clause
    cnt = 4
```

- **Test results**

a(VARCHAR)	b (TIMESTAMP)	cnt (BIGINT)
test2	`2017 - 11 - 06   16 : 54 : 10`	4

6.6 DDL statements

6.6.1 DDL overview

This topic describes the DDL syntax in Realtime Compute and the issues that require your attention during the DDL use, including field mapping and case sensitivity.

Syntax

```
CREATE   TABLE   tableName
     ( columnName   dataType  [,  columnName   dataType  ]*)
     [  WITH  ( propertyNa  me = propertyVa  lue  [,  propertyNa  me
= propertyVa  lue  ]*) ];
```

Description

Realtime Compute does not provide a built-in data storage feature. Therefore, all DDL statements that involve table creation are reference declarations of external data tables or storage systems. The sample code is as follows:

```
CREATE   TABLE   mq_stream (
  a   VARCHAR ,
  b   VARCAHR ,
  c   VARCAHR
)  WITH  (
  type =' mq ',
  topic =' blink_mq_t  est ',
  accessId =' yourAccess  Id ',
  accessKey =' yourAccess  Key '
);
```

The preceding code does not create a `topic` of the MQ source table in Flink SQL. Instead, it declares a reference to a table named `mq_stream`. For all DML operations related to this MQ topic in downstream operators, the topic name can be replaced with the alias `mq_stream`.

- The declaration of a table is valid only in the current job in Realtime Compute. A Realtime Compute job is generated after a SQL file is submitted. The preceding declaration of the `mq_stream` table is valid only in the current SQL file. Other SQL files in the same Realtime Compute project can also declare the `mq_stream` table.
- According to the standard SQL definition, keywords, table names, and field names in DDL statements are case-insensitive.
- Table and field names must start with a letter or digit, and can contain only letters, digits, and underscores (_).

- Depending on the nature of the upstream plug-in used, DDL declarations may establish the field mappings between the declaration table and external table based on other factors rather than solely on the field names. We recommend that you declare the same field names and number of fields as those in the referenced external tables. This can prevent data errors caused by confusing declarations.

> **Note:**
> If the upstream plug-in supports retrieving values based on keys of key-value pairs, the declaration table and its referenced external table do not need to have the same number of fields. However, the field names must be the same. If the upstream plug-in does not support retrieving values based on keys, the number of fields and their order must be the same between the declaration table and external table.

Field mapping

Two field mapping methods are supported for a declaration table depending on whether the external data source has a schema.

- Sequential mapping

  This method applies to data sources without a schema, for example, MQ. These data sources are usually unstructured storage systems that do not support retrieving values based on keys. We recommend that you customize field names in DDL SQL statements and use the same field types and number of fields in the declaration table as those in the external table.

  A sample record in MQ is provided as follows:

```
asavfa , sddd32 , sdfdsv
```

  Specify MQ field names according to the naming conventions.

```
CREATE   TABLE   mq_stream (
  a   VARCHAR ,
  b   VARCHAR ,
  c   VARCAHR
)  WITH  (
  type =' mq ',
  topic =' blink_mq_t  est ',
  accessId =' yourAccess  Id ',
  accessKey =' yourAccess  Secret '
```

```
);
```

- Name mapping

  This method applies to data sources with a schema. These data storage systems define field names and field types at the table storage level, and support retrieving values based on keys. We recommend that you use the same schema definition in Flink SQL declarations as that of the external data storage system. Specifically, the names, number, and order of fields in the declaration table must be the same as those in the external table.

  Note:
  If field names in the external data storage system are case-sensitive (for example, Table Store), enclose the case-sensitive field names in backticks (`). In the DDL syntax, field names in the declaration table must be the same as those in the external table.

Case-sensitivity

In the standard SQL definition, fields are case-insensitive. For example, the following two statements have the same meaning:

```
create table stream_res ult (
    name varchar ,
    value varchar
);
```

```
create table STREAM_RES ULT (
    NAME varchar ,
    VALUE varchar
);
```

However, most external data sources referenced by Realtime Compute are case-sensitive. For example, Table Store is case-sensitive. The following statement shows how to define the uppercase `NAME` field for Table Store:

```
create table STREAM_RES ULT (
   ` NAME ` varchar ,
   ` VALUE ` varchar
);
```

In all subsequent DML statements, enclose the field in backticks (`) whenever it is referenced, as shown in the following statement:

```
INSERT INTO table_a
SELECT
 ` NAME `,
```

```
  ` VALUE `
FROM
  table_b ;
```

Related topics

For more information about how to create source tables, dimension tables, and result tables in Realtime Compute, see the following topics:

- Source table overview
- Result table overview
- Dimension table overview

6.6.2 Create a source table

6.6.2.1 Source table overview

In Realtime Compute, source tables are streaming data storage tables. Streaming data storage provides the input to drive the running of Realtime Compute. Therefore, at least one streaming data storage table is required for each Realtime Compute job.

Syntax

```
  CREATE   TABLE   tableName
     ( columnName   dataType  [,  columnName   dataType  ]*)
     [  WITH  ( propertyNa  me = propertyVa  lue  [,  propertyNa  me
= propertyVa  lue  ]*) ];
```

Examples

```
CREATE   TABLE   metaq_stre  am (
 x   VARCHAR ,
 y   VARCHAR ,
 z   VARCHAR
)  WITH  (
 type =' mq ',
 topic ='< yourTopicN  ame >',
 endpoint ='< yourEndpoi  nt >',
 pullInterv  alMs =' 1000 ',
 accessId ='< yourAccess  Id >',
 accessKey ='< yourAccess  Secret >',
 startMessa  geOffset =' 1000 ',
 consumerGr  oup =' yourConsum  erGroup ',
 fieldDelim  iter ='|'
```

```
);
```

Obtain attribute fields of a source table

- Syntax for obtaining attribute fields of a source table

Realtime Compute provides the keyword `HEADER` in the DDL statement of a source table. You can use this keyword to obtain attribute fields of the source table.

```
CREATE TABLE sourcetabl e
(
 ` timestamp `   VARCHAR   HEADER ,
   name          VARCHAR ,
   MsgID         VARCHAR
) WITH (
      type =' sls '
);
```

The `` ` timestamp ` `` field in the preceding example is defined as `HEADER` to read values from data attribute fields. This field can be used as a common field subsequently.

Note:
Different types of source tables (such as DataHub, Log Service, and MQ source tables) have different default attribute fields. Some source tables also support custom attribute fields. For more information, see the documentation of the corresponding source table type.

- Example for obtaining attribute fields of a source table

The following describes an example of how to obtain attribute fields of a Log Service source table. Currently, Log Service supports the following three attribute fields by default.

Field	Description
`__source__`	The message source.
`__topic__`	The message topic.
`__timestam p__`	The time when the log was generated.

Note:

To obtain attribute fields, you need to first declare the fields according to the normal logic. Then add the keyword `HEADER` to the end of the type declaration.

Example:

- Test data

```
__topic__ :   ens_altar_  flow
        result :  {" MsgID ":" ems0a "," Version ":" 0 . 0 . 1
"}
```

- Test statements

```
 CREATE   TABLE   sls_log  (
   __topic__     VARCHAR   HEADER ,
   result        VARCHAR
) WITH (
   type  = ' sls '
);
 CREATE   TABLE   sls_out  (
   name        varchar ,
   MsgID       varchar ,
   Version     varchar
) WITH (
   type  =' RDS '
);
 INSERT   INTO   sls_out
 SELECT
 __topic__ ,
 JSON_VALUE ( result ,'$.  MsgID '),
 JSON_VALUE ( result ,'$.  Version ')
 FROM
 sls_log
```

- Test results

name(VARCHAT)	MsgID(VARCHAT)	Version(VARCHAT)
ens_altar_flow	ems0a	`0 . 0 . 1`

Source table with window functions

Realtime Compute supports window aggregation over data based on two time attributes: Event Time and Processing Time. For Realtime Compute jobs that involve window functions, the Watermarkand Computed columnare required in the declaration of the source table. For more information about the time attribute-based aggregation operations in Realtime Compute, see Time attributes.

Supported source table types

Realtime Compute allows multiple types of source tables to be created. For more information, see the following topics:

- Create a Log Service source table
- Create a MQ source table
- Create a Kafka source table

6.6.2.2 Create a Log Service source table

This topic describes how to create a Log Service source table in Realtime Compute. It also describes the attribute fields, WITH parameters, and field type mapping involved in the table creation process.

Introduction to Log Service

Log Service is an all-in-one real-time data logging service that Alibaba Group has developed and tested in many big data scenarios. Based on Log Service, you can quickly finish tasks such as data ingestion, consumption, delivery, query, and analysis without any extra development work. This can help you improve O&M and operational efficiency, and build up the capability to process large amounts of logs in the data technology era. Log Service is a streaming data storage system. Realtime Compute supports creating a Log Service table as the source table. In Log Service, each data record is in a format similar to JSON. An example is as follows:

```
{
    " a ":  1000 ,
    " b ":  1234 ,
    " c ": " li "
}
```

Realtime Compute needs to define the following DDL (in which sls indicates Log Service):

```
 create   table   sls_stream (
   a   int ,
   b   int ,
   c   VARCHAR
)  with  (
   type  =' sls ',
   endPoint  =' yourEndpoi  nt ',
   accessId  =' yourAccess  Id ',
   accessKey  =' yourAccess  Key ',
   startTime  = ' yourStartT  ime ',
   project  =' yourProjec  tName ',
   logStore  =' yourLogSto  reName ',
   consumerGr  oup  =' yourConsum  erGroupNam  e '
);
```

Attribute fields

Currently, Flink SQL supports obtaining the following three attribute fields of Log Service by default, and writing other custom fields.

Field	Description
`__source__`	The message source.
`__topic__`	The message topic.
`__timestam p__`	The time when the log was generated.

Notes on attribute fields

To obtain attribute fields, you need to first declare the fields according to the normal logic. Then add the keyword `HEADER` to the end of the type declaration. Example:

- Test data

```
      __topic__ :   ens_altar_  flow
         result :  {" MsgID ":" ems0a "," Version ":" 0 . 0 . 1 "}
```

- Test statements

```
 CREATE   TABLE   sls_log  (
   __topic__    varchar   HEADER ,
   result       varchar
)
 WITH
(
   type  = ' sls '
);

 CREATE   TABLE   sls_out  (
   name       varchar ,
   MsgID      varchar ,
   Version    varchar
)
 WITH
(
   type  =' RDS '
);

 INSERT   INTO   sls_out
 SELECT
 __topic__ ,
 JSON_VALUE ( result ,'$.  MsgID '),
 JSON_VALUE ( result ,'$.  Version ')
 FROM
 sls_log
```

- Test results

name(VARCAHR)	MsgID(VARCAHR)	Version(VARCAHR)
ens_altar_flow	ems0a	0.0.1

WITH parameters

Name	Description	Remarks
endPoint	The consumption endpoint information.	Service endpoint
accessId	The AccessKey ID of Log Service.	None
accessKey	The AccessKey Secret of Log Service.	None
project	The Log Service project to be accessed.	None
logStore	The LogStore in the Log Service project.	None
consumerGroup	The name of the consumer group.	You can customize the consumer group name (with no fixed format).
startTime	The start time that the log is consumed.	None
heartBeatIntervalMills	The heartbeat interval of the consumption client.	Optional. Default value: 10 seconds.
maxRetryTimes	The maximum number of read retries.	Optional. Default value: 5.
batchGetSize	The number of log items read at a time in a log group.	Optional. Default value: 10.

Name	Description	Remarks
lengthCheck	The policy for checking the number of fields in a single line.	Optional. Default value: NONE. Valid values: NONE, SKIP, EXCEPTION, and PAD. · SKIP: skips a data record when the number of fields in the record does not match the specified number. · EXCEPTION: throws an exception when the number of fields in the record does not match the specified number. · PAD: pads fields in sequence. Pad with null when a field does not exist.
columnErrorDebug	Indicates whether to enable debugging. If this parameter is set to true, logs about parsing exceptions are displayed.	Optional. Default value: false.

Note:

- Log Service does not support data of the MAP type.
- Fields can be unordered. However, we recommend that you use the same field order as that defined in the referenced table.
- If the input data source is in JSON format, define a delimiter and use a built-in function to analyze JSON_VALUE. Otherwise, the parsing fails and the following error information is generated:

```
2017 - 12 - 25   15 : 24 : 43 , 467   WARN  [ Topology - 0  ( 1
/ 1 )]  com . alibaba . blink . streaming . connectors . common .
source . parse . DefaultSou  rceCollect  or  -  Field   missing
error ,  table   column   number :  3 ,  data   column   number :
3 ,  data   filed   number :  1 ,  data : [{" lg_order_c  ode ":"
LP00000005 "," activity_c  ode ":" TEST_CODE1 "," occur_time ":"
2017 - 12 - 10   00 : 00 : 01 "}]
```

- The batchGetSize value must not exceed 1000. Otherwise, an error is returned.
- The batchGetSize parameter specifies the number of log items read at a time in a log group. If both the size of a single log item and the batchGetSize value are too large, frequent GC may be triggered. To avoid this, you need to set the parameters to smaller values.

Field type mapping

The following table lists the mapping between Log Service field types and Realtime Compute field types. We recommend that you use the mapping in the DDL declaration .

Log Service field type	Realtime Compute field type
STRING	VARCHAR

6.6.2.3 Create a MQ source table

This topic describes how to create a MQ source table in Realtime Compute. It also describes the CSV format, WITH parameters, and field type mapping involved in the table creation process.

Introduction to MQ

MQ is a professional message middleware that Alibaba Cloud has developed and put into commercial use. It is a core product for the enterprise-level Internet architecture (Aliware). Based on the high-availability distributed cluster technology, MQ provides a complete set of high-performance messaging cloud services, including publishing /subscription, message tracing, resource statistics, message scheduling (delaying), and monitoring and alerting. They implement all asynchronous decoupling functions in distributed computing scenarios. Realtime Compute supports creating an MQ table as the source table. The sample code is as follows.

Examples

```
 create   table   mq_stream (
  x   varchar ,
  y   varchar ,
  z   varchar
)  with  (
  type =' mq ',
  topic =' yourTopicN  ame ',
  endpoint =' yourEndpoi  nt ',
  pullInterv  alMs =' 1000 ',
  accessId =' yourAccess  Id ',
  accessKey =' yourAccess  Secret ',
```

```
  startMessa  geOffset =' 1000 ',
  consumerGr  oup =' yourConsum  erGroup ',
  fieldDelim  iter ='|'
);
```

Note:

MQ uses an unstructured storage format that does not force you to define a data schema. The data schema is specified at the business layer. Currently, Realtime Compute supports messages in CSV and binary formats.

CSV format

Assume that you have an MQ message in the following CSV format:

```
 1 , name , male
 2 , name , female
```

Note:

An MQ message can contain zero to multiple data records separated with `\ n` .

In a Realtime Compute job, the DDL statement used to declare an MQ source table is as follows:

```
 create   table   mq_stream (
  x   varchar ,
  y   varchar ,
  z   varchar
)  with  (
  type =' mq ',
  topic =' yourTopicN  ame ',
  endpoint =' yourEndpoi  nt ',
  pullInterv  alMs =' 1000 ',
  accessId =' yourAccess  Id ',
  accessKey =' yourAccess  Secret ',
  startMessa  geOffset =' 1000 ',
  consumerGr  oup =' yourConsum  erGroup ',
  fieldDelim  iter ='|'
);
```

Binary format

The sample code for the binary format is as follows:

```
 create   table   source_tab  le  (
   mess   varbinary
)  with  (
   type  = ' mq ',
   endpoint  = ' yourEndpoi  nt ',
   pullInterv  alMs =' 500 ',
   accessId =' yourAccess  Id ',
   accessKey =' yourAccess  Secret ',
   topic  = ' yourTopicN  ame ',
   consumerGr  oup =' yourConsum  erGroup '
);
```

```
 create   table   out_table  (
   commodity   varchar
) with (
   type =' print '
);

 INSERT   INTO   out_table
 SELECT
   cast ( mess   as   varchar )
 FROM   source_tab  le
```

Note:

- The `cast ( mess   as   varbinary )` statement is supported in Realtime Compute V2.0 and later. If your Realtime Compute version is earlier than V2.0, upgrade it first.
- The VARBINARY type can be passed in only once.

WITH parameters

Name	Description	Remarks
topic	The topic name.	None
endPoint	The endpoint.	• Intranet access to Alibaba Cloud public cloud (Alibaba Cloud classic network or VPC): The endpoint for China (Hangzhou), China (Shanghai), China (Qingdao), China (Beijing), China (Shenzhen), and Hong Kong is `onsaddr - internal . aliyun . com : 8080`. • Internet access to Alibaba Cloud public cloud: The endpoint is `http :// onsaddr - internet . aliyun . com / rocketmq / nsaddr4cli  ent - internet`.

Name	Description	Remarks
accessId	The AccessKey ID.	None
accessKey	The AccessKey Secret.	None
consumerGroup	The consumer group that subscribes to the topic.	None
pullIntervalMs	The pull interval.	Unit: millisecond.
startTime	The start time of message consumption.	Optional.
startMessageOffset	The start offset of messages.	Optional. If this parameter is set, the loading preferentially starts from the checkpoint determined by the offset.
tag	The subscription tag.	Optional.
lineDelimiter	The line delimiter used to parse message blocks.	Optional. Default value: `\ n`.
fieldDelimiter	The field delimiter.	Optional. Default value: `\ u0001`. This value indicates that `\ u0001` is used as the delimiter in read-only mode and `^ A` is used as the delimiter in edit mode. \u0001 is invisible in read-only mode.
encoding	The encoding format.	Optional. Default value: `UTF - 8`.

Name	Description	Remarks
lengthCheck	The policy for checking the number of fields in a single line.	Optional. Default value: NONE. Valid values: NONE, SKIP, EXCEPTION, and PAD. · SKIP: skips a data record when the number of fields in the record does not match the specified number. · EXCEPTION: throws an exception when the number of fields in the record does not match the specified number. · PAD: pads fields in sequence. Pad with null when a field does not exist.
columnErrorDebug	Indicates whether to enable debugging.	Optional. Default value: false. If this parameter is set to true, logs about parsing exceptions are displayed.

Field type mapping

MQ field type	Recommended Realtime Compute field type
STRING	VARCHAR

6.6.2.4 Create a Kafka source table

This topic describes how to create a Kafka source table in Realtime Compute. It also describes the Kafka version mapping and Kafka message parsing examples.

Note:

This topic applies only to Realtime Compute deployed in exclusive mode.

Introduction to Kafka source tables

Kafka source tables are implemented based on Kafka community edition. The data parsing process of a Kafka source table is Kafka source table -> UDTF -> Realtime Compute -> sink. All data read from Kafka is in the VARBINARY (binary) format. You need to use a UDTF to parse VARBINARY data into formatted data.

DDL definition

The DDL definition of the Kafka source table must be the same as that in the following SQL statement. The five fields in the table must use the following order.

```
-- Define the source table . Note that the DDL fields
   of the Kafka source table must be the same as
 those in the following example . WITH parameters are
 modifiable .
 create table kafka_stre am (
   messageKey VARBINARY ,
  ` message `  VARBINARY ,
   topic      VARCHAR ,
  ` partition `  INT ,
  ` offset `     BIGINT
) with (
   type =' kafka010 ',
   topic = '< yourTopicN ame >',
  ` group . id ` = '< yourGroupI d >',
  ...
);
```

WITH parameters

- General configuration

Name	Description	Remarks
type	The Kafka version name.	Required. Valid values : Kafka08, Kafka09, Kafka010, and Kafka011. For the version mapping , see Mapping between Kafka version names and version numbers.
topic	The topic read.	None
topicPattern	The expression for reading multiple topics at a time.	None

Name	Description	Remarks
startupMode	The start offset.	- EARLIEST: reads data from the earliest Kafka partition. - Group_OFFSETS: reads data by group. - LATEST: reads data from the latest Kafka checkpoint. - TIMESTAMP: reads data from the specified checkpoint. This value is supported by Kafka010 and Kafka011.
partitionDiscoveryIntervalMS	The interval for checking whether any new partition is generated. Unit: millisecond.	Default value: 60000, indicating 1 minute.
extraConfig	The additional kafkaConsumer configuration items.	Optional. You can set configuration items that are required in special occasions but are not included in the optional configuration items.

- Required configuration for Kafka08

Name	Description	Remarks
group.id	The name of the consumer group.	The ID of the consumer group.
zookeeper.connect	The ZooKeeper URL.	The ZooKeeper connection ID.

- (Optional) Key
  - **consumer.id**
  - **socket.timeout.ms**
  - **fetch.message.max.bytes**
  - **num.consumer.fetchers**
  - **auto.commit.enable**
  - **auto.commit.interval.ms**
  - **queued.max.message.chunks**
  - **rebalance.max.retries**
  - **fetch.min.bytes**
  - **fetch.wait.max.ms**
  - **rebalance.backoff.ms**
  - **refresh.leader.backoff.ms**
  - **auto.offset.reset**
  - **consumer.timeout.ms**
  - **exclude.internal.topics**
  - **partition.assignment.strategy**
  - **client.id**
  - **zookeeper.session.timeout.ms**
  - **zookeeper.connection.timeout.ms**
  - **zookeeper.sync.time.ms**
  - **offsets.storage**
  - **offsets.channel.backoff.ms**
  - **offsets.channel.socket.timeout.ms**
  - **offsets.commit.max.retries**
  - **dual.commit.enabled**
  - **partition.assignment.strategy**
  - **socket.receive.buffer.bytes**
  - **fetch.min.bytes**

- Required configuration for Kafka09, Kafka010, and Kafka011

Name	Description	Remarks
group.id	The name of the consumer group.	The ID of the consumer group.
bootstrap.servers	The Kafka cluster address .	None

For more information about other optional configuration items, see Kafka official documentation.

- Kafka09
- Kafka010
- Kafka011

When you need to configure any items, add the corresponding parameters in the WITH section of the DDL statement. For example, when you configure the SASL logon, you need to add the `security.protocol`, `sasl.mechanism`, and `sasl.jaas.config` parameters. The sample code is as follows:

```
create   table   kafka_stre  am (
  messageKey   varbinary ,
  ` message `  varbinary ,
  topic   varchar ,
  ` partition ` int ,
  ` offset ` bigint
)  with  (
  type  =' kafka010 ',
  topic  = '< yourTopicN  ame >',
  ` group . id ` = '< yourGroupI  d >',
  ...,
  ` security . protocol `= SASL_PLAIN  TEXT ,
  ` sasl . mechanism `= PLAIN ,
  ` sasl . jaas . config `=' org . apache . kafka . common .
 security . plain . PlainLogin  Module   required   username ="
 USERNAME "  password =" PASSWORD ";'--  Enter   the   actual
 username   and   password ,  respective  ly .
);
```

Mapping between Kafka version names and version numbers

Kafka version name	Kafka version number
Kafka08	V0.8.22
Kafka09	V0.9.0.1
Kafka010	V0.10.2.1
Kafka011	V0.11.0.2

Kafka message parsing examples

- Example 1:
  - Scenario

    Assume that you want to compute Kafka data and write the output data to RDS. Kafka data is stored in JSON format and must be computed by Realtime Compute . The message format is as follows:

```
{
  " name ":" Alice ",
  " age ": 13 ,
  " grade ":" A "
}
```

    The entire computing process is Kafka source table -> UDTF -> Realtime Compute -> RDS sink.

  - Sample code

    ■ SQL

```
--  Define   a   UDTF   that   parses   Kafka   messages .
 CREATE   FUNCTION   kafkapaser   AS  ' com . alibaba .
 kafkaUDTF ';

--  Define   the   source   table .  Note   that   the   DDL
 fields   of   the   Kafka   source   table   must   be   the
   same   as   those   in   the   following   example .  WITH
 parameters   are   modifiable .
 create   table   kafka_src  (
     messageKey    VARBINARY ,
    ` message `    VARBINARY ,
     topic         VARCHAR ,
    ` partition `  INT ,
    ` offset `     BIGINT
)  WITH  (
     type  = ' kafka010 ',    --  The   Kafka   source
 type ,  which   is   strongly   related   to   the   Kafka
 version .  For   the   version   mapping ,  see   Mapping
 between   Kafka   version   names   and   version   numbers .
     topic  = ' test_kafka  _topic ',
    ` group . id ` = ' test_kafka  _consumer_  group ',
     bootstrap . servers  = ' ip1 : port1 , ip2 : port2 , ip3 :
 port3 '
);
 create   table   rds_sink  (
   name         VARCHAR ,
   age          INT ,
   grade        VARCHAR ,
   updateTime   TIMESTAMP
)  WITH (
  type =' rds ',
  url =' jdbc : mysql :// localhost : 3306 / test ',
  tableName =' test4 ',
  userName =' test ',
  password ='< yourDataba  sePassword >'
```

```
);

-- Use the UDTF to parse binary data into
 formatted data .
 CREATE VIEW input_view (
     name ,
     age ,
     grade ,
     updateTime
) AS
 SELECT
     T . name ,
     T . age ,
     T . grade ,
     T . updateTime
 from
     kafka_src as S ,
     LATERAL TABLE ( kafkapaser (` message `)) as T (
         name ,
         age ,
         grade ,
         updateTime
    );

-- Compute the formatted data and write the
 output data to RDS .
 insert into rds_sink
   SELECT
       name ,
       age ,
       grade ,
       updateTime
   from input_view ;
```

- **UDTF**

Note:

The Flink version of the following Maven dependencies is determined by the Realtime Compute version of your job. For example, when you run a job in Realtime Compute V2.2.4, the Flink version of Maven dependencies is blink-2.2.4-SNAPSHOT. For more information about the download addresses of the dependency packages, see the Build the development environment section of the UDX overview topic. Maven dependencies:

```
    < dependenci es >
        < dependency >
            < groupId > org . apache . flink </ groupId >
            < artifactId > flink - core </ artifactId >
            < version > blink - 2 . 2 . 4 - SNAPSHOT </ version
 >
            < scope > provided </ scope >
        </ dependency >
        < dependency >
            < groupId > org . apache . flink </ groupId >
            < artifactId > flink - streaming - java_2 . 11 </
 artifactId >
```

```
            < version > blink - 2 . 2 . 4 - SNAPSHOT </ version
 >
            < scope > provided </ scope >
        </ dependency >
        < dependency >
            < groupId > org . apache . flink </ groupId >
            < artifactId > flink - table_2 . 11 </ artifactId
 >
            < version > blink - 2 . 2 . 4 - SNAPSHOT </ version
 >
            < scope > provided </ scope >
        </ dependency >
        < dependency >
            < groupId > com . alibaba </ groupId >
            < artifactId > fastjson </ artifactId >
            < version > 1 . 2 . 9 </ version >
        </ dependency >
    </ dependenci  es >
```

```
package   com . alibaba ;

import   com . alibaba . fastjson . JSONObject ;
import   org . apache . flink . table . functions .
TableFunct  ion ;
import   org . apache . flink . table . types . DataType ;
import   org . apache . flink . table . types . DataTypes ;
import   org . apache . flink . types . Row ;

import   java . io . Unsupporte  dEncodingE  xception ;
import   java . sql . Timestamp ;

public   class   kafkaUDTF   extends   TableFunct  ion < Row >
{
    public   void   eval ( byte []  message ) {
        try  {
           /*  input   message  :
               {
                 " name ":" Alice ",
                 " age ": 13  ,
                 " grade ":" A ",
                 " updateTime ": 1544173862
               }
           */
            String   msg  =  new   String ( message , " UTF -
8 ");
            try  {
                JSONObject   data  =  JSON . parseObjec  t (
msg );
                if  ( data  ! =  null ) {
                    String   name  =  data . getString (" name
") ==  null  ? " null " :  data . getString (" name ");
                    Integer   age  =  data . getInteger (" age
") ==  null  ?  0  :  data . getInteger (" age ");
                    String   grade  =  data . getString ("
grade ") ==  null  ? " null " :  data . getString (" grade ");
                    Timestamp   updateTime  =  data .
getTimesta  mp (" updateTime ");

                    Row   row  =  new   Row ( 4 );
                    row . setField ( 0 ,  name );
                    row . setField ( 1 ,  age );
                    row . setField ( 2 ,  grade );
```

```
                    row . setField ( 3 , updateTime  );

                    System . out . println (" Kafka   message
   str  ==>" +  row . toString ());
                    collect ( row );
                }
            }  catch  ( ClassCastE  xception   e ) {
                System . out . println (" Input   data
 format   error .  Input   data  " +  msg  + " is   not   json
   string ");
            }
        }  catch  ( Unsupporte  dEncodingE  xception   e ) {
            e . printStack  Trace ();
        }
    }
    @ Override
    //  If   the   return   value   is   declared   as   Row
 ,  you   must   reload   the   getResultT  ype   method   to
   explicitly   inform   the   system   of   the   returned
 field   types .
     public   DataType   getResultT  ype ( Object []  arguments
 ,  Class []  argTypes ) {
         return   DataTypes . createRowT  ype ( DataTypes .
 STRING ,  DataTypes . INT ,  DataTypes . STRING ,  DataTypes .
 TIMESTAMP );
    }
}
```

- Example 2:
  - Scenario

    Window computation is required for data read from Kafka. In the current design of Realtime Compute, to perform operations related to windows such as tumbling windows and sliding windows, you must define a watermark in the DDL statement of the source table. The Kafka source table is special. To perform window operations based on the Event Time of the message field in Kafka, you must first use a UDX to parse the Event Time from the message field, and then define a watermark. You need to use a computed column for the Kafka source table. Assume that data written to Kafka is as follows:

    `2018 - 11 - 11   00 : 00 : 00 | 1 | Anna | female` . The entire computing process is Kafka source table -> UDTF -> Realtime Compute -> RDS sink.
  - Sample code
    - SQL

```
-- Define  a  UDTF  that  parses  Kafka  messages .
 CREATE   FUNCTION   kafkapaser   AS  ' com . alibaba .
 kafkaUDTF ';
 CREATE   FUNCTION   kafkaUDF   AS  ' com . alibaba . kafkaUDF
 ';
```

```
-- Define the source table . Note that the DDL
 fields of the Kafka source table must be the
  same as those in the following example . WITH
 parameters are modifiable .
 create table kafka_src (
     messageKey    VARBINARY ,
    ` message `    VARBINARY ,
     topic         VARCHAR ,
    ` partition `  INT ,
    ` offset `     BIGINT ,
     ctime  AS  TO_TIMESTA MP ( kafkaUDF (` message `)),
 -- Define a computed column . The computed column
   can be understood as a placeholde r . It
 does not exist in the source table . Data of
   the computed column can be computed at the
 downstream operator . Note that the data type of
   the computed column must be TIMESTAMP if you
 want to define a watermark .
     watermark for ` ctime ` as withoffset (` ctime `, 0
 ) -- Define a watermark based on the computed
 column .
) WITH (
     type = ' kafka010 ',   -- The Kafka source
 type , which is strongly related to the Kafka
 version . For the version mapping , see Mapping
 between Kafka version names and version numbers .
     topic = ' test_kafka _topic ',
    ` group . id ` = ' test_kafka _consumer_ group ',
     bootstrap . servers = ' ip1 : port1 , ip2 : port2 , ip3 :
 port3 '
);

 create table rds_sink (
   name         VARCHAR ,
   age          INT ,
   grade        VARCHAR ,
   updateTime   TIMESTAMP
) WITH (
  type =' rds ',
  url =' jdbc : mysql :// localhost : 3306 / test ',
  tableName =' test4 ',
  userName =' test ',
  password ='< yourDataba sePassword >'
);

-- Use the UDTF to parse binary data into
 formatted data .
 CREATE VIEW input_view (
     name ,
     age ,
     grade ,
     updateTime
) AS
 SELECT
     COUNT (*) as cnt ,
     T . ctime ,
     T . order ,
     T . name ,
     T . sex
 from
     kafka_src as S ,
     LATERAL TABLE ( kafkapaser (` message `)) as T (
         ctime ,
```

```
          order ,
          name ,
          sex
     )
 Group   BY   T . sex ,
          TUMBLE ( ctime ,  INTERVAL  ' 1 '  MINUTE );

--  Compute   the   output   data   from   input_view .
 CREATE   VIEW   view2  (
     cnt ,
     sex
)  AS
 SELECT
     COUNT (*)  as   cnt ,
     T . sex
 from
     input_view
 Group   BY   sex ,  TUMBLE ( ctime ,  INTERVAL  ' 1 '  MINUTE
 );



--  Compute   the   formatted   data   and   write   the
 output   data   to   RDS .
 insert   into   rds_sink
   SELECT
       cnt , sex
   from   view2 ;
```

- UDF&UDTF

Note:

The Flink version of the following Maven dependencies is determined by the Realtime Compute version of your job. For example, when you run a job in Realtime Compute V2.2.4, the Flink version of Maven dependencies is blink-2.2.4-SNAPSHOT. For more information about the download addresses of the dependency packages, see the Build the development environment section of the UDX overview topic. Maven dependencies:

```
   < dependenci  es >
         < dependency >
             < groupId > org . apache . flink </ groupId >
             < artifactId > flink - core </ artifactId >
             < version > blink - 2 . 2 . 4 - SNAPSHOT </ version
 >
             < scope > provided </ scope >
         </ dependency >
         < dependency >
             < groupId > org . apache . flink </ groupId >
             < artifactId > flink - streaming - java_2 . 11 </
 artifactId >
             < version > blink - 2 . 2 . 4 - SNAPSHOT </ version
 >
             < scope > provided </ scope >
         </ dependency >
         < dependency >
             < groupId > org . apache . flink </ groupId >
```

```
            < artifactId > flink - table_2 . 11 </ artifactId >
            < version > blink - 2 . 2 . 4 - SNAPSHOT </ version
>
            < scope > provided </ scope >
        </ dependency >
        < dependency >
            < groupId > com . alibaba </ groupId >
            < artifactId > fastjson </ artifactId >
            < version > 1 . 2 . 9 </ version >
        </ dependency >
    </ dependenci  es >
```

■ **UDTF**

```
package   com . alibaba ;

import   com . alibaba . fastjson . JSONObject ;
import   org . apache . flink . table . functions .
TableFunct  ion ;
import   org . apache . flink . table . types . DataType ;
import   org . apache . flink . table . types . DataTypes ;
import   org . apache . flink . types . Row ;

import   java . io . Unsupporte  dEncodingE  xception ;

/**
   The   following   example   shows   how   to   parse   the
   input   JSON   strings   of   Kafka   and   generate
formatted   data .
**/
public   class   kafkaUDTF   extends   TableFunct  ion < Row
> {

    public   void   eval ( byte []  message ) {
        try  {
         //  Read   a   binary   data   record   and
convert   it   to   the   STRING   type .
            String   msg  =  new   String ( message , " UTF
- 8 ");

               //  Extract   fields   from   the   JSON
object .
                    String   ctime  =  Timestamp . valueOf (
data . split ('\\|')[ 0 ]);
                    String   order  =  data . split ('\\|')[
1 ];
                    String   name  =  data . split ('\\|')[
2 ];
                    String   sex  =  data . split ('\\|')[ 3
];

                   //  Place   the   formatted   fields
into   the   Row ()  object   for   output .
                    Row   row  =  new   Row ( 4 );
                    row . setField ( 0 ,  ctime );
                    row . setField ( 1 ,  age );
                    row . setField ( 2 ,  grade );
                    row . setField ( 3 ,  updateTime );


                    System . out . println (" Kafka
message   str  ==>" +  row . toString ());
```

```
                    // Generate a row .
                     collect ( row );


             } catch ( ClassCastE xception e ) {
                 System . out . println (" Input data
 format error . Input data " + msg + " is not
 json string ");
             }


         } catch ( Unsupporte dEncodingE xception e ) {
             e . printStack Trace ();
         }

     }

     @ Override
     // If the return value is declared as Row
 , you must reload the getResultT ype method to
  explicitly inform the system of the returned
 field types .
     // Define the field type of the Row ()
 object for output .
      public DataType getResultT ype ( Object []
 arguments , Class [] argTypes ) {
          return DataTypes . createRowT ype ( DataTypes
 . TIMESTAMP , DataTypes . STRING , DataTypes . Integer ,
 DataTypes . STRING , DataTypes . STRING );
     }

 }
```

■ **UDF**

```
package com . alibaba ;
package com . hjc . test . blink . sql . udx ;
import org . apache . flink . table . functions .
FunctionCo ntext ;
import org . apache . flink . table . functions .
ScalarFunc tion ;


public class KafkaUDF extends ScalarFunc tion {
   // The open method is optional .
   // You need to run import org . apache . flink
. table . functions . FunctionCo ntext ;.

    public String eval ( byte [] message ) {

        // Read a binary data record and
convert it to the STRING type .
        String msg = new String ( message , " UTF - 8
");
        return msg . split ('\\|')[ 0 ];
   }
    public long eval ( String b , String c ) {
        return eval ( b ) + eval ( c );
   }
   // The close method is optional .
   @ Override
    public void close () {
```

```
        }
}
```

Self-built Kafka

- Examples

```
 create   table   kafka_stre  am (
   messageKey   VARBINARY ,
  ` message `   VARBINARY ,
   topic   varchar ,
  ` partition `  int ,
  ` offset `  bigint
)  with  (
   type  =' kafka011 ',
   topic  = ' kafka_01 ',
  ` group . id ` = ' CID_blink ',
   bootstrap . servers  = ' 192 . 168 . 0 . 251 :****'
);
```

- WITH parameters

  For more information about WITH parameters of self-built Kafka, see descriptions of WITH parameters in this topic. Note that you must enter the address and port number of your self-built Kafka for the `bootstrap . servers` parameter.

> Note:
> Only Realtime Compute V2.2.6 and later support displaying metric information such as TPS and RPS of Alibaba Cloud Kafka or your self-built Kafka.

6.6.3 Create a result table

6.6.3.1 Result table overview

Realtime Compute uses the `CREATE  TABLE` statement to define the format of the output data and to define how data is written to the destination data storage system.

Data can be written to the destination data storage system in two modes: append and update.

- Append: If the output data is stored in a log system, a message system, or an RDS database with the primary key undefined, the output data is written to the data storage system in append mode. In this case, the original data in the data storage system is not modified.

- Update: If the output data is stored in a database with the primary key declared, such as an RDS or HBase database with a primary key, the output data is written in the following two ways:
  - If a data record queried based on the primary key does not exist in the destination database, the data record is inserted into the database.
  - If a data record queried based on the primary key exists in the database, the data record is updated based on the primary key.

Syntax

```
CREATE   TABLE   tableName
   ( columnName   dataType  [,  columnName   dataType  ]*)
   [  WITH  ( propertyNa  me = propertyVa  lue  [,  propertyNa  me =
propertyVa  lue  ]*) ];
```

Examples

```
create   table   rds_output (
id   int ,
len   int ,
content   VARCHAR ,
primary   key ( id )
)  with  (
type =' rds ',
url ='< yourDataba  seURL >',
tableName ='< yourTableN  ame >',
userName ='< yourDataba  seUserName >',
password ='< yourDataba  sePassword >'
);
```

6.6.3.2 Create a Log Service result table

This topic describes how to create a Log Service result table in Realtime Compute.

Introduction to Log Service

As an all-in-one real-time data logging service, Log Service allows you to quickly finish tasks such as data ingestion, consumption, delivery, query, and analysis without any extra development work. This can help you improve O&M and operational efficiency, and build up the capability to process large amounts of logs in the data technology era.

DDL definition

Realtime Compute supports creating a Log Service table as the result table.

```
create   table   sls_stream (
 name   varchar ,
 age   BIGINT ,
```

```
  birthday   BIGINT
) with (
  type =' sls ',
  endPoint =‘< yourEndpoi  nt >’,
  accessId =‘< yourAccess  Id >’,
  accessKey =‘< yourAccess  Secret >’,
  project =‘< yourProjec  tName >’,
  logStore ='< yourLogsto  reName >'
);
```

Note:

We recommend that you use the data storage registration method to connect to Log Service. For more information, see Log Service.

WITH parameters

Name	Description	Remarks
endPoint	The endpoint.	Service endpoint
project	The project name.	None
topic	The table name of the topic.	None
accessId	The AccessKey ID.	None
accessKey	The AccessKey Secret.	None
mode	The write mode.	Optional. Default value: `random`. If this parameter is set to `partition`, data is written by partition.
partitionColumn	The partition column.	This parameter is required if `mode` is set to `partition`.
topic	The Log Service topic.	Optional. This parameter is left empty by default.
source	The source of the log. For example, you can set this parameter to the IP address of the machine where the log was generated.	Optional. This parameter is left empty by default.

6.6.3.3 Create a MQ result table

This topic describes how to create an MQ result table in Realtime Compute.

Introduction to MQ

MQ is a professional message middleware that Alibaba Cloud has developed and put into commercial use. It is a core product for the enterprise-level Internet architecture (Aliware). Based on the high-availability distributed cluster technology, MQ provides a complete set of high-performance messaging cloud services, including publishing/subscription, message tracing, resource statistics, message scheduling (delaying), and monitoring and alerting. Realtime Compute supports creating an MQ table as the result table. The sample code is as follows:

```
 CREATE   TABLE   stream_tes  t_hotline_  agent  (
 id   INTEGER ,
 len   BIGINT ,
 content   VARCHAR
)  WITH  (
 type =' mq ',
  endPoint =‘< yourEndpoi  nt >’,
  accessId =‘< yourAccess  Id >’,
  accessKey =‘< yourAccess  Secret >’,
  project =‘< yourProjec  tName >’,
  producerGr  oup ='< yourGroupN  ame >',
  tag ='< yourTagNam  e >',
  encoding =' utf - 8 ',
  fieldDelim  iter =',',
  retryTimes =' 5 ',
  sleepTimeM  s =' 500 '
);
```

CSV format

```
 CREATE   TABLE   stream_tes  t_hotline_  agent  (
 id   INTEGER ,
 len   BIGINT ,
 content   varchar
)  WITH  (
 type =' mq ',
  endPoint =‘< yourEndpoi  nt >’,
  accessId =‘< yourAccess  Id >’,
  accessKey =‘< yourAccess  Secret >’,
  project =‘< yourProjec  tName >’,
  producerGr  oup ='< yourGroupN  ame >',
  tag ='< yourTagNam  e >',
  encoding =' utf - 8 ',
  fieldDelim  iter =',',
  retryTimes =' 5 ',
  sleepTimeM  s =' 500 '
```

```
);
```

Binary format

The sample code for the binary format is as follows:

```
 create   table   source_tab  le  (
   commodity   VARCHAR
) with (
   type =' random '
);

 create   table   result_tab  le  (
   mess   varbinary
)  with  (
   type  = ' mq ',
   topic  = '< yourTopicN  ame >',
   endPoint =‘< yourEndpoi  nt >’,
   accessId =‘< yourAccess  Id >’,
   accessKey =‘< yourAccess  Secret >’,
   producerGr  oup ='< yourGroupN  ame >'
);

 INSERT   INTO   result_tab  le
 SELECT
 cast ( substring ( commodity , 0 , 5 )  as   varbinary )  as   mess

 FROM   source_tab  le
```

Note:

- The `cast ( varchar   as   varbinary )` statement is supported in Realtime Compute V2.0 and later. If your Realtime Compute version is earlier than V2.0, upgrade it first.
- The VARBINARY type can be passed in only once.

WITH parameters

Name	Description	Remarks
topic	The name of the MQ queue .	None

Name	Description	Remarks
endpoint	The endpoint.	· Intranet access to Alibaba Cloud public cloud (Alibaba Cloud classic network or VPC): The endpoint for China (Hangzhou), China (Shanghai), China (Qingdao), China (Beijing), China (Shenzhen), and Hong Kong is `onsaddr - internal . aliyun . com : 8080`. · Internet access to Alibaba Cloud public cloud: The endpoint is `http :// onsaddr - internet . aliyun . com / rocketmq / nsaddr4cli ent - internet`.
accessID	The AccessKey ID.	None
accessKey	The AccessKey Secret.	None
producerGroup	The name of the producer group.	None
tag	The message tag.	Optional. This parameter is left empty by default.
fieldDelimiter	The field delimiter.	Optional. Default value: `\ u0001`. This value indicates that `\ u0001` is used as the delimiter in read-only mode and `^ A` is used as the delimiter in edit mode. `\ u0001` is invisible in read-only mode.
encoding	The encoding format.	Optional. Default value: `UTF - 8`.

Name	Description	Remarks
retryTimes	The maximum number of write retries allowed.	Optional. Default value: 10.
sleepTimeMs	The retry interval, in milliseconds.	Optional. Default value: 1000.

6.6.3.4 Create a Table Store result table

This topic describes how to create a Table Store result table in Realtime Compute. It also describes the mapping between Table Store field types and Realtime Compute field types.

Introduction to Table Store

Table Store is a distributed NoSQL data storage service built on Alibaba Cloud's Apsara system. It is designed to provide 99.99% high availability and 99.999999999% data reliability. By using data sharding and load balancing technologies, Table Store implements seamless scale-out in terms of data scale and access parallelism. It also supports the storage of and real-time access to large amounts of structured data.

DDL definition

Realtime Compute supports creating a Table Store table as the result table. The sample code is as follows:

```
CREATE   TABLE   stream_tes  t_hotline_  agent  (
  name   varchar ,
  age   BIGINT ,
  birthday   BIGINT ,
  primary   key ( name , age )
)  WITH  (
  type =' ots ',
  instanceNa  me ='< yourInstan  ceName >',
  tableName ='< yourTableN  ame >',
  accessId ='< yourAccess  Id >',
  accessKey ='< yourAccess  Secret >',
  endPoint ='< yourEndpoi  nt >',
  valueColum  ns =' birthday '
);
```

Note:

- We recommend that you use the data storage registration method to connect to Table Store. For more information, see Register Table Store resources.
- The value of the valueColumns parameter must not be a declared primary key. It can be any field other than the primary key.

WITH parameters

Name	Description	Remarks
instanceName	The instance name.	None
tableName	The table name.	None
endPoint	The endpoint for accessing the instance.	See Endpoint.
accessId	The AccessKey ID.	None
accessKey	The AccessKey Secret.	None
valueColumns	The column names of fields to be inserted. Separate multiple fields with commas (`,`).	The format of inserting two fields is `' ID , NAME '`.
bufferSize	The buffer size after deduplication.	Optional. Default value: 5000, indicating that the output starts once there are 5,000 input data records.
batchWriteTimeoutMs	The write timeout period. Unit: millisecond.	Optional. Default value: 5000. This value indicates that if no data is written to Table Store within 5000 milliseconds (5 seconds), all buffered data is written.
batchSize	The number of data records written at a time.	Optional. Default value: 100.
retryIntervalMs	The retry interval, in milliseconds.	Optional. Default value: 1000.
maxRetryTimes	The maximum number of retries allowed.	Optional. Default value: 100.
ignoreDelete	Indicates whether to ignore the delete operation.	Default value: false.

Field type mapping

Table Store field type	Realtime Compute field type
INTEGER	BIGINT

Table Store field type	Realtime Compute field type
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

Note:
The Table Store result table must have a declared `primary key`. The output data is appended to the Table Store result table in Update mode.

6.6.3.5 Create a RDS or DRDS result table

This topic describes how to create a Relational Database Service (RDS) or a Distributed Relational Database Service (DRDS) result table in Realtime Compute. It also describes the mapping between RDS or DRDS field types and Realtime Compute field types.

RDS

ApsaraDB for RDS is a stable and reliable online database service that supports auto scaling. Based on Apsara Distributed File System and high-performance storage, RDS supports a variety of engines such as MySQL, SQL Server, PostgreSQL, and Postgres Plus Advanced Server (PPAS, which is highly compatible with Oracle). It provides a comprehensive set of solutions for diversified requirements, such as disaster tolerance, data backup, data recovery, monitoring, and data migration, to resolve database O&M difficulties.

Note:
RDS and DRDS plug-ins use the same `WITH` parameters. The following WITH parameter descriptions apply to both plug-ins. When you use an RDS or a DRDS result table, make sure that an actual table exists in RDS or DRDS.

DDL definition

Realtime Compute supports creating an RDS or a DRDS table as the result table. Currently, only MySQL tables are supported. The sample code is as follows:

```
create   table   rds_output (
 id   int ,
 len   int ,
 content   VARCHAR ,
 primary   key ( id , len )
) with  (
```

```
  type =' rds ',
  url =' yourDataba  seURL ',
  tableName =' yourDataba  seTable ',
  userName =' yourDataba  seUserName ',
  password =' yourDataba  sePassword '
);
```

Note:

- Realtime Compute can write data to an RDS or a DRDS result table. To do so, Realtime Compute concatenates a SQL statement based on each row of the result data, and then executes the SQL statement against the destination database. If you want Realtime Compute to write multiple data records at a time, you need to append `? rewriteBat chedStatem ents = true` to the end of the URL. Otherwise, the performance will deteriorate.
- RDS for MySQL supports an auto-increment primary key. If you want Realtime Compute to support an auto-increment primary key during data writes, do not declare the auto-increment field in the DDL statement. For example, ID is an auto-increment field. If this field is not declared in the Realtime Compute DDL, the database automatically adds the ID field when a row of data is written to the database.
- We recommend that you use the data storage registration method to connect to RDS. For more information, see Register ApsaraDB for RDS resources.

WITH parameters

Name	Description	Remarks
url	The database connection address.	For more information about the address, see: · Apply for an Internet IP address for RDS
tableName	The table name.	None
userName	The logon username.	None
password	The logon password.	None
maxRetryTimes	The maximum number of insertion retries.	Optional. Default value: 3.
batchSize	The number of data records written at a time.	Optional. Default value: 50.

Name	Description	Remarks
bufferSize	The buffer size after deduplication. This parameter takes effect only when the primary key is specified.	Optional. Default value : 1. This value indicates that the output starts once there is one input data record. The default value is changed to 1000 in V1.4.1.
flushIntervalMs	The write timeout period. Unit: millisecond.	Optional. Default value: 5000. This value indicates that if no data is written within 5000 milliseconds (5 seconds), all buffered data is written.
excludeUpdateColumns	The column to be excluded when Realtime Compute updates data records with the same key value.	Optional. This parameter is left empty by default. The primary key fields are excluded by default.
ignoreDelete	Indicates whether to ignore the delete operation .	Default value: false.
partitionBy	Before writing data to the sink operator, Realtime Compute performs the hash operation based on the parameter value . The data then flows to the corresponding hash operator.	Optional. This parameter is left empty by default.

Field type mapping

RDS field type	Realtime Compute field type
TEXT	VARCHAR
BYTE	VARCHAR
INTEGER	INT
LONG	BIGINT
DOUBLE	DOUBLE
DATE	VARCHAR
DATETIME	VARCHAR

RDS field type	Realtime Compute field type
TIMESTAMP	VARCHAR
TIME	VARCHAR
YEAR	VARCHAR
FLOAT	FLOAT
DECIMAL	DECIMAL
CHAR	VARCHAR

JDBC connection parameters

Name	Description	Default value	Minimum version required
useUnicode	Indicates whether to use the Unicode character set. If the characterEncoding parameter is set to gb2312 or gbk, this parameter must be set to true.	false	1.1g
characterEncoding	The character encoding used when useUnicode is set to true. For example, you can set this parameter to gb2312 or gbk.	false	1.1g
autoReconnect	Indicates whether to automatica lly re-establish a connection when the database connection is unexpectedly interrupted.	false	1.1

Name	Description	Default value	Minimum version required
autoReconnectForPools	Indicates whether to use the reconnection policy applicable a database connection pool.	false	3.1.3
failOverReadOnly	Indicates whether to set the connection to be read-only after the database is automatically reconnected.	true	3.0.12
maxReconnects	The maximum number of reconnection retries allowed when autoReconnect is set to true.	3	1.1
initialTimeout	The interval between two reconnection retries when autoReconnect is set to true. Unit: second.	2	1.1
connectTimeout	The timeout period for establishing a socket connection with the database server. Unit: millisecond. The value 0 indicates no timeout, which applies to JDK V1.4 and later.	0	3.0.1

Name	Description	Default value	Minimum version required
socketTimeout	The timeout period for a socket operation (read or write). Unit: millisecond. The value 0 indicates no timeout.	0	3.0.1

FAQ

- Q: Is a new record generated or is data updated by primary key when the Realtime Compute result data is written to an RDS table?

  A: If the primary key is defined in the DDL, the `insert into on duplicate key update` statement is used to update records. That means, for a data record, if its primary key does not exist in the table, the data record is inserted into the table. If its primary key exists, the data record is updated. If the primary key is not declared in the DDL, the `insert into` statement is used to append the data record to the table.
- Q: Is there anything to pay attention to when I perform the GROUP BY operation by using the unique index of the RDS table?

  A: An RDS table has only one auto-increment primary key, which cannot be declared as a primary key in a Realtime Compute job. If you want to use the unique indexes in the RDS table to execute GROUP BY, declare these unique indexes for the primary key in the job.

6.6.3.6 Create a MaxCompute result table

This topic describes how to create a MaxCompute result table in Realtime Compute and FAQs during the creation process.

DDL definition

Realtime Compute supports creating a MaxCompute table as the result table. The sample code is as follows:

```
create   table   odps_outpu  t (
    id   INT ,
    user_name   VARCHAR ,
    content   VARCHAR
)  with  (
```

```
      type  = ' odps ',
      endPoint  = ' http :// service . cn . maxcompute . aliyun - inc .
  com / api ',
      project  = '< projectNam  e >',
      tableName  = '< tableName >',
      accessId  = '< yourAccess  KeyId >',
      accessKey  = '< yourAccess  KeySecret >',
     ` partition ` = ' ds = 2018 ****'
);
```

WITH parameters

Name	Description	Remarks
endPoint	The MaxCompute endpoint .	Required. For more information, see Configure Endpoint.
project	The MaxCompute project name.	Required.
tableName	The table name.	Required.
accessId	The AccessKey ID.	Required
accessKey	The AccessKey Secret.	Required
partition	The partition name.	Optional. This parameter must be specified for a partition table. To view detailed partition information, log on to Data Map. For example, if the partition name of a table is `ds = 20180905`, you can specify the parameter as `` ` partition ` = ' ds = 20180905 ' ``. Use commas (,) to separate multiple levels of partitions, for example, `` ` partition ` = ' ds = 20180912 , dt = xxxyyy ' ``.

Note:

Realtime Compute writes the cached data to MaxCompute every time when it creates a checkpoint.

FAQ

1. Q: Does a Realtime Compute job clear the result table before it writes data to the MaxCompute sink that is in Stream mode when `isOverwrite` is set to `true`?

   A: The `isOverwrite` parameter is set to `true` by `default`. That is, Realtime Compute clears the result table and result data before it writes data to the sink. Every time Realtime Compute starts a job or resumes a paused job, it clears data of the existing result table or the result partition before it writes data. Data loss may occur when data is cleared after a paused Realtime Compute job is resumed.
2. Q: What is the mapping between MaxCompute data types and Realtime Compute data types?

   A: The following table lists the mapping.

MaxCompute data type	Realtime Compute data type
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
STRING	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY

Note:

- Currently, MaxCompute connectors do not support converting other MaxCompute data types.

- VARCHAR is a new data type of MaxCompute. It is not currently supported by Realtime Compute connectors. We recommend that you set the VARCHAR type in the MaxCompute schema to the STRING type.

6.6.3.7 Create an HBase result table

This topic describes how to create an HBase result table in Realtime Compute.

Note:

This topic applies only to Realtime Compute deployed in exclusive mode.

DDL definition

Realtime Compute supports creating an HBase table as the result table. The sample code is as follows:

```
 create   table   liuxd_user  _behavior_  test_front  (
     row_key   varchar ,
     from_topic   varchar ,
     origin_dat  a   varchar ,
     record_cre  ate_time   varchar ,
     PRIMARY   KEY  ( row_key )
)  with  (
     type  = ' cloudhbase ',
     zkQuorum  = ' 2 '
     columnFami  ly  = '< yourColumn  Family >',
     tableName  = '< yourTableN  ame >',
     batchSize  = ' 500 '
)
```

Note:

- You can define multiple fields for the `primary key`. Multiple fields are concatenated by the `rowkeyDeli miter` into a `row_key`. The default row key delimiter is a colon (:).
- When you perform an undo operation in HBase, if a column stores multiple versions of a value, all versions of the value are deleted.

WITH parameters

Name	Description	Remarks
zkQuorum	The ZooKeeper address of the HBase cluster.	You can find the configuration of `hbase.zookeeper.quorum` in the *hbase - site . xml* file.

Name	Description	Remarks
zkNodeParent	The path of the cluster on the ZooKeeper server.	You can find the configuration of `hbase.zookeeper.quorum` in the *hbase - site . xml* file.
tableName	The name of the HBase table.	None
userName	The logon username.	None
password	The logon password.	None
partitionBy	When this parameter is set to true, Realtime Compute partitions the table based on joinKey and distributes data to the JOIN operator. This helps improve the cache hit ratio.	Optional. Default value: false.
shuffleEmptyKey	When this parameter is set to false, Realtime Compute distributes empty keys in the input data to the same JOIN operator. When this parameter is set to true, Realtime Compute distributes empty keys in the input data to random JOIN operators.	We recommend that you set this parameter to true.
columnFamily	The name of the column family.	Currently, Realtime Compute only supports inserting data of the same column family.
maxRetryTimes	The maximum number of insertion retries.	Optional. Default value: 10.
bufferSize	The maximum number of data records allowed before deduplication.	Default value: 5000.
batchSize	The number of data records written at a time.	Optional. Default value: 100.

Name	Description	Remarks
flushIntervalMs	The maximum length of insertion time.	Optional. Default value: 2000.
writePkValue	Indicates whether to write the primary key value.	Optional. Default value: false.
stringWriteMod	Indicates whether to insert all data records as the STRING type.	Optional. Default value: false.
rowkeyDelimiter	The delimiter of row keys.	Optional. The default delimiter is a colon (:).
isDynamicTable	Indicates whether the table is a dynamic table.	Optional. Default value: false.

6.6.3.8 Create an ElasticSearch result table

This topic describes how to create an ElasticSearch result table in Realtime Compute.

Note:

This topic applies only to Realtime Compute deployed in exclusive mode.

DDL definition

REST API is used to implement ElasticSearch result tables. Theoretically, REST API is compatible with all ElasticSearch versions. Realtime Compute supports creating an ElasticSearch table as the result table. The sample code is as follows:

```
 create   table   es_stream_  sink (
   field1   long ,
   field2   varbinary ,
   field3   varchar ,
   PRIMARY   KEY ( field1 )
)  with  (
   type  =' elasticsea  rch ',
   endPoint  = '< yourEndPoi  nt >',
   accessId  = '< yourAccess  Id >',
   accessKey  = '< yourAccess  Secret >',
   index  = '< yourIndex >',
   typeName  = '< yourTypeNa  me >'
);
```

Note:

ElasticSearch supports data update based on the primary key. You can define only one field for the primary key.

- When the primary key is specified, the document IDs are the values of the primary key field.
- When the primary key is not specified, the document IDs are generated at random. For more information, see Index API.
- In full update mode, later documents overwrite earlier documents instead of updating fields in earlier documents.
- In incremental update mode, the corresponding fields are updated based on the passed-in field values.
- All updates use the upsert semantics by default, indicating insert or update.

WITH parameters

General configuration

Name	Description	Default value	Required
endPoint	The server address, for example: `http :// 127 . 0 . 0 . 1 : 9211 .`	None	Yes
accessId	The AccessKey ID.	None	Yes
accessKey	The AccessKey Secret.	None	Yes
index	The index name, which is similar to the database name.	None	Yes
typeName	The type name, which is similar to the database table name.	None	Yes
bufferSize	The number of data records written at a time.	1000	No
maxRetryTimes	The maximum number of retries allowed in the case of an exception.	30	No
timeout	The read timeout period. Unit: millisecond.	600000	No

Name	Description	Default value	Required
discovery	Indicates whether to enable operator discovery. If operator discovery is enabled, the client refreshes the server list every 5 minutes.	false	No
compression	Indicates whether to use GZIP to compress request bodies.	true	No
multiThread	Indicates whether to enable multithreading for JestClient.	true	No
ignoreWriteError	Indicates whether to ignore write exceptions.	false	No
Settings	The settings used to create indexes.	None	No
updateMode	The update mode after the primary key is specified.	full Note: · full: full data overwriting · inc: incremental data update	No

Dynamic index-related WITH parameters

Name	Description	Default value	Required
dynamicIndex	Indicates whether to enable dynamic indexes.	false(true/false)	No

indexField	The field name of the index.	None	This parameter is required only when dynamicInd ex is set to true. It supports only the TIMESTAMP (in seconds), DATE , and LONG data types.
indexInterval	The interval between index changes.	d	This parameter is required only when dynamicIndex is set to true. Valid values: · d: Day · m: Month · w: Week

Note:

1. When dynamicIndex is set to true, the `index` name in basic settings is used as the unified alias for indexes created subsequently. The alias and indexes have an one-to-many relationship.
2. Actual index names corresponding to different values of `indexInter val` are as follows:
   - d -> Alias + "yyyyMMdd"
   - m -> Alias + "yyyyMM"
   - w -> Alias + "yyyyMMW"
3. You can use Index API to modify an actual index, but you can only `get` the alias. If you want to modify the alias, see Index Aliases.

6.6.3.9 Create an HiTSDB result table

This topic describes how to create an High Performance Time Series Database (HiTSDB) result table in Realtime Compute.

Introduction to HiTSDB

HiTSDB is a high performance, cost-effective, stable, and reliable online time series database service. HiTSDB provides a range of functions such as efficient read and write, storage with a high compression ratio, time series data interpolation, and aggregation. HiTSDB has been widely used in diversified industrial scenarios, such as Internet of Things (IoT) monitoring systems, enterprise-level energy management systems (EMSs), production safety monitoring systems, and electric power detection systems.

DDL definition

Realtime Compute supports creating a HiTSDB table as the result table. The sample code is as follows:

> **Note:**
> To reference an HiTSDB result table in Realtime Compute, you need to configure the data storage whitelist. For more information, see Configure the data storage whitelist.

```
 CREATE   TABLE   stream_tes  t_hitsdb  (
     metric   varchar ,
     timestamp   INTEGER ,
     value   DOUBLE ,
     tagk1   varchar
)  WITH  (
     type =' hitsdb ',
     host ='< yourHostNa  me >',
     virtualDom  ainSwitch   = ' ture ',
     httpConnec  tionPool  = ' 20 ',
     batchPutSi  ze   = ' 1000 '
);
```

The default format for table creation is described as follows:

- Zeroth column: metric(VARCHAR).
- First column: timestamp(INTEGER), in seconds.
- Second column: value(DOUBLE)
- Third column: tag(VARCHAR)
- Fourth to Nth columns: Use the field name as the tag key and field value as the tag value.

> **Note:**
> You must declare `metric` , `timestamp` , and `value` . Their data types must be the same as those in HiTSDB. Multiple `tag` columns are allowed.

WITH parameters

Name	Description	Remarks
host	The IP address or virtual IP address.	Enter the host address of the registered instance. For more information, see Connect to the instance.
port	The port number.	Default value: 8242.
virtualDomainSwitch	Indicates whether to use the VIP server.	Default value: false. If you need to use the VIP server, set this parameter to true.
httpConnectionPool	The maximum number of HTTP connections in the HTTP connection pool.	Default value: 10.
httpCompress	Indicates whether to use GZIP to compress request bodies.	Default value: false, indicating no compression.
httpConnectTimeout	The HTTP connection timeout period.	Default value: 0.
ioThreadCount	The number of I/O threads.	Default value: 1.
batchPutBufferSize	The buffer size.	Default value: 10000.
batchPutRetryCount	The maximum number of write retries allowed.	Default value: 3.
batchPutSize	The data volume submitted at a time.	By default, 500 data points are submitted at a time.
batchPutTimeLimit	The wait time in the buffer. Unit: millisecond.	Default value: 200.
batchPutConsumerThreadCount	The number of serialized threads.	Default value: 1.

FAQ

Q: An error occurred during failover, indicating that the LONG type cannot be converted to the INT type. Why?

A: Realtime Compute versions earlier than V2.2.5 only support the INT type. Realtime Compute V2.2.5 and later support the BIGINT type.

6.6.3.10 Create a Kafka result table.

This topic describes how to create a Kafka result table in Realtime Compute.

Note:

This topic applies only to Realtime Compute deployed in exclusive mode.

DDL definition

The DDL statement used to create a Kafka result table is defined as follows:

```
 create   table   sink_kafka  (
     messageKey   VARBINARY ,
    ` message `   VARBINARY ,
     PRIMARY   KEY  ( messageKey )
)  with  (
     type  = ' kafka010 ',
     topic  = '< yourTopicN  ame >',
     bootstrap . servers  = '< yourServer  Address >'
);
```

Note:

- You must explicitly declare the PRIMARY KEY (messageKey) when you create a Kafka result table.
- Only Realtime Compute V2.2.6 and later support displaying metric information such as TPS and RPS of Alibaba Cloud Kafka or your self-built Kafka.

WITH parameters

General configuration

Name	Description	Remarks
type	The Kafka version name.	Required. Valid values: Kafka08, Kafka09, Kafka010, and Kafka011. For the version mapping, see Mapping between Kafka version names and version numbers.
topic	The topic to be written.	The topic name.

- Required configuration
  - Required configuration for Kafka08

Name	Description	Remarks
zookeeper.connect	The ZooKeeper URL.	The ZooKeeper connection ID.

  - Required configuration for Kafka09, Kafka010, and Kafka011

Name	Description	Remarks
bootstrap.servers	The Kafka cluster address.	None

- **Optional configuration**
  - `consumer . id`
  - `socket . timeout . ms`
  - `fetch . message . max . bytes`
  - `num . consumer . fetchers`
  - `auto . commit . enable`
  - `auto . commit . interval . ms`
  - `queued . max . message . chunks`
  - `rebalance . max . retries`
  - `fetch . min . bytes`
  - `fetch . wait . max . ms`
  - `rebalance . backoff . ms`
  - `refresh . leader . backoff . ms`
  - `auto . offset . reset`
  - `consumer . timeout . ms`
  - `exclude . internal . topics`
  - `partition . assignment . strategy`
  - `client . id`
  - `zookeeper . session . timeout . ms`
  - `zookeeper . connection . timeout . ms`
  - `zookeeper . sync . time . ms`
  - `offsets . storage`
  - `offsets . channel . backoff . ms`
  - `offsets . channel . socket . timeout . ms`
  - `offsets . commit . max . retries`
  - `dual . commit . enabled`
  - `partition . assignment . strategy`
  - `socket . receive . buffer . bytes`
  - `fetch . min . bytes`

Note:

For more information about other optional configuration items, see Kafka official documentation.

- Kafka09
- Kafka010
- Kafka011

Mapping between Kafka version names and version numbers

Kafka version name	Kafka version number
Kafka08	V0.8.22
Kafka09	V0.9.0.1
Kafka010	V0.10.2.1
Kafka011	V0.11.0.2

Examples

```
 create   table   datahub_in  put  (
 id   VARCHAR ,
 nm   VARCHAR
)  with  (
 type  = ' datahub '
);

 create   table   sink_kafka  (
  messageKey   VARBINARY ,
` message `  VARBINARY ,
  PRIMARY   KEY  ( messageKey )
)  with  (
     type  = ' kafka010 ',
     topic  = '< yourTopicN  ame >',
     bootstrap . servers  = '< yourServer  Address >'
);

 INSERT   INTO
     sink_kafka
 SELECT
    cast ( id   as   VARBINARY )  as   messageKey ,
    cast ( nm   as   VARBINARY )  as  ` message `
 FROM
```

```
    datahub_in  put ;
```

6.6.3.11 Create a HybridDB for MySQL result table

This topic describes how to create a HybridDB for MySQL result table in Realtime Compute.

Introduction to HybridDB for MySQL

HybridDB for MySQL is a relational hybrid transaction/analytical processing (HTAP) database that supports both online transaction processing (OLTP) and online analytical processing (OLAP). HTAP combines transaction processing (TP) and analytical processing (AP) to ensure real-time data processing and analysis.

HybridDB for MySQL is compatible with MySQL syntax and functions, and supports common Oracle analytic functions. HybridDB for MySQL is fully compatible with the TPC-H and TPC-DS benchmarks.

DDL definition

Realtime Compute supports creating a HybridDB for MySQL table as the result table. The sample code is as follows:

```
 create   table   petadata_o  utput (
  id   INT ,
  len   INT ,
  content   VARCHAR ,
  primary   key ( id , len )
)  with  (
  type =' petaData ',
  url ='< yourDataba  seURL >',
  tableName ='< yourTableN  ame >',
  userName ='< yourDataba  seUserName >',
  password ='< yourDataba  sePassword >'
);
```

Note:

- Realtime Compute supports writing data to a HybridDB for MySQL result table. To do so, Realtime Compute concatenates a SQL statement based on each row of the result data, and then executes the SQL statement against the destination database.
- The default value of bufferSize is 1000. If the bufferSize threshold (buffer hashmap size) is reached, data writing is triggered. Therefore, you need to set bufferSize together with batchSize. You can set bufferSize and batchSize to the same value.
- The value of the batchSize parameter cannot be too large. We recommend you set the value as `4096`.

WITH parameters

Name	Description	Remarks
url	The address.	Switch network type
tableName	The table name.	None
userName	The logon username.	None
password	The logon password.	None
maxRetryTimes	The maximum number of insertion retries.	Optional. Default value: 3.
batchSize	The number of data records written at a time.	Optional. Default value: 1000.
bufferSize	The buffer size after deduplication. This parameter takes effect only when the primary key is specified.	Optional.
flushIntervalMs	The write timeout period. Unit: millisecond.	Optional. Default value: 3000. This value indicates that if no data is written within 3 seconds, all buffered data is written.
ignoreDelete	Indicates whether to ignore the delete operation.	Default value: false.

6.6.3.12 Create a custom result table

This topic describes how to create a custom result table in Realtime Compute.

Note:

This topic applies only to Realtime Compute deployed in exclusive mode.

To meet diversified output requirements, Realtime Compute now allows you to customize the sink plug-in. The following dependency packages are required for a Maven project. The scope setting is `< scope > provided </ scope >`.

JAR package download

1. blink-connector-common-blink-2.2.4
2. blink-connector-custom-blink-2.2.4

3. blink-table-blink-2.2.4
4. flink-table_2.11-blink-2.2.4
5. flink-core-blink-2.2.4

Dependencies for Realtime Compute V2.0 and later

```
 < dependenci  es >
  < dependency >
    < groupId > com . alibaba . blink </ groupId >
    < artifactId > blink - table </ artifactId >
    < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
    < scope > provided </ scope >
  </ dependency >
  < dependency >
    < groupId > org . apache . flink </ groupId >
    < artifactId > flink - table_2 . 11 </ artifactId >
    < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
    < scope > provided </ scope >
  </ dependency >
  < dependency >
    < groupId > org . apache . flink </ groupId >
    < artifactId > flink - core </ artifactId >
    < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
    < scope > provided </ scope >
  </ dependency >
  < dependency >
    < groupId > com . alibaba . blink </ groupId >
    < artifactId > blink - connector - common </ artifactId >
    < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
    < scope > provided </ scope >
  </ dependency >
  < dependency >
    < groupId > com . alibaba . blink </ groupId >
    < artifactId > blink - connector - custom </ artifactId >
    < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
    < scope > provided </ scope >
  </ dependency >
</ dependenci  es >
```

API description

The custom result table class needs to inherit the base class of the custom sink plug-in and implement the following methods:

```
protected   Map < String , String >  userParams  Map ;//  The
key - value   pairs   defined   by   you   in   the   SQL   WITH
statement .  All   keys   are   in   lowercase .
protected   Set < String >  primaryKey  s ;//  The   custom   primary
  key   fields .
protected   List < String >  headerFiel  ds ;//  The   list   of
fields   marked   as   header .
protected   RowTypeInf  o   rowTypeInf  o ;//  The   field   type
and   name .
/**
```

```
 *  The   initializa  tion   method .  This   method   is   called
 when   you   create   a   table   for   the   first   time   or   in
   the   case   of   failover .
 *
 * @ param   taskNumber   //  The   ID   of   the   current   operator
 .
 * @ param   numTasks   //  The   total   number   of   sink
 operators .
 * @ throws   IOExceptio  n
 */
 public   abstract   void   open ( int   taskNumber , int   numTasks
 )  throws   IOExceptio  n ;

/**
 *  The   close   method ,  which   is   used   to   release
 resources .
 *
 * @ throws   IOExceptio  n
 */
 public   abstract   void   close ()  throws   IOExceptio  n ;

/**
 *  Insert   a   single   row   of   data .
 *
 * @ param   row
 * @ throws   IOExceptio  n
 */
 public   abstract   void   writeAddRe  cord ( Row   row )  throws
 IOExceptio  n ;

/**
 *  Delete   a   single   row   of   data .
 *
 * @ param   row
 * @ throws   IOExceptio  n
 */
 public   abstract   void   writeDelet  eRecord ( Row   row )  throws
   IOExceptio  n ;

/**
 *  If   you   want   to   use   this   method   to   insert
 multiple   rows   at   a   time ,  you   need   to   load   all
 data   cached   in   threads   to   the   downstream   operator .
 If   you   do   not   want   to   insert   multiple   rows   at   a
 time ,  this   method   is   not   required .
 *
 * @ throws   IOExceptio  n
 */
 public   abstract   void   sync ()  throws   IOExceptio  n ;

/**
*  Return   the   class   name .
*/
 public   String   getName ();
```

After uploading JAR packages to Realtime Compute and referencing resources, you need to specify `type  = ' custom '` for your custom sink plug-in. In addition,

specify the class that implements the method. The following shows an example of a custom Redis result table (download the demo here).

```
 create   table   in_table (
     kv   varchar
) with (
     type  = ' random '
);

 create   table   out_table (
    ` key `  varchar ,
    ` value `  varchar
) with (
     type  = ' custom ',
     class  = ' com . alibaba . blink . connector . custom . demo .
 RedisSink ',
    -- ** You   can   define   more   user   parameters , which   can
   be   obtained   by   using   userParams Map   in   the   open ()
 function .**
     host  = ' r - uf ****. redis . rds . aliyuncs . com ',
     port  = ' 6379 ',
     db  = ' 0 ',
     batsize  = ' 10 ',
     password  = '‘< yourDataba  sePassword >'
);

 insert   into   out_table
 select
 substring ( kv , 0 , 4 )  as  ` key `,
 substring ( kv , 0 , 6 )  as  ` value `
 from   in_table ;
```

The following table describes parameters of the Redis sink plug-in.

Name	Description	Required	Remarks
host	The intranet connection address of the Redis instance.	Yes	None
port	The port number of the Redis instance.	Yes	None
password	The password used to connect to the Redis instance.	Yes	None
db	The database ID.	No	Default value: 0, indicating db0.

Name	Description	Required	Remarks
batchsize	The number of data records written at a time.	No	Default value: 1, indicating that writing multiple data records at a time is not supported.

6.6.4 Create a dimension table

6.6.4.1 Dimension table overview

No special DDL syntax is designed for dimension tables in Realtime Compute. You can use the standard `CREATE TABLE` syntax by simply adding the `PERIOD FOR SYSTEM_TIM E` statement. This statement defines the change period of the dimension table, which indicates that the dimension table is changeable.

Examples

```
CREATE   TABLE   white_list  (
  id   varchar ,
  name   varchar ,
  age   int ,
  PRIMARY   KEY  ( id ),  -- If   the   table   is   used   as   a
dimension   table  ,  it   must   have   a   declared   primary   key
.
  PERIOD   FOR   SYSTEM_TIM  E  --  Define   the   change   period
of   the   dimension   table .
)  with  (
  type   = ' rds ',
  ...
)
```

Note:

- When you declare a dimension table, you must specify the primary key. When you join a dimension table to another table, the ON condition must contain the equivalent conditions for all primary keys.
- Currently, Realtime Compute only supports joining the source table to the dimension table in `INNER JOIN` or `LEFT JOIN` mode.

- The declared unique key of the dimension table must be the unique key of the database table. Otherwise, you may find the following negative impacts:
  1. The dimension table becomes slow to read.
  2. When you join the dimension table to another table, table joining starts from the first data record. During the job processing, multiple data records with the same key are updated in the database in order. This may cause errors in the joining result.

INDEX syntax

Note:
We recommend that you use the `INDEX` syntax in Realtime Compute V2.2.7 and later.

In versions earlier than Realtime Compute V2.2, you must declare the `PRIMARY KEY` when you declare a dimension table. This only works with one-to-one table joining. To meet the one-to-many table joining requirements, the `INDEX` syntax is introduced. Currently, when cache is not set to `ALL`, dimension table joining is implemented based on the `INDEX LOOKUP` method. For a batch job, we may use the `SCAN` method based on `COST` in the future.

```
 CREATE   TABLE   Persons   (
     ID   bigint ,
     LastName   varchar ,
     FirstName   varchar ,
     Nick   varchar ,
     Age   int ,
    [ UNIQUE ]  INDEX ( LastName , FirstName , Nick ), --  Define
 the   index .  You   do   not   need   to   specify   the   index
 type  ( such   as   fulltext   or   clustered ).
     PERIOD   FOR   SYSTEM_TIM  E
)  with  (
   type =' xxx ',
  ...
);
```

`UNIQUE INDEX` indicates `one - to - one` table joining. `INDEX` indicates `one - to - many` table joining.

Note:
1. `UNIQUE CONSTRAINT` (`UNIQUE KEY`) is supported in Realtime Compute V2.2.7 and later. Like most relational database management systems (RDBMSs),

Realtime Compute indirectly provides the `UNIQUE INDEX` attribute after you declare `UNIQUE CONSTRAINT`. If you are using an earlier Realtime Compute version, you can use the `PRIMARY KEY` definition.

2. When generating an execution plan, Realtime Compute preferentially uses `UNIQUE INDEX`. That is, when INDEX is used in DDL and both `UNIQUE INDEX` and `NON - UNIQUE INDEX` are contained in the equijoin conditions, Realtime Compute preferentially uses `UNIQUE INDEX` to search for data in the right table.
3. RDS and MaxCompute dimension tables support one-to-many table joining. MaxCompute dimension tables support only the `ALL` cache policy and do not allow random access.
4. The `maxJoinRow s` parameter specifies the maximum number of rows from the right table to which a row from the left table can be joined. The default value is 1024. Note that, if one row is joined to too many rows, consider to increase the cache memory. The `cacheSize` parameter limits the number of row keys in the left table. The job performance may be severely affected if one row from the left table is joined to too many rows from the right table.

Differences between the dimension table, source table, and result table

	Source table	Result table	Dimension table
Whether it drives computing	Yes	No	No
Whether it supports reading data	Yes, it supports reading data directly.	No	Yes, but it only supports reading data by joining the source table to the dimension table.
Whether it supports writing data	No	Yes	No
Whether it supports caching	No	No	Yes

6.6.4.2 Create a Table Store dimension table

This topic describes how to create a Table Store dimension table in Realtime Compute.

Introduction to Table Store

Table Store is a distributed NoSQL data storage service built on Alibaba Cloud's Apsara system. It is designed to provide 99.99% high availability and 99.999999999% data reliability. By using data sharding and load balancing technologies, Table Store implements seamless scale-out in terms of data scale and access parallelism. It also supports the storage of and real-time access to large amounts of structured data.

Examples

Realtime Compute supports creating a Table Store table as the dimension table. The sample code is as follows:

```
CREATE   TABLE   ots_dim_ta  ble  (
  id   int ,
  len   int ,
  content   VARCHAR ,
  PRIMARY   KEY  ( id ),
  PERIOD   FOR   SYSTEM_TIM  E  --  Define   the   change   period
   of   the   dimension   table ,  which   indicates   that   the
 dimension   table   is   changeable .
)  WITH  (
  type =' ots ',
  endPoint =‘< yourEndpoi  nt >’,
  instanceNa  me ='< yourInstan  ceName >',
  tableName ='< yourTableN  ame >',
  accessId =‘< yourAccess  Id >’,
  accessKey =‘< yourAccess  Secret >’
);
```

Note:
When you declare a dimension table, you must specify the primary key. When you join a dimension table to another table, the ON condition must contain the equivalent conditions for all primary keys. The primary key of a Table Store table is the rowkey field of the table.

WITH parameters

Name	Description
instanceName	The instance name.
tableName	The table name.

Name	Description
endPoint	The endpoint for accessing the instance.
accessId	The AccessKey ID.
accessKey	The AccessKey Secret.

Cache parameters

Name	Description	Remarks
cache	The cache policy.	Default value: `None`. Valid values: None and `LRU`.
cacheSize	The cache size, in lines.	When cache is set to LRU , this parameter specifies the cache size. Default value: 10000.
cacheTTLMs	The time before the cache expires, in milliseconds.	When cache is set to LRU , this parameter specifies the time before the cache expires.

Sample code

```
 CREATE   TABLE   datahub_in  put1  (
 id              BIGINT ,
 name          VARCHAR ,
 age             BIGINT
)  WITH  (
 type =' datahub '
);

 create   table   phoneNumbe  r (
 name   VARCHAR ,
 phoneNumbe  r   bigint ,
 primary   key ( name ),
 PERIOD   FOR   SYSTEM_TIM  E  --  Define   the   change   period   of
   the   dimension   table .
) with (
 type =' ots '
);

 CREATE   table   result_inf  or (
 id   bigint ,
 phoneNumbe  r   bigint ,
 name   VARCHAR
) with (
 type =' rds '
);

 INSERT   INTO   result_inf  or
 SELECT
```

```
 t . id
, w . phoneNumbe  r
, t . name
 FROM   datahub_in  put1   as   t
 JOIN   phoneNumbe  r   FOR   SYSTEM_TIM  E   AS   OF   PROCTIME ()
 as   w  -- This   statement   must   be   specified   for   joining
   the   dimension   table .
 ON   t . name  =  w . name ;
```

For detailed syntax of dimension tables, see Dimension table JOIN statement.

6.6.4.3 Create an RDS or a DRDS dimension table

This topic describes how to create an RDS or a DRDS dimension table in Realtime Compute.

ApsaraDB for RDS

ApsaraDB for RDS is a stable and reliable online database service that supports auto scaling. Based on Apsara Distributed File System and high-performance storage, RDS supports a variety of engines such as MySQL, SQL Server, PostgreSQL, and Postgres Plus Advanced Server (PPAS, which is highly compatible with Oracle). It provides a comprehensive set of solutions for diversified requirements, such as disaster tolerance, data backup, data recovery, monitoring, and data migration, to resolve database O&M difficulties.

Note:

RDS and DRDS plug-ins use the same WITH parameters. The following WITH parameter descriptions apply to both plug-ins. When you use an RDS or a DRDS dimension table, make sure that an actual table exists in RDS or DRDS.

Examples

Realtime Compute supports creating an RDS or a DRDS table as the dimension table. Currently, only MySQL tables are supported. The sample code is as follows:

```
 CREATE   TABLE   rds_dim_ta  ble (
  id   int ,
  len   int ,
  content   VARCHAR ,
  PRIMARY   KEY  ( id ),
  PERIOD   FOR   SYSTEM_TIM  E  -- Define   the   change   period
   of   the   dimension   table ,  which   indicates   that   the
 dimension   table   is   changeable .
)  with  (
  type =' rds ',
  url ='< yourDataba  seURL >',
  tableName =‘< yourTableN  ame >’,
  userName =‘< yourDataba  seUserName >',
  password =‘< yourDataba  sePassword >'
```

```
);
```

Note:

When you declare a dimension table, you must specify the primary key. When you join a dimension table to another table, the ON condition must contain the equivalent conditions for all primary keys. You can define the primary key or unique index column of the table to which the RDS or DRDS dimension table is joined as the primary key of the dimension table.

WITH parameters

Name	Description	Remarks
url	The address.	For more information about the address, see: · Apply for an Internet IP address for RDS
tableName	The table name.	None
userName	The logon username.	None
password	The logon password.	None
maxRetryTimes	The maximum number of insertion retries.	Optional. Default value: 3.

Cache parameters

Name	Description	Remarks
cache	The cache policy.	Default value: `None`. Valid values: None, `LRU`, and `ALL`.
cacheSize	The cache size, in lines.	When cache is set to `LRU`, this parameter specifies the cache size. Default value: 10000.

Name	Description	Remarks
cacheTTLMs	The time before the cache expires, in milliseconds.	When cache is set to `LRU`, this parameter specifies the time before the cache expires. The cache will not expire by default. When cache is set to `ALL`, this parameter specifies the cache reload interval. The cache will not be reloaded by default.
cacheReloadTimeBlackList	The reload time blacklist. This parameter is available when cache is set to `ALL`. It is used to prevent the cache from being reloaded during the blacklist period (for example, during the Double 11 Shopping Festival).	Optional. This parameter is left empty by default. An example of the custom input value is `2017 - 10 - 24 14 : 00 -> 2017 - 10 - 24 15 : 00 , 2017 - 11 - 10 23 : 30 -> 2017 - 11 - 11 08 : 00`. Use a comma (`,`) to separate two blacklist records. Use an arrow sign (`->`) to separate the start time and end time of a blacklist record.

Currently, RDS and DRDS provide the following three cache policies:

- None: indicates that no data is cached.
- LRU: indicates that the recently used data is cached. When this cache policy is used, you need to set the cacheSize and cacheTTLMs parameters.
- ALL: indicates that all data is cached. Before Realtime Compute runs a job, it loads all data in the remote table to the memory. Then Realtime Compute searches the cache for data in all subsequent dimension table query operations. In the case of a cache miss, the corresponding data does not exit. All data is cached again after the cache expires. The ALL cache policy applies to scenarios where the remote table is small but there are a large number of missing keys. When this cache policy is used, you need to set the cacheTTLMs and cacheReloadTimeBlackList parameters.

Note:

- When cache is set to ALL, Realtime Compute reloads data asynchronously. Therefore, you need to increase the memory of the JOIN operator. The size of the increased memory is twice the data size of the remote table.
- When cache is set to ALL, pay special attention to the memory of the JOIN operator to prevent out of memory (OOM) errors.

Sample code

```
 CREATE   TABLE   datahub_in  put1  (
 id               BIGINT ,
 name           VARCHAR ,
 age              BIGINT
)  WITH  (
 type =' datahub '
);
 create   table   phoneNumbe  r (
 name   VARCHAR ,
 phoneNumbe  r   bigint ,
 primary   key ( name ),
 PERIOD   FOR   SYSTEM_TIM  E  -- Define   the   change   period   of
   the   dimension   table .
) with (
 type =' rds '
);
 CREATE   table   result_inf  or (
 id   bigint ,
 phoneNumbe  r   bigint ,
 name   VARCHAR
) with (
 type =' rds '
);
 INSERT   INTO   result_inf  or
 SELECT
 t . id
, w . phoneNumbe  r
, t . name
 FROM   datahub_in  put1   as   t
 JOIN   phoneNumbe  r   FOR   SYSTEM_TIM  E   AS   OF   PROCTIME ()
 as   w  --  This   statement   must   be   specified   for   joining
   the   dimension   table .
 ON   t . name  =  w . name ;
```

For detailed syntax of dimension tables, see Dimension table JOIN statement.

6.6.4.4 Create an HBase dimension table

This topic describes how to create an HBase dimension table in Realtime Compute.

Note:

For more information about the JOIN syntax of an HBase dimension table, see Dimension table JOIN statement.

Examples

```
CREATE   TABLE   hbase  (
  ` key `  varchar ,
  ` name `  varchar ,
    PRIMARY   KEY  (` key `), --  The   rowkey   field   in   HBase
.
    PERIOD   FOR   SYSTEM_TIM  E  --  Specify   that   this   is   a
  dimension   table .
  )  with  (
    type  = ' cloudhbase ',
    zkQuorum  = ' 2 '
    columnFami  ly  = '< yourColumn  Family >',
    tableName  = '< yourTableN  ame >'
);
```

Note:

When you declare a dimension table, you must specify the primary key. When you join a dimension table to another table, the ON condition must contain the equivalent conditions for all primary keys. The primary key of an HBase dimension table is the rowkey field of the table.

Parameters

Name	Description	Remarks
zkQuorum	The ZooKeeper address of the HBase cluster. The value is a list of host IP addresses separated by commas (,).	You can find the configuration of `hbase.zookeeper.quorum` in the *hbase - site . xml* file.
zkNodeParent	The path of the cluster on the ZooKeeper server.	You can find the configuration of `hbase.zookeeper.quorum` in the *hbase - site . xml* file.
tableName	The name of the HBase table.	None
columnFamily	The name of the column family.	Currently, Realtime Compute only supports inserting data of the same column family.
userName	The logon username.	None
password	The logon password.	None

Name	Description	Remarks
maxRetryTimes	The maximum number of insertion retries.	Default value: 10.
partitionedJoin	When this parameter is set to true, Realtime Compute partitions the table based on joinKey and distributes data to the JOIN operator. This helps improve the cache hit ratio.	Optional. Default value: false.
shuffleEmptyKey	When this parameter is set to false, Realtime Compute distributes empty keys in the input data to the same JOIN operator. When this parameter is set to true, Realtime Compute distributes empty keys in the input data to random JOIN operators.	We recommend that you set this parameter to true.

Cache parameters

Name	Description	Remarks
cache	The cache policy.	Default value: `None`. Valid values: None, `LRU`, and `ALL`.
cacheSize	The cache size, in lines.	When cache is set to `LRU`, this parameter specifies the cache size. Default value: 10000.
cacheTTLMs	The time before the cache expires, in milliseconds.	When cache is set to `LRU`, this parameter specifies the time before the cache expires. The cache will not expire by default. When cache is set to `ALL`, this parameter specifies the cache reload interval. The cache will not be reloaded by default.

Name	Description	Remarks
cacheReloadTimeBlackList	The reload time blacklist. This parameter is available when cache is set to `ALL`. It is used to prevent the cache from being reloaded during the blacklist period (for example, during the Double 11 Shopping Festival).	Optional. This parameter is left empty by default. An example of the custom input value is `2017 - 10 - 24 14 : 00 -> 2017 - 10 - 24 15 : 00 , 2017 - 11 - 10 23 : 30 -> 2017 - 11 - 11 08 : 00`. Use a comma (`,`) to separate two blacklist records. Use an arrow sign (`->`) to separate the start and end time of a blacklist record.
cacheScanLimit	The maximum number of rows returned by the server to the client for each remote procedure call (RPC) when Realtime Compute loads all data of an HBase database. This parameter is available when cache is set to `All`. Optional.	Default value: 100.

Currently, HBase provides the following three cache policies:

- None: indicates that no data is cached.
- LRU: indicates that the recently used data is cached. When this cache policy is used, you need to set the `cacheSize` and `cacheTTLMs` parameters.
- ALL: indicates that all data is cached. Before Realtime Compute runs a job, it loads all data in the remote table to the memory. Then Realtime Compute searches the cache for data in all subsequent dimension table query operations. In the case of a cache miss, the corresponding data does not exit. All data is cached again after the cache expires. The ALL cache policy applies to scenarios where the remote table is small but there are a large number of missing keys. When this cache policy is

used, you need to set the `cacheTTLMs` and `cacheReloa dTimeBlack List` parameters.

> **Note:**
> When cache is set to ALL, Realtime Compute reloads data asynchronously. Therefore, you need to increase the memory of the JOIN operator. The size of the increased memory is twice the data size of the remote table.

6.6.4.5 Create a MaxCompute dimension table

> **Note:**
>
> - We recommend that you use Realtime Compute V2.1.1 and later to perform this operation.
> - For more information about the query syntax of a dimension table, see Dimension table JOIN statement.
> - To use a MaxCompute dimension table, you must grant the read permission to your MaxCompute account first.

Examples

```
CREATE   TABLE   white_list  (
  id   varchar ,
  name   varchar ,
  age   int ,
  PRIMARY   KEY  ( id ),
  PERIOD   FOR   SYSTEM_TIM  E  --  Specify   that   this   is   a
dimension   table  .
)  with  (
  type  =  ' odps ',
  endPoint  =  ' http :// service  .  cn  .  maxcompute  .  aliyun  -  inc  .
com  /  api  ',
  project  =  '< projectNam  e >',
  tableName  =  '< tableName  ',
  accessId  =  '< yourAccess  KeyId >',
  accessKey  =  '< yourAccess  KeySecret >',
 `  partition  `  =  '  ds  =  20180905  ',
  cache  =  '  ALL  '
)
```

> **Note:**
>
> - When you declare a dimension table, you must specify the primary key. When you join a dimension table to another table, the ON condition must contain the equivalent conditions for all primary keys.

- Primary key values for each row of a MaxCompute dimension table must be unique. Otherwise, the duplicate records are removed.
- `partition` is a keyword. It must be enclosed in backticks (`) whenever it is referenced.
- If the dimension table is a partition table, Realtime Compute does not currently support writing the partition column to the schema definition.
- Realtime Compute V2.2.0 and later allow you to load the latest partition table by configuring `partition =' max_pt ()'`. `max_pt ()` indicates the largest numbered partition in the lexicographic order of all partitions.

WITH parameters

Name	Description	Remarks
endPoint	The MaxCompute endpoint .	Required. For more information, see Configure Endpoint.
project	The MaxCompute project name.	Required.
tableName	The table name.	Required.
accessId	The AccessKey ID.	Required.
accessKey	The AccessKey Secret.	Required.
partition	The partition name.	Optional. This parameter must be specified for a partition table. To view detailed partition information, log on to Data Map. For example, if the partition name of a table is `ds = 20180905`, you can specify the parameter as `` ` partition ` = ' ds = 20180905 ' ``. Use commas (,) to separate multiple levels of partitions, for example, `` ` partition ` = ' ds = 20180912 , dt = xxxyyy ' ``.

Name	Description	Remarks
maxRowCount	The maximum number of rows that you can load from a table to the memory.	Optional. Default value: 100000. Note: If the number of rows of your data table is greater than 100,000, increase the value of the maxRowCount parameter. We recommend that you set it to a value that is greater than the actual number of rows.

Cache parameters

Name	Description	Remarks
cache	The cache policy.	Default value: `None`. Valid values: None, `LRU`, and `ALL`.
cacheSize	The cache size, in lines.	When cache is set to `LRU`, this parameter specifies the cache size. Default value: 100000.
cacheTTLMs	The time before the cache expires, in milliseconds.	When cache is set to `LRU`, this parameter specifies the time before the cache expires. The cache will not expire by default. When cache is set to `ALL`, this parameter specifies the cache reload interval. The cache will not be reloaded by default.

Name	Description	Remarks
cacheReloadTimeBlackList	The reload time blacklist. This parameter is available when cache is set to ALL . It is used to prevent the cache from being reloaded during the blacklist period (for example, during the Double 11 Shopping Festival).	Optional. This parameter is left empty by default. Example: `2017 - 10 - 24 14 : 00 -> 2017 - 10 - 24 15 : 00 , 2017 - 11 - 10 23 : 30 -> 2017 - 11 - 11 08 : 00` . Use a comma (,) to separate two blacklist records. Use an arrow sign (`->`) to separate the start time and end time of a blacklist record.
cacheScanLimit	The maximum number of rows returned by the server to the client for each RPC when Realtime Compute loads all data of a MaxCompute instance. This parameter is available when cache is set to ALL.	Optional. Default value: 100.

Currently, a MaxCompute dimension table only supports the `ALL` cache policy. To use this policy, you must declare the MaxCompute dimension table explicitly.

The ALL cache policy indicates that all data is cached. Before Realtime Compute runs a job, it loads all data in the remote table to the memory. Then Realtime Compute searches the cache for data in all subsequent dimension table query operations. In the case of a cache miss, the corresponding data does not exit. All data is cached again after the cache expires. The ALL cache policy applies to scenarios where the remote table is small but there are a large number of missing keys. When cache is set to ALL, you need to set the `cacheTTLMs` and `cacheReloa dTimeBlack List` parameters.

Note:

Note: When cache is set to ALL, Realtime Compute reloads data asynchronously. Therefore, you need to increase the memory of the JOIN operator. The size of the increased memory is twice the data size of the remote table.

Metrics

When you join the dimension table to another table, you can view metrics such as the correlation degree and cache hit ratio. Currently, you cannot view the metrics on the Bayes platform. You can only use K-Monitor to view the metrics.

Query statement	Description
fetch qps	Queries the total number of queries per second (QPS) against the dimension table, including hits and misses. The metric name is `blink.projectName.jobName.dimJoin.fetchQPS`.
fetchHitQPS	Queries the number of hits (in QPS) against the dimension table, including cache hits and hits against the physical dimension table. The metric name is `blink.projectName.jobName.dimJoin.fetchHitQPS`.
cacheHitQPS	Queries the number of cache hits (in QPS) against the dimension table. The metric name is `blink.projectName.jobName.dimJoin.cacheHitQPS`.
dimJoin.fetchHit	Queries the correlation degree of the dimension table and the table to which the dimension table is joined. The metric name is `blink.projectName.jobName.dimJoin.fetchHit`.
dimJoin.cacheHit	Queries the cache hit ratio of the dimension table. The metric name is `blink.projectName.jobName.dimJoin.cacheHit`.

FAQ

Q: What can I do if the failover message `RejectedExecutionException: Task java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTas` occurs when I run a job?

A: Dimension table joining in Realtime Compute V1.x has certain issues. We recommend that you upgrade Realtime Compute to V2.1.1 or later. If you insist on using the existing version, you need to pause the job and resume it after troubleshooting. To troubleshoot the failover, check the specific error information that was generated for the first failover record in the failover history.

Q: What is the mapping between MaxCompute data types and Realtime Compute data types?

A: The following table lists the mapping.

MaxCompute data type	Realtime Compute data type
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	STRING
DECIMAL	DECIMAL
BINARY	VARBINARY

Note:

Currently, MaxCompute connectors do not support converting other MaxCompute data types.

6.7 DML statement

6.7.1 EMIT statement

Note:

The EMIT statement is supported in Realtime Compute `V2 . 0` and later.

Policy

Users may require different output policies (such as the maximum delay allowed) for a query in different scenarios. For example, a user wants to view the latest result every 1 minute before a 1-hour window ends, and does not want to lose data that arrives late within one day after the window ends. The traditional ANSI SQL does not provide the syntax to meet this requirement. Flink SQL abstracts the EMIT syntax from such requirement and extends the SQL syntax with the EMIT syntax.

Purpose

Currently, the EMIT syntax is used to control delay and improve data accuracy.

1. Control delay. For a large window, you can set the result output frequency before the window ends to shorten the delay in displaying the result to users.
2. Improve data accuracy. The system does not discard data that arrives after a window ends and updates the data to the output result.

When configuring EMIT policies, you also need to weigh the overhead. A lower output delay and a higher data accuracy mean a higher computing overhead.

Syntax

The EMIT syntax is used to define output policies, that is, to define actions in the INSERT INTO statement. If no EMIT policy is configured, the default behavior takes effect. In this case, a window generates a result only when the window ends, that is, when the watermark is triggered. The syntax is as follows:

```
INSERTINTO   tableName
query
EMIT   strategy  [,  strategy ]*

strategy  ::= { WITH   DELAY   timeInterv  al  |  WITHOUT   DELAY }
                  [ BEFORE   WATERMARK  | AFTER   WATERMARK ]

timeInterv  al  ::=' string '  timeUnit
```

- `WITH   DELAY` : specifies the maximum result output delay allowed. Results are generated at the specified interval.
- `WITHOUT   DELAY` : specifies that no delay is allowed. A result is generated immediately data is received.
- `BEFORE   WATERMARK` : specifies the policy before the window ends, that is, before the watermark is triggered.

- `AFTER WATERMARK` : specifies the policy after the window ends, that is, after the watermark is triggered.

Note:

1. Multiple strategies can be defined, and the BEFORE WATERMARK and AFTER WATERMARK policies can be defined at the same time. However, you cannot define two BEFORE WATERMARK policies or two AFTER WATERMARK policies at the same time.
2. If the AFTER WATERMARK policy is defined, you must set the `blink . state . ttl . ms` parameter to explicitly define the maximum delay allowed.

Example

```
-- Before a window ends , results are generated at a
   delay of 1 minute . After the window ends , results
 are generated without delay .
 EMIT
   WITH DELAY ' 1 ' MINUTE BEFORE WATERMARK ,
   WITHOUT DELAY AFTER WATERMARK

-- Before a window ends , no result is generated .
 After the window ends , results are generated without
 delay .
 EMIT WITHOUT DELAY AFTER WATERMARK

-- Results are generated at a delay of 1 minute
   globally . ( The delay is used by MiniBatch to
 accumulate data .)
 EMIT WITH DELAY ' 1 ' MINUTE -- Before a window ends ,
 results are generated at a delay of 1 minute .
 EMIT WITH DELAY ' 1 ' MINUTE BEFORE WATERMARK
```

- Maximum delay allowed

  When the AFTER WATERMARK policy is configured, the window status is retained for a period of time to wait for late data. But how long will the window wait for late data? You can set the `blink . state . ttl . ms` parameter to customize a state time to live (TTL) for the window. This parameter is not set by default. If the AFTER WATERMARK policy is configured, you must explicitly set the `blink . state . ttl . ms` parameter. For example, `blink . state . ttl . ms = 3600000` indicates that the window will wait for late data for as long as 1 hour. Data that arrives more than 1 hour late will be directly discarded.
- Example

  Take the 1-hour tumbling window `tumble_win dow` as an example.

```
CREATE VIEW tumble_win dow AS
```

```
SELECT
  id ,
  TUMBLE_STA  RT ( rowtime ,  INTERVAL  ' 1 '  HOUR )  as
start_time ,
  COUNT (*)  as   cnt
FROM   source
GROUP   BY   id ,  TUMBLE ( rowtime ,  INTERVAL  ' 1 '  HOUR )
```

By default, you need to wait for 1 hour before obtaining the result of `tumble_win dow`. Sometimes, you may want to obtain the result as early as possible even though the result is incomplete. For example, you want to see the latest result from the window every 1 minute. In this case, define the EMIT policy as follows:

```
INSERT   INTO   result
SELECT  *  FROM   tumble_win  dow
EMIT   WITH   DELAY  ' 1 '  MINUTE   BEFORE   WATERMARK  --  Before
  the   window   ends ,  results   are   generated   every   1
minute .
```

By default, `tumble_win  dow` discards any data that arrives after the window ends. The late data may be important to you, and you want to incorporate the data into the final result. You know that the data will not arrive more than one day late and want to update the result immediately late data is received. In this case, you can define the EMIT policy as follows:

```
INSERT   INTO   result
SELECT  *  FROM   tumble_win  dow
EMIT   WITH   DELAY  ' 1 '  MINUTE   BEFORE   WATERMARK ,
     WITHOUT   DELAY   AFTER   WATERMARK   --  After   the   window
  ends ,  results   are   generated   immediatel  y   data   is
received .

--  Add   a   state   TTL   configurat  ion   of   one   day .
blink . state . ttl . ms  =  86400000
```

Delay concept

In this topic, the delay refers to the duration that starts when user data enters Realtime Compute and ends when result data exits Realtime Compute. The delay is tolerable to users and can be either in event time or processing time. The delay is calculated based on the system time. In other words, the delay is the interval between the time when a row is changed in a dynamic table and the time when the new row can be viewed in a result table. The dynamic table is the data stream storage media inside Realtime Compute, and the result table is the storage media outside Realtime Compute.

If the processing time of Realtime Compute is 0, the delay is generated when MiniBatch accumulates data and a window waits for window data. If a user specifies that a maximum delay of 30 seconds can be tolerated, the 30 seconds can be used by MiniBatch to accumulate data. For a query of a 1-hour window, a maximum delay of 30 seconds means that changed rows are exported every 30 seconds. The delay specifies the trigger interval of the window result.

Take `EMIT WITH DELAY '1' MINUTE` as an example. For a common GROUP BY aggregation, MiniBatch takes 1 minute to accumulate data. For a window whose size is greater than 1 minute, the window generates a result every 1 minute. If the window size is smaller than or equal to 1 minute, the configuration is ignored. This is because the window can ensure the delay SLA by using the watermark.

Another example is `EMIT WITHOUT DELAY`. For a common GROUP BY aggregation, MiniBatch is not enabled, and data is computed and exported immediately it is received. For a window, data is also computed and exported immediately it is received.

Current status and future plans

Current status and future plans:

1. Currently, only tumbling and sliding windows support EMIT policies. Session windows do not support EMIT policies.
2. Currently, if a job has multiple outputs, the same EMIT policy must be defined for these outputs. Different policies will be supported in the future.
3. Currently, the EMIT syntax cannot be used to configure allowLateness for MiniBatch. We plan to enable you to declare allowLateness in EMIT policies in the future.

6.7.2 INSERT INTO statement

This topic describes the method and constraints for using an INSERT INTO statement in Realtime Compute.

Syntax

```
INSERT INTO tableName
     [ ( columnName [ , columnName ]*) ]
         queryState ment ;
```

Examples

```
INSERT INTO LargeOrder s
SELECT * FROM Orders WHERE units > 1000 ;
```

```
INSERT INTO Orders ( z , v )
```

```
SELECT   c , d   FROM   OO ;
```

Note:

- A single Realtime Compute job allows one SQL job to contain multiple DML operations, data sources, data destinations, and dimension tables. For example, a job file can contain two paragraphs of SQL statements for independent businesses , which will write data to different data destinations.
- Realtime Compute does not allow you to use a separate SELECT statement for query. A SELECT statement must be used together with a CREATE VIEW statement or be contained in an INSERT INTO statement.
- An INSERT INTO statement supports updating information in an existing record. For example, you can insert a key value into an RDS table. If this key value exists, it will be updated. If it does not exist, a new key value will be inserted.

Operation constraints

Table type	Operation constraint
Source table	Can only be referenced in a FROM statement and does not support INSERT.
Dimension table	Can only be referenced in a JOIN statement and does not support INSERT.
Result table	Supports only INSERT.
View	Can only be referenced in a FROM statement.

6.8 Query statements

6.8.1 SELECT statement

A SELECT statement selects data from tables.

Syntax

```
 SELECT   [  DISTINCT  ]
{ * |  projectIte  m  [,  projectIte  m  ]* }
 FROM   tableExpre  ssion ;
```

Test data

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Example 1

- **Test statement**

```
SELECT  *  FROM   Table   name ;
```

- **Test result**

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Example 2

- **Test statement**

```
SELECT   a ,  c   AS   d   FROM   Table   name ;
```

- **Test result**

a (VARCHAR)	d (DATE)
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

Example 3

- Test statement

```
SELECT   DISTINCT   a   FROM   Table   name ;
```

- Test result

a (VARCHAR)
a1
b1
c1

Subquery

Generally, a SELECT statement reads data from several tables, such as `SELECT column_1 , column_2 … FROM table_name`. A SELECT statement can also read data from another SELECT statement, which is called a subquery.

Note:
A subquery must use aliases, as shown in the following example.

- SQL statement example:

```
INSERT   INTO   result_tab  le
SELECT   *  FROM
              ( SELECT     t . a ,
                         sum ( t . b )   AS   sum_b
                FROM       t1   t
                GROUP   BY   t . a
              )  t1
WHERE    t1 . sum_b   >   100 ;
```

- Example result

a (VARCHAR)	b (INT)
a1	211
b1	120
a1	257

6.8.2 WHERE statement

A WHERE statement can be used to filter data generated from a SELECT statement.

Syntax

```
SELECT   [  ALL   |  DISTINCT  ]
```

```
{ * |  projectIte  m  [,  projectIte  m  ]* }
 FROM   tableExpre  ssion
[  WHERE   booleanExp  ression  ];
```

The following table describes the operators can be used in a WHERE statement.

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Smaller than
<=	Smaller than or equal to

Example

- Test data

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	Shanghai

- Test statement

```
SELECT  *  FROM   table_a   WHERE   City =' Beijing '
```

- Test result

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

6.8.3 HAVING statement

When using an aggregate function, you need to add a HAVING statement to achieve the same filtering effect as a WHERE statement.

Syntax

```
  SELECT  [  ALL  |  DISTINCT  ]{ * |  projectIte  m  [,  projectIte
m  ]* }
  FROM   tableExpre  ssion
 [  WHERE   booleanExp  ression  ]
 [  GROUP   BY  {  groupItem  [,  groupItem  ]* } ]
```

```
[ HAVING   booleanExp  ression  ];
```

Example

- Test data

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- Test statement

```
SELECT   Customer , SUM ( OrderPrice )  FROM   XXX
GROUP   BY   Customer
HAVING   SUM ( OrderPrice )< 2000 ;
```

- Test result

Customer	SUM(OrderPrice)
Carter	1700

6.8.4 GROUP BY statement

A GROUP BY statement groups a result set by one or more columns.

Syntax

```
 SELECT  [  DISTINCT  ]
{ * |  projectIte  m  [,  projectIte  m  ]* }
 FROM   tableExpre  ssion
[  GROUP   BY  {  groupItem  [,  groupItem  ]* } ];
```

Example

- Test data

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700

Customer	OrderPrice
Bush	300
Adams	2000
Carter	100

- Test statement

```
SELECT   Customer , SUM ( OrderPrice )  FROM   xxx
GROUP   BY   Customer ;
```

- Test result

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

6.8.5 JOIN statement

A JOIN statement used in Realtime Compute has the same semantics as a traditional JOIN statement for batch processing. Both statements are used to join two tables. However, a Realtime Compute JOIN statement joins two dynamic tables and the join results are updated dynamically to ensure that the final result is consistent with the batch processing result.

Syntax

```
 tableRefer  ence  [,  tableRefer  ence  ]* |  tableexpre  ssion
[  LEFT  ]  JOIN   tableexpre  ssion  [  joinCondit  ion  ];
```

Note:

- An equijoin is supported, whereas a non-equijoin is not supported.
- Only INNER JOIN and LEFT OUTER JOIN are supported.

Example 1

- Test data

Orders:

rowtime	productId	orderId	units
10 : 17 : 00	30	5	4

rowtime	productId	orderId	units
`10 : 17 : 05`	10	6	1
`10 : 18 : 05`	20	7	2
`10 : 18 : 07`	30	8	20
`11 : 02 : 00`	10	9	6
`11 : 04 : 00`	10	10	1
`11 : 09 : 30`	40	11	12
`11 : 24 : 11`	10	12	4

Products:

productId	name	unitPrice
30	Cheese	`17`
10	Beer	`0 . 25`
20	Wine	`6`
30	Cheese	`17`
10	Beer	`0 . 25`
10	Beer	`0 . 25`
40	Bread	`100`
10	Beer	`0 . 25`

- **Test statement**

```
  SELECT   o . rowtime ,  o . productId ,  o . orderId ,  o . units
, p . name ,  p . unitPrice
  FROM   Orders   AS   o
  JOIN   Products   AS   p
  ON   o . productId  =  p . productId ;
```

- **Test result**

rowtime	productId	orderId	units	name	unitPrice
`10 : 17 : 00`	30	5	4	Cheese	`17`
`10 : 17 : 05`	10	6	1	Beer	`0 . 25`

rowtime	productId	orderId	units	name	unitPrice
`10 : 18 : 05`	20	7	2	Wine	`6`
`10 : 18 : 07`	30	8	20	Cheese	`17`
`11 : 02 : 00`	10	9	6	Beer	`0 . 25`
`11 : 04 : 00`	10	10	1	Beer	`0 . 25`
`11 : 09 : 30`	40	11	12	Bread	`100`
`11 : 24 : 11`	10	12	4	Beer	`0 . 25`

Example 2

- Test data

  datahub_stream1:

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21

  datahub_stream2:

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21
0	10	test31
1	10	test41

- Test statement

```
SELECT   s1 . c , s2 . c
FROM   datahub_st  ream1   AS   s1
JOIN   datahub_st  ream2   AS   s2
ON   s1 . a  = s2 . a
```

```
WHERE  s1 . a  =  0 ;
```

- Test result

s1_c (VARCHAR)	s2_c (VARCHAR)
test11	test11
test11	test31

6.8.6 Dimension table JOIN statement

A dimension table is constantly changing. Therefore, when joining a record to a dimension table, you must specify the time the record is associated with the dimension table snapshot. Currently, a record can be associated with the dimension table snapshot taken only at the current moment. (In the future, we will allow a record to be associated with the dimension table snapshot taken at the time specified by rowtime in the left table.)

Dimension table JOIN syntax

```
SELECT   column - names
FROM   table1   [ AS  < alias1 >]
[ LEFT ]  JOIN   table2   FOR   SYSTEM_TIM  E   AS   OF   PROCTIME ()
 [ AS  < alias2 >]
ON   table1 . column - name1  =  table2 . key - name1
```

For example, the following SQL statement joins an event stream to a whitelist dimension table:

```
SELECT   e .*,  w . *
FROM   event   AS   e
JOIN   white_list   FOR   SYSTEM_TIM  E   AS   OF   PROCTIME ()  AS
w
ON   e . id  =  w . id
```

Note:

- Dimension tables support `INNER  JOIN` and `LEFT  JOIN`, and do not support `RIGHT  JOIN` or `FULL  JOIN`.
- You must append `FOR  SYSTEM_TIM E  AS  OF  PROCTIME ()` to the end of the dimension table. Then, the data in the dimension table that can be viewed at the current moment is joined.
- The JOIN operation is performed only in processing time. Therefore, even if data in the dimension table is added, updated, or deleted, the associated data is not revoked or changed.

- The ON condition must contain an equivalent condition for the primary key of the dimension table (and must be the consistent with the definition of the table that is actually referenced). In addition to the required equivalent condition, the ON condition can contain other equivalent conditions.
- Two dimension tables cannot be joined.

Example

- Test data

  nameinfo:

id (BIGINT)	name (VARCHAR)	age (BIGINT)
1	lilei	22
2	hanmeimei	20
3	libai	28

  phoneNumber:

name (VARCHAR)	phoneNumber (BIGINT)
dufu	18867889855
baijuyi	18867889856
libai	18867889857
lilei	18867889858

- Test statements

```
 CREATE   TABLE   datahub_in  put1  (
 id               BIGINT ,
 name           VARCHAR ,
 age              BIGINT
)  WITH  (
 type =' datahub '
);

 create   table   phoneNumbe  r (
 name   VARCHAR ,
 phoneNumbe  r   bigint ,
 primary   key ( name ),
 PERIOD   FOR   SYSTEM_TIM  E
) with (
 type =' rds '
);

 CREATE   table   result_inf  or (
 id   bigint ,
 phoneNumbe  r   bigint ,
 name   VARCHAR
) with (
```

```
 type =' rds '
);

 INSERT   INTO   result_inf  or
 SELECT
 t . id ,
 w . phoneNumbe  r ,
 t . name
 FROM   datahub_in  put1   as   t
 JOIN   phoneNumbe  r   FOR   SYSTEM_TIM  E   AS   OF   PROCTIME ()
 as   w
 ON   t . name  =  w . name ;
```

- Test result

id (BIGINT)	phoneNumber (BIGINT)	name (VARCHAR)
1	18867889858	lilei
3	18867889857	libai

6.8.7 UNION ALL statement

A UNION ALL statement is used to combine two data streams. The fields of the two data streams must be the same in terms of the field type and sequence.

Syntax

```
select_sta  tement
UNION   ALL
select_sta  tement ;
```

Note:

Realtime Compute also supports the `UNION` function. `UNION ALL` allows duplicate values, whereas `UNION` does not. At the underlying layer of Realtime Compute, `UNION` is implemented as a combination of `UNION ALL` and `DISTINCT`. Due to its low execution efficiency, we recommend that you do not use `UNION`.

Examples

- Test data

  test_source_union1:

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10

  test_source_union2:

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20

  test_source_union3:

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20
test1	1	10

- Test statements

```
SELECT
    a ,
    sum ( b ),
    sum ( c )
FROM
   ( SELECT  *  from   test_sourc  e_union1
    UNION   ALL
    SELECT  *  from   test_sourc  e_union2
    UNION   ALL
    SELECT  *  from   test_sourc  e_union3
   ) t
 GROUP   BY   a ;
```

- Test result

d (VARCHAR)	e (BIGINT)	f (BIGINT)
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40

d (VARCHAR)	e (BIGINT)	f (BIGINT)
test1	4	40

6.8.8 TopN statement

A TopN statement is used to compute the largest or smallest N data records of an indicator in real-time data. Flink SQL can use an `OVER` window function to flexibly implement TopN computing.

Syntax

```
SELECT  *
FROM  (
  SELECT  *,
    ROW_NUMBER () OVER ([ PARTITION  BY  col1 [,  col2 ..]]
    ORDER  BY  col1  [ asc | desc ][,  col2  [ asc | desc ]...])
AS  rownum
  FROM  table_name )
WHERE  rownum  <=  N  [ AND  conditions ]
```

Note:

- `ROW_NUMBER ()`: specifies an `OVER` window function for computing the number of a row. The value starts from 1.
- `PARTITION  BY  col1 [,  col2 ..]`: specifies the columns used for partitioning. This parameter is optional.
- `ORDER  BY  col1  [ asc | desc ][,  col2  [ asc | desc ]...]`: specifies the columns used for sorting and the sorting order of each column.

As shown in the preceding syntax, TopN requires two levels of queries.

- In the subquery, the `ROW_NUMBER ()` window function is used to sort data by the specified columns and mark the data with rankings.
- In the outer query, only the first N data records in a ranking list are obtained. For example, if N = 10, the first 10 data records are obtained.

During execution, Flink SQL sorts an input data stream based on the sort key. If the first N data records in a partition are changed, the updated data is sent downstream as an update steam.

Note:

Therefore, if you want to export the TopN data to external storage, the target result table must contain primary keys.

Constraints of the WHERE condition

To enable Flink SQL to identify a TopN query, use the `rownum <= N` format in the outer loop to specify the first N data records. Do not place `rownum` in an expression such as `rownum - 5 <= N` for that purpose. The WHERE condition can also include other conditions that are joined with `AND`.

Example 1

In the following example, the number of times each keyword is queried is computed by hour and city. Then, the top 100 most-queried keywords are exported. The hour , city, and ranking columns in the output table together identify a unique record. Therefore, the three columns must be declared as composite keys. (The keys must also be set in the external storage.)

```
CREATE   TABLE   rds_output  (
  rownum   BIGINT ,
  start_time   BIGINT ,
  city   VARCHAR ,
  keyword   VARCHAR ,
  pv   BIGINT ,
  PRIMARY   KEY  ( rownum ,  start_time ,  city )
)  WITH  (
  type  = ' rds ',
 ...
)

INSERT   INTO   rds_output
SELECT   rownum ,  start_time ,  city ,  keyword ,  pv
FROM  (
  SELECT  *,
     ROW_NUMBER ()  OVER  ( PARTITION   BY   start_time ,  city
ORDER   BY   pv   desc )  AS   rownum
  FROM  (
        SELECT   SUBSTRING ( time_str , 1 , 12 )  AS   start_time ,
            keyword ,
            count ( 1 )  AS   pv ,
            city
        FROM   tmp_search
        GROUP   BY   SUBSTRING ( time_str , 1 , 12 ),  keyword ,
city
    )  a
)
WHERE   rownum  <=  100
```

Example 2

- Test data

ip (VARCHAR)	time (VARCHAR)
`192 . 168 . 1 . 1`	100000000

ip (VARCHAR)	time (VARCHAR)
`192 . 168 . 1 . 2`	100000000
`192 . 168 . 1 . 2`	100000000
`192 . 168 . 1 . 3`	100030000
`192 . 168 . 1 . 3`	100000000
`192 . 168 . 1 . 3`	100000000

- **Test statements**

```
 CREATE   TABLE   source_tab  le  (
   IP   VARCHAR ,
  ` TIME `  VARCHAR
) WITH (
   type =' datahub ',
   endPoint =' xxxxxxx ',
   project =' xxxxxxx ',
   topic =' xxxxxxx ',
   accessId =' xxxxxxx ',
   accessKey =' xxxxxxx '
);

 CREATE   TABLE   result_tab  le  (
   rownum   BIGINT ,
   start_time   VARCHAR ,
   IP   VARCHAR ,
   cc   BIGINT ,
   PRIMARY   KEY  ( start_time ,  IP )
)  WITH  (
   type  = ' rds ',
   url =' xxxxxxx ',
   tableName =' blink_rds_  test ',
   userName =' xxxxxxx ',
   password =' xxxxxxx '
);
 INSERT   INTO   result_tab  le
 SELECT   rownum , start_time , IP , cc
 FROM  (
   SELECT  *,
      ROW_NUMBER ()  OVER  ( PARTITION   BY   start_time   ORDER
 BY   cc   DESC )  AS   rownum
   FROM  (
         SELECT   SUBSTRING (` TIME `, 1 , 2 )  AS   start_time ,
 --  You   can   specify   a   value   based   on   the   actual
 time .  The   data   specified   in   this   example   is   test
 data .
         COUNT ( IP )  AS   cc ,
         IP
         FROM   source_tab  le
         GROUP   BY   SUBSTRING (` TIME `, 1 , 2 ),  IP
     ) a
)
 WHERE   rownum  <=  3  --  You   can   specify   a   value   based
   on   the   number   of   data   records   you   want   to
 obtain .  The   data   specified   in   this   example   is   test
   data .
```

```

```

- Test result

rownum (BIGINT)	start_time (VARCHAR)	ip (VARCHAR)	cc (BIGINT)
1	10	`192 . 168 . 1 . 3`	6
2	10	`192 . 168 . 1 . 2`	4
3	10	`192 . 168 . 1 . 1`	2

No ranking

- No ranking solves the data bloat problem.
  - Data bloat problem

    Based on the TopN syntax, the `rownum` field is written into a result table as one of the primary keys of the table. This may lead to data bloat. For example, if the ranking of a record is improved from the ninth to the first place after an update, the records ranked from the first to the ninth places are all changed. The changes must be updated in the result table. As a result, data bloat occurs. The update speed of the result table may decrease because an excessive amount of data is received.
  - Method of no ranking

    To avoid data bloat, exclude `rownum` from the result table and compute `rownum` at the front-end. Generally, the amount of top N data records is not large, and the top 100 data records can be sorted quickly at the front-end. In this case, if the ranking of a record is improved from the ninth to the first place after an update, only this record needs to be delivered. This greatly improves the update speed of the result table.
- Syntax of no ranking

```
SELECT   col1 ,  col2 ,  col3
FROM  (
 SELECT   col1 ,  col2 ,  col3
   ROW_NUMBER ()  OVER  ([ PARTITION   BY   col1 [,  col2 ..]]
   ORDER   BY   col1  [ asc | desc ][,  col2  [ asc | desc ]...])
AS   rownum
 FROM   table_name )
WHERE   rownum  <=  N  [ AND   conditions ]
```

The syntax is similar to the original TopN syntax. You only need to exclude the rownum field from the outer query.

Note:
If rownum is excluded, pay special attention to the definition of the primary keys of the result table. If the definition is incorrect, the TopN query result is incorrect. If rownum is excluded, the primary keys must be those in the key list at the GROUP BY node before the TopN statement.

- Example of no ranking

  This example is a simplified case from a customer in the video industry. Heavy traffic is generated when each video is distributed. Based on the video traffic, you can identify the most popular videos. The following example identifies the top 5 videos that consume the most traffic per minute.

  - Test statements

```
-- Read the original data storage table from Log
 Service .
 CREATE TABLE sls_cdnlog _stream (
 vid VARCHAR , -- video id
 rowtime TIMESTAMP , -- Identify the time when the
 videos are watched .
 response_s ize BIGINT , -- Identify the traffic for
 watching the videos .
 WATERMARK FOR rowtime as withOffset ( rowtime , 0 )
) WITH (
 type =' sls ',
...
);

-- Compute the consumed bandwidth by video ID in
 the 1 - minute window .
 CREATE VIEW cdnvid_gro up_view AS
 SELECT vid ,
 TUMBLE_STA RT ( rowtime , INTERVAL ' 1 ' MINUTE ) AS
 start_time ,
 SUM ( response_s ize ) AS rss
 FROM sls_cdnlog _stream
 GROUP BY vid , TUMBLE ( rowtime , INTERVAL ' 1 ' MINUTE
 );

-- Create the result table .
 CREATE TABLE hbase_out_ cdnvidtopl og (
 vid VARCHAR ,
 rss BIGINT ,
 start_time VARCHAR ,
   -- Do not store the rownum field in the result
   table .
   -- Pay special attention to the definition of
 the primary keys . The primary keys must be those
```

```
  in   the   key   list   at   the   GROUP   BY   node   before
 the   TopN   statement .
 PRIMARY   KEY ( start_time ,  vid )
)  WITH  (
 type =' RDS ',
...
);

--  Identify   and   export   the   IDs   of   the   top   5
 videos   that   consume   the   most   traffic   per   minute .
 INSERT   INTO   hbase_out_  cdnvidtopl  og

--  The   outer   query   cannot   include   the   rownum   field
 .
 SELECT   vid ,  rss ,  start_time   FROM
(
 SELECT
 vid ,  start_time ,  rss ,
 ROW_NUMBER ()  OVER  ( PARTITION   BY   start_time   ORDER   BY
 rss   DESC )  as   rownum ,
 FROM
 cdnvid_gro  up_view
)
 WHERE   rownum  <=  5 ;
```

- **Test data**

vid (VARCHAR)	rowtime (Timestamp)	response_size (BIGINT)
10000	`2017 - 12 - 18 15 : 00 : 10`	2000
10000	`2017 - 12 - 18 15 : 00 : 15`	4000
10000	`2017 - 12 - 18 15 : 00 : 20`	3000
10001	`2017 - 12 - 18 15 : 00 : 20`	3000
10002	`2017 - 12 - 18 15 : 00 : 20`	4000
10003	`2017 - 12 - 18 15 : 00 : 20`	1000
10004	`2017 - 12 - 18 15 : 00 : 30`	1000
10005	`2017 - 12 - 18 15 : 00 : 30`	5000
10006	`2017 - 12 - 18 15 : 00 : 40`	6000

vid (VARCHAR)	rowtime (Timestamp)	response_size (BIGINT)
10007	`2017 - 12 - 18 15 : 00 : 50`	8000

- Test result

start_time (VARCHAR)	vid (VARCHAR)	rss (BIGINT)
`2017 - 12 - 18 15 : 00 : 00`	10000	9000
`2017 - 12 - 18 15 : 00 : 00`	10007	8000
`2017 - 12 - 18 15 : 00 : 00`	10006	6000
`2017 - 12 - 18 15 : 00 : 00`	10005	5000
`2017 - 12 - 18 15 : 00 : 00`	10002	4000

6.8.9 CEP statement

As a complex event processing (CEP) statement, MATCH_RECOGNIZE is used to identify events that conform to specified rules from input data streams and generate output events in the specified way.

Syntax

```
 SELECT  [  ALL  |  DISTINCT  ]
{ * |  projectIte  m  [,  projectIte  m  ]* }
 FROM  tableExpre  ssion
[ MATCH_RECO  GNIZE  (
[ PARTITION  BY  { partitionI  tem  [,  partitionI  tem ]*}]
[ ORDER  BY  { orderItem  [,  orderItem ]*}]
[ MEASURES  { measureIte  m  AS  col  [,  measureIte  m  AS  col
 ]*}]
[ ONE  ROW  PER  MATCH | ALL  ROWS  PER  MATCH | ONE  ROW  PER
   MATCH  WITH  TIMEOUT  ROWS | ALL  ROWS  PER  MATCH  WITH
 TIMEOUT  ROWS ]
[ AFTER  MATCH  SKIP ]
 PATTERN  ( patternVar  iable [ quantifier ] [  patternVar  iable [
 quantifier ]]*)  WITHIN  intervalEx  pression
 DEFINE  { patternVar  iable  AS  patternDef  inationExp  ression
 [,  patternVar  iable  AS  patternDef  inationExp  ression ]*}
)];
```

Name	Description
PARTITION BY	The column used for partitioning. This parameter is optional.
ORDER BY	The column used for sorting. You can specify multiple columns. However, the first column used for sorting must be the `EVENT TIME` or `PROCESS TIME` column. This parameter is optional.
MEASURES	The way to construct an output event based on the input events that are successfully matched.
ONE ROW PER MATCH	Specifies that only one output event will be generated upon each successful match .
ONE ROW PER MATCH WITH TIMEOUT ROWS	Specifies that an output event will be generated upon each successful match or each timeout. The timeout interval is defined by the `WITHIN` statement in the `PATTERN` statement.
ALL ROW PER MATCH	Specifies that an output event will be generated for each input event upon each successful match.
ALL ROW PER MATCH WITH TIMEOUT ROWS	Specifies that an output event will be generated for each input event upon each successful match or each timeout. The timeout interval is defined by the `WITHIN` statement in the `PATTERN` statement.
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]	This parameter is optional. The default value is `ONE ROW PER MATCH`.
AFTER MATCH SKIP TO NEXT ROW	Specifies that the next match following a successful match will start from the next event following the first event in the sequence of successfully matched events.

Name	Description
AFTER MATCH SKIP PAST LAST ROW	Specifies that the next match following a successful match will start from the next event following the last event in the sequence of successfully matched events.
AFTER MATCH SKIP TO FIRST patternItem	Specifies that the next match following a successful match will start from the first event corresponding to patternItem in the sequence of successfully matched events.
AFTER MATCH SKIP TO LAST patternItem	Specifies that the next match following a successful match will start from the last event corresponding to patternItem in the sequence of successfully matched events.
PATTERN	The rule to which the sequence of events to be identified conforms. The rule is defined in parentheses `()` and consists of a series of custom patternVariables. Note: · If two patternVariables are separated with a space, the events that conform to the patternVariables are next to each other and there are no other events between these events. · If two patternVariables are separated with an arrow sign (`->`), other events may exist between the events that conform to the patternVariables.

- `Quantifier`

  The `quantifier` specifies the number of occurrences of events that meet the `patternVar iable` definition.

Value	Description
*	Zero or multiple times
+	Once or multiple times
?	Zero or once

Value	Description
{n}	n times
{n,}	Greater than or equal to n times
{n, m}	Greater than or equal to n times, and smaller than or equal to m times
{,m}	Smaller than or equal to m times

A greedy match is performed by default. For example, if the pattern is `A -> B +` and the input is `a b1 , b2 , b3` , the output is `a b1 , a b1 b2 , a b1 b2 b3` . To perform a non-greedy match, append the quantifier with a question mark (`?`).

- `*?`
- `+?`
- `{ n }?`
- `{ n ,}?`
- `{ n , m }?`
- `{, m }?`

Then, the output generated for the pattern and input in the preceding example changes to `a b1 , a b2 , a b1 b2 , a b3 , a b2 b3 , a b1 b2 b3` .

Note:

- The WITHIN statement defines the maximum time span of events that conform to the specified rule in a sequence.
- Static window format: `INTERVAL ‘ string ’ timeUnit [ TO timeUnit ]`. Example: `INTERVAL ‘ 10 ’ SECOND , INTERVAL ‘ 45 ’ DAY , INTERVAL ‘ 10 : 20 ’ MINUTE TO SECOND , INTERVAL ‘ 10 : 20 . 10 ’ MINUTE TO SECOND , INTERVAL ‘ 10 : 20 ’ HOUR TO MINUTE , INTERVAL ‘ 1 - 5 ’ YEAR TO MONTH` .
- Dynamic window format: `INTERVAL intervalEx pression` . Example: `INTERVAL A . windowTime + 10` , where A indicates the first patternVariable in the pattern definition. The intervalExpression definition

can use a patternVariable in the pattern definition. However, only the first patternVariable in the pattern definition can be used currently. In intervalExpression, you can use a UDF. The result of intervalExpression indicates the window size. The result must be of the long type and in the unit of milliseconds.

- The DEFINE statement defines the meanings of patternVariables in the PATTERN statement. If a patternVariable is not defined in the DEFINE statement, the patternVariable is valid for each event.

- MEASURES and DEFINE statement functions

Function	Description
Row Pattern Column References	This function is in the format of `patternVariable . col`. It is used to access the specified column of an event that conforms to `patternVariable`.
PREV	This function can be used only in the DEFINE statement and generally works with `Row Pattern Column References`. The PREV function is used to access the specified column of the previous event with a specified offset before the event that conforms to the specified pattern. Example: `DOWN AS DOWN . price < PREV ( DOWN . price )`. `PREV ( A . price )` indicates the `price` column value of the previous event before the current event. Note that `DOWN . price` is equivalent to `PREV ( DOWN . price , 0 )` and `PREV ( DOWN . price )` is equivalent to `PREV ( DOWN . price , 1 )`.

Function	Description
FIRST or LAST	These functions generally work with `Row Pattern Column References`. The FIRST or LAST function is used to access the event with a specified offset in the sequence of events that conform to the specified pattern. For example, `FIRST ( A . price , 3 )` indicates the fourth event in the sequence of events that conform to pattern A, and `LAST ( A . price , 3 )` indicates the last but three event in the sequence of events that conform to pattern A.

- `Output columns`

Function	Output column
ONE ROW PER MATCH	The columns specified by `PARTITION BY` and `MEASURES` are included. The columns specified by `PARTITION BY` do not need to be specified in `MEASURES` again.
ONE ROW PER MATCH WITH TIMEOUT ROWS	An output event will be generated upon a successful match or a timeout. The timeout interval is defined by the `WITHIN` statement in the `PATTERN` statement.

Note:

- When you define the PATTERN statement, we recommend that you also define the WITHIN statement. If the WITHIN statement is not defined, the state size may grow larger.
- The first column specified by ORDER BY must be the EVENT TIME or PROCESS TIME column.

Example

- Example syntax

```
SELECT *
```

```
 FROM   Ticker   MATCH_RECO  GNIZE  (
 PARTITION   BY   symbol
 ORDER   BY   tstamp
 MEASURES   STRT . tstamp   AS   start_tsta  mp ,
 LAST ( DOWN . tstamp )  AS   bottom_tst  amp ,
 LAST ( UP . tstamp )  AS   end_tstamp
 ONE   ROW   PER   MATCH
 AFTER   MATCH   SKIP   TO   NEXT   ROW
 PATTERN  ( STRT   DOWN +  UP +)  WITHIN   INTERVAL  ' 10 '  SECOND
 DEFINE
 DOWN   AS   DOWN . price  <  PREV ( DOWN . price ),
 UP   AS   UP . price  >  PREV ( UP . price )
)  MR
 ORDER   BY   MR . symbol ,  MR . start_tsta  mp ;
```

- **Test data**

timestamp (TIMESTAMP)	card_id (VARCHAR)	location (VARCHAR)	action (VARCHAR)
`2018 - 04 - 13 12 : 00 : 00`	1	WW	Tom
`2018 - 04 - 13 12 : 05 : 00`	1	WW1	Tom
`2018 - 04 - 13 12 : 10 : 00`	1	WW2	Tom
`2018 - 04 - 13 12 : 20 : 00`	1	WW	Tom

- **Test case syntax**

```
 CREATE   TABLE   datahub_st  ream  (
    ` timestamp `               TIMESTAMP ,
     card_id                    VARCHAR ,
     location                   VARCHAR ,
    ` action `                  VARCHAR ,
     WATERMARK   wf   FOR  ` timestamp `  AS   withOffset (`
 timestamp `,  1000 )
)  WITH  (
     type  = ' datahub '
    ...
);
 CREATE   TABLE   rds_out  (
     start_time  stamp               TIMESTAMP ,
     end_timest  amp                 TIMESTAMP ,
     card_id                       VARCHAR ,
     event                         VARCHAR
)  WITH  (
     type = ' rds '
    ...
);

--  Case   descriptio  n
--  When   payments   with   a   card   with   a   unique   ID   (
 card_id )  occur   at   two   different   locations   within   10
```

```
 minutes , an  alert  is  triggered . This  helps  monitor
 credit  card  identity  fraud .

-- Define  the  computatio nal  logic  as  follows :
 insert  into  rds_out
 select
` start_time  stamp `,
` end_timest  amp `,
 card_id , ` event `
 from  datahub_st  ream
 MATCH_RECO  GNIZE  (
     PARTITION  BY  card_id   --  Partition  data  by  card_id
 . The  data  with  the  same  card  ID  is  allocated  to
   the  same  compute  node .
     ORDER  BY  ` timestamp `  --  Sort  events  by  time  in
   a  window .
     MEASURES                 --  Define  how  to  construct  an
   output  event  based  on  the  input  events  that  are
 successful  ly  matched .
         e2 .` action `  as  ` event `,
         e1 .` timestamp `  as  ` start_time  stamp `,  --  Define
   the  time  of  the  first  event  as  start_time  stamp .
         LAST ( e2 .` timestamp `)  as  ` end_timest  amp `--
 Define  the  time  of  the  last  event  as  end_timest
 amp .
     ONE  ROW  PER  MATCH            --  Generate  an  output
 event  upon  a  successful  match .
     AFTER  MATCH  SKIP  TO  NEXT  ROW --  Start  the  next
 match  from  the  next  row  upon  a  successful  match .
     PATTERN  ( e1  e2 +)  WITHIN  INTERVAL  ' 10 '  MINUTE  --
 Define  two  events  e1  and  e2 .
     DEFINE                  --  Define  the  meanings  of
 patternVar  iables  in  the  PATTERN  statement .
         e1  as  e1 . action  = ' Tom ',   --  Mark  the
 action  of  e1  as  Tom .
         e2  as  e2 . action  = ' Tom '  and  e2 . location  <>
 e1 . location  --  Mark  the  action  of  e2  as  Tom .  The
   locations  of  e1  and  e2  are  different .

);
```

- **Test result**

start_timestamp (TIMESTAMP)	end_timestamp (TIMESTAMP)	card_id (VARCHAR)	event (VARCHAR)
`2018 - 04 - 13 20 : 00 : 00 . 0`	`2018 - 04 - 13 20 : 05 : 00 . 0`	1	Tom
`2018 - 04 - 13 20 : 05 : 00 . 0`	`2018 - 04 - 13 20 : 10 : 00 . 0`	1	Tom

6.9 Window functions

6.9.1 Window function overview

This topic describes the window functions, time attributes, and window types that Flink SQL supports.

Window functions

Flink SQL supports aggregation over infinite windows (you do not need to explicitly add any windows in your SQL query statement). In addition, Flink SQL supports aggregation over a specific window. For example, to count the number of users who clicked a specific URL in the past minute, you can define a window for collecting user clicks in the past minute. Then, you can compute the data in the window to obtain the result.

Flink SQL supports window aggregate and over aggregate. This topic describes window aggregate. Window aggregate supports the following two time attributes : event time and processing time. For each time attribute, Flink SQL supports the following three window types: tumbling window, sliding window, and session window.

Time attributes

Flink SQL supports two time attributes. Realtime Compute aggregates data in windows based on these two time attributes.

- Event time: The event time that you provide in the table schema, which is generally the original creation time of the data.
- Processing time: The local time at which the system processes an event.

For more information about the time attributes supported by Realtime Compute, see Time attributes.

6.9.2 Tumbling window

This topic describes how to use the tumbling window function of Realtime Compute.

What is a tumbling window

By using tumbling windows, you assign each element to a window with the specified size. Generally, tumbling windows are fixed in size and do not overlap each other. For example, if a 5-minute tumbling window is defined, an infinite data stream is divided by period into windows such as `[ 0 : 00 , 0 : 05 )`, `[ 0 : 05 , 0 : 10 )`, `[ 0 : 10 , 0 : 15 )`. The following figure shows a 30-second tumbling window.

Syntax

The TUMBLE function is used to define a tumbling window in a GROUP BY clause.

```
 TUMBLE (< time - attr >, < size - interval >)
< size - interval >:  INTERVAL  ' string '  timeUnit
```

> **Note:**
>
> The <time-attr data-spm-anchor-id="a2762.11472859.0.i151.7ca4203bEk6mXa">`< time - attr >` parameter must be a valid time attribute in a stream to specify whether the time is the processing time or event time.</time-attr> For more information about how to define the time attribute and watermark, see Window function overview.

Window identifier functions

A window identifier function specifies the start or end time of a window, or the window time attribute for aggregation of lower-level windows.

Window identifier function	Return type	Description
`TUMBLE_STA RT ( time - attr , size - interval )`	TIMESTAMP	Return the start time (border value) of the window. For example, if the window is `[ 00 : 10 , 00 : 15 )`, `00 : 10` is returned.
`TUMBLE_END ( time - attr , size - interval )`	TIMESTAMP	Return the end time (border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 ]`, `00 : 15` is returned.

Window identifier function	Return type	Description
`TUMBLE_ROW TIME ( time - attr , size - interval )`	TIMESTAMP (rowtime-attr)	Return the end time (not the border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 ]`, `00 : 14 : 59 . 999` is returned. The return value is a rowtime attribute, based on which time type operations such as window cascading can be performed.

Example

The following describes how to compute the number of clicks per user and minute on the specified website.

- Test data

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 00 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 10 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 49 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 01 : 05 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 01 : 58 . 0`
Timo	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 02 : 10 . 0`

- Test statements

```
CREATE   TABLE   user_click  s (
username   varchar ,
click_url   varchar ,
ts   timeStamp ,
WATERMARK   wk   FOR   ts   as   withOffset ( ts ,  2000 ) --
Define   a   watermark   for   rowtime .
```

```
) with (
 type =' datahub ',
...
);

 CREATE  TABLE  tumble_out  put (
 window_sta rt  TIMESTAMP ,
 window_end  TIMESTAMP ,
 username  VARCHAR ,
 clicks  BIGINT
) with (
 type =' RDS '
);

 INSERT  INTO  tumble_out  put
 SELECT
 TUMBLE_STA RT ( ts , INTERVAL  ' 1 '  MINUTE ),
 TUMBLE_END ( ts , INTERVAL  ' 1 '  MINUTE ),
 username ,
 COUNT ( click_url )
 FROM  user_click  s
 GROUP  BY  TUMBLE ( ts , INTERVAL  ' 1 '  MINUTE ), username
```

- Test result

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
`2017 - 10 - 10 10 : 00 : 00 . 0`	`2017 - 10 - 10 10 : 01 : 00 . 0`	Jark	3
`2017 - 10 - 10 10 : 01 : 00 . 0`	`2017 - 10 - 10 10 : 02 : 00 . 0`	Jark	2
`2017 - 10 - 10 10 : 02 : 00 . 0`	`2017 - 10 - 10 10 : 03 : 00 . 0`	Timo	1

6.9.3 Hop window

This article describes how to use the real-time calculation sliding window function.

Note:

Real-time computing HOP window (HOP) cannot be used together with last_value, first_value, or TopN functions.

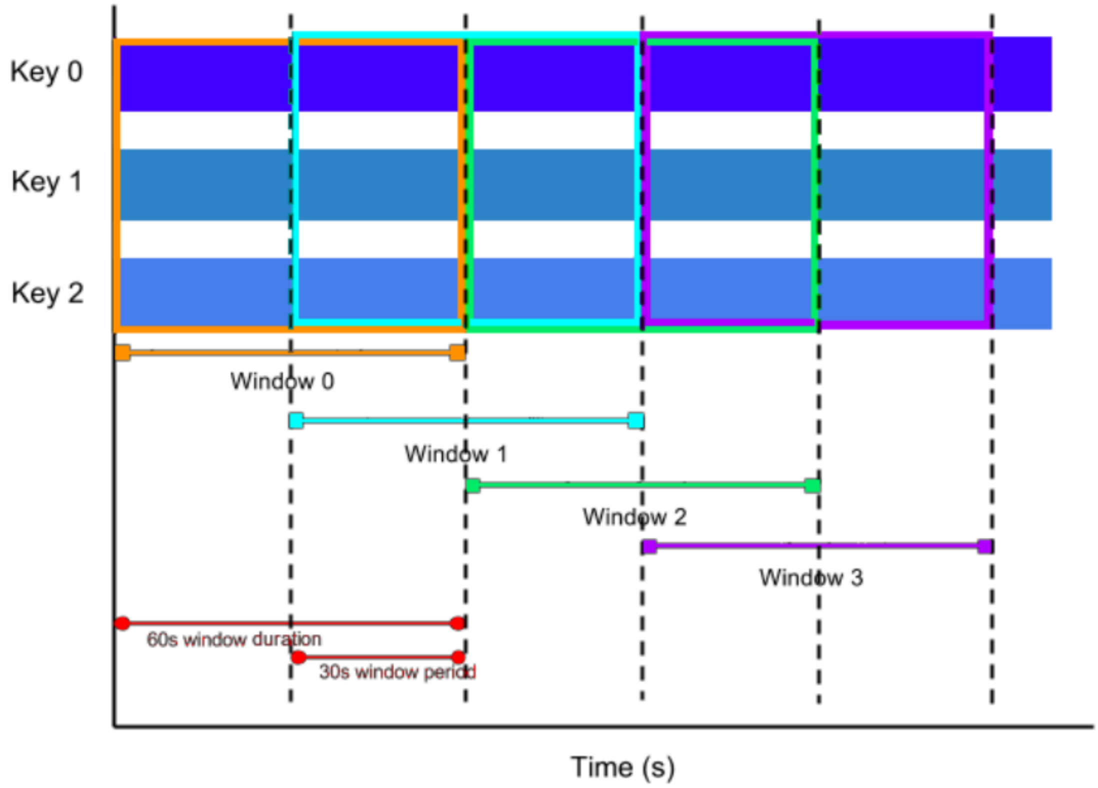
What is a sliding window?

A hop window is also called a sliding window. Unlike tumble windows, hop windows can overlap each other.

The sliding window has two parameters: slide and size. The slide parameter specifies the length of the sliding step, and the size parameter specifies the size of the window.

- `Slide < Size` :The windows overlap each other, and each element is assigned to multiple windows.
- `Slide = Size` :The windows are tumbling windows.
- `Slide > Size` :The windows do not overlap each other but are separated by gaps.

Generally, most elements match multiple windows, and the windows overlap. Sliding windows are useful in computing moving averages. For example, to compute the data average in the past 5 minutes every 10 seconds, you can set slide to 10 seconds and size to 5 minutes. The following figure shows sliding windows whose slide is 30 seconds and size is 1 minute.



Sliding Window function syntax

The HOP function is used to define a sliding window in a GROUP BY clause.

```
 HOP  (< time - attr >, < slide - interval >, < size - interval >)
< slide - interval >:  INTERVAL  ' string '  timeUnit
< size - interval >:  INTERVAL  ' string '  timeUnit
```

Note:

The `< time - attr >` parameter must be a valid time attribute in a stream to specify whether the time is the processing time or event time. Refer Window function overviewto learn how to define Time attributes and Watermark.

Window identifier functions

A window identifier function specifies the start or end time of a window, or the window time attribute for aggregation of lower-level windows.

Window identifier function	Return type	Description
`HOP_START (< time - attr >, < slide - interval >, < size - interval >)`	TIMESTAMP	Return the start time (include border value) of the window. For example, if the window is `[ 00 : 10 , 00 : 15 )`, `00 : 10` is returned.
`HOP_END (< time - attr >, < slide - interval >, < size - interval >)`	TIMESTAMP	Return the end time (include border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 )`, `00 : 15` is returned.
`HOP_ROWTIM E (< time - attr >, < slide - interval >, < size - interval >)`	TIMESTAMP (rowtime-attr)	Return the end time (exclude the border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 )`, `00 : 14 : 59 . 999` is returned. The return value is a rowtime attribute, based on which time type operations can be performed. The function is applicable only to windows based on event time.

Window identifier function	Return type	Description
`HOP_PROCTI ME (< time - attr >, < slide - interval >, < size - interval >)`	TIMESTAMP (rowtime-attr)	Return the end time (exclude the border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 )]`, `00 : 14 : 59 . 999` is returned. The return value is a proctime attribute, based on which time type operations can be performed. It can only be used in a window based on processing time.

Example

The following example describes how to compute the number of clicks per user over the past minute every 30 seconds. That is, the 1-minute window slides every 30 seconds.

- Test data

username(VARCHAR)	click_url(VARCHAR)	ts(TIMESTAMP)
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 00 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 10 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 49 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 01 : 05 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 01 : 58 . 0`
Timo	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 02 : 10 . 0`

- Test statement

```
Create  table  user_click  s  (
    Username  VARCHAR ,
    Click_url  VARCHAR ,
```

```
    Ts   TIMESTAMP ,
    WATERMARK   wk   FOR   ts   as   withoffset  ( ts ,  2000 ) --
 Define   a   watermark   for   rowtime .
)  WITH  ( TYPE  = ' datahub  ',
        ...) ;
 CREATE   TABLE   hop_output  (
    window_sta  rt   TIMESTAMP ,
    window_end   TIMESTAMP ,
    username   VARCHAR ,
    clicks   BIGINT
)  WITH  ( TYPE  = ' rds  ',
        ...) ;
 INSERT   INTO
    hop_output
 SELECT   statement
    HOP_START  ( ts ,  INTERVAL  ' 30 '  SECOND ,  INTERVAL  ' 1 '
 MINUTE  ),
    HOP_END  ( ts ,  INTERVAL  ' 30 '  SECOND ,  INTERVAL  ' 1 '
 MINUTE  ),
    username ,
    COUNT  ( click_url )
 FROM
    user_click  s
 GROUP   BY
    HOP  ( ts ,  INTERVAL  ' 30 '  SECOND ,  INTERVAL  ' 1 '
 MINUTE  ),
    username
```

- **Test results**

window_start(TIMESTAMP)	window_end(TIMESTAMP)	username(VARCHAR)	clicks(BIGINT)
`2017 - 10 - 10 10 : 00 : 00 . 0`	`2017 - 10 - 10 10 : 01 : 00 . 0`	Jark	3
`2017 - 10 - 10 10 : 00 : 30 . 0`	`2017 - 10 - 10 10 : 01 : 30 . 0`	Jark	2
`2017 - 10 - 10 10 : 01 : 00 . 0`	`2017 - 10 - 10 10 : 02 : 00 . 0`	Jark	2
`2017 - 10 - 10 10 : 01 : 30 . 0`	`2017 - 10 - 10 10 : 02 : 30 . 0`	Jark	1
`2017 - 10 - 10 10 : 01 : 30 . 0`	`2017 - 10 - 10 10 : 02 : 30 . 0`	Timo	1

window_start(TIMESTAMP)	window_end(TIMESTAMP)	username(VARCHAR)	clicks(BIGINT)
`2017 - 10 - 10 10 : 02 : 00 . 0`	`2017 - 10 - 10 10 : 03 : 00 . 0`	Timo	1

6.9.4 Session window

This topic describes how to use the session window function of Realtime Compute.

What is a session window

A session window groups elements by session activity. Unlike tumbling and sliding windows, session windows do not overlap and are not fixed in size. If a session window does not receive any elements within a certain period, the session is disconnected and the window is closed.

A session window is configured by using a gap, which defines the length of an inactive period. For example, a data stream that represents a mouse click activity may include highly clustered mouse click events, separated with idle periods. If data arrives after the specified shortest gap, a new window is opened.

The following figure shows a session window. Note that different keys have different windows due to data distribution differences.

Session window function syntax

The SESSION function is used to define a session window in a GROUP BY clause.

```
 SESSION (< time - attr >, < gap - interval >)
< gap - interval >:  INTERVAL  ' string '  timeUnit
```

Note:

The <time-attr data-spm-anchor-id="a2762.11472859.0.i151.7ca4203bEk6mXa">< `time - attr >` parameter must be a valid time attribute in a stream to specify whether the time is the processing time or event time.</time-attr> For more information about how to define the time attribute and watermark, see Window function overview.

Window identifier functions

A window identifier function specifies the start or end time of a window, or the window time attribute for aggregation of lower-level windows.

Window identifier function	Return type	Description
`SESSION_START (< time - attr >, < gap - interval >)`	TIMESTAMP	Return the start time (border value) of the window. For example, if the window is `[ 00 : 10 , 00 : 15 )`, `00 : 10` is returned.
`SESSION_END (< time - attr >, < gap - interval >)`	TIMESTAMP	Return the end time (border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 )`, `00 : 15` is returned.
`SESSION_ROWTIME (< time - attr >, < gap - interval >)`	TIMESTAMP (rowtime-attr)	Return the end time (not the border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 )`, `00 : 14 : 59 . 999` is returned. The return value is a rowtime attribute, based on which time type operations such as window cascading can be performed. The function is applicable only to windows based on event time.

Window identifier function	Return type	Description
`SESSION_PR OCTIME (< time - attr >, < gap - interval >)`	TIMESTAMP (rowtime-attr)	Return the end time (not the border value) of the window. For example, if the window is `[ 00 : 00 , 00 : 15 )`, `00 : 14 : 59 . 999` is returned. The return value is a proctime attribute, based on which time type operations such as window cascading can be performed. The function is applicable only to windows based on processing time.

Example

The following example describes how to compute the number of clicks per user during each active session. The session timeout interval is 30 seconds.

- Test data

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 00 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 10 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 00 : 49 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 01 : 05 . 0`
Jark	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 01 : 58 . 0`
Timo	`http :// taobao . com / xxx`	`2017 - 10 - 10 10 : 02 : 10 . 0`

- Test statements

```
CREATE TABLE user_click s (
username varchar ,
click_url varchar ,
```

```
 ts   timeStamp ,
 WATERMARK   wk   FOR   ts   as   withOffset ( ts ,  2000 ) --
 Define   a   watermark   for   rowtime .
)  with  (
 type =' datahub ',
...
);

 CREATE   TABLE   session_ou  tput (
 window_sta  rt   TIMESTAMP ,
 window_end   TIMESTAMP ,
 username   VARCHAR ,
 clicks   BIGINT
)  with  (
 type =' rds '
);

 INSERT   INTO   session_ou  tput
 SELECT
 SESSION_ST  ART ( ts ,  INTERVAL  ' 30 '  SECOND ),
 SESSION_EN  D ( ts ,  INTERVAL  ' 30 '  SECOND ),
 username ,
 COUNT ( click_url )
 FROM   user_click  s
 GROUP   BY   SESSION ( ts ,  INTERVAL  ' 30 '  SECOND ),  username
```

- Test result

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
`2017 - 10 - 10 10 : 00 : 00 . 0`	`2017 - 10 - 10 10 : 00 : 40 . 0`	Jark	2
`2017 - 10 - 10 10 : 00 : 49 . 0`	`2017 - 10 - 10 10 : 01 : 35 . 0`	Jark	2
`2017 - 10 - 10 10 : 01 : 58 . 0`	`2017 - 10 - 10 10 : 02 : 28 . 0`	Jark	1
`2017 - 10 - 10 10 : 02 : 10 . 0`	`2017 - 10 - 10 10 : 02 : 40 . 0`	Timo	1

6.9.5 OVER window

An OVER window is a standard window used by traditional databases. It is different from the GROUP BY window. In a stream that applies the OVER window, each element corresponds to an OVER window, whose elements are a set of elements adjacent to the current element. Elements of a stream are distributed across multiple windows. In the implementation of Flink SQL windows, the row determined by each element that triggers computing is the last row of the window where the element is located.

In a stream that applies the OVER window, each element corresponds to an OVER window and triggers data computing once. In the underlying implementation of Realtime Compute, the OVER window data is managed in a global and unified manner (only one copy of data is stored). Logically, an OVER window is maintained to perform window computing for each element. Expired data will be cleared after the computing is completed.

Syntax

```
SELECT
    agg1 ( col1 )  OVER  ( definition  1 )  AS   colName ,
   ...
    aggN ( colN )  OVER  ( definition  1 )  AS   colNameN
FROM   Tab1
```

Note:

- OVER (definition1) must be the same for agg1 through aggN.
- The alias specified by AS can be queried by using an outer SQL statement.

Type

In Flink SQL, the OVER window definition follows standard SQL syntax. Traditionally, OVER windows are not classified into finer-grained window types. To help you gain a better understanding of the OVER window semantics, we classify OVER windows into the following two types based on the ways of determining the computed row:

- ROWS OVER window: Each row of elements is treated as a new computed row. That is, each row corresponds to a new window.
- RANGE OVER window: All element rows with the same timestamp value are treated as the same computed row and belong to the same window.

Attribute

Orthogonal attribute	proctime	eventtime
rows	√	√
range	√	√

- rows: A window is determined based on the actual row of an element.

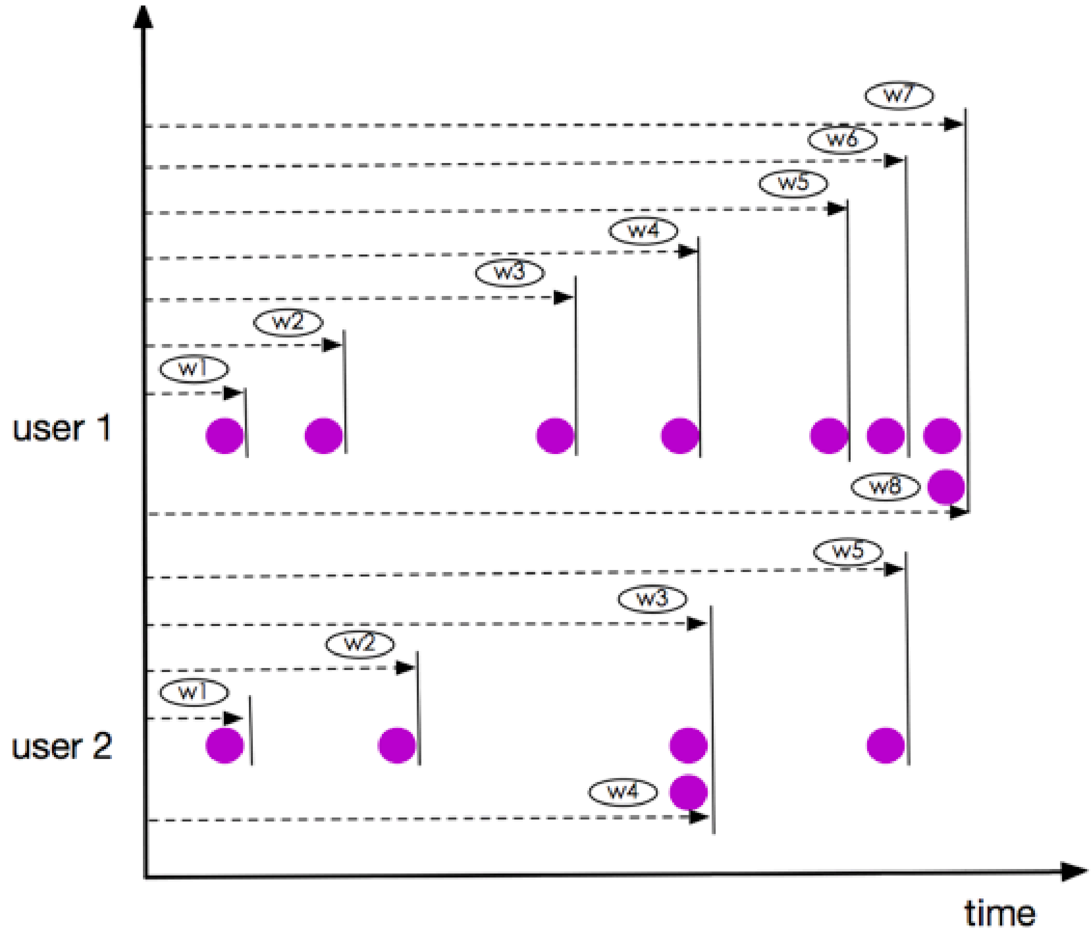
- range: A window is determined based on the actual value (timestamp value) of an element.

ROWS OVER window semantics

- Window data

  In a ROWS OVER window, each element determines a window. ROWS OVER windows are classified into Unbounded and Bounded ROWS OVER windows.

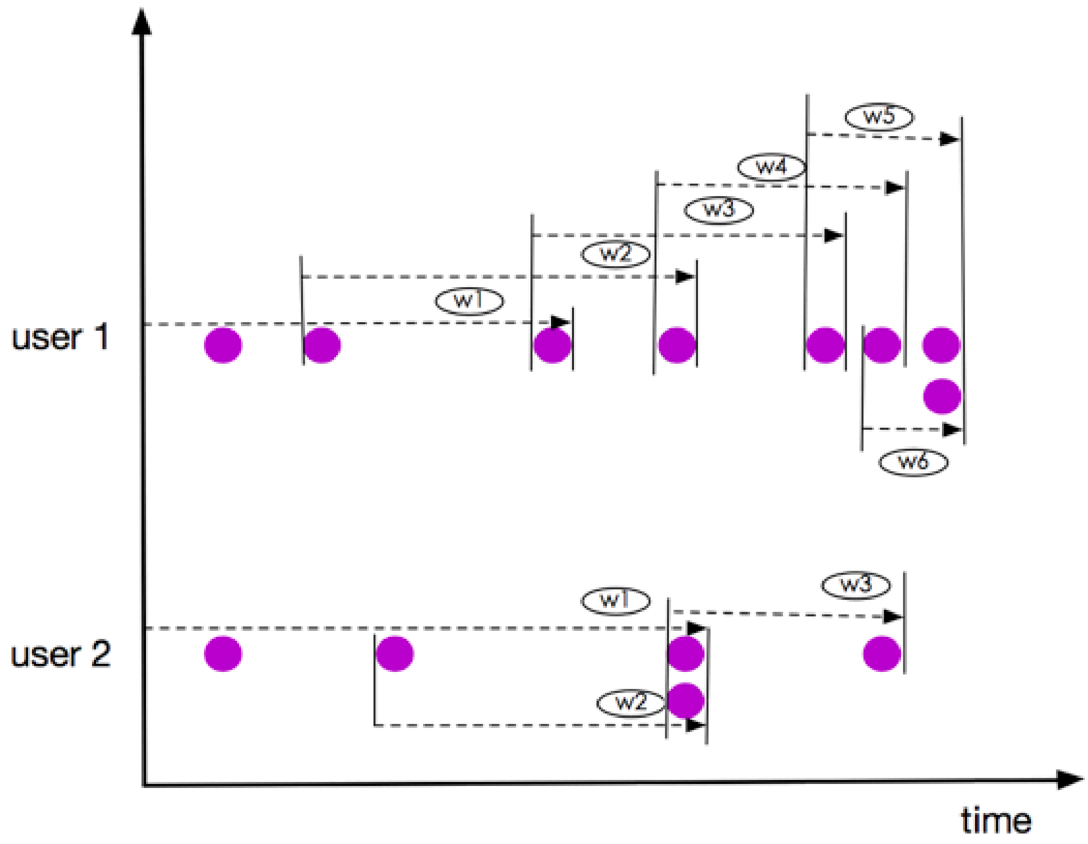
  The following figure shows Unbounded ROWS OVER window data.



> Note:
> As shown in the preceding figure, elements of windows w7 and w8 of user 1 arrive at the same time, and so do elements of windows w3 and w4 of user 2. However,

the elements are in different windows. In this regard, a ROWS OVER window is different from a RANGE OVER window.

The following figure shows Bounded ROWS OVER window data, in which a window has three elements (two elements in PRECEDING state).



Note:
As shown in the preceding figure, windows w5 and w6 of user 1 have elements that arrive at the same time, and so do windows w2 and w3 of user 2. However, the elements are in different windows. In this regard, a ROWS OVER window is different from a RANGE OVER window.

- Window syntax

```
SELECT
    agg1 ( col1 )  OVER (
    [ PARTITION   BY  ( value_expr  ession1 ,..., value_expr
essionN )]
     ORDER   BY   timeCol
     ROWS
     BETWEEN  ( UNBOUNDED  |  rowCount )  PRECEDING   AND   CURRENT
  ROW )  AS   colName , ...
FROM   Tab1
```

- value_expression: specifies the value expression used for partitioning.
- timeCol: specifies the time field used for sorting elements.
- rowCount: specifies the number of rows to be traced before the current row.

· Example

Use a Bounded ROWS OVER window as an example. Here is a merchandise shelving table, which lists the ID, type, shelving time, and price of merchandises. Compute the highest price among three similar merchandises before the current merchandise hits shelves.

Test data

itemID	itemType	onSellTime	price
ITEM001	Electronic	`2017 - 11 - 11 10 : 01 : 00`	20
ITEM002	Electronic	`2017 - 11 - 11 10 : 02 : 00`	50
ITEM003	Electronic	`2017 - 11 - 11 10 : 03 : 00`	30
ITEM004	Electronic	`2017 - 11 - 11 10 : 03 : 00`	60
ITEM005	Electronic	`2017 - 11 - 11 10 : 05 : 00`	40
ITEM006	Electronic	`2017 - 11 - 11 10 : 06 : 00`	20
ITEM007	Electronic	`2017 - 11 - 11 10 : 07 : 00`	70
ITEM008	Clothes	`2017 - 11 - 11 10 : 08 : 00`	20

Test code

```
CREATE   TABLE   tmall_item (
   itemID   VARCHAR ,
   itemType   VARCHAR ,
   onSellTime   TIMESTAMP ,
   price   DOUBLE ,
   WATERMARK   onSellTime   FOR   onSellTime   as   withOffset (
onSellTime ,  0 )
)
```

```
 WITH  (
   type  = ' sls ',
   ...
) ;

 SELECT
     itemID ,
     itemType ,
     onSellTime ,
     price ,
     MAX ( price )  OVER  (
         PARTITION   BY   itemType
         ORDER   BY   onSellTime
         ROWS   BETWEEN   2   preceding   AND   CURRENT   ROW )  AS
   maxPrice
   FROM   tmall_item
```

Test result

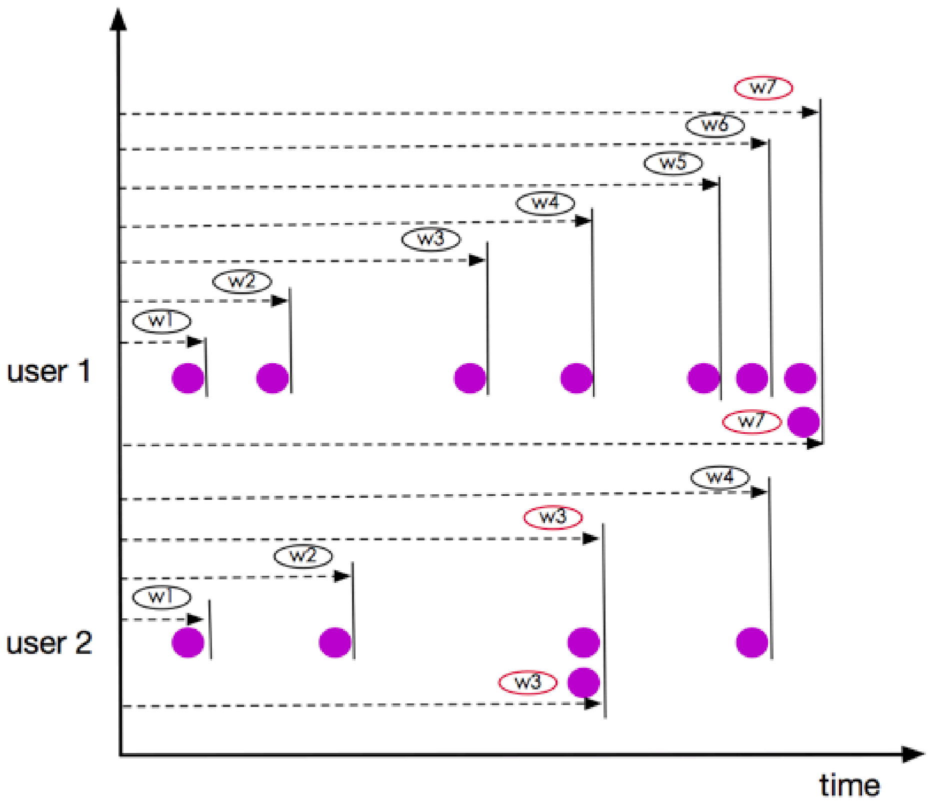
itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	`2017 - 11 - 11  10 : 01 : 00`	20	20
ITEM002	Electronic	`2017 - 11 - 11  10 : 02 : 00`	50	50
ITEM003	Electronic	`2017 - 11 - 11  10 : 03 : 00`	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	`2017 - 11 - 11  10 : 05 : 00`	40	60
ITEM006	Electronic	`2017 - 11 - 11  10 : 06 : 00`	20	60
ITEM007	Electronic	`2017 - 11 - 11  10 : 07 : 00`	70	70
ITEM008	Clothes	`2017 - 11 - 11  10 : 08 : 00`	20	20

RANGE OVER window semantics

- Window data

  In a RANGE OVER window, all element rows with the same element value (element timestamp) determine a window. RANGE OVER windows are classified into Unbounded and Bounded RANGE OVER windows.

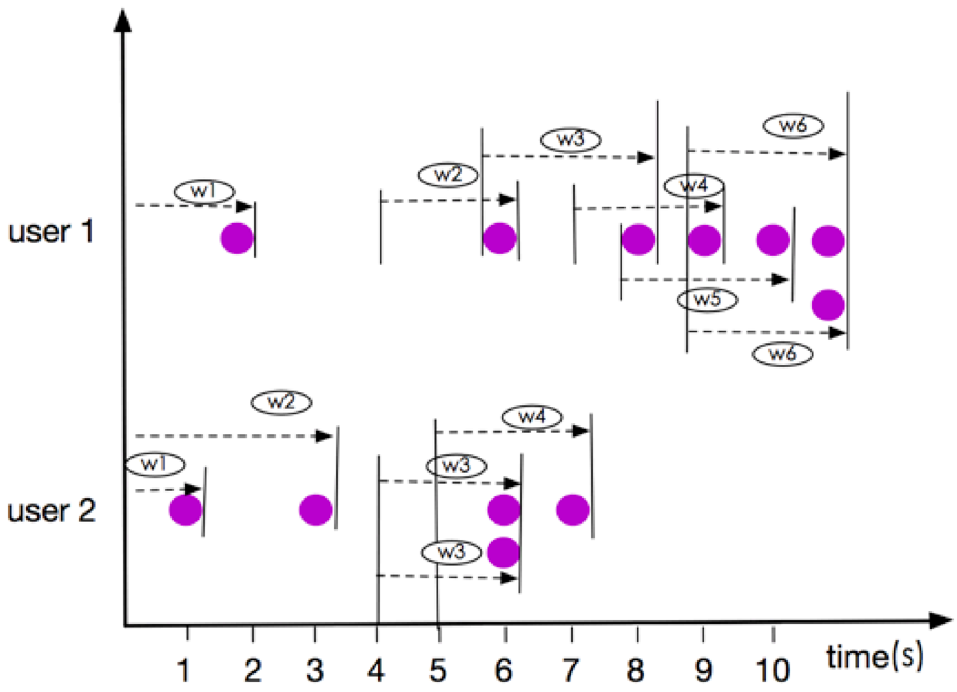
  The following figure shows Unbounded RANGE OVER window data.



As shown in the preceding figure, elements of the two w7 windows of user 1 arrive at the same time, and so do elements of the two w3 windows of user 2. The elements are in the same windows, respectively. In

this regard, a RANGE OVER window is different from a ROWS OVER window.

The following figure shows Bounded RANGE OVER window data, in which a 3-second window has an interval of 2 seconds.



Note:
As shown in the preceding figure, elements of the two w6 windows of user 1 arrive at the same time, and so do elements of the two w3 windows of user 2. The elements are in the same windows, respectively. In this regard, a RANGE OVER window is different from a ROWS OVER window.

- Window syntax

```
SELECT
    agg1 ( col1 )  OVER (
    [ PARTITION   BY  ( value_expr  ession1 ,..., value_expr
essionN )]
     ORDER   BY   timeCol
     RANGE
     BETWEEN  ( UNBOUNDED  |  timeInterv  al )  PRECEDING   AND
CURRENT   ROW ) AS   colName ,
...
FROM   Tab1
```

- value_expression: specifies the value expression used for partitioning.
- timeCol: specifies the time field used for sorting elements.
- timeInterval: specifies the time span from the current element row to the earliest element row to be traced backwards.

· Example

Use a Bounded RANGE OVER window as an example. Here is a merchandise shelving table, which lists the ID, type, shelving time, and price of merchandises. Compute the highest price among similar merchandises that hit shelves 2 minutes earlier than the current merchandise.

Test data

itemID	itemType	onSellTime	price
ITEM001	Electronic	2017 - 11 - 11 10 : 01 : 00	20
ITEM002	Electronic	2017 - 11 - 11 10 : 02 : 00	50
ITEM003	Electronic	2017 - 11 - 11 10 : 03 : 00	30
ITEM004	Electronic	2017 - 11 - 11 10 : 03 : 00	60
ITEM005	Electronic	2017 - 11 - 11 10 : 05 : 00	40
ITEM006	Electronic	2017 - 11 - 11 10 : 06 : 00	20
ITEM007	Electronic	2017 - 11 - 11 10 : 07 : 00	70
ITEM008	Clothes	2017 - 11 - 11 10 : 08 : 00	20

Test code

```
CREATE   TABLE   tmall_item (
   itemID   VARCHAR ,
   itemType   VARCHAR ,
   onSellTime   TIMESTAMP ,
   price   DOUBLE ,
```

```
    WATERMARK   onSellTime   FOR   onSellTime   as   withOffset (
 onSellTime ,  0 )
)
 WITH  (
   type  = ' sls ',
   ...
) ;

 SELECT
     itemID ,
     itemType ,
     onSellTime ,
     price ,
     MAX ( price )  OVER  (
         PARTITION   BY   itemType
         ORDER   BY   onSellTime
         RANGE   BETWEEN   INTERVAL  ' 2 '  MINUTE   preceding   AND
   CURRENT   ROW )  AS   maxPrice
   FROM   tmall_item
```

Test result

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	`2017 - 11 - 11 10 : 01 : 00`	20	20
ITEM002	Electronic	`2017 - 11 - 11 10 : 02 : 00`	50	50
ITEM003	Electronic	`2017 - 11 - 11 10 : 03 : 00`	30	50
ITEM004	Electronic	`2017 - 11 - 11 10 : 03 : 00`	60	60
ITEM005	Electronic	`2017 - 11 - 11 10 : 05 : 00`	40	60
ITEM006	Electronic	`2017 - 11 - 11 10 : 06 : 00`	20	40
ITEM007	Electronic	`2017 - 11 - 11 10 : 07 : 00`	70	70

itemID	itemType	onSellTime	price	maxPrice
ITEM008	Clothes	`2017 - 11 - 11 10 : 08 : 00`	20	20

6.10 Logical functions

6.10.1 =

This topic describes how to use the logical operation function = of Realtime Compute.

Syntax

```
A = B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 = int2 ;
```

- Test result

aa (INT)
97

6.10.2 >

This topic describes how to use the logical operation function > of Realtime Compute.

Syntax

```
A > B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is greater than B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	65	100

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2 ;
```

- Test result

aa (INT)
97

6.10.3 >=

This topic describes how to use the logical operation function >= of Realtime Compute.

Syntax

```
A  >=  B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is greater than or equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65
9	6	61

- Test statement

```
SELECT  int1  as  aa
FROM  T1
WHERE  int3  >=  int2 ;
```

- Test result

aa (INT)
97
9

6.10.4 <=

This topic describes how to use the logical operation function <= of Realtime Compute.

Syntax

```
A  <=  B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is smaller than or equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Test statement

```
SELECT  int1  as  aa
FROM  T1
WHERE  int3  <=  int2 ;
```

- Test result

aa (INT)
97
9

6.10.5 <

This topic describes how to use the logical operation function < of Realtime Compute.

Syntax

```
A < B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is smaller than B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2 ;
```

- Test result

aa (INT)
97
9

6.10.6 <>

This topic describes how to use the logical operation function <> of Realtime Compute.

Syntax

```
A <> B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is not equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	66	6

- Test statement

```
SELECT  int1  as  aa
FROM  T1
WHERE  int3  <>  int2 ;
```

- Test result

aa (INT)
97

6.10.7 AND

This topic describes how to use the logical operation function AND of Realtime Compute.

Syntax

```
A AND B
```

Input parameter

Name	Data type
A	BOOLEAN
B	BOOLEAN

Function description

TRUE is returned if both A and B are TRUE. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1 = 255 AND int3 = 65 ;
```

- Test result

aa (INT)
97

6.10.8 BETWEEN AND

This topic describes how to use the logical operation function BETWEEN AND of Realtime Compute.

Syntax

```
A BETWEEN AND B
```

```

```

Input parameter

Name	Data type
A	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
B	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
C	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME

Function description

This function selects a value within a data range defined by two other values.

Example 1

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
90	80	100
11	10	7

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   int1   BETWEEN   int2   AND   int3 ;
```

- Test result

aa (INT)
90

Example 2

- Test data

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
b	a	c

- Test statement

```
SELECT   var1   as   aa
FROM   T1
WHERE   var1   BETWEEN   var2   AND   var3 ;
```

- Test result

aa (VARCHAR)
b

Example 3

- Test data

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

- Test statement

```
SELECT   TIMESTAMP1   as   aa
FROM   T1
WHERE   TIMESTAMP1   BETWEEN   TIMESTAMP2   AND   TIMESTAMP3 ;
```

- Test result

aa (TIMESTAMP)
1969-07-20 20:17:30

6.10.9 IS NOT FALSE

This topic describes how to use the logical operation function IS NOT FALSE of Realtime Compute.

Syntax

```
A   IS   NOT   FALSE
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If A is TRUE, TRUE is returned. If A is FALSE, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT   int2   as   aa
FROM   T1
WHERE   int1 = 255   IS   NOT   FALSE ;
```

- Test result

aa (INT)
97

6.10.10 IS NOT NULL

This topic describes how to use the logical operation function IS NOT NULL of Realtime Compute.

Syntax

```
value   IS   NOT   NULL
```

Input parameter

Name	Data type
value	Any data type

Function description

If the value is `NULL`, `FALSE` is returned. Otherwise, `TRUE` is returned.

Example

- Test data

int1 (INT)	int2 (VARCHAR)
97	NULL
9	ww123

- Test statement

```
SELECT   int1   as   aa
FROM   T1
```

```
WHERE   int2   IS   NOT   NULL ;
```

- Test result

aa (INT)
9

6.10.11 IS NOT TRUE

This topic describes how to use the logical operation function IS NOT TRUE of Realtime Compute.

Syntax

```
A   IS   NOT   TRUE
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If A is TRUE, FALSE is returned. If A is FALSE, TRUE is returned.

Example

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   int1 = 25   IS   NOT   TRUE ;
```

- Test result

aa (INT)
97

6.10.12 IS NOT UNKNOWN

This topic describes how to use the logical operation function IS NOT UNKNOWN of Realtime Compute.

Syntax

```
A  IS  NOT  UNKNOWN
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

A is a logical comparison expression, such as 6 < 8.

In normal cases, when A compares two numbers, the value of A can be determined, which is either TRUE or FALSE. However, if either operand is not a number, the value of A cannot be determined. `IS NOT UNKNOWN` is used to determine whether this situation occurs. If the value of A cannot be determined (that is, the value is neither `TRUE` nor `FALSE`), `FALSE` is returned. If the value of A can be determined (that is, the value is `TRUE` or `FALSE`), `TRUE` is returned.

Example 1

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT  int2  as  aa
FROM  T1
WHERE  int1 = 25  IS  NOT  UNKNOWN ;
```

- Test result

aa (INT)
97

Example 2

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT   int2   as   aa
FROM   T1
WHERE   int1  <  null   IS   NOT   UNKNOWN ;
```

- Test result

aa (INT)
null

6.10.13 IS NULL

This topic describes how to use the logical operation function IS NULL of Realtime Compute.

Syntax

```
value   IS   NULL
```

Input parameter

Name	Data type
value	Any data type

Function description

If the value is `NULL` , `TRUE` is returned. Otherwise, `FALSE` is returned.

Example

- Test data

int1 (INT)	int2 (VARCHAR)
97	NULL
9	www

- Test statement

```
SELECT   int1   as   aa
FROM   T1
```

```
WHERE   int2   IS   NULL ;
```

- Test result

aa (INT)
97

6.10.14 IS TRUE

This topic describes how to use the logical operation function IS TRUE of Realtime Compute.

Syntax

```
A   IS   TRUE
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If A is TRUE, TRUE is returned. If A is FALSE, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   int1 = 255   IS   TRUE ;
```

- Test result

aa (INT)
97

6.10.15 IS UNKNOWN

This topic describes how to use the logical operation function IS UNKNOWN of Realtime Compute.

Syntax

```
A   IS   UNKNOWN
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If the value of A (a logical comparison expression) cannot be determined (that is, the value is neither `TRUE` nor `FALSE`), `TRUE` is returned. If the value of A can be determined (that is, the value is `TRUE` or `FALSE`), `FALSE` is returned. In normal cases, when A compares two numbers (for example, `6 <> 8`), the value of A can be determined, which is either TRUE or FALSE. However, if either operand is not a number, the value of A cannot be determined. `IS   UNKNOWN` is used to determine whether this situation occurs.

Example 1

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT   int2   as   aa
FROM   T1
WHERE   int1 = 25   IS   UNKNOWN ;
```

- Test result

aa (INT)
null

Example 2

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT   int2   as   aa
FROM   T1
WHERE   int1  >  null   IS   UNKNOWN ;
```

- Test result

aa (INT)
97

6.10.16 LIKE

This topic describes how to use the logical operation function LIKE of Realtime Compute.

Syntax

```
A   LIKE   B
```

Input parameter

Name	Data type
A	VARCHAR
B	VARCHAR

Function description

TRUE is returned if A matches B. Otherwise, FALSE is returned.

Note:
You can use the percent sign `(%)` as a wildcard.

Example 1

- Test data

int1 (INT)	VARCHAR2 (VARCHAR)	VARCHAR3 (VARCHAR)
90	ss97	97ss

int1 (INT)	VARCHAR2 (VARCHAR)	VARCHAR3 (VARCHAR)
99	ss10	7ho7

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   VARCHAR2   LIKE  ' ss %';
```

- Test result

aa (INT)
90
99

Example 2

- Test data

int1 (INT)	VARCHAR2 (VARCHAR)	VARCHAR3 (VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   VARCHAR3   LIKE  '% ho %';
```

- Test result

aa (INT)
99

6.10.17 NOT

This topic describes how to use the logical operation function NOT of Realtime Compute.

Syntax

```
NOT   A
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If A is `TRUE` , `FALSE` is returned. If A is `FALSE` , `TRUE` is returned.

Example

- Test data

int2 (INT)	int3 (INT)
97	65

- Test statement

```
SELECT   int2   as   aa
FROM   T1
WHERE   NOT   int3 = 62 ;
```

- Test result

aa (INT)
97

6.10.18 NOT BETWEEN AND

This topic describes how to use the logical operation function NOT BETWEEN AND of Realtime Compute.

Syntax

```
A   NOT   BETWEEN   B   AND   C
```

Input parameter

Name	Data type
A	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
B	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
C	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME

Function description

This function selects a value not within a data range defined by two other values.

Example 1

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
90	97	80
11	10	7

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   int1   NOT   BETWEEN   int2   AND   int3 ;
```

- Test result

aa (INT)
11

Example 2

- Test data

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
d	a	c

- Test statement

```
SELECT   int1   as   aa
FROM   T1
WHERE   var1   NOT   BETWEEN   var2   AND   var3 ;
```

- Test result

aa (VARCHAR)
d

Example 3

- Test data

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

- Test statement

```
SELECT   TIMESTAMP1   as   aa
FROM   T1
```

```
WHERE   TIMESTAMP1   NOT   BETWEEN   TIMESTAMP2   AND   TIMESTAMP3
;
```

- Test result

aa (TIMESTAMP)
1969-07-20 20:17:30

6.10.19 OR

This topic describes how to use the logical operation function OR of Realtime Compute.

Syntax

```
A   OR   B
```

Input parameter

Name	Data type
A	BOOLEAN
B	BOOLEAN

Function description

FALSE is returned if both A and B are FALSE. Otherwise, TRUE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

- Test statement

```
SELECT   int2   as   aa
FROM   T1
WHERE   int1 = 255   OR   int3 = 65 ;
```

- Test result

aa (INT)
97

6.10.20 IN

This topic describes how to use the logical operation function IN of Realtime Compute.

Syntax

```
SELECT   column_nam  e ( s )
FROM   table_name
WHERE   column_nam  e   IN  ( value1 , value2 ,...)
```

Input parameter

Name	Data type
value1	Constant
value2	Constant

Function description

This function queries records that match the input parameters.

Example

- Test data

id (INT)	LastName (VARCHAR)
1	Adams
2	Bush
3	Carter

- Test statement

```
SELECT  *
FROM   T1
WHERE   LastName   IN  (' Adams ',' Carter ')
```

- Test result

id (INT)	LastName (VARCHAR)
1	Adams
3	Carter

6.10.21 IS DISTINCT FROM

This topic describes how to use the logical operation function IS DISTINCT FROM of Realtime Compute.

Syntax

```
A IS DISTINCT FROM B
```

Input parameter

Name	Data type
A	Any data type
B	Any data type

Function description

- `TRUE` is returned if the data types or values of A and B are different.
- `FALSE` is returned if the data types and values of A and B are the same.
- If both A and B are null, FALSE is returned even when their data types are different.

Example

- Test data

A (INT)	B (VARCHAR)
97	97
null	sss
null	null

- Test statement

```
SELECT
A IS DISTINCT FROM B as ` result `
FROM T1
```

- Test result

result (BOOLEAN)
TRUE
TRUE
FALSE

6.10.22 IS NOT DISTINCT FROM

This topic describes how to use the logical operation function IS NOT DISTINCT FROM of Realtime Compute.

Syntax

```
A   IS   NOT   DISTINCT   FROM   B
```

Input parameter

Name	Data type
A	Any data type
B	Any data type

Function description

- `FALSE` is returned if the data types or values of A and B are different.
- `TRUE` is returned if the data types and values of A and B are the same.
- If both A and B are null, TRUE is returned even when their data types are different.

Example

- Test data

A (INT)	B (VARCHAR)
97	97
null	sss
null	null

- Test statement

```
SELECT
A   IS   NOT   DISTINCT   FROM   B   as   ` result `
FROM   T1
```

- Test result

result (BOOLEAN)
FALSE
FALSE
TRUE

6.10.23 NOT IN

This topic describes how to use the logical operation function NOT IN of Realtime Compute.

Syntax

```
SELECT   column_nam  e ( s )
FROM   table_name
WHERE   column_nam  e   NOT   IN  ( value1 , value2 ,...)
```

Input parameter

Name	Data type
value1	Constant
value2	Constant

Function description

This function queries records that do not match the input parameters.

Example

- Test data

id (INT)	LastName (VARCHAR)
1	Adams
2	Bush
3	Carter

- Test statement

```
SELECT  *
FROM   T1
WHERE   LastName   NOT   IN  (' Adams ',' Carter ')
```

- Test result

id (INT)	LastName (VARCHAR)
2	Bush

6.11 Built-in functions

6.11.1 String functions

6.11.1.1 REGEXP_EXTRACT

This topic describes how to use the string function REGEXP_EXTRACT in Realtime Compute.

Syntax

```
VARCHAR   REGEXP_EXT  RACT ( VARCHAR   str ,  VARCHAR   pattern ,
INT   index )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string.
pattern	VARCHAR	The regular expression pattern.
index	INT	The index number of the substring to be extracted from the source string.

Notice:

Comply with Java code conventions to write regular expression constants. When you run the codegen tool, it automatically converts SQL constant strings to Java code. Write the string \d as '\d' in the regular expression, just in the same way as you write a regular expression in Java.

Function description

This function extracts the substring with the specified index number from a string based on the specified regular expression pattern. The index number starts from 1. If any input parameter is NULL or the regular expression is invalid, the return value is NULL.

Examples

- Test data

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	foo(. *?)(bar)	2

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
100-200	(\\d+)-(\\d+)	1
null	foo(. *?)(bar)	2
foothebar	null	2
foothebar	Empty string	2
foothebar	(	2

- Test statements

```
SELECT   REGEXP_EXT  RACT ( str1 ,  pattern1 ,  index1 )  as
result
FROM   T1
```

- Test results

result(VARCHAR)
bar
100
null
null
null
null

6.11.1.2 REGEXP_REPLACE

This topic describes how to use the string function REGEXP_REPLACE in Realtime Compute.

Syntax

```
VARCHAR   REGEXP_REP  LACE ( VARCHAR   str ,  VARCHAR   pattern ,
VARCHAR   replacemen  t )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string.
pattern	VARCHAR	The substring to be replaced in the source string.
replacement	VARCHAR	The replacement substring .

Notice:
Comply with Java code conventions to write regular expression constants. When you run the codegen tool, it automatically converts SQL constant strings to Java code. Write the string \d as '\d' in the regular expression, just in the same way as you write a regular expression in Java.

Function description

This function replaces a substring that matches the specified regular expression pattern in the source string with another substring, and returns a new string. If any input parameter is NULL or the regular expression is invalid, the return value is NULL .

Examples

- Test data

str1(VARCHAR)	pattern1(VARCHAR)	replace1(VARCHAR)
2014-03-13	-	Empty string
null	-	Empty string
2014-03-13	-	null
2014-03-13	Empty string	s
2014-03-13	(	s
100-200	(\d+)	num

- Test statements

```
SELECT  REGEXP_REP LACE ( str1 ,  pattern1 ,  replace1 )  as
result
FROM  T1
```

- Test results

result(VARCHAR)
20140313
null
null
2014-03-13
null
num-num

6.11.1.3 REPEAT

This topic describes how to use the string function REPEAT in Realtime Compute.

Syntax

```
VARCHAR   REPEAT ( VARCHAR   str ,  INT   n )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string to be repeated.
n	INT	The number of times to repeat the string.

Function description

This function repeats a string a specified number of times and returns a new string . If str is NULL, the return value is NULL. If n is 0 or negative, the return value is an empty string.

Examples

- Test data

str(VARCHAR)	n(INT)
J	9
Hello	2
Hello	-9
null	9

- Test statements

```
SELECT   REPEAT ( str , n )  as   var1
FROM   T1
```

- Test results

var1(VARCHAR)
JJJJJJJJJ
HelloHello
Empty string
null

6.11.1.4 REPLACE

This topic describes how to use the string function REPLACE in Realtime Compute.

Syntax

```
VARCHAR  REPLACE ( str1 ,  str2 ,  str3 )
```

Input parameters

Parameter	Data type	Description
str1	VARCHAR	The source string.
str2	VARCHAR	The substring to be replaced in the source string.
str3	VARCHAR	The replacement substring .

Function description

This function replaces a substring of a string with another substring.

Examples

- Test data

str1(INT)	str2(INT)	str3(INT)
alibaba blink	blink	flink

- Test statements

```
SELECT  REPLACE ( str1 ,  str2 ,  str3 )  as  ` result `
FROM  T1
```

- Test results

result(VARCHAR)
alibaba flink

6.11.1.5 REVERSE

This topic describes how to use the string function REVERSE in Realtime Compute.

Syntax

```
VARCHAR  REVERSE ( VARCHAR  str )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string.

Function description

This function returns a string in the reverse order of the specified string. If any input parameter is NULL, the return value is NULL.

Examples

- Test data

str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)	str4(VARCHAR)
iPhoneX	Alibaba	World	null

- Test statements

```
SELECT  REVERSE ( str1 )  as  var1 , REVERSE ( str2 )  as  var2
,
        REVERSE ( str3 )  as  var3 , REVERSE ( str4 )  as  var4
FROM  T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

6.11.1.6 RPAD

This topic describes how to use the string function RPAD in Realtime Compute.

Syntax

```
VARCHAR  RPAD ( VARCHAR  str ,  INT  len ,  VARCHAR  pad )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string.

Parameter	Data type	Description
len	INT	The length of the new string after padding.
pad	VARCHAR	The string to be repeatedly padded to the source string.

Function description

This function right-pads a source string with another string several times until the new string reaches the specified length. If any input parameter is NULL, the return value is NULL.

If `len` is negative, the return value is NULL.

If `pad` is an empty string and the value of `len` is less than or equal to the length of `str` , `str` is trimmed to the specified length. If pad is an empty string and the value of `len` is greater than the length of `str` , the return value is NULL.

Examples

- Test data

str(VARCHAR)	len(INT)	pad(VARCHAR)
Empty string	-2	Empty string
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	Empty string
Empty string	2	s
asd	4	Empty string
Empty string	0	Empty string

- Test statements

```
SELECT  RPAD ( str , len , pad ) as  result
```

```
FROM  T1
```

- Test results

result(VARCHAR)
null
HelloWorldJohnJ
Jo
CHel
null
null
as
ss
null
Empty string

6.11.1.7 SPLIT_INDEX

This topic describes how to use the string function SPLIT_INDEX in Realtime Compute.

Syntax

```
VARCHAR  SPLIT_INDE  X ( VARCHAR  str ,  VARCHAR  sep ,  INT
index )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string to be split.
sep	VARCHAR	The separator.
index	INT	The index number of the substring to be extracted from the source string.

Function description

This function uses the separator specified by `sep` to split the string specified by `str` into several substrings and returns the substring indexed as `index`. The value of `index` starts from 0. If the substring with the specified index number does not exist, the return value is NULL.

If any input parameter is NULL, the return value is NULL.

Examples

- Test data

str(VARCHAR)	sep(VARCHAR)	index(INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0
null	,	0

- Test statements

```
SELECT   SPLIT_INDE  X ( str ,  sep ,  index )  as   var1
FROM   T1
```

- Test results

var1(VARCHAR)
Mary
null
null
null

6.11.1.8 STR_TO_MAP

This topic describes how to use the string function STR_TO_MAP in Realtime Compute.

Syntax

```
MAP   STR_TO_MAP ( VARCHAR   text )
MAP   STR_TO_MAP ( VARCHAR   text ,  VARCHAR   listDelimi  ter ,
VARCHAR   keyValueDe  limiter )
```

Function description

This function first uses the separator specified by listDelimiter to split the given text into key-value pairs. Then, this function uses the separator specified by keyValueDe limiter to separate the key and value in each key-value pair. Finally, this function assembles and returns a MAP. The default value of listDelimiter is a comma (,). The default value of keyValueDelimiter is an equal sign (=).

Input parameters

Parameter	Data type	Description
text	VARCHAR	The input text.
listDelimiter	VARCHAR	The separator between key-value pairs in the input text. The default value is a comma (,).
keyValueDelimiter	VARCHAR	The separator between the key and value in each key-value pair. The default value is an equal sign (=).

Notice:

The listDelimiter and keyValueDelimiter parameters are defined by Java regular expressions. If a special character is used, it needs to be escaped.

Test statements

```
SELECT
  STR_TO_MAP (' k1 = v1 , k2 = v2 ')[' k1 ']  as   a
FROM   T1
```

Test results

a(VARCHAR)
v1

6.11.1.9 SUBSTRING

This topic describes how to use the string function SUBSTRING in Realtime Compute.

Syntax

```
VARCHAR   SUBSTRING ( VARCHAR   a ,  INT   start )
VARCHAR   SUBSTRING ( VARCHAR   a ,  INT   start ,  INT   len )
```

Input parameters

Parameter	Data type	Description
a	VARCHAR	The source string.

Parameter	Data type	Description
start	INT	The start position of the substring to be extracted from the source string.
len	INT	The length of the substring to be extracted.

Function description

This function returns a substring of the specified length from a string, starting from the specified position. If the length is not specified, this function returns the substring from the specified position to the end of the string. The value of start begins with 1. If the value is 0, it is regarded as 1. If the value is negative, this function counts backward from the end of the string to find the first character of the substring .

Examples

- Test data

str(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	null

- Test statements

```
SELECT   SUBSTRING ('',  222222222 )  as   var1 ,
       SUBSTRING ( str ,  2 )  as   var2 ,
       SUBSTRING ( str , - 2 )  as   var3 ,
       SUBSTRING ( str , - 2 ,  1 )  as   var4 ,
       SUBSTRING ( str ,  2 ,  1 )  as   var5 ,
       SUBSTRING ( str ,  22 )  as   var6 ,
       SUBSTRING ( str , - 22 )  as   var7 ,
       SUBSTRING ( str ,  1 )  as   var8 ,
       SUBSTRING ( str ,  0 )  as   var9 ,
       SUBSTRING ( nullstr ,  0 )  as   var10
FROM   T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)
Empty string	1=v1;k2=v2	v2	v	1	Empty string	Empty string	k1=v1;k2=v2	k1=v1;k2=v2	null

6.11.1.10 TO_BASE64

This topic describes how to use the string function TO_BASE64 in Realtime Compute.

Syntax

```
VARCHAR  TO_BASE64 ( bin )
```

Input parameters

Parameter	Data type
bin	BINARY

Function description

This function converts binary data to a Base64-encoded string.

Examples

- Test data

c(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

- Test statements

```
SELECT  TO_BASE64 ( FROM_BASE6  4 ( c ))  as  var1
FROM  T1
```

- Test results

var1(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

6.11.1.11 TRIM

This topic describes how to use the string function TRIM in Realtime Compute.

Syntax

```
VARCHAR  TRIM ( VARCHAR  x )
```

Input parameters

Parameter	Data type
x	VARCHAR

Function description

This function removes leading and trailing characters from a string. The most common use is to remove leading and trailing spaces.

Examples

- Test statements

```
SELECT  TRIM ('   Sample   ')  as  result
FROM  T1
```

- Test results

result(VARCHAR)
Sample

Note:

The return value is 'Sample'.

6.11.1.12 UPPER

This topic describes how to use the string function UPPER in Realtime Compute.

Syntax

```
VARCHAR  UPPER ( A )
```

Input parameters

Parameter	Data type
A	VARCHAR

Function description

This function converts all the letters in a string to uppercase.

Examples

- Test data

var1(VARCHAR)
ss
ttee

- Test statements

```
SELECT   UPPER ( var1 )  as   aa
FROM   T1 ;
```

- Test results

aa(VARCHAR)
SS
TTEE

6.11.1.13 CHAR_LENGTH

This topic describes how to use the string function CHAR_LENGTH in Realtime Compute.

Syntax

```
INT   CHAR_LENGT  H ( A )
```

Input parameters

Parameter	Data type
A	INT

Function description

This function returns the number of characters contained in a string.

Examples

- Test data

var1(INT)
ss

var1(INT)
231ee

- Test statements

```
SELECT   CHAR_LENGT  H ( var1 )  as   aa
FROM   T1 ;
```

- Test results

aa(INT)
2
5

6.11.1.14 CHR

This topic describes how to use the string function CHR in Realtime Compute.

Syntax

```
VARCHAR   CHR ( INT   ascii )
```

Input parameters

Parameter	Data type	Description
ascii	INT	The integer ranging from 0 to 255. If the input parameter falls out of this range, the return value is NULL.

Function description

This function converts an ASCII code into a character.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
255	97	65

- Test statements

```
SELECT   CHR ( int1 )  as   var1 ,  CHR ( int2 )  as   var2 ,  CHR
( int3 )  as   var3
```

```
FROM T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

6.11.1.15 CONCAT

This topic describes how to use the string function CONCAT in Realtime Compute.

Syntax

```
VARCHAR CONCAT ( VARCHAR var1 , VARCHAR var2 , ...)
```

Input parameters

Parameter	Data type	Description
var1	VARCHAR	The string.
var2	VARCHAR	The string.

Function description

This function concatenates two or more strings into a single string. If any input parameter is NULL, the parameter is skipped.

Examples

- Test data

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

- Test statements

```
SELECT CONCAT ( var1 , var2 , var3 ) as var
FROM T1
```

- Test results

var(VARCHAR)
HelloMyWorld
HelloWorld

var(VARCHAR)
World
null

6.11.1.16 CONCAT_WS

This topic describes how to use the string function CONCAT_WS in Realtime Compute.

Syntax

```
VARCHAR   CONCAT_WS ( VARCHAR   separator ,  VARCHAR   var1 ,
VARCHAR   var2 , ...)
```

Input parameters

Parameter	Data type	Description
separator	VARCHAR	The separator.
var1	VARCHAR	The parameter to be concatenated.
var2	VARCHAR	The parameter to be concatenated.

Function description

This function concatenates every two parameter values with a separator and returns a new string. The length and type of the new string depend on the input values.

Note:

If the separator value is NULL, it is regarded as an empty string for concatenating the parameter values. If any other parameter is NULL, the parameter is skipped during concatenation.

Examples

- Test data

sep(VARCHAR)	str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)
|	Jack	Harry	John
null	Jack	Harry	John
|	null	Harry	John
|	Jack	null	null

- Test statements

```
SELECT   CONCAT_WS ( sep ,  str1 ,  str2 ,  str3 )  as   var
FROM   T1
```

- Test results

var(VARCHAR)
Jack|Harry|John
JackHarryJohn
Harry|John
Jack

6.11.1.17 FROM_BASE64

This topic describes how to use the string function FROM_BASE64 in Realtime Compute.

Syntax

```
BINARY   FROM_BASE6  4 ( str )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The Base64-encoded string .

Function description

This function decodes a Base64-encoded string into binary data.

Examples

- Test data

a(INT)	b(BIGINT)	c(VARCHAR)
1	1L	null

- Test statements

```
SELECT
from_base6  4 ( c )  as   var1 , from_base6  4 (' SGVsbG8gd2  9ybGQ
=')  as   var2
```

```
FROM  T1
```

- Test results

var1(BINARY)	var2(BINARY)
null	Byte Array: [72('H'), 101('e'), 108('l'), 108('l'), 111('o'), 32(' '), 119('w'), 111('o'), 114('r'), 108('l'), 100('d')]

6.11.1.18 HASH_CODE

This topic describes how to use the string function HASH_CODE in Realtime Compute.

Syntax

```
INT  HASH_CODE ( VARCHAR  str )
```

Input parameters

str	VARCHAR

Function description

This function generates a hash code for the specified string based on the HASH_CODE () method, and then returns the absolute value of the hash code.

Examples

- Test data

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

- Test statements

```
SELECT  HASH_CODE ( str1 )  as  var1 ,  HASH_CODE ( str2 )  as
var2 ,  HASH_CODE ( nullstr )  as  var3
FROM  T1
```

- Test results

var1(INT)	var2(INT)	var3(INT)
1099348823	401392878	null

6.11.1.19 INITCAP

This topic describes how to use the string function INITCAP in Realtime Compute.

Syntax

```
VARCHAR  INITCAP ( A )
```

Input parameters

A	VARCHAR

Function description

This function returns a string with the first letter of each word in uppercase and all other letters in lowercase.

Examples

- Test data

var1(VARCHAR)
aADvbn

- Test statements

```
SELECT  INITCAP ( var1 ) as  aa
FROM  T1 ;
```

- Test results

aa(VARCHAR)
Aasvbn

6.11.1.20 INSTR

This topic describes how to use the string function INSTR in Realtime Compute.

Note:
The INSTR function is available only in Realtime Compute V2.2.0 and later.

Syntax

```
INT  instr (  string1 ,  string2  )
```

```
INT   instr (  string1 ,  string2  [,  start_posi  tion  [,
nth_appear  ance  ] ] )
```

Input parameters

Parameter	Data type	Description
string1	VARCHAR	The source string to search .
string2	VARCHAR	The substring to search for in the source string.
start_position	INT	The position in the source string where the search starts.
nth_appearance	INT	Which occurrence of the substring to be searched for in the source string.

Function description

This function returns the position of the substring in the source string. If the substring is not found in the source string, the return value is 0.

Examples

- Test data T1

string1(VARCHAR)
helloworld

- Test statements

```
SELECT
instr (' helloworld ',' lo ')  as   res1 ,
instr (' helloworld ',' l ',- 1 , 1 )  as   res2 ,
instr (' helloworld ',' l ', 3 , 2 )  as   res3
FROM   T1
```

- Test results

res1(INT)	res2(INT)	res3(INT)
4	9	4

6.11.1.21 JSON_VALUE

This topic describes how to use the string function JSON_VALUE in Realtime Compute.

Syntax

```
VARCHAR   JSON_VALUE ( VARCHAR   content ,  VARCHAR   path1 )
```

Input parameters

- content

  The JSON object to be parsed, which is represented as a string. This parameter is of the VARCHAR type.
- path

  The path expression that is used to parse the JSON object. This parameter is of the VARCHAR type. The following table lists the supported path expressions.

Symbol	Description
$	The root object.
[]	The array subscript.
*	The array wildcard.
.	The child element.

Function description

This function extracts the value at the specified path from a JSON string. If the JSON string is invalid or any input parameter is NULL, the return value is NULL.

Examples

- Test data

id(INT)	json(VARCHAR)	path1(VARCHAR)
1	[10, 20, [30, 40]]	$[2][*]
2	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}	$.ccc.hhh[*]
3	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg",hhh":["h0","h1","h2"]},"iii":"jjj"}	$.ccc.hhh[1]
4	[10, 20, [30, 40]]	NULL

id(INT)	json(VARCHAR)	path1(VARCHAR)
5	NULL	$[2][*]
6	"{xx]"	"$[2][*]"

- Test statements

```
SELECT
 id ,
 JSON_VALUE ( json ,  path1 )  AS  ` value `
FROM
 T1
```

- Test results

id (INT)	value (VARCHAR)
1	[30,40]
2	["h0","h1","h2"]
3	H1
4	NULL
5	NULL
6	NULL

6.11.1.22 KEYVALUE

This topic describes how to use the string function KEYVALUE in Realtime Compute.

Syntax

```
VARCHAR   KEYVALUE ( VARCHAR   str ,  VARCHAR   split1 ,  VARCHAR
split2 ,  VARCHAR   key_name )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The key-value pairs in the specified string.
split1	VARCHAR	The separator between key-value pairs.
split2	VARCHAR	The separator between the key and value in each key-value pair.
key_name	VARCHAR	The name of the key whose value is to be extracted.

Function description

This function parses the key-value pairs in a string based on the key-value pair separator and key-value separator. Then, this function returns the value for the specified key name. If the key name does not exist or an exception occurs, the return value is NULL.

Examples

- Test data

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)
k1=v1;k2=v2	;	=	k2
null	;	|	:
k1:v1|k2:v2	null	=	:
k1:v1|k2:v2	|	=	null
k1:v1|k2:v2	|	=	:
k1:v1|k2:v2	|	=	:

- Test statements

```
SELECT   KEYVALUE ( str ,  split1 ,  split2 ,  key1 )  as  ` result
`
FROM   T1
```

- Test results

result(VARCHAR)
v2
null
null
null
null
null

6.11.1.23 LOCALTIMESTAMP

This topic describes how to use the string function LOCALTIMESTAMP in Realtime Compute.

Syntax

```
timestamp   LOCALTIMES  TAMP
```

Input parameters

- None

Function description

This function returns the current timestamp of the system.

Examples

- Test statements

```
SELECT
LOCALTIMES  TAMP   as  ` result `
FROM   T1
```

- Test results

result (TIMESTAMP)
2018-07-27 14:04:38.998

6.11.1.24 LOWER

This topic describes how to use the string function LOWER in Realtime Compute.

Syntax

```
VARCHAR   LOWER ( A )
```

Input parameters

- A

  VARCHAR

Function description

This function converts all the letters in a string to lowercase.

Examples

- Test data

var1(VARCHAR)
Ss
yyT

- Test statements

```
SELECT   LOWER ( var1 )  as   aa
FROM   T1 ;
```

- Test results

aa(VARCHAR)
ss
yyt

6.11.1.25 LPAD

This topic describes how to use the string function LPAD in Realtime Compute.

Syntax

```
VARCHAR   LPAD ( VARCHAR   str ,  INT   len ,  VARCHAR   pad )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string.
len	INT	The length of the new string after padding.
pad	VARCHAR	The string to be repeatedly padded to the source string.

Function description

This function left-pads a source string with another string several times until the new string reaches the specified length.

If any input parameter is NULL, the return value is NULL.

If `len` is negative, the return value is NULL.

If `pad` is an empty string and the value of `len` is less than or equal to the length of `str`, `str` is trimmed to the specified length. If pad is an empty string and the value of `len` is greater than the length of `str`, the return value is NULL.

Examples

- Test data

str(VARCHAR)	len(INT)	pad(VARCHAR)
Empty string	-2	Empty string
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	Empty string
Empty string	2	s
asd	4	Empty string
Empty string	0	Empty string

- Test statements

```
SELECT   LPAD ( str ,  len ,  pad )  AS   result
FROM   T1
```

- Test results

result(VARCHAR)
null
JohnJHelloWorld
Jo
HelC
null
null
as
ss
null

result(VARCHAR)
Empty string

6.11.1.26 MD5

This topic describes how to use the string function MD5 in Realtime Compute.

Syntax

```
VARCHAR  MD5 ( VARCHAR  str )
```

Input parameters

- str

  VARCHAR

Function description

This function returns the MD5 value of the specified string. If the input parameter is an empty string (''), the return value is an empty string.

Examples

- Test data

str1(VARCHAR)	str2(VARCHAR)
k1=v1;k2=v2	Empty string

- Test statements

```
SELECT
   MD5 ( str1 )  as  var1 ,
   MD5 ( str2 )  as  var2
FROM  T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	Empty string

6.11.1.27 OVERLAY

This topic describes how to use the string function OVERLAY in Realtime Compute.

Syntax

```
VARCHAR   OVERLAY  ( ( VARCHAR   x   PLACING   VARCHAR   y   FROM
INT   start_posi  tion  [  FOR   INT   length  ]) )
```

Input parameters

Parameter	Data type
x	VARCHAR
y	VARCHAR
start_position	INT
length (optional)	INT

Function description

This function replaces a substring of x with y. The replacement starts from the character position specified by start_position. The total number of characters are to be replaced is the length value plus one.

Examples

- Test statements

```
OVERLAY (' abcdefg '  PLACING  ' hij '  FROM   2   FOR   2 )  as
result
FROM   T1
```

- Test results

result(VARCHAR)
ahijdefg

6.11.1.28 PARSE_URL

This topic describes how to use the string function PARSE_URL in Realtime Compute.

Syntax

```
VARCHAR   PARSE_URL ( VARCHAR   urlStr ,  VARCHAR   partToExtr  act
[,  VARCHAR   key ])
```

Input parameters

Parameter	Data type	Description
urlStr	VARCHAR	The URL string.
partToExtract	VARCHAR	The part to be parsed from the URL.
key (optional)	VARCHAR	The name of the key whose value is to be extracted.

Function description

This function parses a URL and returns the specified part from the URL. If the value of partToExtract is QUERY, this function returns the value of the specified key in the URL. Valid values of partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, FILE, AUTHORITY, and USERINFO.

Notice:

If the input URL string is NULL, the return value is NULL.

Examples

- Test data

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

- Test statements

```
SELECT   PARSE_URL ( url1 , ' QUERY ', ' query ')  as   var1 ,
       PARSE_URL ( url1 , ' QUERY ')  as   var2 ,
       PARSE_URL ( url1 , ' HOST ')  as   var3 ,
       PARSE_URL ( url1 , ' PATH ')  as   var4 ,
       PARSE_URL ( url1 , ' REF ')  as   var5 ,
       PARSE_URL ( url1 , ' PROTOCOL ')  as   var6 ,
       PARSE_URL ( url1 , ' FILE ')  as   var7 ,
       PARSE_URL ( url1 , ' AUTHORITY ')  as   var8 ,
       PARSE_URL ( nullstr , ' QUERY ')  as   var9 ,
       PARSE_URL ( url1 , ' USERINFO ')  as   var10 ,
```

```
        PARSE_URL ( nullstr , ' QUERY ', ' query ')  as   var11
FROM   T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)	var11(VARCHAR)
1	query=1	facebook.com	/path/p1.php	null	http	/path/p1.php?query=1	facebook.com	null	null	null

6.11.1.29 POSITION

This topic describes how to use the string function POSITION in Realtime Compute.

Syntax

```
INTEGER   POSITION (  x   IN   y )
```

Input parameters

Parameter	Data type
x	VARCHAR
y	VARCHAR

Function description

This function returns the position of the first occurrence of string x in string y. This function is similar to LOCATE(substr,str).

Examples

- Test statements

```
POSITION (' in '  IN  ' china ')  as   result
FROM   T1
```

- Test results

result(INT)
3

6.11.1.30 REGEXP

This topic describes how to use the string function REGEXP in Realtime Compute.

Syntax

```
BOOLEAN   REGEXP ( VARCHAR   str ,  VARCHAR   pattern )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string.
pattern	VARCHAR	The regular expression pattern.

Function description

This function performs regular expression matching on a string to check whether it matches the specified pattern. If the string or pattern is empty or NULL, the return value is false.

Examples

- Test data

str1(VARCHAR)	pattern1(VARCHAR)
k1=v1;k2=v2	k2*
k1:v1|k2:v2	k3
null	k3
k1:v1|k2:v2	null
k1:v1|k2:v2	(

- Test statements

```
SELECT   REGEXP ( str1 ,  pattern1 )  as   result
FROM   T1
```

- Test results

result(BOOLEAN)
true
false
null
null

result(BOOLEAN)
false

6.11.2 Mathematical functions

6.11.2.1 Addition

This topic describes how to use the mathematical function addition in Realtime Compute.

Syntax

```
A + B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns the result of A plus B.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
10	20	30

- Test statements

```
SELECT   int1 + int2 + int3   as   aa
FROM   T1
```

- Test results

aa(int)
60

6.11.2.2 Subtraction

This topic describes how to use the mathematical function subtraction in Realtime Compute.

Syntax

```
A - B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns the result of A minus B.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
10	10	30

- Test statements

```
SELECT int3 - int2 - int1 as aa
FROM T1
```

- Test results

aa(int)
10

6.11.2.3 Multiplication

This topic describes how to use the mathematical function multiplication in Realtime Compute.

Syntax

```
A  *  B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns the result of A multiplied by B.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
10	20	3

- Test statements

```
SELECT   int1 * int2 * int3   as   aa
FROM   T1
```

- Test results

aa(int)
600

6.11.2.4 Division

This topic describes how to use the mathematical function division in Realtime Compute.

Syntax

```
A / B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns the result of A divided by B.

Examples

- Test data

int1(INT)	int2(INT)
8	4

- Test statements

```
SELECT   int1 / int2   as   aa
FROM   T1
```

- Test results

aa(int)
2

6.11.2.5 ABS

This topic describes how to use the mathematical function ABS in Realtime Compute.

Syntax

```
DOUBLE   ABS ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the absolute value of input parameter A.

Examples

- Test data

in1(DOUBLE)
4.3

- Test statements

```
SELECT   ABS ( in1 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
4.3

6.11.2.6 ACOS

This topic describes how to use the mathematical function ACOS in Realtime Compute.

Syntax

```
ACOS ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the arccosine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.7173560908995228
0.4

- Test statements

```
SELECT  ACOS ( in1 ) as  aa
FROM  T1
```

- Test results

aa(DOUBLE)
0.7707963267948966
1.1592794807274085

6.11.2.7 BIN

This topic describes how to use the mathematical function BIN in Realtime Compute.

Syntax

```
VARCHAR  BIN ( BIGINT  number )
```

Input parameters

Parameter	Data type
number	BIGINT Note: You can set this parameter only to a BIGINT or INT value. The INT value is implicitly converted to a BIGINT value for computation.

Function description

This function converts a BIGINT value to a binary string.

Examples

- Test data

id(INT)	x(BIGINT)
1	12L
2	10L
3	0L

id(INT)	x(BIGINT)
4	10000000000L

Note:
In the test data, letter `L` in the x(BIGINT) column indicates the data type Long, which is not involved in binary conversion.

- Test statements

```
SELECT   id ,  bin ( x )  as   var1
FROM   T1
```

- Test results

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	1001010100000010111110010000000000

6.11.2.8 ASIN

This topic describes how to use the mathematical function ASIN in Realtime Compute.

Syntax

```
DOUBLE   ASIN ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the arcsine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.7173560908995228

in1(DOUBLE)
0.4

- Test statements

```
SELECT   ASIN ( in1 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
0.8
0.41151684606748806

6.11.2.9 ATAN

This topic describes how to use the mathematical function ATAN in Realtime Compute.

Syntax

```
DOUBLE   ATAN ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the arctangent value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.7173560908995228
0.4

- Test statements

```
SELECT   ATAN ( in1 )  as   aa
```

```
FROM   T1
```

- Test results

aa(DOUBLE)
0.6222796222326533
0.3805063771123649

6.11.2.10 BITAND

This topic describes how to use the mathematical function BITAND in Realtime Compute.

Syntax

```
INT   BITAND ( INT   number1 ,  INT   number2 )
```

Input parameters

Parameter	Data type
number1	INT
number2	INT

Function description

This function performs a bitwise AND operation on the specified values. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)	b(INT)
2	3

- Test statements

```
SELECT   BITAND ( a ,  b )  as   intt
FROM   T1
```

- Test results

intt(INT)
2

6.11.2.11 BITNOT

This topic describes how to use the mathematical function BITNOT in Realtime Compute.

Syntax

```
INT   BITNOT ( INT   number )
```

Input parameters

Parameter	Data type
number	INT

Function description

This function performs a bitwise NOT operation on the specified value. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)
7

- Test statements

```
SELECT   BITNOT ( a )  as   var1
FROM   T1
```

- Test results

var1(INT)
0xfff8

6.11.2.12 BITOR

This topic describes how to use the mathematical function BITOR in Realtime Compute.

Syntax

```
INT   BITOR ( INT   number1 ,  INT   number2 )
```

Input parameters

Parameter	Data type
number1	INT
number2	INT

Function description

This function performs a bitwise OR operation on the specified values. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)	b(INT)
2	3

- Test statements

```
SELECT   BITOR ( a ,  b )  as   var1
FROM   T1
```

- Test results

var1(INT)
3

6.11.2.13 BITXOR

This topic describes how to use the mathematical function BITXOR in Realtime Compute.

Syntax

```
INT   BITXOR ( INT   number1 ,  INT   number2 )
```

Input parameters

Parameter	Data type
number1	INT
number2	INT

Function description

This function performs a bitwise XOR operation on the specified values. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)	b(INT)
2	3

- Test statements

```
SELECT   BITXOR ( a ,  b )  as   var1
FROM   T1
```

- Test results

var1(INT)
1

6.11.2.14 CARDINALITY

This topic describes how to use the mathematical function CARDINALITY in Realtime Compute.

Syntax

```
CARDINALIT  Y ( str )
```

Input parameters

Parameter	Data type
number1	Array

Function description

This function returns the number of elements in an array.

Examples

- Test statements

```
SELECT   cardinalit  y ( array [ 1 , 2 , 3 ])  AS  ` result `
FROM   T1
```

- Test results

result(INT)
3

6.11.2.15 COS

This topic describes how to use the mathematical function COS in Realtime Compute.

Syntax

```
DOUBLE   COS ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the cosine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.8
0.4

- Test statements

```
SELECT   COS ( in1 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

6.11.2.16 COT

This topic describes how to use the mathematical function COT in Realtime Compute.

Syntax

```
DOUBLE   COT ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the cotangent value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.8
0.4

- Test statements

```
SELECT   COT ( in1 )  as   aa
```

```
FROM   T1
```

- Test results

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

6.11.2.17 EXP

This topic describes how to use the mathematical function EXP in Realtime Compute.

Syntax

```
DOUBLE   EXP ()
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns e raised to the power of the specified number. The constant e is the base of natural logarithms.

Examples

- Test data

in1(DOUBLE)
8.0
10.0

- Test statements

```
SELECT   EXP ( in1 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
2980.9579870417283
22026.465794806718

6.11.2.18 E

This topic describes how to use the mathematical function E in Realtime Compute.

Syntax

```
DOUBLE   E ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the DOUBLE type value of natural constant e.

Examples

- Test data

in1(DOUBLE)
8.0
10.0

- Test statements

```
SELECT   id ,  e ()  as   dou1 ,  E ()  as   dou2
FROM   T1
```

- Test results

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.718281828459045	2.718281828459045

6.11.2.19 FLOOR

This topic describes how to use the mathematical function FLOOR in Realtime Compute.

Syntax

```
B   FLOOR ( A )
```

Input parameters

Parameter	Data type
A	INT, BIGINT, FLOAT, or DOUBLE

Function description

This function rounds down the decimal portion of input parameter A and returns the largest integer less than or equal to input parameter A. The data type of output parameter B is the same as that of input parameter A.

Examples

- Test data

in1(DOUBLE)	in2(BIGINT)
8.123	3

- Test statements

```
SELECT
FLOOR ( in1 )  as   out1 ,
FLOOR ( in2 )  as   out2
FROM   T1
```

- Test results

out1(DOUBLE)	out2(BIGINT)
8.0	3

6.11.2.20 LN

This topic describes how to use the mathematical function LN in Realtime Compute.

Syntax

```
DOUBLE  ln ( DOUBLE  number )
```

Input parameters

Parameter	Data type
number	DOUBLE Note: If the input parameter is of the VARCHAR or BIGINT type, it is implicitly converted to the DOUBLE type for computation.

Function description

This function returns the natural logarithm of the specified number. The return value is a logarithm of the DOUBLE type.

Examples

- Test data

ID(INT)	X(DOUBLE)
1	100.0
2	8.0

- Test statements

```
SELECT  id , ln ( x ) as  dou1 , ln ( e ()) as  dou2
FROM  T1
```

- Test results

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	4.605170185988092	1.0
2	2.0794415416798357	1.0

6.11.2.21 LOG

This topic describes how to use the mathematical function LOG in Realtime Compute.

Syntax

```
DOUBLE  LOG ( DOUBLE  base ,  DOUBLE  x )
DOUBLE  LOG ( DOUBLE  x )
```

Input parameters

Parameter	Data type
base	DOUBLE
x	DOUBLE

Function description

This function returns the logarithm of x to the specified base. The return value is a logarithm of the DOUBLE type. If base is not specified, this function returns the logarithm of x to base e.

Examples

- Test data

ID(INT)	BASE(DOUBLE)	X(DOUBLE)
1	10.0	100.0
2	2.0	8.0

- Test statements

```
SELECT  id ,  LOG ( base ,  x )  as  dou1 ,  LOG ( 2 )  as
dou2
FROM  T1
```

- Test results

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

6.11.2.22 LOG10

This topic describes how to use the mathematical function LOG10 in Realtime Compute.

Syntax

```
DOUBLE   LOG10 ( DOUBLE   x )
```

Input parameters

Parameter	Data type
x	DOUBLE

Function description

This function returns the base-10 logarithm of x. If x is NULL, the return value is NULL. If x is negative, an exception occurs.

Examples

- Test data

id(INT)	X(INT)
1	100
2	10

- Test statements

```
SELECT   id ,  log10 ( x )  as   dou1
FROM   T1
```

- Test results

id(INT)	dou1(DOUBLE)
1	2.0
2	1.0

6.11.2.23 LOG2

This topic describes how to use the mathematical function LOG2 in Realtime Compute.

Syntax

```
DOUBLE   LOG2 ( DOUBLE   x )
```

Input parameters

Parameter	Data type
x	DOUBLE

Function description

This function returns the base-2 logarithm of x. If x is NULL, the return value is NULL . If x is negative, an exception occurs.

Examples

- Test data

id(INT)	X(INT)
1	8
2	2

- Test statements

```
SELECT   id ,  log2 ( x )  as   dou1
FROM   T1
```

- Test results

id(INT)	dou1(DOUBLE)
1	3.0
2	1.0

6.11.2.24 PI

This topic describes how to use the mathematical function PI in Realtime Compute.

Syntax

```
DOUBLE  PI ()
```

Function description

This function returns the value of Pi.

Examples

- Test data

ID(INT)	X(INT)
1	8

- Test statements

```
SELECT  id , PI () as  dou1
FROM  T1
```

- Test results

ID(INT)	dou1(DOUBLE)
1	3.141592653589793

6.11.2.25 POWER

This topic describes how to use the mathematical function POWER in Realtime Compute.

Syntax

```
DOUBLE  POWER ( A ,  B )
```

Input parameters

Parameter	Data type
A	DOUBLE
B	DOUBLE

Function description

This function returns the result of A raised to the power of B. The result is a DOUBLE value.

Examples

- Test data

in1(DOUBLE)	in2(DOUBLE)
2.0	4.0

- Test statements

```
SELECT   POWER ( in1 ,  in2 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
16.0

6.11.2.26 RAND

This topic describes how to use the mathematical function RAND in Realtime Compute.

Syntax

```
DOUBLE   RAND ([ BIGINT   seed ])
```

Input parameters

Parameter	Data type
seed	BIGINT

Note:
The value of seed is a random number, which determines the start value of a random number sequence.

Function description

This function returns a random number between 0 (inclusive) and 1 (exclusive). The return value is of the DOUBLE type.

Examples

- Test data

id(INT)	X(INT)
1	8

- Test statements

```
SELECT   id ,  rand ( 1 )  as   dou1 ,  rand ( 3 )  as   dou2
FROM   T1
```

- Test results

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

6.11.2.27 SIN

This topic describes how to use the mathematical function SIN in Realtime Compute.

Syntax

```
DOUBLE   SIN ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the sine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
8.0
0.4

- Test statements

```
SELECT   SIN ( in1 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
0.9893582466233818
0.3894183423086505

6.11.2.28 SQRT

This topic describes how to use the mathematical function SQRT in Realtime Compute.

Syntax

```
DOUBLE  SQRT ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the square root of input parameter A.

Examples

- Test data

in1(DOUBLE)
8.0

- Test statements

```
SELECT  SQRT ( in1 )  as  aa
FROM  T1
```

- Test results

aa(DOUBLE)
2.8284271247461903

6.11.2.29 TAN

This topic describes how to use the mathematical function TAN in Realtime Compute.

Syntax

```
DOUBLE  TAN ( A )
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the tangent value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.8
0.4

- Test statements

```
SELECT   TAN ( in1 )  as   aa
FROM   T1
```

- Test results

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

6.11.2.30 CEIL

This topic describes how to use the mathematical function CEIL in Realtime Compute.

Syntax

```
B   CEIL ( A )
```

Input parameters

Parameter	Data type
A	INT, BIGINT, FLOAT, or DOUBLE
B	INT, BIGINT, FLOAT, or DOUBLE

Function description

This function rounds input parameter A up to the nearest integer greater than or equal to A. The data type of output parameter B is the same as that of input parameter A.

Examples

- Test data

in1(INT)	in2(DOUBLE)
1	2.3

- Test statements

```
SELECT
CEIL ( in1 )  as   out1
CEIL ( in2 )  as   out2
FROM   T1
```

- Test results

out1(INT)	out2(DOUBLE)
1	3.0

6.11.2.31 CHARACTER_LENGTH

This topic describes how to use the mathematical function CHARACTER_LENGTH in Realtime Compute.

Syntax

```
INTEGER   CHARACTER_  LENGTH (  VARCHAR   x  )
```

Input parameters

Parameter	Data type
x	VARCHAR

Function description

This function returns the number of characters contained in string x.

Examples

- Test data

ID(INT)	X(VARCHAR)
1	StreamCompute

- Test statements

```
SELECT   CHARACTER_  LENGTH (  x  )  as   result
```

```
FROM   T1
```

- Test results

ID(INT)	result(INT)
1	13

6.11.2.32 DEGREES

This topic describes how to use the mathematical function DEGREES in Realtime Compute.

Syntax

```
DOUBLE   DEGREES (  double   x  )
```

Input parameters

Parameter	Data type
x	DOUBLE

Function description

This function converts a radian value x to a degree value.

Examples

- Test statements

```
SELECT   DEGREES (  PI () )  as   result
FROM   T1
```

- Test results

result(DOUBLE)
180.0

6.11.2.33 MOD

This topic describes how to use the mathematical function MOD in Realtime Compute.

Syntax

```
INTEGER  MOD ( INTEGER  x , INTEGER  y )
```

Input parameters

Parameter	Data type
x	INTEGER
y	INTEGER

Function description

This function returns the remainder of integer x divided by integer y. When x is negative , the result is negative.

Examples

- Test data

X(INT)	Y(INT)
29	3
-29	3
-29	-3

- Test statements

```
SELECT  MOD ( x , y )  as  result
FROM  T1
```

- Test results

result(INT)
2
-2
-2

6.11.2.34 ROUND

This topic describes how to use the mathematical function ROUND in Realtime Compute.

Syntax

```
DECIMAL   ROUND (  DECIMAL   x ,  INT   n )
```

Input parameters

Parameter	Data type
x	DECIMAL
n	INT

Function description

This function rounds the x parameter to n decimal places.

Examples

- Test data

in1(DECIMAL)
0.7173560908995228
0.4

- Test statements

```
SELECT   ROUND ( in1 , 2 )  as  ` result `
FROM   T1
```

- Test results

result(DECIMAL)
0.72
0.40

6.11.3 Date functions

6.11.3.1 CURRENT_DATE

This topic describes how to use the date function CURRENT_DATE in Realtime Compute.

Syntax

```
CURRENT_DA  TE
```

Function description

This function returns the current system date.

Examples

- Test statements

```
SELECT   CURRENT_DA  TE   as   res
FROM   T1
```

- Test results

res(DATE)
2018-09-20

6.11.3.2 CURRENT_TIMESTAMP

This topic describes how to use the date function CURRENT_TIMESTAMP in Realtime Compute.

Syntax

```
TIMESTAMP   CURRENT_TI  MESTAMP
```

Function description

This function returns the current UTC timestamp.

Examples

- Test statements

```
SELECT   CURRENT_TI  MESTAMP   as   var1
FROM   T1
```

- Test results

var1(TIMESTAMP)
2007-04-30 13:10:02.047

6.11.3.3 DATEDIFF

This topic describes how to use the date function DATEDIFF in Realtime Compute.

Syntax

```
INT   DATEDIFF ( VARCHAR   enddate ,  VARCHAR   startdate )
INT   DATEDIFF ( TIMESTAMP   enddate ,  VARCHAR   startdate )
INT   DATEDIFF ( VARCHAR   enddate ,  TIMESTAMP   startdate )
INT   DATEDIFF ( TIMESTAMP   enddate ,  TIMESTAMP   startdate )
```

Input parameters

Parameter	Data type
startdate	TIMESTAMP or VARCHAR
enddate	TIMESTAMP or VARCHAR

Note:

The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.

Function description

This function computes the number of days between the end date and start date. The return value is an integer. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

datetime1(VARCHAR)	datetime2(VARCHAR)	nullstr(VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

- Test statements

```
SELECT   ROUND ( in1 , 2 )  as  ` result `
 DATEDIFF ( TIMESTAMP  ' 2017 - 10 - 15   23 : 00 : 00 ', datetime2
)  as   int2 ,
 DATEDIFF ( datetime2 , TIMESTAMP  ' 2017 - 10 - 15   23 : 00 : 00
')  as   int3 ,
 DATEDIFF ( datetime2 , nullstr )  as   int4 ,
 DATEDIFF ( nullstr , TIMESTAMP  ' 2017 - 10 - 15   23 : 00 : 00
')  as   int5 ,
 DATEDIFF ( nullstr , datetime2 )  as   int6 ,
 DATEDIFF ( TIMESTAMP  ' 2017 - 10 - 15   23 : 00 : 00 ', TIMESTAMP
 ' 2017 - 9 - 15   00 : 00 : 00 ') as   int7
```

```
FROM   T1
```

- Test results

int1(INT)	int2(INT)	int3(INT)	int4(INT)	int5(INT)	int6(INT)	int7(INT)
30	31	-31	null	null	null	31

6.11.3.4 DATE_ADD

This topic describes how to use the date function DATE_ADD in Realtime Compute.

Syntax

```
VARCHAR   DATE_ADD ( VARCHAR   startdate ,  INT   days )
VARCHAR   DATE_ADD ( TIMESTAMP   time ,  INT   days )
```

Input parameters

Parameter	Data type
startdate	TIMESTAMP or VARCHAR Note: The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
enddate	TIMESTAMP
days	INT

Function description

This function adds an interval (specified by days) to the specified date and returns a new date. The return value is a VARCHAR type date in `yyyy - MM - dd` format. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

datetime1(VATCHAR)	nullstr(VATCHAR)
2017-09-15 00:00:00	null

- Test statements

```
SELECT   DATE_ADD ( datetime1 ,  30 )  as   var1 ,
 DATE_ADD ( TIMESTAMP  ' 2017 - 09 - 15   23 : 00 : 00 ', 30 )  as
  var2 ,
 DATE_ADD ( nullstr , 30 )  as   var3
```

```
FROM   T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-10-15	2017-10-15	null

6.11.3.5 DATE_FORMAT

This topic describes how to use the date function DATE_FORMAT in Realtime Compute.

Syntax

```
VARCHAR   DATE_FORMA  T ( TIMESTAMP   time ,  VARCHAR   to_format )
VARCHAR   DATE_FORMA  T ( VARCHAR   date ,  VARCHAR   to_format )
VARCHAR   DATE_FORMA  T ( VARCHAR   date ,  VARCHAR   from_forma  t
,  VARCHAR   to_format )
```

Input parameters

Parameter	Data type
date	VARCHAR Note: The default date format is yyyy-MM-dd HH:mm:ss.
time	TIMESTAMP
from_format	VARCHAR
to_format	VARCHAR

Function description

This function converts a VARCHAR type date from the source format to the target format. The time or date parameter specifies the source string. The from_format parameter is optional. It specifies the source format of the date. The default format is yyyy-MM-dd hh:mm:ss. The to_format parameter specifies the target format of the date. The return value is a VARCHAR type date in the target format. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

date1(VARCHAR)	datetime1(VARCHAR)	nullstr(VARCHAR)
0915-2017	2017-09-15 00:00:00	null

- Test statements

```
SELECT   DATE_FORMA  T ( datetime1 , ' yyMMdd ')  as   var1 ,
 DATE_FORMA  T ( nullstr , ' yyMMdd ')  as   var2 ,
 DATE_FORMA  T ( datetime1 ,  nullstr )  as   var3 ,
 DATE_FORMA  T ( date1 , ' MMdd - yyyy ',  nullstr )  as   var4 ,
 DATE_FORMA  T ( date1 , ' MMdd - yyyy ', ' yyyyMMdd ')  as   var5
,
 DATE_FORMA  T ( TIMESTAMP  ' 2017 - 09 - 15   23 : 00 : 00 ', '
yyMMdd ')  as   var6
FROM   T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)
170915	null	null	null	20170915	170915

6.11.3.6 DATE_SUB

This topic describes how to use the date function DATE_SUB in Realtime Compute.

Syntax

```
VARCHAR   DATE_SUB ( VARCHAR   startdate ,  INT   days )
VARCHAR   DATE_SUB ( TIMESTAMP   time ,  INT   days )
```

Input parameters

Parameter	Data type
startdate	VARCHAR Note: The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
time	TIMESTAMP
days	INT

Function description

This function subtracts an interval (specified by days) from the specified date and returns a new date. The return value is a VARCHAR type date in yyyy-MM-dd format. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

date1(VARCHAR)	nullstr(VARCHAR)
2017-10-15	null

- Test statements

```
SELECT   DATE_SUB ( date1 ,  30 )  as   var1 ,
 DATE_SUB ( TIMESTAMP  ' 2017 - 10 - 15   23 : 00 : 00 ', 30 )  as
  var2 ,
 DATE_SUB ( nullstr , 30 )  as   var3
FROM   T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15	2017-09-15	null

6.11.3.7 DAYOFMONTH

This topic describes how to use the date function DAYOFMONTH in Realtime Compute.

Syntax

```
BIGINT   DAYOFMONTH ( TIMESTAMP   time )
BIGINT   DAYOFMONTH ( DATE   date )
```

Input parameters

Parameter	Data type
date	DATE
time	TIMESTAMP

Function description

This function returns the day in the specified date or time value. The return value ranges from 1 to 31.

Examples

- Test data

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- Test statements

```
SELECT   DAYOFMONTH ( TIMESTAMP  ' 2016 - 09 - 15   00 : 00 : 00
')  as   int1 ,
 DAYOFMONTH ( DATE  ' 2017 - 09 - 22 ')  as   int2 ,
 DAYOFMONTH ( tdate )  as   int3 ,
 DAYOFMONTH ( ts )  as   int4 ,
 DAYOFMONTH ( CAST ( dateStr   AS   DATE ))  as   int5 ,
 DAYOFMONTH ( CAST ( tsStr   AS   TIMESTAMP ))  as   int6
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
15	22	10	15	15	15

6.11.3.8 EXTRACT

This topic describes how to use the date function EXTRACT in Realtime Compute.

Syntax

```
BIGINT   EXTRACT ( unit   FROM   time )
```

Input parameters

Parameter	Data type
time	Any date expression.

Parameter	Data type
unit	Valid values of unit are as follows: · MICROSECOND · SECOND · MINUTE · HOUR · DAY · WEEK · MONTH · QUARTER · YEAR · SECOND_MICROSECOND · MINUTE_MICROSECOND · MINUTE_SECOND · HOUR_MICROSECOND · HOUR_SECOND · HOUR_MINUTE · DAY_MICROSECOND · DAY_SECOND · DAY_MINUTE · DAY_HOUR · YEAR_MONTH

Function description

This function returns one or two separate parts from the date or time value, for example, the year, month, day, hour, minute, or week.

Examples

- Test statements

```
EXTRACT ( YEAR   FROM   CURRENT_TI  MESTAMP )  AS   OrderYear ,
EXTRACT ( MONTH   FROM   CURRENT_TI  MESTAMP )  AS   OrderMonth ,
EXTRACT ( DAY   FROM   CURRENT_TI  MESTAMP )  AS   OrderDay ,
EXTRACT ( WEEK   FROM   CURRENT_TI  MESTAMP )  AS   OrderWeek
```

- Test results

OrderYear(BIGINT)	OrderMonth(BIGINT)	OrderDay(BIGINT)	OrderWeek(BIGINT)
2018	10	11	41

6.11.3.9 FROM_UNIXTIME

This topic describes how to use the date function FROM_UNIXTIME in Realtime Compute.

Syntax

```
VARCHAR   FROM_UNIXT  IME ( BIGINT   unixtime [,  VARCHAR   format ])
```

Input parameters

Parameter	Data type
unixtime	BIGINT
format	VARCHAR

Function description

- This function converts the timestamp (in seconds) specified by unixtime to a VARCHAR type date in the specified date format. The default format is yyyy-MM-dd HH:mm:ss.
- If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

unixtime1(INT)	nullstr(VARCHAR)
1505404800	null

- Test statements

```
SELECT   FROM_UNIXT  IME ( unixtime1 )  as   var1 ,
 FROM_UNIXT  IME ( unixtime1 ,' MMdd - yyyy ')  as   var2 ,
 FROM_UNIXT  IME ( unixtime1 , nullstr )  as   var3
FROM   T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-10-15	2017-10-15	null

6.11.3.10 HOUR

This topic describes how to use the date function HOUR in Realtime Compute.

Syntax

```
BIGINT   HOUR ( TIME   time )
```

```
BIGINT   HOUR ( TIMESTAMP   timestamp )
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the hours (in 24-hour format) in the specified time or timestamp value as a number. The return value ranges from 0 to 23.

Examples

- Test data

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- Test statements

```
SELECT   HOUR ( TIMESTAMP  ' 2016 - 09 - 20   23 : 33 : 33 ')  as
int1 ,
 HOUR ( TIME  ' 23 : 30 : 33 ')  as   int2 ,
 HOUR ( time2 )  as   int3 ,
 HOUR ( timestamp1 )  as   int4 ,
 HOUR ( CAST ( time1   AS   TIME ))  as   int5 ,
 HOUR ( CAST ( datetime1   AS   TIMESTAMP ))  as   int6
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
23	23	22	11	22	11

6.11.3.11 LOCALTIME

This topic describes how to use the date function LOCALTIME in Realtime Compute.

Syntax

```
TIME  LOCALTIME
```

Function description

This function returns the current time of the TIME type in the session time zone. You can use LOCALTIME as a variable.

Examples

- Test statements

```
SELECT  LOCALTIME  as  ` result `
FROM  T1
```

- Test results

result(TIME)
19:00:47

6.11.3.12 MINUTE

This topic describes how to use the date function MINUTE in Realtime Compute.

Syntax

```
BIGINT  MINUTE ( TIME  time )
BIGINT  MINUTE ( TIMESTAMP  timestamp )
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the minutes in the specified time or timestamp value as a number. The return value ranges from 0 to 59.

Examples

- Test data

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- Test statements

```
SELECT   MINUTE ( TIMESTAMP  ' 2016 - 09 - 20   23 : 33 : 33 ')  as
  int1 ,
 MINUTE ( TIME  ' 23 : 30 : 33 ')  as   int2 ,
 MINUTE ( time2 )  as   int3 ,
 MINUTE ( timestamp1 )  as   int4 ,
 MINUTE ( CAST ( time1   AS   TIME ))  as   int5 ,
 MINUTE ( CAST ( datetime1   AS   TIMESTAMP ))  as   int6
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	30	23	12	23	12

6.11.3.13 MONTH

This topic describes how to use the date function MONTH in Realtime Compute.

Syntax

```
BIGINT   MONTH ( TIMESTAMP   timestamp )
BIGINT   MONTH ( DATE   date )
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the month in the specified time value as a number. The return value ranges from 1 to 12.

Examples

- Test data

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- Test statements

```
SELECT   MONTH ( TIMESTAMP  ' 2016 - 09 - 15   00 : 00 : 00 ')  as
  int1 ,
 MONTH ( DATE  ' 2017 - 09 - 22 ')  as   int2 ,
 MONTH ( tdate )  as   int3 ,
 MONTH ( ts )  as   int4 ,
 MONTH ( CAST ( dateStr   AS   DATE ))  as   int5 ,
 MONTH ( CAST ( tsStr   AS   TIMESTAMP ))  as   int6
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
9	9	11	10	9	10

6.11.3.14 NOW

This topic describes how to use the date function NOW in Realtime Compute.

Syntax

```
BIGINT   NOW ()
```

Input parameters

If no input parameter is specified, the UNIX timestamp (in seconds) of the current system time is returned.

Function description

This function returns the UNIX timestamp (in seconds) in the current time zone. You can specify an INT type parameter as an offset (in seconds) and add the offset to the current timestamp to return a value. For example, the NOW(100) function adds 100 seconds to the current timestamp and returns a value of the BIGINT type.

Examples

- Test data

b(VARCHAR)
null

- Test statements

```
SELECT
 NOW ()  as   big1 ,
 NOW ( b )  as   big2
FROM   T1
```

- Test results

big1(BIGINT)	big2(BIGINT)
1403006911	null

6.11.3.15 SECOND

This topic describes how to use the date function SECOND in Realtime Compute.

Syntax

```
BIGINT   SECOND ( TIMESTAMP   timestamp )
 BIGINT   SECOND ( TIME   time )
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the seconds in the specified time value as a number. The return value ranges from 0 to 59.

Examples

- Test data

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- Test statements

```
SELECT   SECOND ( TIMESTAMP  ' 2016 - 09 - 20   23 : 33 : 33 ')  as
  int1 ,
 SECOND ( TIME  ' 23 : 30 : 33 ')  as   int2 ,
 SECOND ( time2 )  as   int3 ,
 SECOND ( timestamp1 )  as   int4 ,
 SECOND ( CAST ( time1   AS   TIME ))  as   int5 ,
 SECOND ( CAST ( datetime1   AS   TIMESTAMP ))  as   int6
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	33	24	13	24	13

6.11.3.16 TIMESTAMPADD

This topic describes how to use the date function TIMESTAMPADD in Realtime Compute.

Syntax

```
TIMESTAMP   TIMESTAMPA  DD ( interval , INT   int_expr , TIMESTAMP
datetime_e  xpr )
DATE   TIMESTAMPA  DD ( interval , INT   int_expr , DATE
datetime_e  xpr )
```

Input parameters

Parameter	Data type
interval	VARCHAR
int_expr	INT
datetime_expr	TIMESTAMP or DATE

Note:

The following table lists the valid units of interval.

Unit of interval	Description
FRAC_SECOND	Millisecond
SECOND	Second
MINUTE	Minute
HOUR	Hour

Unit of interval	Description
DAY	Day
WEEK	Week
MONTH	Month
QUARTER	Quarter
YEAR	Year

Function description

This function adds the integer expression int_expr to the date or datetime expression datetime_expr, and returns the current time of the TIME type in the session time zone . The data type of the return value of this function is the same as that of datetime_expr .

Examples

- Test data

a(TIMESTAMP)	b(DATE)
2018-07-09 10:23:56	1990-02-20

- Test statements

```
SELECT
TIMESTAMPA  DD ( HOUR , 3 , a )  AS  ` result1 `
TIMESTAMPA  DD ( DAY , 3 , b )  AS  ` result2 `
FROM   T1
```

- Test results

result1(TIMESTAMP)	result2(DATE)
2018-07-09 13:23:56.0	1990-02-23

6.11.3.17 TO_DATE

This topic describes how to use the date function TO_DATE in Realtime Compute.

Syntax

```
Date   TO_DATE ( INT   time )
Date   TO_DATE ( VARCHAR   date )
```

```
Date   TO_DATE ( VARCHAR   date , VARCHAR   format )
```

Input parameters

Parameter	Data type
time	INT Note: This parameter specifies the number of days that have elapsed since 00:00:00 Thursday, 1 January, 1970.
date	VARCHAR Note: The default format is yyyy-MM-dd.
format	VARCHAR

Function description

This function converts a date of the INT or VARCHAR type to a date of the DATE type.

Examples

- Test data

date1(INT)	date2(VARCHAR)	date3(VARCHAR)
100	2017-09-15	20170915

- Test statements

```
SELECT   TO_DATE ( date1 )  as   var1 ,
 TO_DATE ( date2 )  as   var2 ,
 TO_DATE ( date3 ,' yyyy - MM - dd ')  as   var3
FROM   T1
```

- Test results

var1(DATE)	var2(DATE)	var3(DATE)
1970-04-11	2017-09-15	2017-09-15

6.11.3.18 TO_TIMESTAMP

This topic describes how to use the date function TO_TIMESTAMP in Realtime Compute.

Syntax

```
TIMESTAMP   TO_TIMESTA  MP ( BIGINT   time )
TIMESTAMP   TO_TIMESTA  MP ( VARCHAR   date )
TIMESTAMP   TO_TIMESTA  MP ( VARCHAR   date ,   VARCHAR   format )
```

Input parameters

Parameter	Data type
time	BIGINT Note: The unit is millisecond.
date	VARCHAR Note: The default format is yyyy-MM-dd HH:mm:ss[.SSS]. [DO NOT TRANSLATE]
format	VARCHAR

Function description

This function converts a date of the BIGINT or VARCHAR type to a date of the TIMESTAMP type.

Examples

- Test data

timestamp1(bigint)	timestamp2(VARCHAR)	timestamp3(VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

- Test statements

```
SELECT   TO_TIMESTA  MP ( timestamp1 )  as   var1 ,
 TO_TIMESTA  MP ( timestamp2 )  as   var2 ,
 TO_TIMESTA  MP ( timestamp3 , ' yyyyMMddHH  mmss ')  as   var3
```

```
FROM   T1
```

- Test results

var1(TIMESTAMP)	var2(TIMESTAMP)	var3(TIMESTAMP)
2017-12-13 03:27:57.0	2017-09-15 00:00:00.0	2017-09-15 00:00:00.0

6.11.3.19 UNIX_TIMESTAMP

This topic describes how to use the date function UNIX_TIMESTAMP in Realtime Compute.

Syntax

```
BIGINT   UNIX_TIMES  TAMP ()
BIGINT   UNIX_TIMES  TAMP ( VARCHAR   date )
BIGINT   UNIX_TIMES  TAMP ( TIMESTAMP   timestamp )
BIGINT   UNIX_TIMES  TAMP ( VARCHAR   date ,  VARCHAR   format )
```

Input parameters

Parameter	Data type
timestamp	TIMESTAMP
date	VARCHAR Note: The default date format is `yyyy - MM - dd   HH : mm : ss`.
format	VARCHAR Note: The default format is `yyyy - MM - dd   hh : mm : ss`.

Function description

This function converts the specified date to a UNIX timestamp (in seconds) of the BIGINT type. If no input parameter is specified, the UNIX timestamp (in seconds) of the current time is returned. In this case, this function has the same semantics as NOW. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

nullstr(VARCHAR)
null

- Test statements

```
SELECT   UNIX_TIMES  TAMP ()  as   big1 ,
       UNIX_TIMES  TAMP ( nullstr )  as   big2
FROM   T1
```

- Test results

big1(BIGINT)	big2(BIGINT)
1403006911	null

6.11.3.20 WEEK

This topic describes how to use the date function WEEK in Realtime Compute.

Syntax

```
BIGINT   WEEK ( DATE   date )
BIGINT   WEEK ( TIMESTAMP   timestamp )
```

Input parameters

Parameter	Data type
date	DATE
timestamp	TIMESTAMP

Function description

This function computes the week number of the specified date in a year. The week number ranges from 1 to 53.

Examples

- Test data

dateStr(VARCHAR)	date1(DATE)	ts1(TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

- Test statements

```
SELECT   WEEK ( TIMESTAMP  ' 2017 - 09 - 15   00 : 00 : 00 ')  as
int1 ,
```

```
 WEEK ( date1 )  as   int2 ,
 WEEK ( ts1 )  as   int3 ,
 WEEK ( CAST ( dateStr   AS   DATE ))  as   int4
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)
37	45	41	37

6.11.3.21 YEAR

This topic describes how to use the date function YEAR in Realtime Compute.

Syntax

```
BIGINT   YEAR ( TIMESTAMP   timestamp )
BIGINT   YEAR ( DATE   date )
```

Input parameters

Parameter	Data type
date	DATE
timestamp	TIMESTAMP

Function description

This function returns the year in the specified time value.

Examples

- Test data

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- Test statements

```
SELECT   YEAR ( TIMESTAMP  ' 2016 - 09 - 15   00 : 00 : 00 ')  as
int1 ,
 YEAR ( DATE  ' 2017 - 09 - 22 ')  as   int2 ,
 YEAR ( tdate )  as   int3 ,
 YEAR ( ts )  as   int4 ,
 YEAR ( CAST ( dateStr   AS   DATE ))  as   int5 ,
 YEAR ( CAST ( tsStr   AS   TIMESTAMP ))  as   int6
FROM   T1
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
2016	2017	2017	2017	2015	2017

6.11.4 Conditional functions

6.11.4.1 CASE WHEN

This topic describes how to use the conditional function CASE WHEN in Realtime Compute.

Syntax

```
CASE   WHEN   a   THEN   b  [ WHEN   c   THEN   d ]* [ ELSE   e ]
END
```

Input parameters

Parameter	Data type
a	BOOLEAN
b	BOOLEAN
c	BOOLEAN
d	BOOLEAN
e	BOOLEAN

Function description

If a is true, this function returns b. If a is false and c is true, this function returns d. If both a and c are false, this function returns e.

Examples

Note:

When the CASE WHEN function returns a constant string, it pads spaces to the right-side of the string. In the following example, when the else condition is met, the return value is 'ios' followed by several spaces.

```
case   when   device_typ  e  = ' android '
then  ' android '
else  ' ios '
```

```
end   as   os
```

You can resolve this issue in the following two ways:

- Use the TRIM function to remove spaces. For the preceding example, use trim(os).
- Use the CAST function to convert a constant string to a string of the VARCHAR type .

- Test data

device_type(VARCHAR)
android
ios
win

- Test statement 1 (using the TRIM function)

```
SELECT
   trim ( os ), //  The   trim ()  function   is   used   here .
   CHAR_LENGT  H ( trim ( os )) //  The   trim ()  function   is
used   here .
from (
   SELECT
     case   when   device_typ  e  = ' android '
     then  ' android '
     else  ' ios '
end   as   os
FROM   T1
);
```

Test statement 2 (using the CAST function)

```
SELECT
 os ,
CHAR_LENGT  H ( os )
from
( SELECT
 case   when   device_typ  e  = ' android '
 then   cast (' android '  as   varchar ) //  The   CAST   function
  is   used   here .
 else   cast (' ios '  as   varchar ) //  The   CAST   function
is   used   here .
end   as   os
FROM   T1
);
```

- Test results

os(VARCHAR)	length(INT)
android	7
ios	3

os(VARCHAR)	length(INT)
ios	3

6.11.4.2 COALESCE

This topic describes how to use the conditional function COALESCE in Realtime Compute.

Syntax

```
COALESCE ( A , B ,...)
```

Input parameters

Parameter	Data type
A	Any data type
B	Any data type

Note:
All values must be of the same type or be NULL. Otherwise, an exception occurs.

Function description

This function returns the first non-NULL value in the specified list. The return value is of the same type as the input parameter values. If all values in the list are NULL, the return value is NULL.

Note:
The list must contain at least one parameter. Otherwise, an exception occurs.

Examples

- Test data

var1(VARCHAR)	var2(VARCHAR)
null	30

- Test statements

```
SELECT  COALESCE ( var1 , var2 )  as  aa
```

```
FROM    T1
```

- Test results

aa(VARCHAR)
30

6.11.4.3 IF

This topic describes how to use the conditional function IF in Realtime Compute.

Syntax

```
T   IF ( BOOLEAN   testCondit  ion ,  T   valueTrue ,  T
valueFalse  OrNull )
```

Note:

T represents a return value of any type.

Input parameters

Parameter	Data type
testCondition	BOOLEAN
valueTrue	Any data type (The valueTrue and valueFalseOrNull parameters must be of the same type.)
valueFalseOrNull	Any data type (The valueTrue and valueFalseOrNull parameters must be of the same type.)

Function description

This function uses the BOOLEAN value of testCondition as the judgment criterion. If testCondition is true, this function returns valueTrue. If testCondition is false, this function returns valueFalseOrNull. If testCondition is NULL, it is also regarded as false and this function returns valueFalseOrNull. If any other parameter is NULL, this function works based on normal semantics. The data type of the return value is determined by T.

Examples

- Test data

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null
1	2	null	Harry

- Test statements

```
SELECT   IF ( int1  <  int2 , str1 ,  str2 )  as   int1
FROM   T1
```

- Test results

int1(VARCHAR)
Jack
Jack
null

6.11.4.4 IS_ALPHA

This topic describes how to use the conditional function IS_ALPHA in Realtime Compute.

Syntax

```
BOOLEAN   IS_ALPHA ( VARCHAR   str )
```

Input parameters

Parameter	Data type
str	VARCHAR

Function description

This function checks whether the specified string contains only letters. If yes, the return value is true. If not, the return value is false.

Examples

- Test data

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- Test statements

```
SELECT   IS_ALPHA ( e )  as   boo1 , IS_ALPHA ( f )  as   boo2 ,
IS_ALPHA ( g )  as   boo3
FROM   T1
```

- Test results

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
false	true	false

6.11.4.5 IS_DECIMAL

This topic describes how to use the conditional function IS_DECIMAL in Realtime Compute.

Syntax

```
BOOLEAN   IS_DECIMAL ( VARCHAR   str )
```

Input parameters

Parameter	Data type
str	VARCHAR

Function description

This function checks whether the specified string can be converted to a decimal value . If yes, the return value is true. If not, the return value is false.

Examples

- Test data

a(VARCHAR)	b(VARCHAR)	c(VARCHAR)	d(VARCHAR)	e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
1	123	2	11.4445	3	asd	null

- Test statements

```
SELECT
IS_DECIMAL ( a )  as   boo1 ,
IS_DECIMAL ( b )  as   boo2 ,
IS_DECIMAL ( c )  as   boo3 ,
IS_DECIMAL ( d )  as   boo4 ,
IS_DECIMAL ( e )  as   boo5 ,
IS_DECIMAL ( f )  as   boo6 ,
IS_DECIMAL ( g )  as   boo7
FROM   T1
```

- Test results

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)	boo4(BOOLEAN)	boo5(BOOLEAN)	boo6(BOOLEAN)	boo7(BOOLEAN)
true	true	true	true	true	false	false

6.11.4.6 IS_DIGIT

This topic describes how to use the conditional function IS_DIGIT in Realtime Compute.

Syntax

```
BOOLEAN   IS_DIGIT ( VARCHAR   str )
```

Input parameters

Parameter	Data type
str	VARCHAR

Function description

This function checks whether the specified string contains only digits. If yes, the return value is true. If not, the return value is false. The return value is of the BOOLEAN type.

Examples

- Test data

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- Test statements

```
SELECT
```

```
IS_DIGIT ( e )  as   boo1 ,
IS_DIGIT ( f )  as   boo2 ,
IS_DIGIT ( g )  as   boo3
FROM   T1
```

- Test results

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

6.11.4.7 NULLIF

This topic describes how to use the conditional function NULLIF in Realtime Compute.

Syntax

```
NULLIF ( A , B )
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns NULL if the two specified parameters have the same value, and returns the value of the first parameter if the parameters have different values.

Examples

- Test data

var1(INT)	var2(INT)
30	30

- Test statements

```
SELECT   NULLIF ( var1 , var2 )  as   aa
FROM   T1
```

- Test results

aa(INT)
null

6.11.5 Table-valued functions

6.11.5.1 GENERATE_SERIES

This topic describes how to use the table-valued function GENERATE_SERIES in Realtime Compute.

Syntax

```
GENERATE_S  ERIES ( INT   from ,  INT   to )
```

Input parameters

Parameter	Data type
from	The lower bound of a consecutive series of values (including the lower bound) to be generated. This parameter is of the INT type.
to	The upper bound of a consecutive series of values (excluding the upper bound) to be generated. This parameter is of the INT type.

Function description

This function generates a consecutive series of values from the lower bound to the upper bound minus one.

Examples

- Test data

s(INT)	e(INT)
1	3
-2	1

- Test statements

```
SELECT   s ,  e ,  v
FROM   T1 ,  lateral   table ( GENERATE_S  ERIES ( s ,  e ))
as   T ( v )
```

- Test results

s(INT)	e(INT)	v(INT)
1	3	1

s(INT)	e(INT)	v(INT)
1	3	2
-2	1	-2
-2	1	-1
-2	1	-0

6.11.5.2 JSON_TUPLE

This topic describes how to use the table-valued function JSON_TUPLE in Realtime Compute.

Syntax

```
JSON_TUPLE ( str ,  path1 ,  path2  ...,  pathN )
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The JSON string.
path1 to pathN	VARCHAR	The path string, which does not start with special characters such as a dollar sign ($) or a period (.).

Function description

This function returns the value represented by each path string from the JSON string.

Examples

- Test data

d(VARCHAR)	s(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	qwe3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	qwe2

- Test statements

```
SELECT   d ,  v
FROM   T1 ,  lateral   table ( JSON_TUPLE ( d , ' qwe ',  s ))
 as   T ( v )
```

```

```

- Test results

d(VARCHAR)	v(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	asd
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	asd3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	asd4
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	asd5

6.11.5.3 STRING_SPLIT

This topic describes how to use the table-valued function STRING_SPLIT in Realtime Compute.

Syntax

```
string_spl  it ( varchar   string ,  varchar   separator )
```

Input parameters

Parameter	Data type
string	VARCHAR
separator	VARCHAR

Function description

This function splits a string into several substrings based on a separator.

Examples

- Test data

d(varchar)	s(varchar)
abc-bcd	-
hhh	-

- Test statements

```
select   d ,  v
from   T1 ,
```

```
lateral  table ( string_spl it ( d ,  s )) as  T ( v )
```

- Test results

d(varchar)	v(varchar)
abc-bcd	abc
abc-bcd	bcd
hhh	hhh

Note:
Currently, the separator must be a single character.

6.11.5.4 MULTI_KEYVALUE

This topic describes how to use the table-valued function MULTI_KEYVALUE in Realtime Compute.

Note:
The MULTI_KEYVALUE function is available only in Realtime Compute V2.2.2 and later.

Syntax

```
MULTI_KEYV  ALUE ( VARCHAR  str ,  VARCHAR  split1 ,  VARCHAR
split2 ,  VARCHAR  key_name1 ,  VARCHAR  key_name2 , ...)
```

Input parameters

- str

  The string of key-value pairs. This parameter is of the VARCHAR type.
- split1

  The key-value pair separator of the VARCHAR type. If split1 is null, a whitespace is used as the key-value pair separator. If the value of split1 contains more than one character, split1 specifies a set of separators and each character in the value specifies a valid separator.
- split2

  The key-value separator of the VARCHAR type. If split2 is null, a whitespace is used as the key-value separator. If the value of split2 contains more than one character, split2 specifies a set of separators and each character in the value specifies a valid separator.

- key_name1, key_name2, ...

  The list of key names whose values are to be extracted. This parameter is of the VARCHAR type.

Function description: This function parses the key-value pairs in a string based on the key-value pair separator and key-value separator. Then, this function returns a list of values for the key names in the key_name1, key_name2, ... list. If any key_name does not exist, null is returned as the value. Examples

- Test data

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)	key2(VARCHAR)
k1=v1;k2=v2	;	=	k1	k2
null	;	=	k1	k2
k1:v1;k2:v2	;	:	k1	k3
k1:v1;k2:v2	;	=	k1	k2
k1:v1;k2:v2	,	:	k1	k2
k1:v1;k2=v2	;	:	k1	k2
k1:v1abck2:v2	cab	:	k1	k2
k1:v1;k2=v2	;	:=	k1	k2
k1:v1 k2:v2	null	:	k1	k2
k1 v1;k2 v2	;	null	k1	k2

- Test statements

```
SELECT   c1 ,  c2
FROM   T1 ,  lateral   table ( MULTI_KEYV  ALUE ( str ,  split1 ,
split2 ,  key1 ,  key2 ))
as   T ( c1 ,  c2 )
```

- Test results

c1(VARCHAR)	c2(VARCHAR)
v1	v2
null	null
v1	null
null	null

c1(VARCHAR)	c2(VARCHAR)
null	null
v1	null
v1	v2
v1	v2
v1	v2
v1	v2

6.11.6 Type conversion function

6.11.6.1 CAST

This topic describes how to use the type conversion function CAST in Realtime Compute.

Syntax

```
CAST ( A  AS  type )
```

Input parameters

Parameter	Data type
A	See Overview.

Function description

This function converts the value of input parameter A to a specified type.

Examples

- Test data

var1(VARCHAR)	var2(INT)
1000	30

- Test statements

```
SELECT  CAST ( var1  AS  INT )  as  aa
FROM  T1
```

- Test results

aa(INT)
1000

6.11.7 Aggregate functions

6.11.7.1 AVG

This topic describes how to use the aggregate function AVG in Realtime Compute. In Flink SQL, the AVG function returns the average value of all values in the specified expression.

Syntax

```
AVG ( A )
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, or DOUBLE

Function description

This function returns the average value of a column.

Examples

- Test data

var1(INT)	var2(INT)
4	30
6	30

- Test statements

```
SELECT   AVG ( var1 )  as   aa
FROM   T1
```

- Test results

aa(INT)
5

6.11.7.2 CONCAT_AGG

This topic describes how to use the aggregate function CONCAT_AGG in Realtime Compute. In Flink SQL, the CONCAT_AGG function concatenates the strings of all specified fields and returns a new string.

Syntax

```
CONCAT_AGG ([ linedelimi  ter ,]  value )
```

Input parameters

Parameter	Data type
linedelimiter (optional)	Only a string constant is currently supported.

Function description

This function concatenates the strings of all specified fields and returns a new string. The default connector is `\ n`. The return value is of the VARCHAR type.

Examples

- Test data

c(VARCHAR)
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi

- Test statements

```
SELECT
concat_agg ( c )  as   var1 ,
concat_agg ('-',  c )  as   var2
FROM   MyTable
```

```
GROUP   BY   c
```

- Test results

var1(VARCHAR)	var2(VARCHAR)
Hi\nHi\nHi\nHi\nHi\nHi\nHi\nHi\nHi\nHi	Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi

6.11.7.3 COUNT

This topic describes how to use the aggregate function COUNT in Realtime Compute.

In Flink SQL, the COUNT function returns the number of rows in a given column.

Syntax

```
COUNT ( A )
```

Input parameters

Parameter	Data type
A	• Supported data types: TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, DOUBLE, BOOLEAN, and VARCHAR • Unsupported data types: DATE, TIME, TIMESTAMP, and VARBINARY

Function description

This function returns the number of rows in a given column.

Examples

- Test data

var1(VARCHAR)
1000
100
10
1

- Test statements

```
SELECT   COUNT ( var1 )  as   aa
FROM   T1
```

- Test results

aa(BIGINT)
4

6.11.7.4 FIRST_VALUE

This topic describes how to use the aggregate function FIRST_VALUE in Realtime Compute. In Flink SQL, the FIRST_VALUE function returns the first non-null record of a data stream.

Syntax

```
T   FIRST_VALU  E ( T   value )
T   FIRST_VALU  E ( T   value ,  Long   order )
```

Input parameters

Parameter	Data type
value	Any data type (The input parameters must be of the same type.)
order	INT

Function description

This function returns the first non-null record of a data stream. A record with the smallest order value is obtained as the first non-null record.

Examples

- Test data

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	"Hello"
2L	2	"Hello"
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"

a(BIGINT)	b(INT)	c(VARCHAR)
20L	20	"Hello World"

- Test statements

```
SELECT   c ,
 first_valu  e ( b )
OVER  (
PARTITION   BY   c
ORDER   BY   PROCTIME ()  RANGE   UNBOUNDED   preceding
)  as   var1
from   T1
```

- Test results

c(VARCHAR)	var1(INT)
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello World	7
Hello World	7
Hello World	7

6.11.7.5 LAST_VALUE

This topic describes how to use the aggregate function LAST_VALUE in Realtime Compute. In Flink SQL, the LAST_VALUE function returns the last non-null record of a data stream.

Syntax

```
T   LAST_VALUE ( T   value )
T   LAST_VALUE ( T   value ,  Long   order )
```

Input parameters

Parameter	Data type
value	Any data type (The input parameters must be of the same type.)
order	INT

Function description

This function returns the last non-null record of a data stream. A record with the greatest order value is obtained as the last non-null record.

Examples

- Test data

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	"Hello"
2L	2	"Hello"
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

- Test statements

```
SELECT   c ,
 last_value ( b )
OVER  (
PARTITION   BY   c
ORDER   BY   PROCTIME ()  RANGE   UNBOUNDED   preceding
)  as   var1
from   T1
```

- Test results

c(VARCHAR)	var1(INT)
Hello	1
Hello	2
Hello	3
Hello	4
Hello	5
Hello	6
Hello World	7

c(VARCHAR)	var1(INT)
Hello World	8
Hello World	20

6.11.7.6 MAX

This topic describes how to use the aggregate function MAX in Realtime Compute. In Flink SQL, the MAX function returns the maximum value among all input values.

Syntax

```
MAX ( A )
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, DOUBLE, BOOLEAN, or VARCHAR Note: The following data types are not supported: DATE, TIME, TIMESTAMP, and VARBINARY.

Function description

This function returns the maximum value among all input values.

Examples

- Test data

var1(INT)
4
8

- Test statements

```
SELECT   MAX ( var1 )  as   aa
```

```
FROM   T1
```

- Test results

aa(INT)
8

6.11.7.7 MIN

This topic describes how to use the aggregate function MIN in Realtime Compute. In Flink SQL, the MIN function returns the minimum value among all input values.

Syntax

```
MIX ( A )
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, DOUBLE, BOOLEAN, or VARCHAR Note: The following data types are not supported: DATE, TIME, TIMESTAMP, and VARBINARY.

Function description

This function returns the minimum value among all input values.

Examples

- Test data

var1(INT)
4
8

- Test statements

```
SELECT   MIX ( var1 )  as   aa
```

```
FROM   T1
```

- Test results

aa(INT)
4

6.11.7.8 SUM

This topic describes how to use the aggregate function SUM in Realtime Compute. In Flink SQL, the SUM function returns the sum of all input values.

Syntax

```
SUM ( A )
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, or DOUBLE

Function description

This function returns the sum of all input values.

Examples

- Test data

var1(INT)
4
4

- Test statements

```
SELECT   sum ( var1 )  as   aa
FROM   T1
```

- Test results

aa(INT)
8

6.11.7.9 VAR_POP

This topic describes how to use the aggregate function VAR_POP in Realtime Compute. In Flink SQL, the VAR_POP function returns the population variance of all input values in the specified expression.

Syntax

```
T   VAR_POP ( T   value )
```

Input parameters

Parameter	Data type
value	Numeric type, such as BIGINT or DOUBLE

Function description

This function returns the population variance of all input values.

Examples

- Test data

a(BIGINT)	c(VARCHAR)
2900	Hi
2500	Hi
2600	Hi
3100	Hello
11000	Hello

- Test statements

```
SELECT
VAR_POP ( a )  as  ` result `,
 c
FROM   MyTable
GROUP   BY   c
```

- Test results

result(BIGINT)	c
28889	Hi
15602500	Hello

6.11.7.10 STDDEV_POP

This topic describes how to use the aggregate function STDDEV_POP in Realtime Compute. In Flink SQL, the STDDEV_POP function returns the population standard deviation of a set of values.

Syntax

```
T  STDDEV_POP ( T  value )
```

Input parameters

Parameter	Data type
value	BIGINT or DOUBLE

Function description

This function returns the population standard deviation of a set of values.

Examples

- Test data

a(DOUBLE)	c(VARCHAR)
0	Hi
1	Hi
2	Hi
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

- Test statements

```
SELECT  c ,  STDDEV_POP ( a )  as  dou1
FROM  MyTable
```

```
GROUP BY c
```

- Test results

c(VARCHAR)	dou1(DOUBLE)
Hi	2.8722813232690143

6.11.8 Other functions

6.11.8.1 UUID

This topic describes how to use the UUID function in Realtime Compute. In Flink SQL, the UUID function returns a universally unique identifier.

Syntax

```
VARCHAR UUID ()
```

Function description

This function returns a universally unique identifier.

Examples

- Test statements

```
SELECT uuid () as ` result `
FROM T1
```

- Test results

result(VARCHAR)
a364e414-e68b-4e5c-9166-65b3a153e257

6.11.8.2 DISTINCT

This topic describes how to use the DISTINCT function in Realtime Compute. The DISTINCT function removes duplicate records from the query result of your `SELECT` statement and returns only unique records.

DISTINCT syntax

```
SELECT DISTINCT expression s
FROM tables
...
```

- `DISTINCT` must be placed before expressions.
- `expression s` can be one or more expressions, specific columns, or any other valid expressions such as functions.

DISTINCT syntax examples

- Test statements

  The following provides an example of `DISTINCT` in Flink SQL:

```
 CREATE   TABLE   distinct_t  ab_source (
     FirstName   VARCHAR ,
     LastName   VARCHAR
) WITH (
    type =' random '
);
 CREATE   TABLE   distinct_t  ab_sink (
     FirstName   VARCHAR ,
     LastName   VARCHAR
) WITH (
     type  = ' print '
);
 INSERT   INTO   distinct_t  ab_sink
 SELECT   DISTINCT   FirstName ,  LastName  //  Remove   duplicate
 records   based   on   the   FirstName   and   LastName   columns .
 FROM   distinct_t  ab_source ;
```

- Test data

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- Test results

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN

Note:

- The test data contains four records. `DISTINCT   FirstName ,  LastName` removes one duplicate record `SUN , SHENGRAN` and returns three unique records.
- The `SUNS , HENGRAN` and `SUN , SHENGRAN` records are retained. This indicates that `DISTINCT   FirstName ,  LastName` processes the FirstName

and LastName columns separately, instead of concatenating them for deduplication.

- Alternative for DISTINCT

`GROUP BY` in SQL statements also provides a deduplication function similar to that of `DISTINCT`. The `GROUP BY` syntax is as follows:

```
SELECT   expression  s
FROM   tables
GROUP   BY   expression  s
;
```

The following writes an SQL multi-insert query to reach the equivalent effect as the DISTINCT function:

```
CREATE   TABLE   distinct_t  ab_source (
     FirstName   VARCHAR ,
     LastName   VARCHAR
) WITH (
    type =' random '
);
CREATE   TABLE   distinct_t  ab_sink (
     FirstName   VARCHAR ,
     LastName   VARCHAR
) WITH (
     type  = ' print '
);
CREATE   TABLE   distinct_t  ab_sink2 (
     FirstName   VARCHAR ,
     LastName   VARCHAR
) WITH (
     type  = ' print '
);
INSERT   INTO   distinct_t  ab_sink
     SELECT   DISTINCT   FirstName ,  LastName  //  Remove
duplicate   records   based   on   the   FirstName   and   LastName
  columns .
         FROM   distinct_t  ab_source ;
INSERT   INTO   distinct_t  ab_sink2
     SELECT   FirstName ,  LastName
         FROM   distinct_t  ab_source
          GROUP   BY   FirstName ,  LastName ; //  Remove
duplicate   records   based   on   the   FirstName   and   LastName
  columns .
```

Given the same test data, the output of GROUP BY FirstName, LastName; is the same as that of the DISTINCT function in the preceding example. This indicates that the GROUP BY statement has the same semantics as the DISTINCT function.

Use of DISTINCT in the aggregate function COUNT

The use of `DISTINCT` enables `COUNT` to count the number of records after deduplication.

```
COUNT ( DISTINCT   expression )
```

Note:
Currently, only a single expression is supported.

COUNT DISTINCT syntax examples

- Test statements

```
 CREATE   TABLE   distinct_t  ab_source (
     FirstName   VARCHAR ,
     LastName   VARCHAR
) WITH (
    type =' random '
);
 CREATE   TABLE   distinct_t  ab_sink (
     cnt   BIGINT ,
     distinct_c  nt   BIGINT
) WITH (
     type   = ' print '
);
 INSERT   INTO   distinct_t  ab_sink
     SELECT
       COUNT ( FirstName ), //  Do   not   remove   duplicate
 records .
       COUNT ( DISTINCT   FirstName ) //  Remove   duplicate
 records   based   on   the   FirstName   column .
     FROM   distinct_t  ab_source ;
```

- Test data

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- Test results

cnt	distinct_cnt
1	1
2	2
3	2

cnt	distinct_cnt
4	2

6.12 UDX

6.12.1 UDX overview

This topic describes how to build a development environment and use user defined extensions (UDXs) in Realtime Compute.

Note:
Currently, Realtime Compute does not support UDXs in shared mode. UDXs are supported only in exclusive mode.

Overview

Realtime Compute supports the following UDXs:

- UDF

  A user defined function (UDF) maps zero, one, or multiple scalar values of one record to a new scalar value.
- UDAF

  A user defined aggregation function (UDAF) aggregates multiple records into a single value.
- UDTF

  A user defined table function (UDTF) converts multiple records before generating output records. The number of output records does not need to match the number of input records. UDTFs are the only type of UDXs that can return multiple fields.

Build the development environment

The development of UDXs depends on some JAR packages of Realtime Compute. Alibaba Cloud provides a UDX development demo (`RealtimeCo mpute - udxDemo . gz`) to help you quickly build the development environment. The demo is a Maven project. You can open it in IntelliJ IDEA and develop your UDXs based on this demo.

The demo implements three simple UDXs (a UDF, a UDAF, and a UDTF) for your reference.

`RealtimeCo mpute - udxDemo . gz`

The demo depends on the following JAR packages. If you need to use a package separately, click the corresponding link to download it.

`flink - streaming - java_2 . 11`

`flink - table_2 . 11`

`flink - core - blink - 2 . 2 . 4`

Note:

After the demo package is downloaded, modify the `pom . xml` file by referring to the following example:

```
    < dependency >
        < groupId > org . apache . flink </ groupId >
        < artifactId > flink - core </ artifactId >
        < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
        < scope > provided </ scope >
    </ dependency >
    < dependency >
        < groupId > org . apache . flink </ groupId >
        < artifactId > flink - table_2 . 11 </ artifactId >
        < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
        < scope > provided </ scope >
    </ dependency >
    < dependency >
         < groupId > org . apache . flink </ groupId >
         < artifactId > flink - streaming - java_2 . 11 </ artifactId
 >
         < version > blink - 2 . 2 . 4 - SNAPSHOT </ version >
         < scope > provided </ scope >
    </ dependency >
```

Register and use a UDX

After a UDX is developed, compress it into a JAR package. On the Development page, click Upload.

After the JAR package is uploaded, select a job and declare the UDX in the job as follows:

```
CREATE  FUNCTION  stringLeng thUdf  AS  ' com . hjc . test .
blink . sql . udx . StringLeng thUdf ';
```

1. Login Realtime Compute Console.
2. Click Development at the top menu .
3. Click Resources at the left side navigation bar.

4. Click #Create Resource on the Resources Tab.
5. Input resource configuration information.

Configuration name	Description	
Upload mode	Only Upload locally is supported for now.	
Resource	Click Upload Resource icon and select the resource your need to upload.	
Resource Name	Input your resource name .	
Resource Description	Input your resource description.	
Resource Type	Choose your uploading resource type, JAR、DICTIONARY or PYTHON.	

6. In Resources tab, Hover your mouse on more.
7. Select Reference.
8. Add UDX function statement at the top of SQL query in the job edit window.

   Example as below.

```
CREATE   FUNCTION   stringLeng  thUdf   AS  ' com . hjc . test .
blink . sql . udx . StringLeng  thUdf ';
```

6.12.2 UDF

This topic describes how to build the development environment, write business code, and publish a user defined function (UDF) for Realtime Compute.

Note:

Currently, Realtime Compute does not support UDXs in shared mode. UDXs are supported only in exclusive mode.

Definition

A UDF maps zero, one, or multiple scalar values to a new scalar value.

Build the development environment

For more information, see Build the development environment.

Write business logic code

A UDF needs to implement the `eval` method in the ScalarFunction class. The `open` and `close` methods are optional. The following sample code is written in Java:

```
package   com . hjc . test . blink . sql . udx ;


import   org . apache . flink . table . functions . FunctionCo  ntext
;
import   org . apache . flink . table . functions . ScalarFunc  tion
;

public   class   StringLeng  thUdf   extends   ScalarFunc  tion  {
   //  The   open   method   is   optional .
   //  To   write   the   open   method , you   must   import   org
. apache . flink . table . functions . FunctionCo  ntext .
   @ Override
    public   void   open ( FunctionCo  ntext   context ) {
       }
    public   long   eval ( String   a ) {
        return   a  ==  null  ?  0  :  a . length ();
   }
    public   long   eval ( String   b ,  String   c ) {
        return   eval ( b ) +  eval ( c );
   }
   //  The   close   method   is   optional .
   @ Override
    public   void   close () {
       }
}
```

Publish

Locate the required class, write SQL statements, and click Publish. On the Administration page, click Start to run the function.

```
--  udf   str . length ()
 CREATE   FUNCTION   stringLeng  thUdf   AS  ' com . hjc . test .
 blink . sql . udx . StringLeng  thUdf ';
 create   table   sls_stream (
     a   int ,
     b   int ,
     c   varchar
)  with  (
     type =' sls ',
     endPoint =' yourEndpoi  nt ',
     accessKeyI  d =' yourAccess  Id ',
     accessKeyS  ecret =' yourAccess  Secret ',
     startTime  = ' 2017 - 07 - 04   00 : 00 : 00 ',
     project =' yourProjec  tName ',
     logStore =' yourLogSto  reName ',
     consumerGr  oup =' consumerGr  oupTest1 '
);
 create   table   rds_output (
     id   int ,
     len   bigint ,
     content   VARCHAR
```

```
) with (
    type =' rds ',
    url =' yourDataba seURL ',
    tableName =' yourDataba seTableNam e ',
    userName =' yourDataba seUserName ',
    password =' yourDataba sePassword '
);
 insert into rds_output
 select
    a ,
    stringLeng thUdf ( c ),
    c as content
 from sls_stream
```

FAQ

Q: Why does a user defined random number generator always generate the same value at run time?

A: If a UDF has no parameters and you do not declare the function as nondeterministic, the function may be optimized to be a constant value during compilation. To avoid this, you can override `isDetermin istic ()` to make it return `false` in the UDF.

6.12.3 UDAF

This topic describes how to build the development environment, write business code, and publish a user defined aggregation function (UDAF) for Realtime Compute.

Note:
Currently, Realtime Compute does not support UDXs in shared mode. UDXs are supported only in exclusive mode.

Definition

A UDAF aggregates multiple records into a single value.

Methods of the UDAF abstract class

The following code describes some core methods of the AggregateFunction abstract class.

Note:
Although a UDAF can be implemented in Java or Scala, we recommend that you use Java because Scala data types sometimes may result in unnecessary performance overhead.

```
/*
```

```
* @ param  < T >  The   type   of   the   UDAF   output   result .
* @ param  < ACC >  The   accumulato  r   type   of   a   UDAF .  An
 accumulato  r   stores   the   intermedia  te   computing   results
 of   a   UDAF .
*  You   can   design   an   accumulato  r   for   each   UDAF   as
 required .
*/
 public   abstract   class   AggregateF  unction < T ,  ACC >  extends
   UserDefine  dFunction  {
/*
*  Initialize   the   accumulato  r   of   AggregateF  unction .
*  The   system   calls   the   following   method   once   before
 performing   aggregate   computing   for   the   first   time :
*/
 public   ACC   createAccu  mulator ();
/*
*  The   system   calls   the   following   method   after
 completing   each   aggregate   computing :
*/
 public   T   getValue ( ACC   accumulato  r );
}
```

The input and output of the `createAccumulator` **and** `getValue` **methods are certain, and therefore the two methods can be defined in the** `AggregateFunction` **abstract class. In addition to the preceding two methods, a most basic UDAF requires an** `accumulate` **method.**

```
/*
*  You   need   to   implement   an   accumulate   method   to
 describe   how   to   compute   input   data   and   update   an
 accumulato  r   with   the   computing   result .
*  The   first   parameter   of   the   accumulate   method   must
   be   an   accumulato  r   of   the   ACC   type   defined   in
 AggregateF  unction .
*  During   the   system   operation ,  the   underlying   runtime
 code   sends   the   accumulato  r   in   the   historical   state
   and   user - specified   input   data   ( of   any   amount   and
 type )
 to   the   accumulate   method   for   computing .
*/
 public   void   accumulate ( ACC   accumulato  r , ...[  User -
 specified   parameters ]...); ;
```

The createAccumulator, getValue, and accumulate methods can be used together to design a most basic UDAF. However, Realtime Compute also requires the retract and merge methods in some special scenarios.

```
/*
*  In   Realtime   Compute ,  computing   is   an   early   firing
 for   an   infinite   stream   most   of   the   time .
*  You   may   need   to   modify   the   computing   result ,  which
   is   called   a   retraction .
*  The   SQL   optimizer   automatica  lly   determines   the
 conditions   in   which   data   to   be   retracted   is   generated
```

```
  and   the   operations   during   which   data   marked   with
 retract   tags   needs   to   be   processed .
* You   must   implement   a   retract   method   to   define   how
  the   retracted   data   is   processed . The   retract   method
 is   a   reverse   operation   of   the   accumulate   method .
* For   example , in   a   count   UDAF , the   computing   result
  increments   by   1   once   the   accumulate   method   processes
  a   data   record , and   decrements   by   1   once   the
 retract   method   processes   a   data   record .
* Similar   to   the   accumulate   method , the   first   parameter
  of   the   retract   method   must   be   an   accumulato  r   of
 the   ACC   type   defined   in   AggregateF  unction .
* During   the   system   operation , the   underlying   runtime
 code   sends   the   accumulato  r   in   the   historical   state
  and   user - specified   input   data  ( of   any   amount   and
 type )
 to   the   retract   method   for   computing .
*/
 public   void   retract ( ACC   accumulato  r , ...[  User -
 specified   parameters ]...);

/*
* The   merge   method   is   widely   used   in   batch   computing
 , as   well   as   in   some   Realtime   Compute   scenarios ,
 such   as   a   session   window .
* Because   Realtime   Compute   possesses   an   out - of - order
 feature , late - arriving   data   may   be   located   in   two
 separate   sessions , which   results   in   the   merge   of   the
  two   sessions   as   one .
* In   this   case , a   merge   method   is   required   to   merge
  multiple   accumulato  rs   into   one   accumulato  r .
* The   first   parameter   of   the   merge   method   must
 be   an   accumulato  r   of   the   ACC   type   defined   in
 AggregateF  unction .
* This   accumulato  r   stores   state   data   after   the   merge
  method   is   called .
* The   second   parameter   of   the   merge   method   is   an
 ACC - type   accumulato  r   traverse   iterator , which   may
 include   one   or   more   accumulato  rs .
*/
 public   void   merge ( ACC   accumulato  r ,  Iterable < ACC >  its
 );
```

Build the development environment

For more information, see Build the development environment.

Write business logic code

The following sample code is written in Java:

```
import   org . apache . flink . table . functions . AggregateF
unction ;

public   class   CountUdaf   extends   AggregateF  unction < Long ,
CountUdaf . CountAccum > {
   //  Define   the   data   structure   of   the   accumulato  r
that   stores   state   data   of   the   count   UDAF .
    public   static   class   CountAccum  {
        public   long   total ;
   }
```

```
    // Initialize the accumulato r of the count UDAF .
     public CountAccum createAccu mulator () {
         CountAccum acc = new CountAccum ();
         acc . total = 0 ;
         return acc ;
    }

    // getValue is a method for computing the result
 of the count UDAF based on the accumulato r that
 stores state data .
     public Long getValue ( CountAccum accumulato r ) {
         return accumulato r . total ;
    }

    // accumulate is a method for updating the
 accumulato r used by the count UDAF to store state
 data based on input data .
     public void accumulate ( CountAccum accumulato r , Object
   iValue ) {
         accumulato r . total ++;
    }

     public void merge ( CountAccum accumulato r , Iterable <
 CountAccum > its ) {
          for ( CountAccum other : its ) {
             accumulato r . total += other . total ;
         }
    }
}
```

Note:

A subclass of AggregateFunction supports both the open and close methods as optional methods. For more information, see the use of a UDF or UDTF.

Publish

Locate the required class, write SQL statements, and click Publish. On the Administration page, click Start to run the function.

```
-- UDAF count
 CREATE FUNCTION countUdaf AS ' com . hjc . test . blink . sql
 . udx . CountUdaf ';
 create table sls_stream (
 a int ,
 b bigint ,
 c varchar
) with (
 type =' sls ',
 endPoint =' yourEndpoi nt ',
 accessKeyI d =' yourAccess Key ',
 accessKeyS ecret =' yourAccess Secret ',
 startTime = ' 2017 - 07 - 04 00 : 00 : 00 ',
 project =' yourProjec tName ',
 logStore =' stream - test2 ',
 consumerGr oup =' consumerGr oupTest3 '
);
```

```
create   table   rds_output (
len1   bigint ,
len2   bigint
) with  (
type =' rds ',
url =' yourDataba  seURL ',
tableName =' yourTableN  ame ',
userName =' yourUserNa  me ',
password =' yourDataba  sePassword '
);

insert   into   rds_output
select
count ( a ),
countUdaf ( a )
from   sls_stream
```

6.12.4 UDTF

This topic describes how to build the development environment, write business code, and publish a user defined table function (UDTF) for Realtime Compute.

Note:

Currently, Realtime Compute does not support UDXs in shared mode. UDXs are supported only in exclusive mode.

Definition

Similar to a UDF, a UDTF uses zero, one, or multiple scalar values as input parameters . Different from a UDF, a UDTF returns any number of rows, rather than a single value . The returned rows can consist of one or more columns.

Build the development environment

For more information, see Build the development environment.

Write business logic code

A UDTF needs to implement the `eval` method in the TableFunction class. The `open` and `close` methods are optional. The following sample code is written in Java:

```
package   com . hjc . test . blink . sql . udx ;

import   org . apache . flink . table . functions . FunctionCo  ntext
;
import   org . apache . flink . table . functions . TableFunct  ion ;

public   class   SplitUdtf   extends   TableFunct  ion < String > {

   //  The   open   method   is   optional . To   write   the   open
  method ,  you   must   import   org . apache . flink . table .
functions . FunctionCo  ntext .
   @ Override
```

```
    public   void   open ( FunctionCo  ntext   context ) {
        // ...
        }

    public   void   eval ( String   str ) {
        String []  split  =  str . split ("\\|");
        for  ( String   s  :  split ) {
            collect ( s );
        }
    }

    //  The   close   method   is   optional .
    @ Override
    public   void   close () {
        // ...
        }

}
```

Return multiple rows

A UDTF can convert the output result from a single row to multiple rows by calling `collect()` multiple times.

Return multiple columns

A UDTF can also convert the output result from a single column to multiple columns . If you want a UDTF to return multiple columns, declare the return value as Tuple or Row. Realtime Compute supports Tuple1 to Tuple25, which define 1 to 25 fields, respectively. The following example is a UDTF that uses Tuple3 to return three fields:

```
 import   org . apache . flink . api . java . tuple . Tuple3 ;
 import   org . apache . flink . table . functions . TableFunct  ion ;

//  If   the   return   value   is   declared   as   Tuple ,  you
 must   explicitly   declare   the   generic   types   of   Tuple ,
 such   as   String ,  Long ,  and   Integer   in   this   example .
 public   class   ParseUdtf   extends   TableFunct  ion < Tuple3 <
 String ,  Long ,  Integer >> {

 public   void   eval ( String   str ) {
 String []  split  =  str . split (",");
//  The   code   is   for   demonstrat  ion   only .  In   practice ,
 more   verificati  on   logic   needs   to   be   added .
 String   first  =  split [ 0 ];
 long   second  =  Long . parseLong ( split [ 1 ]);
 int   third  =  Integer . parseInt ( split [ 2 ]);
 Tuple3 < String ,  Long ,  Integer >  tuple3  =  Tuple3 . of ( first
 ,  second ,  third );
 collect ( tuple3 );
}
}
```

Note:

If the return value is declared as Tuple, a field value cannot be null and only a maximum of 25 fields are allowed.

The following example is a UDTF that uses Row to return three fields:

```
import   org . apache . flink . table . types . DataType ;
import   org . apache . flink . table . types . DataTypes ;
import   org . apache . flink . table . functions . TableFunct  ion ;
import   org . apache . flink . types . Row ;

public   class   ParseUdtf   extends   TableFunct  ion < Row > {

public   void   eval ( String   str ) {
String []  split  =  str . split (",");
String   first  =  split [ 0 ];
long   second  =  Long . parseLong ( split [ 1 ]);
int   third  =  Integer . parseInt ( split [ 2 ]);
Row   row  =  new   Row ( 3 );
row . setField ( 0 ,  first );
row . setField ( 1 ,  second );
row . setField ( 2 ,  third );
collect ( row );
}

@ Override
//  If   the   return   value   is   declared   as   Row ,  you
must   overload   the   getResultT  ype   method   to   explicitly
inform   the   system   of   the   types   of   the   fields   to
be   returned .
public   DataType   getResultT  ype ( Object []  arguments ,  Class
[]  argTypes ) {
return   DataTypes . createRowT  ype ( DataTypes . STRING ,
DataTypes . LONG ,  DataTypes . INT );
}
}
```

Note:

If the return value is declared as Row, a field value can be null. However, you must overload the `getResultT ype` method.

SQL syntax

A UDTF supports CROSS JOIN and LEFT JOIN. When using a UDTF, you need to add the keywords `LATERAL` and `TABLE`. Take `ParseUdtf` described above as an example. First, you need to register a function name.

```
CREATE   FUNCTION   parseUdtf   AS  ' com . alibaba . blink . sql .
udtf . ParseUdtf ';
```

CROSS JOIN: Each row in the left table correlates with each row of data generated by the UDTF. If the UDTF does not generate any data for a row, the row is not exported.

```
select   S . id ,  S . content ,  T . a ,  T . b ,  T . c
from   input_stre  am   as   S ,
LATERAL   TABLE ( parseUdtf ( content ))  as   T ( a ,  b ,  c );
```

LEFT JOIN: Each row in the left table correlates with each row of data generated by the UDTF. If the UDTF does not generate any data for a row, the UDTF fields in the row are populated with null.

> **Note:**
> A LEFT JOIN statement that uses a UDTF must end with `ON   TRUE`.

```
select   S . id ,  S . content ,  T . a ,  T . b ,  T . c
from   input_stre  am   as   S
LEFT   JOIN   LATERAL   TABLE ( parseUdtf ( content ))  as   T ( a ,
b ,  c )  ON   TRUE ;
```

Publish

Locate the required class, write SQL statements, and click Publish. On the Administration page, click Start to run the function.

```
--  UDTF   str . split ("\\|");
 CREATE   FUNCTION   splitUdtf   AS  ' com . hjc . test . blink . sql
 . udx . SplitUdtf ';

 create   table   sls_stream (
 a   int ,
 b   bigint ,
 c   varchar
)  with  (
 type =' sls ',
 endPoint =' yourEndpoi  nt ',
 accessKeyI  d =' yourAccess  Id ',
 accessKeyS  ecret =' yourAccess  Secret ',
 startTime  = ' 2017 - 07 - 04   00 : 00 : 00 ',
 project =' yourProjec  tName ',
 logStore =' stream - test2 ',
 consumerGr  oup =' consumerGr  oupTest2 '
);

--  Pass   the   c   field   to   splitUdtf   to   generate   table
 T ( s )  that   consists   of   one   column   and   multiple   rows
   after   splitting .  In   the   table   name ,  s   indicates   the
   field   name .
 create   view   v1   as
 select   a , b , c , s
 from   sls_stream ,
 LATERAL   TABLE ( splitUdtf ( c ))  as   T ( s );
```

```
 create   table   rds_output (
 id   int ,
 len   bigint ,
 content   VARCHAR
)  with  (
 type =' rds ',
 url =' yourDataba  seURL ',
 tableName =' yourDataba  seTableNam  e ',
 userName =' yourDataba  seUserName ',
 password =' yourDataba  sePassword '
);

 insert   into   rds_output
 select
 a , b , s
 from   v1
```

6.12.5 Develop a UDX by using IntelliJ IDEA

This topic describes how to use IntelliJ IDEA to develop a UDX for Realtime Compute, including building the development environment and referencing the UDX in Realtime Compute jobs.

Note:

- The operations in this topic are based on IntelliJ IDEA. Download and install the tool first.
- Currently, Realtime Compute does not support UDXs in shared mode. UDXs are supported only in exclusive mode.

Download and configure Maven

1. Download Maven.
    a. Visit the official download page of Maven and download `apache - maven - 3 . 5 . 3 - bin . tar . gz`.
    b. Decompress the downloaded package to the specified directory, such as `/ Users / xxx / Documents / maven`.
2. Configure environment variables.
    a. Open Terminal and run the `vim ~/.bash_profile` command.
    b. In the `. bash_profi  le` file, add the following commands for configuring environment variables:

```
export   M2_HOME =/ Users / xxx / Documents / maven / apache -
maven - 3 . 5 . 3
```

```
export   PATH =$ PATH :$ M2_HOME / bin
```

c. Save the configuration and exit. Run the following command to make the configuration take effect: `source ~/.bash_profile`

3. Verify that the configuration takes effect.

Run the `mvn -v` command. If a similar output is displayed, the configuration takes effect:

```
Apache   Maven   3 . 5 . 0  ( ff8f5e7444  045639af65  f6095c6221
0b5713 ****;  2017 - 04 - 04T03 : 39 : 06 + 08 : 00 )
Maven   home : / Users / xxx / Documents / maven / apache - maven -
3 . 5 . 0
Java   version :  1 . 8 . 0_121 ,  vendor :  Oracle   Corporatio
n
Java   home : / Library / Java / JavaVirtua  lMachines / jdk1 . 8 .
0_121 . jdk / Contents / Home / jre
Default   locale :  zh_CN ,  platform   encoding :  UTF - 8
OS   name : " mac   os   x ",  version : " 10 . 12 . 6 ",  arch : "
x86_64 ",  family : " mac "
```

Build the development environment

1. Visit Build the development environment and download the demo package *RealtimeCo mpute - udxDemo . gz* .

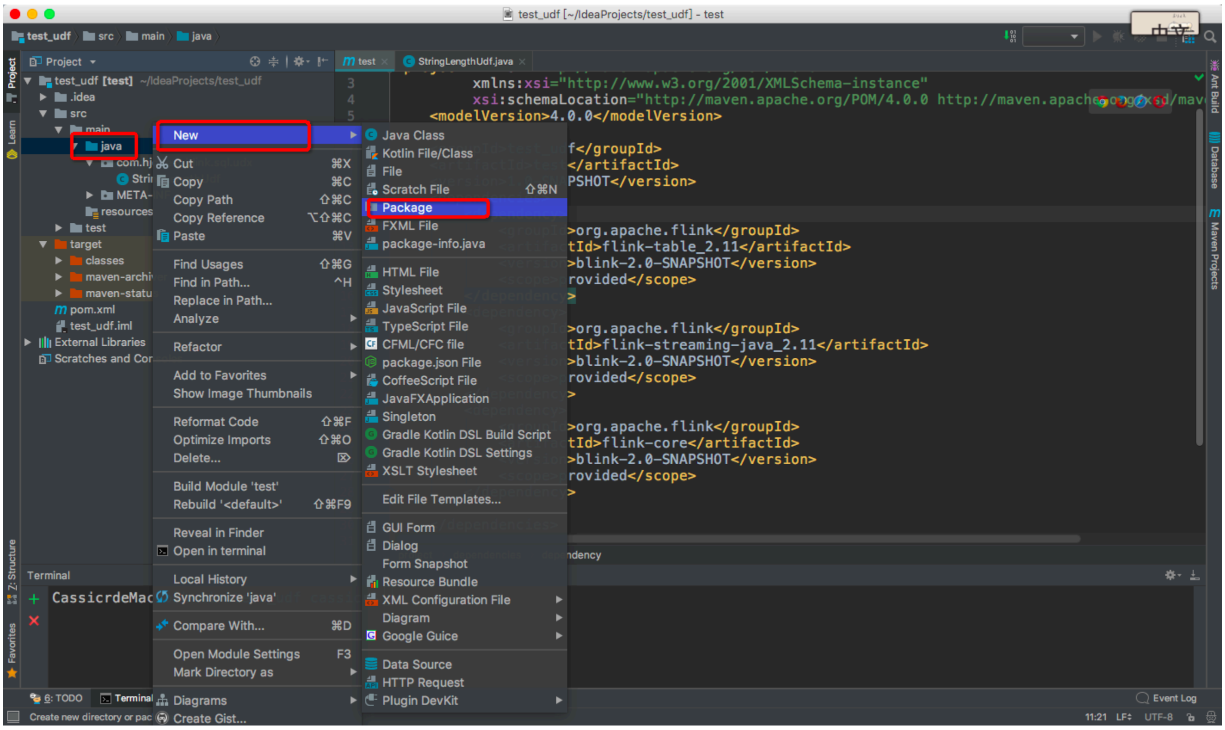
2. Decompress *RealtimeCo mpute - udxDemo . gz* in a Linux environment.

```
tar   xzvf   RealtimeCo  mpute - udxDemo . gz
```

3. Open IntelliJ IDEA and click Open to open the demo.

Create a package

The procedure is shown in the following figure.

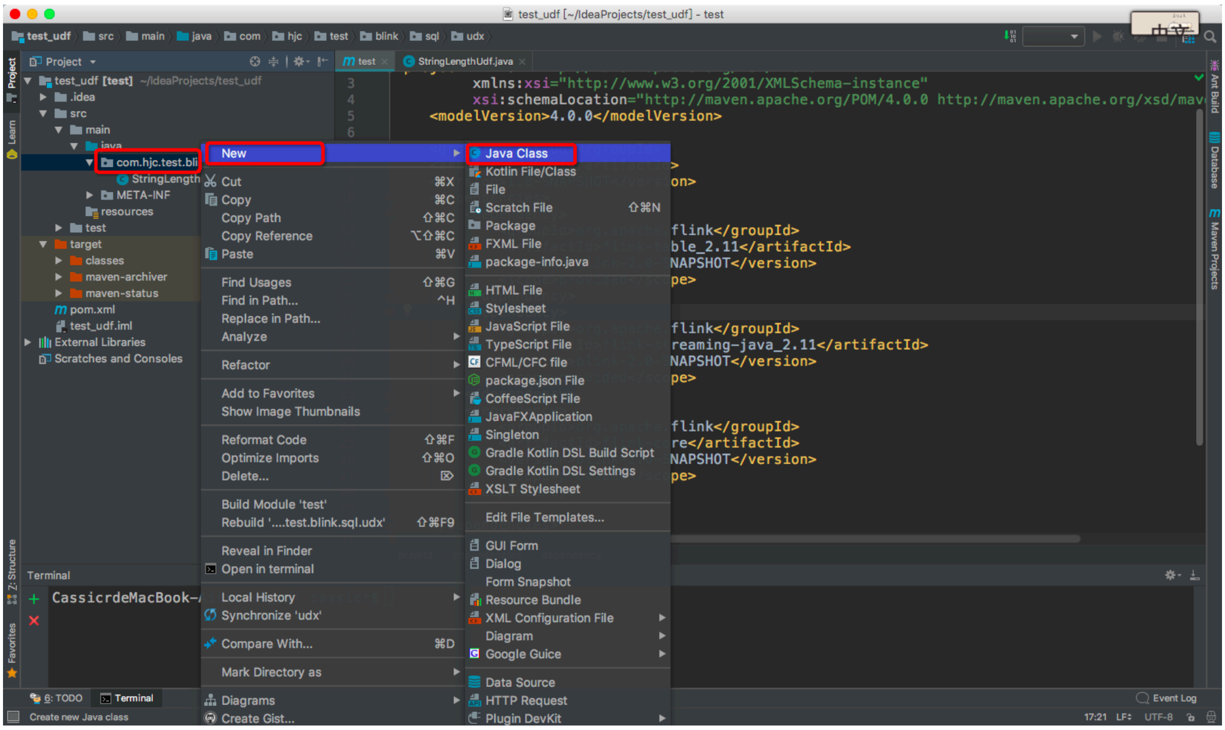


In this example, the `com . hjc . test . blink . sql . udx` package is created, as shown in the following figure.

```
package com.hjc.test.blink.sql.udx;
```

Create a class

The procedure is shown in the following figure.



Paste the test code into the class

Copy and paste the following UDX sample code into the class:

```
package   com . hjc . test . blink . sql . udx ;

import   org . apache . flink . table . functions . FunctionCo  ntext
;
import   org . apache . flink . table . functions . ScalarFunc  tion
;

public   class   StringLeng  thUdf   extends   ScalarFunc  tion  {
    //  The   open   method   is   optional .
    //  To   write   the   open   method ,  you   must   import   org
. apache . flink . table . functions . FunctionCo  ntext .
    @ Override
     public   void   open ( FunctionCo  ntext   context ) {
        }
     public   long   eval ( String   a ) {
         return   a  ==  null  ?  0  :  a . length ();
    }
     public   long   eval ( String   b ,  String   c ) {
         return   eval ( b ) +  eval ( c );
    }
    //  The   close   method   is   optional .
    @ Override
     public   void   close () {
        }
}
```

Compress the project into a JAR package

1. On Terminal, run the `mvn package` or `mvn assembly:assembly` command. If you need to add required third-party packages to the JAR package, use the latter command.
2. The compiled JAR package is `RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT.jar` or `RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT-jar-with-dependencies.jar` (containing required third-party packages).

Reference the JAR package in Realtime Compute jobs

Please refer to Register and use a UDX