# Stream Compute Service

# SQL Developer Guide

# Product Documentation

# Contents

# SQL Developer Guide

## Overview

Last updated：2023-11-08 15:52:57

Because SQL is easier to learn than other programming languages, SQL jobs help lower the barrier for data developers to use Flink.
In this section, you will learn how to develop a SQL job with a private cluster. The following topics are covered:

# Glossary and Data Types

## Glossary

Last updated：2023-11-08 15:51:46

The table below lists some of the commonly used terms related to SQL jobs.

| Term | Description |
|---|---|
| Stream computing | Stream computing is the computing of stream data. It reads data in stream form from one or more data sources, efficiently computes the continuous data streams using multiple operators of the engine, and outputs the results to different sinks such as message queues, databases, data warehouses, and storage services. |
| Data source | The source that continuously generates data for stream computing, such as CKafka. |
| Data sink | The destination of the results of stream computing, such as CKafka, TencentDB for MySQL, and TencentDB for PostgreSQL. |
| Schema | The structure information of a table, such as column headings and types. In the context of PostgreSQL, a schema is smaller than a database and larger than a table. It can be seen as a namespace inside a database. |
| Time mode | The time mode determines how the system obtains timestamp information when processing data. Currently, three time modes are supported, namely Event Time, Processing Time, and Source Time |
| Event Time | In the Event Time mode, timestamp information is offered by an input field. You can use the WATERMARK FOR statement to specify this field and enable the Event Time mode. This mode is suitable for scenarios where the data source offers precise timestamp information. |
| Watermark | A watermark is a time point. All data before this time point have been properly processed. Watermarks are automatically generated. You can use the WATERMARK FOR BOUNDED statement to specify the maximum error tolerance. |
| Processing Time | In the Processing Time mode, the system automatically generates timestamps ( `PROCTIME` ) and adds them to the data source ( `PROCTIME` is invisible to `SELECT *` . You need to specify it explicitly). In this mode, a timestamp is the time each data record is processed. Because such data has some uncertainty, this mode is suitable for scenarios that do not have high requirements on precision. |
| Source Time | In the Source Time mode, you can use the timestamp in the metadata of each Kafka record as the timestamp ( `SOURCETIME` ) for stream computing ( `SOURCETIME` is invisible to `SELECT *` . You need to specify it explicitly). In cases where the input data |

| | |
|---|---|
| | does not include a timestamp field, this mode eliminates the uncertainty of the Processing Time mode. |
| Time window | A time window specifies multiple time ranges and the relationships among them, for example, whether they can overlap or whether the window length is fixed. Currently, three types of time windows are supported, namely TUMBLE, HOP, and SESSION. For details, see Time Window Functions. |
| TencentDB for MySQL | TencentDB is a database management service featuring high performance, availability, and scalability. It helps you easily deploy and use MySQL, PostgreSQL, and other databases on the cloud. |
| CKafka | CKafka is a high-throughput, highly scalable distributed messaging system that is fully compatible with open-source Apache Kafka API 0.10. Currently, Stream Compute Service supports inputs and outputs in CSV and JSON formats. |
| Tuple (Append) stream | A tuple (append) can store stream data that does not contain a primary key. You can append data to a tuple continuously. Already sent data is not affected. Currently, append streams are supported by all data sources and sinks. |
| Upsert stream | Upsert (update or insert) streams are generated by queries such as DISTINCT, GROUP BY without a time window, and JOIN without a time range. A primary key is defined for upsert streams. If a new data record has the same primary key as an existing record, the existing data will be updated. Otherwise, a new record will be added. This ensures that existing data is up-to-date. |
| DDL statements | Data Definition Language (DDL) is a subset of SQL consisting of CREATE statements. You can use DDL statements to define a table, a view, and a user-defined function (UDF). |
| DML statement | Data Manipulation Language (DML) is a subset of SQL consisting of INSERT and SELECT statements. You can use DML statements to select, change, filter, and insert tables and views. |

# Data Type

Last updated：2023-11-08 15:51:16

Stream Compute Service uses definitions that follow the ANSI SQL standard and supports a great variety of data types. When creating a table with `CREATE TABLE`, you can use the following data types to specify the type of each field.

## Supported data types

| Data Type | Description |
|---|---|
| CHAR<br>CHAR(n) | A fixed-length string. `n` indicates the number of characters, which is `1` by default. `CHAR` is equivalent to `CHAR(1)`. |
| VARCHAR<br>VARCHAR(n)<br>STRING | A variable-length string. `n` indicates the maximum number of characters allowed, which is `1` by default. `VARCHAR` is equivalent to `VARCHAR(1)`. `STRING` is equivalent to `VARCHAR(2147483647)`. |
| BINARY<br>BINARY(n) | A fixed-length binary string. `n` indicates the number of bytes of the string, which is `1` by default. `BINARY` is equivalent to `BINARY(1)`. |
| VARBINARY<br>VARBINARY(n)<br>BYTES | A variable-length binary string. `n` indicates the maximum number of bytes allowed. `VARBINARY` is equivalent to `VARBINARY(1)`. `BYTES` is equivalent to `VARBINARY(2147483647)`. |
| DECIMAL<br>DECIMAL(p)<br>DECIMAL(p, s)<br>DEC<br>DEC(p)<br>DEC(p, s)<br>NUMERIC<br>NUMERIC(p)<br>NUMERIC(p, s) | A fixed-precision number (fixed-point number).<br>`p` indicates the total number of digits (precision). The value range is [1, 38], and the default value is `10`.<br>`s` indicates the number of digits to the right of the decimal point (scale). The value range is [0, p], and the default value is `0`. `DEC`, `NUMERIC`, and `DECIMAL` are interchangeable data type names. That is, `DECIMAL(p, s)` is equivalent to `DEC(p, s)` and `NUMERIC(p, s)`. |
| TINYINT | A 1-byte integer. This is equivalent to `Byte` in Java. The value range is [-128, 127]. |
| SMALLINT | A 2-byte integer. This is equivalent to `Short` in Java. The value range is [-32768, 32767]. |
| INT | A 4-byte integer. This is equivalent to `Integer` in Java. The value range is |

| | |
|---|---|
| | [-2147483648, 2147483647]. |
| BIGINT | An 8-byte integer. This is equivalent to `Long` in Java. The value range is [-9223372036854775808, 9223372036854775807]. |
| FLOAT | A 4-byte, single-precision floating-point number. This is equivalent to `Float` in Java. |
| DOUBLE | An 8-byte, double-precision floating-point number. This is equivalent to `Double` in Java. |
| DATE | Date (yyyy-mm-dd). The value range is [0000-01-01, 9999-12-31]. |
| TIME<br>TIME(p) | A time without time zone (hh:mm:ss.nnnnnnnnn).<br>The value range is [00:00:00.000000000, 23:59:59.999999999].<br>`p` indicates the number of digits of fractional seconds. The value range is [0, 9]. The default value is `0` .<br>The leap second is not supported. This data type is similar to `LocalTime` in Java. |
| TIMESTAMP<br>TIMESTAMP(p)<br>TIMESTAMP WITHOUT TIME ZONE<br>TIMESTAMP(p) WITHOUT TIME ZONE | A timestamp without time zone, which can be accurate to the nanosecond.<br>The value range is [0000-01-01 00:00:00.000000000, 9999-12-31 23:59:59.999999999].<br>`p` indicates the number of digits of fractional seconds. The value range is [0, 9]. The default value is `6` .<br>Conversion between this data type and `BIGINT` ( `Long` in Java) is not supported. `TIMESTAMP WITHOUT TIMEZONE` is equivalent to `TIMESTAMP` .<br>The leap second is not supported. This data type is similar to `Timestamp` in Java. |
| TIMESTAMP WITH TIME ZONE<br>TIMESTAMP(p) WITH TIME ZONE | A timestamp with time zone.<br>The value range is [0000-01-01 00:00:00.000000000 +14:59, 9999-12-31 23:59:59.999999999 -14:59].<br>`p` indicates the number of digits of fractional seconds. The value range is [0, 9]. The default value is `6` .<br>Each data entry of this type has its time zone information.<br>The leap second is not supported. This data type is similar to `OffsetDateTime` in Java. |
| TIMESTAMP WITH LOCAL TIME ZONE<br>TIMESTAMP(p) WITH LOCAL TIME ZONE | A timestamp with local time zone.<br><br>The value range is [0000-01-01 00:00:00.000000000 +14:59, 9999-12-31 23:59:59.999999999 -14:59]. |

| | |
|---|---|
| | `p` indicates the number of digits of fractional seconds. The value range is [0, 9]. The default value is `6` .<br>Instead of being stored in each data entry, the time zone information is based on the global time zone settings.<br>The leap second is not supported. This data type is similar to `OffsetDateTime` in Java. |
| INTERVAL YEAR<br>INTERVAL YEAR(p)<br>INTERVAL YEAR(p) TO MONTH<br>INTERVAL MONTH | A year-month interval.<br>The format is "+Number of years-Number of months", for example, "+04-02".<br>The value range is [-9999-11, +9999-11].<br>`p` indicates the number of digits for year. The value range is [1, 4], and the default value is `2` . |
| INTERVAL DAY<br>INTERVAL DAY(p1)<br>INTERVAL DAY(p1) TO HOUR<br>INTERVAL DAY(p1) TO MINUTE<br>INTERVAL DAY(p1) TO SECOND(p2)<br>INTERVAL HOUR<br>INTERVAL HOUR TO MINUTE<br>INTERVAL HOUR TO SECOND(p2)<br>INTERVAL MINUTE<br>INTERVAL MINUTE TO SECOND(p2)<br>INTERVAL SECOND<br>INTERVAL SECOND(p2) | A day-time interval, which can be accurate to the nanosecond.<br>The value range is [-999999 23:59:59.999999999, +999999 23:59:59.999999999].<br>`p1` indicates the number of digits for day. The value range is [1, 6], and the default value is `2` .<br>`p2` indicates the number of digits of fractional seconds. The value range is [0, 9], and the default value is `6` . |
| ARRAY<t><br>t ARRAY | An array, whose length is at most 2147483647.<br>`t` indicates the data type of elements in the array.<br>`ARRAY<t>` is equivalent to `t ARRAY` . For example, `ARRAY<INT>` is the same as `INT ARRAY` . |
| MAP<kt, vt> | A key-value mapping. `kt` indicates the data type of the key, and `vt` indicates the data type of the value. |
| MULTISET<t><br>t MULTISET | A set that can include duplicate element values, which is also known as a bag.<br>`MULTISET<t>` is equivalent to `t MULTISET` . |
| ROW<n0 t0, n1 t1, ...><br>ROW<n0 t0 'd0', n1 t1 'd1', ...> | A complex data type that can store multiple fields, which is similar to `Struct` or `Tuple` in other programming languages.<br>`n` is the field name. |

| | |
|---|---|
| ROW(n0 t0, n1 t1, ...)<br>ROW(n0 t0 'd0', n1 t1 'd1', ...) | `t` is the logical type of the field.<br>`d` is the field description.<br>Angle and round brackets are interchangeable. That is, `ROW(field1 INT, field2 BOOLEAN)` is the same as `ROW<field1 INT, field2 BOOLEAN>`. |
| BOOLEAN | A three-valued Boolean, whose valid values include `TRUE`, `FALSE`, and `UNKNOWN`. If you don't want to allow `UNKNOWN`, use `BOOLEAN NOT NULL`. |
| RAW('class', 'snapshot') | An arbitrary type. You can use this for data types that Fink does not recognize or does not need to recognize.<br>`class` is the original data type.<br>`snapshot` is the serialized `TypeSerializerSnapshot` in Base64 encoding. |
| NULL | A null type, which is similar to `null` in Java. |

# DDL Statements

## CREATE TABLE

Last updated：2023-11-08 15:28:58

The `CREATE TABLE` statement describes a data source or a sink and defines it as a data table.

## Syntax

**Syntax structure**

```
CREATE TABLE Table name
  (
    { <Column definition> | <Calculated column definition> }[ , ...n]
    [ <Watermark definition> ]
    [ <Definitions of table constraints such as the primary key> ][ , ...n]
  )
  [COMMENT Table comments]
  [PARTITIONED BY (grouped column name 1, grouped column name 2, ...)]
  WITH (key 1=value 1, key 2=value 2, ...)
  [ LIKE another table [( <LIKE clause> )] ]
```

## Symbols

A table created by `CREATE TABLE` can be used as a data source or a data sink. However, there must be a corresponding connector; otherwise, an error will occur.

```
<Column definition>:
  Column name Column type [ <Definitions of column constraints> ] [COMMENT Column c

<Definitions of column constraints>:
  [CONSTRAINT constraint name] PRIMARY KEY NOT ENFORCED
```

```
<Definitions of table constraints>:
  [CONSTRAINT constraint name] PRIMARY KEY (column 1, column 2, ...) NOT ENFORCED

<Calculated column definition>:
  Column name AS calculated column expression [COMMENT column comments]

<Watermark definition>:
  WATERMARK FOR a ROWTIME column AS a watermark generation expression

<LIKE clause>:
{
   { INCLUDING | EXCLUDING } { ALL | CONSTRAINTS | PARTITIONS }
 | { INCLUDING | EXCLUDING | OVERWRITING } { GENERATED | OPTIONS | WATERMARKS }
}[, ...]
```

## Clauses

### Calculated column

A calculated column is a virtual column. It is logically defined but does not actually exist in the data source. Calculated columns are usually the result of calculation based on other columns, constants, variables, or functions in the same table. For example, if your data source has "price" and "quantity", you can define a calculated column "cost" using `cost AS price * quantity` so that you can query cost data from the table.
A more common use of calculated columns is timestamp standardization. Assume that a data source has a timestamp field `mytime` , which is in Unix format (such as 1599469771494). You can use `ts AS TO_TIMESTAMP(FROM_UNIXTIME(mytime / 1000, 'yyyy-MM-dd HH:mm:ss'))` to convert the timestamp to `Timestamp(3)` so that it can be recognized by Flink. There may be cases where, although the timestamp is in `Timestamp(3)` format, it is embedded in another JSON field. In such cases, you can also use calculated columns to parse the field and get the timestamp.
**Note**
Calculated columns can only be used in the SELECT statement.
With INSERT, calculated columns in the sink will be ignored.

### Watermark

#### Watermark definition

Watermark determines the time mode (for details, see "Event Time/Processing Time" below) of a Flink job. Here is how you define a watermark:

```
WATERMARK FOR a ROWTIME column AS a watermark generation expression
```

For example, `WATERMARK FOR my_time_field AS my_time_field - INTERVAL '3' SECOND` defines a watermark policy with a tolerance of 3 seconds.

The ROWTIME column must be in `Timestamp(3)` format so that it can be recognized by Flink. Embedded fields are not supported. If the column is not in the required format or is an embedded field, create a virtual calculated column using the method described above.

The watermark generation expression determines how a watermark is generated. It describes a value in `Timestamp(3)` format.

Below is an example:

```
CREATE TABLE StudentRecord (
    Id BIGINT,
    StudentName STRING,
    RegistrationTime TIMESTAMP(3),
    WATERMARK FOR RegistrationTime AS RegistrationTime - INTERVAL '3' MINUTE
) WITH ( ... ... );
```

**Watermark generation policy**

**Watermark policy for monotonically increasing timestamp**

If your timestamp is monotonically increasing, to keep the data latency as low as possible, you can use the following statement.

```
WATERMARK FOR A ROWTIME column AS A ROWTIME column
```

The following statement uses the largest timestamp in each input data record as the watermark. If the timestamp is out of order, data that arrives late will be discarded due to failure to meet the watermark threshold. Example:

```
WATERMARK FOR my_time AS my_time
```

**Watermark policy with out-of-order tolerance**

This kind of policy has a certain degree of out-of-order tolerance. You can define your own tolerance. Note that a large tolerance may result in a high data latency (backlog, waiting), and a small tolerance may cause data that exceeds the threshold to be dropped, leading to inaccurate results.

```
WATERMARK FOR a ROWTIME column AS a ROWTIME column - INTERVAL 'Time window' Time un
```

The following statement uses the largest timestamp in each input data record minus the 3-second tolerance window as the watermark. In case of out-of-order data, if the difference between the data arrival time and the largest timestamp is within 3 seconds, the data will be counted in calculation.

```
WATERMARK FOR my_time AS my_time - INTERVAL '3' SECOND
```

**Event Time/Processing Time**

For window-based operations (such as the specification of time ranges by GROUP BY, OVER, and JOIN), Stream Compute Service supports two time modes: Event Time and Processing Time.

Event Time uses timestamp information in the input data and has a certain degree of out-of-order tolerance (for example, it can tolerate late arrivals caused by reasons such as network fluctuations and the varying processing capabilities of different nodes). You can specify the tolerance (in milliseconds) using the second parameter of

`BOUNDED` . This mode is more accurate, but to use it, you must include timestamp information in the input data. Currently, this mode is only supported for timestamp-type fields in the source. In the future, we will add support for virtual columns, which you can use to convert columns of other data types to timestamp data that the system can recognize.
With the Processing Time mode, the input data does not need to include timestamp information. The processing time is automatically added to the data and is assigned to the field `PROCTIME` (all capitals). This column is hidden and will not be read by `SELECT *` . It is read only if you specify it manually.
**Note**
You need to use the same time mode for all data sources of the same task. If Event Time is used, you must define the timestamp and declare a watermark field for all table sources defined.

### PRIMARY KEY

When defining a table or view, you can declare some fields as the primary key so that their values will not be duplicate or null (equivalent to `NOT NULL + UNIQUE` in SQL).
You can define columns as the primary key or use the CONSTRAINT clause. **You cannot define different primary keys for the same table.** Flink does not guarantee the uniqueness of the primary key of each data record, so only the `PRIMARY KEY NOT ENFORCED` clause is supported. This reminds users that they need to ensure the primary keys are defined as required.

### PARTITIONED BY

If a PARTITIONED BY clause is used for a column, it indicates that Flink can partition the column. This mainly affects the FileSystem sink, which creates a separate catalog for different data partition.
We don't recommend you use the FileSystem sink because the data of the file system will be automatically cleared after all TaskManager tasks are executed.

### WITH parameters

WITH parameters are usually used to specify connector parameters for the source and sink. The syntax is `'key1'='value1', 'key2' = 'value2'` .
For example, to use Kafka (Tencent Cloud or self-built Kafka), you need to specify parameters such as the server address, the topic, and the consumption start time.
For how to use some of the common connectors, see Connectors.

### LIKE clause

When creating a table (table B), you can use the LIKE clause to use the structure of another table (table A). This can significantly reduce the code size of CREATE TABLE. Assume that you want to write the same data to a Kafka sink, an Elasticsearch sink, and a MySQL sink. You can use the LIKE clause to define three tables where the column definition is reused.
Define table A:

```
CREATE TABLE A (
    Id BIGINT,
    StudentName STRING,
    RegistrationTime TIMESTAMP(3)
) WITH ( ... Some parameters ... );
```

Use table A to define table B (with watermark):

```
CREATE TABLE B (
    WATERMARK FOR RegistrationTime AS RegistrationTime - INTERVAL '3' MINUTE
) WITH ( ... Some other parameters ... ) LIKE `A`;
```

By default, the LIKE clause and WITH parameters are unrelated, so you can use different WITH parameters for the two tables. To reuse the WITH parameters of table A, use **LIKE clause options**.

#### LIKE clause options

The LIKE clause offers the following options which you can use to specify the content of table A to reuse:

CONSTRAINTS: Primary key and other constraints

GENERATED: Calculated column
OPTIONS: WITH parameters
PARTITIONS: PARTITIONED BY definition
WATERMARKS: WATERMARK FOR definition
ALL: All above
Flink provides three merging strategies:
INCLUDING: Table B will inherit all specified properties of table A. If table A and table B have conflicting definitions (for example, if they both have definitions for the same field), an error will occur.
EXCLUDING: Table B **will not** include the specified properties of table A.
OVERWRITING: Table B will inherit all specified properties of table A. If table A and table B have conflicting definitions, table B's definitions will overwrite table A's.
If LIKE clause options are not specified, `INCLUDING ALL OVERWRITING OPTIONS` will be executed. This means table B will inherit all definitions and settings of table A, but will overwrite table A's WITH parameters.

# CREATE VIEW

Last updated：2023-11-08 15:29:24

You can use the `CREATE VIEW` statement to create a view. A view is a virtual table that is generated based on a SELECT statement. It is useful in scenarios such as type conversion, column conversion, virtual columns, and long code splitting.

### Syntax

```
CREATE VIEW view name AS
SELECT clause
```

### Example 1

Create a view with the name "MyView":

```
CREATE VIEW MyView AS
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
```

```
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

### Example 2

Due to reasons such as large data volumes or the use of specific function types, you may need to use data types such as TINYINT and SMALLINT. Such data types are not recognized by Kafka and some other data sources. In such cases, you can use the `CREATE VIEW` statement together with the type conversion function `CAST()` (see Type Conversion Functions) to define a virtual table and use it as the data source.
Define a view named "KafkaSource2" to convert the `status_code` column, whose data type is BIGINT, to the data type VARCHAR:

```
CREATE VIEW KafkaSource2 AS
SELECT
  `time_`,
  `client_ip`,
  `method`,
  CAST(`status_code` AS VARCHAR) AS status_code,
FROM KafkaSource1;
```

**Note**

Some `CAST()` conversions, for example, from BIGINT to INTEGER or BIGINT to TINYINT, may lead to loss of precision.

To convert between string (VARCHAR) and timestamp (TIMESTAMP), see `TO_TIMESTAMP` and `DATE_FORMAT` functions in Date and Time Functions.

# CREATE FUNCTION

Last updated：2023-11-08 15:12:12

For a SQL job, you can upload a custom package and reference this package when configuring parameters for the job in the console. You can either upload a local package or use resources in the COS buckets (must be in the same region) of the current account.
The package can be used as additional connector features. You can also use it to create user-defined functions (UDFs).

## Syntax

Currently, Stream Compute Service supports packages written in Java and Scala. After uploading a custom package and referencing it in the console, you can use the following `CREATE FUNCTION` statement to declare it:

```
CREATE TEMPORARY SYSTEM FUNCTION function name
  AS 'Full function class name' [LANGUAGE JAVA|SCALA]
```

You can determine your own **function name**. Make sure it is unique. **Full function class name** should be the full name of a Java or Scala class, such as `'com.example.flink.MyCustomFunction'` .

## Overwriting in case of identical names

If there is a built-in function with the same name, it will be overwritten by the UDF created using the above statement. Therefore, please avoid using a function name that is identical to a built-in function, unless you intend to overwrite the built-in function.

## Function types

Flink supports the following functions.

### Scalar function

A user-defined scalar function converts one value to another value (one-to-one). Examples include the built-in string handling functions `SUBSTRING` and `REPLACE` .

### Table function

A user-defined table function converts a value to a row of data (one-to-many) so that it can be used as the right table in JOIN operations.

### Aggregate function

A user-defined aggregate function aggregates a set of values from multiple data rows into one value (many-to-one). Examples include the built-in functions `MAX` , `MIN` , and `AVG` .

### Table aggregate function

A table aggregate function aggregates a set of values from multiple data rows into a new set of data rows (many-to-many).

### Async table function

An async table function can be used as a special data source. For example, you can use it to connect to an external database or storage.

## UDF development

Flink updates its APIs and documentation on a regular basis. For details, see the Flink document User-defined Functions. Stream Compute Service is compatible with the APIs of the open-source Flink 1.11.

# DML Statements

## SELECT

Last updated：2023-11-08 15:32:35

### SELECT FROM

The SELECT statement must be used together with `CREATE VIEW … AS` or `INSERT INTO` ; otherwise, a no operator error will occur.

**Syntax**

```
SELECT Fields to select, separated with commas
FROM Data source or view
WHERE Filter condition
Other subqueries
```

### Example

```
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

## WHERE

You can use `WHERE` to filter the data to select from. Combine multiple filter conditions with `AND` or `OR` . When a TencentDB table is joined, only `AND` is supported. To use `OR` , see `UNION ALL` below.

## HAVING

You can use `HAVING` to filter the result of `GROUP BY`. `WHERE` filters data before `GROUP BY`, while `HAVING` filters data after `GROUP BY`.

```
SELECT SUM(amount)
FROM Orders
WHERE price > 10
GROUP BY users
HAVING SUM(amount) > 50
```

# GROUP BY

In Stream Compute Service, `GROUP BY` arranges query results into groups. It supports `GROUP BY` with a time window and `GROUP BY` without (continuous query).

`GROUP BY` with a time window does not update previous results and therefore generates append (tuple) data streams. Such data can only be written to MySQL, PostgreSQL, Kafka, and Elasticsearch sinks that do not have a primary key.

`GROUP BY` without a time window **updates** previously sent records and therefore generates upsert data streams. Such data can be written to MySQL, PostgreSQL, and Elasticsearch sinks with a primary key (the primary key must be identical to the upsert field in the `GROUP BY` statement).

## `GROUP BY` with a time window

The example below defines a `GROUP BY` query with a time window. For details about time window functions, see Time Window Functions.

```
SELECT user, SUM(amount)
FROM Orders
GROUP BY TUMBLE(rowtime, INTERVAL '1' DAY), user
```

In Event Time mode (the WATERMARK FOR statement is used to define the timestamp field), the first parameter of the TUMBLE function must be the timestamp field. The same is true for HOP and SESSION.
In Processing Time mode, the first parameter of the TUMBLE function must be the field declared by `proctime()` . The same is true for HOP and SESSION.

### `GROUP BY` without a time window (continuous query)

The example below defines a `GROUP BY` query without a time window. This is known as a continuous query. It determines whether to update previously sent results depending on each arriving data record and therefore generates an upsert stream.

```
SELECT a, SUM(b) as d
FROM Orders
GROUP BY a
```

**Note**

Out of Memory errors may occur for such queries due to too many keys or too much data. Please consider this when setting the timeout period. Do not set it too long.

## JOIN

Currently, Stream Compute Service only supports `Equi-JOIN`. That is to say, the `JOIN` query must include at least one filter condition that equates a field in the left and right tables.

### Inner Equi-JOIN (stream join)

There are two types of stream join: `JOIN` with a time range and `JOIN` without a time range. The former generates append (tuple) streams, while the latter generates upsert streams.

#### Inner JOIN with a time range

A `JOIN` query with a time range is also known as **interval join**. The `WHERE` clause of such queries must have at least one equality join condition and a time range. The time range can be specified using <, <=, >=, > or `BETWEEN … AND`.

```
ltime = rtime
ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND
```

**Example:**

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.orderId AND
o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime
```

**Inner JOIN without a time range**

An inner join without a time range must have at least one equality join condition, but does not need to specify a time range. This means all historical data will be used for calculation (you can specify a timeout period to exclude inactive elements).

**Note**

Such queries may significantly drive up memory usage. We recommend you set an appropriate timeout period to exclude inactive objects.

Such queries generate upsert streams and therefore can only use MySQL, PostgreSQL, and Elasticsearch sinks that have a primary key.

**Example:**

```
SELECT *
FROM Orders INNER JOIN Product ON Orders.productId = Product.id
```

### Outer Equi-JOIN

Outer Equi-JOIN generates upsert streams and therefore can only use MySQL, PostgreSQL, and Elasticsearch sinks that accept such streams.

**Note**

Because the join order is not optimized, the `JOIN` query will follow the order of tables in the `FROM` clause. This may result in high state pressure and cause the query to fail.

```
SELECT *
FROM Orders LEFT JOIN Product ON Orders.productId = Product.id
```

```
SELECT *
FROM Orders RIGHT JOIN Product ON Orders.productId = Product.id

SELECT *
FROM Orders FULL OUTER JOIN Product ON Orders.productId = Product.id
```

### JOIN with temporal tables

Stream Compute Service also supports joining streams and temporal tables (tables that change constantly over time). The syntax is the same, except that the temporal table must be used as the right table.

```
SELECT
  o.amout, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency
```

**Note**

Make sure you include the `FOR SYSTEM_TIME AS OF` clause, without which the `JOIN` query will still be executed, but the database will be read in its entirety only once, and the result may not meet expectations.

### JOIN with user-defined table functions

You can use a user-defined table function (UDTF) as the right table in a `JOIN` query. The syntax is similar to other `JOIN` queries. You just need to put the UDTF in `LATERAL TABLE( )`.

#### Inner UDTF JOIN

```
SELECT users, tag
FROM Orders, LATERAL TABLE(unnest_udtf(tags)) t AS tag
```

**Left Outer UDTF JOIN**

```
SELECT users, tag
FROM Orders LEFT JOIN LATERAL TABLE(unnest_udtf(tags)) t AS tag ON TRUE
```

**Note**

Currently, `Left Outer JOIN` with UDTFs only supports `ON TRUE` , which is similar to `CROSS JOIN` .

### JOIN with arrays

Stream Compute Service supports join operations with a defined array object (you can use value construction functions to construct an array object).

**Example:** Assume that `tags` is a defined array.

```
SELECT users, tag
FROM Orders CROSS JOIN UNNEST(tags) AS t (tag)
```

## UNION ALL

`UNION ALL` combines the results of two queries.

```
SELECT *
FROM (
    (SELECT user FROM Orders WHERE a % 2 = 0)
  UNION ALL
    (SELECT user FROM Orders WHERE b = 0)
)
```

Currently, Stream Compute Service only supports `UNION ALL` and not `UNION`. That is, it does not remove duplicate rows.

To remove duplicates, you can use `DISTINCT` together with `UNION ALL` . `DISTINCT` converts append (tuple) streams into upsert streams and therefore can only use MySQL, PostgreSQL, and Elasticsearch sinks that have a primary key.

## OVER

You can use `OVER` to aggregate data streams based on hopping windows (without using `GROUP BY` ). In the `OVER` clause, you can specify the partition, order, and window frame.
The example below defines an aggregate query based on topping windows. It calculates the total transaction volume ( `amount` ) for a window size of 3. To specify the number of preceding rows, use `PRECEDING` . `FOLLOWING` is not supported currently.
You can specify only one timestamp field for `ORDER BY` . In the example below, the `proctime` field declared in the data source is used.

```
SELECT SUM(amount) OVER (
  PARTITION BY user
  ORDER BY proctime
  ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM Orders
```

```
SELECT COUNT(amount) OVER w, SUM(amount) OVER w
FROM Orders
WINDOW w AS (
  PARTITION BY user
  ORDER BY proctime
  ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
```

## ORDER BY

`ORDER BY` sorts query results. By default, it sorts the data in ascending order ( `ASC` ). You can also use `DESC` to sort the data in descending order.

**Note**

The first field specified for sorting must be the time column (Event Time or Processing Time, i.e., `PROCTIME` ), in ascending order. After that, you can specify your own fields to sort by.

```
SELECT *
FROM Orders
ORDER BY `orderTime`, `username` DESC, `userId` ASC
```

## DISTINCT

`DISTINCT` removes duplicates from query results. It should be added after `SELECT` .

```
SELECT DISTINCT users FROM Orders
```

`DISTINCT` generates upsert streams, so you need to use sinks that accept upsert streams. Please note that continuous use of such queries may result in high memory usage.
We recommend you set an appropriate timeout period to exclude inactive objects and reduce memory usage.

## IN

You can use the `IN` keyword to determine whether an element exists in a specified set, such as a subquery.

**Note**

This operation is demanding on memory.

```
SELECT user, amount
FROM Orders
WHERE product IN (
    SELECT product FROM NewProducts
)
```

## EXISTS

If the result of the subquery following `EXISTS` has one or more rows (data exists), `true` is returned.

**Note**

This operation is demanding on memory.

```
SELECT user, amount
FROM Orders
WHERE product EXISTS (
    SELECT product FROM NewProducts
)
```

## ORDER BY

`ORDER BY` sorts data by a specified field.

**Note**

This operation is demanding on memory.

```
SELECT *
FROM Orders
ORDER BY orderTime
```

## Grouping Sets, Rollup, Cube

`Grouping Sets` , `Rollup` , and `Cube` generate upsert streams. Therefore, you need to use data sinks that accept upsert streams.

```
SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product))
```

## MATCH_RECOGNIZE

`MATCH_RECOGNIZE` performs pattern recognition on an input stream, allowing you to use a SQL query to describe complex event processing (CEP) logic.

```
SELECT T.aid, T.bid, T.cid
FROM MyTable
MATCH_RECOGNIZE (
  PARTITION BY userid
  ORDER BY proctime
  MEASURES
    A.id AS aid,
    B.id AS bid,
    C.id AS cid
  PATTERN (A B C)
  DEFINE
```

```
      A AS name = 'a',
      B AS name = 'b',
      C AS name = 'c'
  ) AS T
```

The above example defines three events, A, B, and C, where the `name` field equates `a` , `b` , and `c` .
`PATTERN` specifies the trigger rule. In the example, the trigger rule is when A, B, and C occur consecutively.
`MEASURES` specifies the output format.
For details, see the Flink document Detecting Patterns in Tables.

## Top-N

`Top-N` gets the N smallest or largest values from a data stream. It generates upsert streams, so you need to use data sinks that accept upsert streams.
To learn more about `Top-N` syntax, see the Flink document Top-N.

## Deduplication

In cases where the input data includes consecutive duplicate values, you can use this clause to remove the duplicates.
To learn more about the syntax, see Flink document Deduplication.

# INSERT

Last updated：2023-11-08 15:31:46

## INSERT INTO

`INSERT INTO` is used together with `SELECT` to write the selected data to the specified sink.

### Syntax

```
INSERT INTO sink name
SELECT subclause
```

### Example

The example below inserts the result of the `SELECT` query to the sink named `KafkaSink1` .

```
INSERT INTO KafkaSink1
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

## About sinks

### Connector package

If a sink is specified using WITH parameters, make sure you select the corresponding built-in connector or upload the connector package.
Without a matching connector package, the error `org.apache.flink.table.api.ValidationException: Could not find any factory` will occur when you run the job.
If data is read from or written to Kafka, **instead of** selecting an old-version package such as `flink-connector-kafka-0.11`, we recommend you use `flink-connector-kafka` (without a version number) and set `connector.version` to `universal` to support the latest features.

## Calculated columns

`INSERT INTO` ignores **calculated columns** in the sink. Assume that a sink is defined as follows. The `SELECT` clause following `INSERT INTO MySink` must include `a (VARCHAR)` and `b (BIGINT)` and cannot include the calculated column `c`.

```
CREATE TABLE MySink (
   a VARCHAR,
   b BIGINT,
   c AS PROCTIME()
) WITH ( ... ... );
```

## Difference between tuple and upsert streams

Make sure you specify the correct WITH parameters for the sink. For example, some connectors can only be used as sources and not sinks, and some only support tuple streams and not upsert streams.

# Merging MySQL CDC Sources

Last updated：2023-11-08 15:58:10

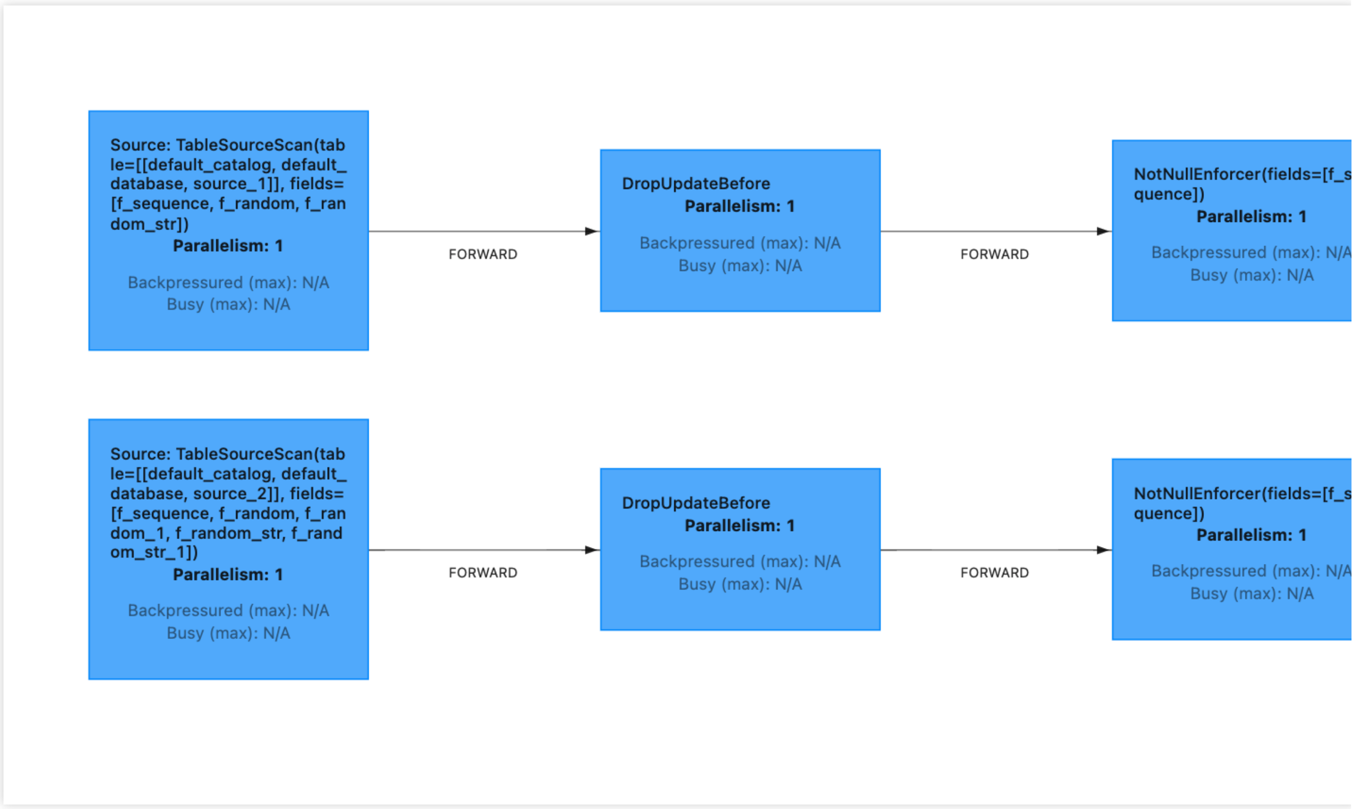
When syncing data, Flink CDC Connector will establish a database connection for each table. This significantly increases the load on the database instance when data is synced among multiple tables or across the database. For this, Stream Compute Service introduced the source merging capability.

## Overview

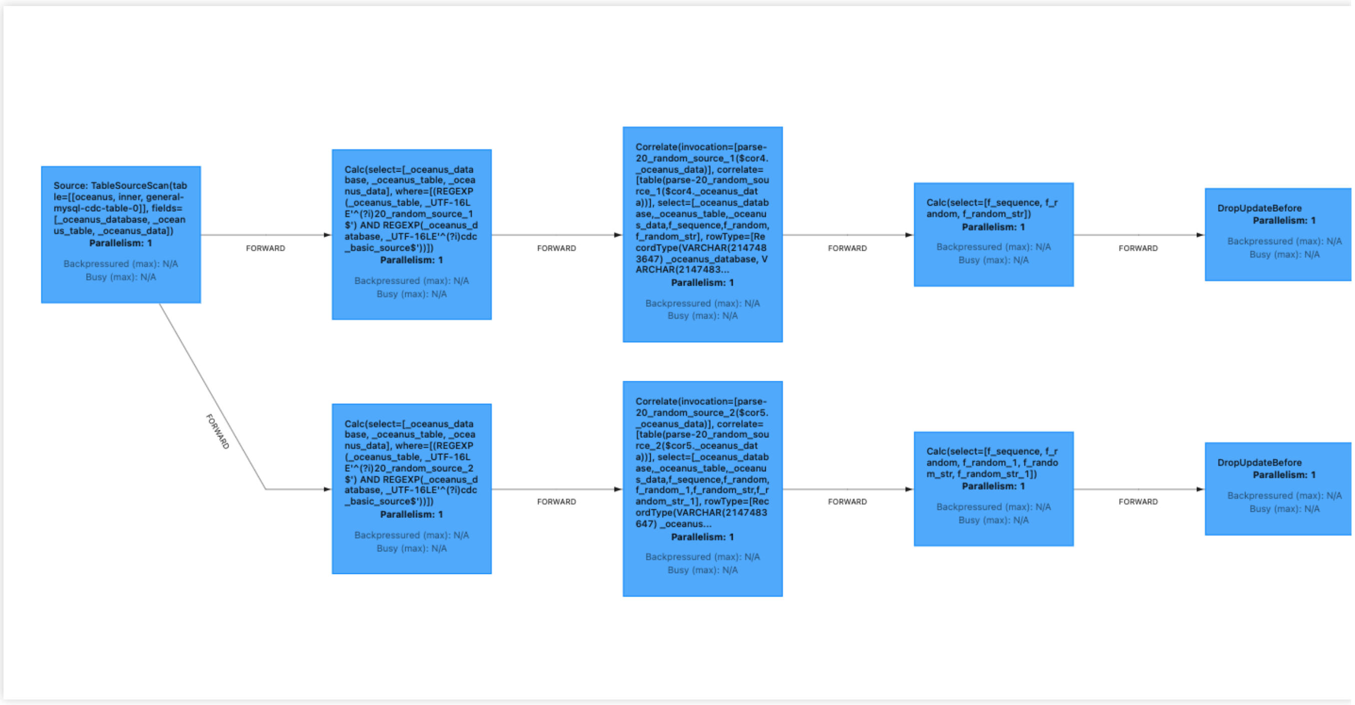
Take the following job for example:

```
CREATE TABLE `source_1`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_str` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'mysql-cdc' ,
      'hostname' = 'ip1',
      'port' = '3306',
      'username' = 'xxx',
      'password' = 'xxx',
      'database-name' = 'db1',
      'table-name' = 'source_1'
);

CREATE TABLE `source_2`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_1`     INT,
    `f_random_str` VARCHAR,
    `f_random_str_1` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'mysql-cdc' ,
      'hostname' = 'ip1',
      'port' = '3306',
      'username' = 'xxx',
      'password' = 'xxx',
      'database-name' = 'db2',
      'table-name' = 'source_2'
);


CREATE TABLE `sink_1`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_str` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'logger'
);

CREATE TABLE `sink_2`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_1`     INT,
    `f_random_str` VARCHAR,
    `f_random_str_1` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'logger'
);

insert into sink_1 select * from source_1;
insert into sink_2 select * from source_2;
```

To sync the two tables, which belong to the same database instance, Flink will generate two pipelines. Each CDC source will establish a connection with the MySQL database. When a job has many database connections, the load on the database will be high.



After source merging is enabled, multiple MySQL CDC sources of the same database instance will be merged into one source, reducing the database load. Also, when new data is synced, binlog data only needs to be read once for multiple sources.


Source: TableSourceScan(table=[[oceanus, inner, general-mysql-cdc-table-0]], fields=[_oceanus_database, _oceanus_table, _oceanus_data])
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Calc(select=[_oceanus_database, _oceanus_table, _oceanus_data], where=[(REGEXP(_oceanus_table, _UTF-16LE'^(?i)20_random_source_1$') AND REGEXP(_oceanus_database, _UTF-16LE'^(?i)cdc_basic_source$'))])
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Correlate(invocation=[parse-20_random_source_1($cor4._oceanus_data)], correlate=[table(parse-20_random_source_1($cor4._oceanus_data))], select=[_oceanus_database,_oceanus_table,_oceanus_data,f_sequence,f_random,f_random_str], rowType=[RecordType(VARCHAR(2147483647) _oceanus_database, VARCHAR(2147483...
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Calc(select=[f_sequence, f_random, f_random_str])
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
DropUpdateBefore
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Calc(select=[_oceanus_database, _oceanus_table, _oceanus_data], where=[(REGEXP(_oceanus_table, _UTF-16LE'^(?i)20_random_source_2$') AND REGEXP(_oceanus_database, _UTF-16LE'^(?i)cdc_basic_source$'))])
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Correlate(invocation=[parse-20_random_source_2($cor5._oceanus_data)], correlate=[table(parse-20_random_source_2($cor5._oceanus_data))], select=[_oceanus_database,_oceanus_table,_oceanus_data,f_sequence,f_random,f_random_1,f_random_str,f_random_str_1], rowType=[RecordType(VARCHAR(2147483647) _oceanus...
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Calc(select=[f_sequence, f_random, f_random_1, f_random_str, f_random_str_1])
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
DropUpdateBefore
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A

## How to enable CDC source merging

At the beginning of a SQL job, use the SET command to enable merging of MySQL CDC sources.

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;
```

SET parameters

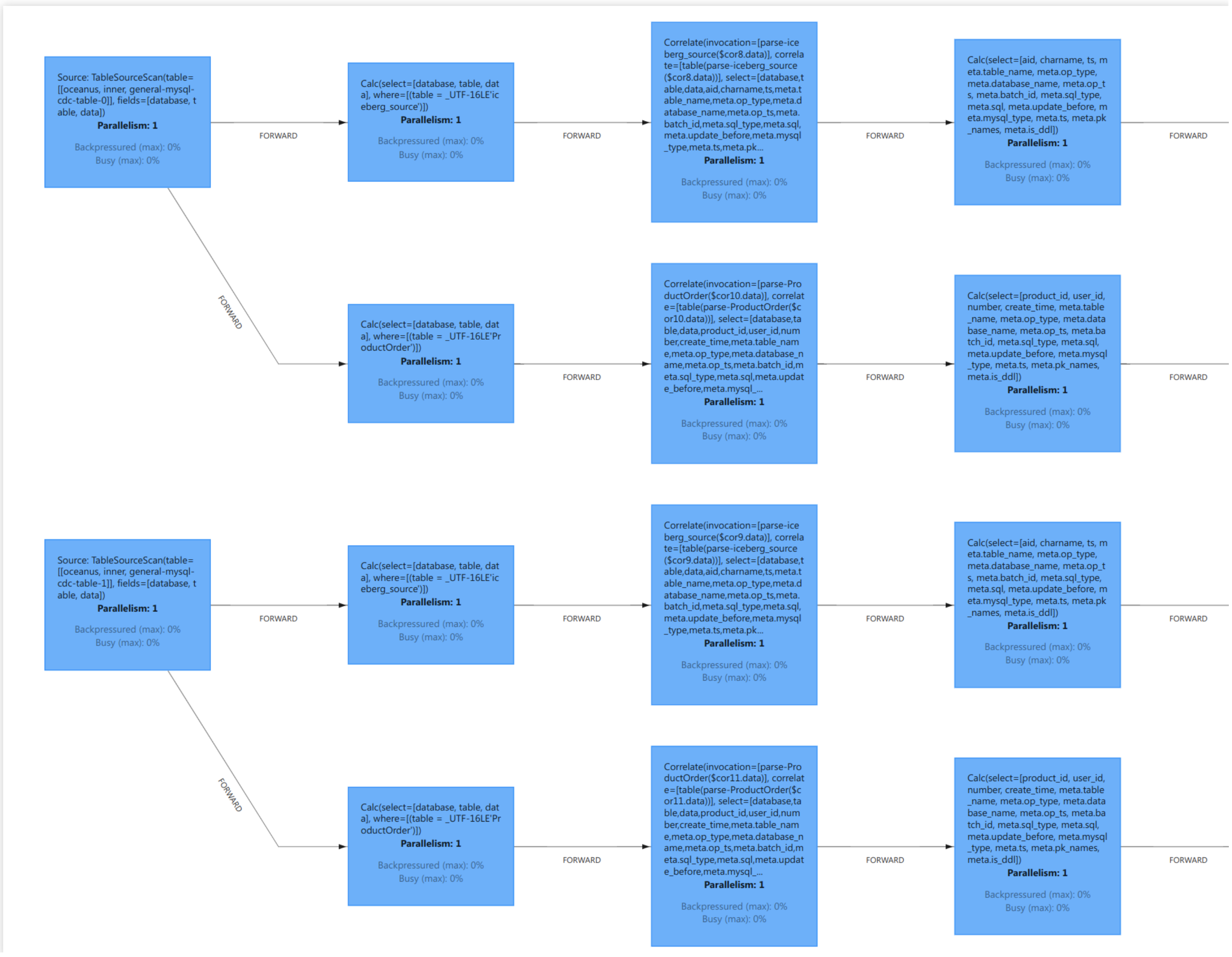
| Option | Default Value | Description |
|---|---|---|
| table.optimizer.mysql-cdc-source.merge.enabled | false | Whether to enable merging of MySQL sources. If it is enabled, Stream Compute Service will automatically merge MySQL CDC sources that belong to the same database in a job into one source. |

| | | |
|---|---|---|
| table.optimizer.mysql-cdc-source.merge.default-group.splits | 1 | The number of sources multiple MySQL CDC sources are merged into (when merging is enabled). If there is a large number of tables, merging them all into one source may not meet performance requirements. In such cases, you can increase the value of this parameter. Stream Compute Service will group the sources as evenly as possible according to the parameter specified. |

Assume that `source_1` , `source_2` , `source_3` , and `source_4` are MySQL CDC sources from the same database instance (source and sink definitions are omitted). You can use the following SET command to configure the source merging feature.

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;
SET table.optimizer.mysql-cdc-source.merge.default-group.splits=2;

insert into sink_1 select * from source_1;
insert into sink_2 select * from source_2;
insert into sink_3 select * from source_3;
insert into sink_4 select * from source_4;
```

This will automatically divide the four CDC sources into two groups.


Source: TableSourceScan(table=[[oceanus, inner, general-mysql-cdc-table-0]], fields=[database, table, data]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Calc(select=[database, table, data], where=[(table = _UTF-16LE'iceberg_source')]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Correlate(invocation=[parse-iceberg_source($cor8.data)], correlate=[table(parse-iceberg_source($cor8.data))], select=[database,table,data,aid,charname,ts,meta.table_name,meta.op_type,meta.database_name,meta.op_ts,meta.batch_id,meta.sql_type,meta.sql,meta.update_before,meta.mysql_type,meta.ts,meta.pk... Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Calc(select=[aid, charname, ts, meta.table_name, meta.op_type, meta.database_name, meta.op_ts, meta.batch_id, meta.sql_type, meta.sql, meta.update_before, meta.mysql_type, meta.ts, meta.pk_names, meta.is_ddl]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
FORWARD
Calc(select=[database, table, data], where=[(table = _UTF-16LE'ProductOrder')]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Correlate(invocation=[parse-ProductOrder($cor10.data)], correlate=[table(parse-ProductOrder($cor10.data))], select=[database,table,data,product_id,user_id,number,create_time,meta.table_name,meta.op_type,meta.database_name,meta.op_ts,meta.batch_id,meta.sql_type,meta.sql,meta.update_before,meta.mysql_... Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Calc(select=[product_id, user_id, number, create_time, meta.table_name, meta.op_type, meta.database_name, meta.op_ts, meta.batch_id, meta.sql_type, meta.sql, meta.update_before, meta.mysql_type, meta.ts, meta.pk_names, meta.is_ddl]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Source: TableSourceScan(table=[[oceanus, inner, general-mysql-cdc-table-1]], fields=[database, table, data]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Calc(select=[database, table, data], where=[(table = _UTF-16LE'iceberg_source')]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Correlate(invocation=[parse-iceberg_source($cor9.data)], correlate=[table(parse-iceberg_source($cor9.data))], select=[database,table,data,aid,charname,ts,meta.table_name,meta.op_type,meta.database_name,meta.op_ts,meta.batch_id,meta.sql_type,meta.sql,meta.update_before,meta.mysql_type,meta.ts,meta.pk... Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Calc(select=[aid, charname, ts, meta.table_name, meta.op_type, meta.database_name, meta.op_ts, meta.batch_id, meta.sql_type, meta.sql, meta.update_before, meta.mysql_type, meta.ts, meta.pk_names, meta.is_ddl]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
FORWARD
Calc(select=[database, table, data], where=[(table = _UTF-16LE'ProductOrder')]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Correlate(invocation=[parse-ProductOrder($cor11.data)], correlate=[table(parse-ProductOrder($cor11.data))], select=[database,table,data,product_id,user_id,number,create_time,meta.table_name,meta.op_type,meta.database_name,meta.op_ts,meta.batch_id,meta.sql_type,meta.sql,meta.update_before,meta.mysql_... Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD
Calc(select=[product_id, user_id, number, create_time, meta.table_name, meta.op_type, meta.database_name, meta.op_ts, meta.batch_id, meta.sql_type, meta.sql, meta.update_before, meta.mysql_type, meta.ts, meta.pk_names, meta.is_ddl]) Parallelism: 1 Backpressured (max): 0% Busy (max): 0%
FORWARD

# Connectors

## Kafka

Last updated：2023-11-08 14:58:59

### Overview

Kafka data pipelines are among the most used sources and sinks in stream computing. You can ingest stream data to a Kafka topic, process the data using Flink operators, and output the result to another Kafka topic on the same or a different instance.
Kafka supports reading data from and writing data to multiple partitions of the same topic. This can increase throughput and reduce data skew.

### Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Supported |
| 1.16 | Supported |

### Limits

Kafka can be used as a source or a sink for tuple streams.
Kafka can also be used together with Debezium and Canal to capture and subscribe to changes to databases such as MySQL and PostgreSQL and to process the changes.

### Defining a table in DDL

#### As a source

**JSON format**

```
CREATE TABLE `kafka_json_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- Replace this with the topic you want to consume data
  'scan.startup.mode' = 'latest-offset', -- Valid values include latest-offset, ear
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your
  'properties.group.id' = 'testGroup',  -- (Required) The group ID.
```

```
    -- Define the data format (JSON).
    'format' = 'json',
    'json.fail-on-missing-field' = 'false'    -- If this is 'false', no errors will oc
    'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors wil
  );
```

**CSV format**

```
CREATE TABLE `kafka_csv_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- Replace this with the topic you want to consume data
  'scan.startup.mode' = 'latest-offset', -- Valid values include latest-offset, ear
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your
  'properties.group.id' = 'testGroup',  -- (Required) The group ID.

  -- Define the data format (CSV).
  'format' = 'csv'
);
```

**Debezium format**

```
CREATE TABLE `kafka_debezium_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- Replace this with the topic you want to consume data
  'scan.startup.mode' = 'latest-offset', -- Valid values include latest-offset, ear
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your
  'properties.group.id' = 'testGroup',  -- (Required) The group ID.
  -- Define the data format (JSON data output by Debezium).
  'format' = 'debezium-json'
);
```

**Canal format**

```
CREATE TABLE `kafka_source`
(
    aid                 BIGINT COMMENT 'unique id',
    charname            string,
    `ts`                timestamp(6),
    origin_database STRING METADATA FROM 'value.database' VIRTUAL,
    origin_table STRING METADATA FROM 'value.table' VIRTUAL,
    origin_es TIMESTAMP(3) METADATA FROM 'value.event-timestamp' VIRTUAL,
    origin_type STRING METADATA FROM 'value.operation-type' VIRTUAL,
    `batch_id` bigint METADATA FROM 'value.batch-id' VIRTUAL,
    `is_ddl` boolean METADATA FROM 'value.is-ddl' VIRTUAL,
    origin_old ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before' VIRTU
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type' VIRTUAL,
    origin_pk_names ARRAY<STRING> METADATA FROM 'value.pk-names' VIRTUAL,
    `sql` STRING METADATA FROM 'value.sql' VIRTUAL,
    origin_sql_type MAP<STRING, INT> METADATA FROM 'value.sql-type' VIRTUAL,
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL
) WITH (
    'connector' = 'kafka',   -- Make sure you specify the corresponding connector.
    'topic' = '$TOPIC', -- Replace this with the topic you want to consume data fro
    'properties.bootstrap.servers' = '$IP:$PORT', --  Replace this with your Kafka
    'properties.group.id' = 'testGroup',  -- (Required) The group ID.
    'scan.startup.mode' = 'latest-offset',
    'scan.topic-partition-discovery.interval' = '5s',
    'format' = 'canal-json',
    'canal-json.ignore-parse-errors' = 'false',  -- Ignore JSON parse errors.
    'canal-json.source.append-mode' = 'true'  -- Only Flink 1.13 or later is suppor
);
```

## As a sink

### JSON format

```
CREATE TABLE `kafka_json_sink_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Output',  -- Replace this with the topic you want to write to.
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your

  -- Define the data format (JSON).
  'format' = 'json',
  'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will oc
  'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors wil
);
```

**CSV format**

```
CREATE TABLE `kafka_csv_sink_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
```

```
    'connector' = 'kafka',
    'topic' = 'Data-Output',  -- Replace this with the topic you want to write to.
    'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your

    -- Define the data format (CSV).
    'format' = 'csv'
);
```

**Canal format**

```
CREATE TABLE `kafka_canal_json_sink_table`
```

```
(
    aid                BIGINT COMMENT 'unique id',
    charname           string,
    `ts`              timestamp(6),
    origin_database STRING METADATA FROM 'value.database',
    origin_table STRING METADATA FROM 'value.table',
    origin_ts TIMESTAMP(3) METADATA FROM 'value.event-timestamp',
    `type` STRING METADATA FROM 'value.operation-type',
    `batch_id` bigint METADATA FROM 'value.batch-id',
    `isDdl` BOOLEAN METADATA FROM 'value.is-ddl',
    `old` ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before',
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type',
    `pk_names` ARRAY<STRING> METADATA FROM 'value.pk-names',
    `sql` STRING METADATA FROM 'value.sql',
    `sql_type` MAP<STRING, INT> METADATA FROM 'value.sql-type',
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp'
) WITH (
    'connector' = 'kafka',   -- Make sure you specify the corresponding connector.
    'topic' = '$TOPIC', -- Replace this with the topic you want to consume data fro
    'properties.bootstrap.servers' = '$IP:$PORT', --  Replace this with your Kafka
    'properties.group.id' = 'testGroup',  -- (Required) The group ID.
    'format' = 'canal-json'
);
```

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | - | Here, it should be `'kafka'` . |
| topic | Yes | - | The name of the Kafka topic to read from or write to. |
| properties.bootstrap.servers | Yes | - | The Kafka bootstrap server addresses, separated with |
| properties.group.id | Yes for source connectors | - | The Kafka consumer group ID. |
| format | Yes | - | The input and output format of Kafka messages. Curren values include `csv` , `json` , `avro` , `debezium-` and `canal-json` . For Flink 1.13, `maxwell-json` supported. |
| | | | |

| | | | |
|---|---|---|---|
| scan.startup.mode | No | group-offsets | The Kafka consumer startup mode. Valid values include `latest-offset` , `earliest-offset` , `speci offsets` , `group-offsets` , and `timestamp` . `'scan.startup.specific-offsets' = 'partition:0,offset:42;partition:1,offse` If `'specific-offsets'` s used, you need to speci offsets of each partition. `'scan.startup.timestamp-millis' = '1631588815000'` . If `'timestamp'` is used, you specify the start timestamp (milliseconds). |
| scan.startup.specific-offsets | No | - | If `scan.startup.mode` is `'specific-offsets` need to use this option to specify the start offset, for exa `'partition:0,offset:42;partition:1,offs` |
| scan.startup.timestamp-millis | No | - | If `scan.startup.mode` is `'timestamp'` , you r this option to specify the start timestamp (Unix format in milliseconds). |
| sink.partitioner | No | - | The Kafka output partitioning. Currently, the following ty partitioning are supported: `fixed` : One Flink partition corresponds to not more t Kafka partition. `round-robin` : One Flink partition is distributed in tu different Kafka partitions. Custom partitioning: You can also implement partitionin inheriting the `FlinkKafkaPartitioner` class. |

## WITH parameters for JSON

| Option | Required | Default Value | Description |
|---|---|---|---|
| json.fail-on-missing-field | No | false | If this is `true` , the job will fail in case of missing parameters. If this is `false` (default), the missing parameters will be set to null and the job will continue to be executed. |
| json.ignore-parse-errors | No | false | If this is `true` , when there is a parse error, the field will be set to null and the job will continue to be executed. If this is `false` , the job will fail in case of a parse error. |
| json.timestamp-format.standard | No | SQL | The JSON timestamp format. The default value is `SQL` , in which case the format will be `yyyy-MM-dd HH:mm:ss.s{precision}` . You can also set it to |

| | | | |
|---|---|---|---|
| | | | `ISO-8601` , and the format will be `yyyy-MM-ddTHH:mm:ss.s{precision}` . |

## WITH parameters for CSV

| Option | Required | Default Value | Description |
|---|---|---|---|
| csv.field-delimiter | No | , | The field delimiter, which is comma by default. |
| csv.line-delimiter | No | U&'\\000A' | The line delimiter, which is `\\n` by default (in SQL, you must use `U&'\\000A'` ). You can also set it to `\\r` (in SQL, you need to use `U&'\\000D'` ). |
| csv.disable-quote-character | No | false | Whether to disable quote characters. If this is `true` , 'csv.quote-character' cannot be used. |
| csv.quote-character | No | " | The quote characters. Text inside quotes will be viewed as a whole. The default value is `"` . |
| csv.ignore-parse-errors | No | false | Whether to ignore parse errors. If this is `true` , fields will be set to null in case of parse failure. |
| csv.allow-comments | No | false | Whether to ignore comment lines that start with # and output them as empty lines (if this is `true` , make sure you set `csv.ignore-parse-errors` to `true` as well). |
| csv.array-element-delimiter | No | ; | The array element delimiter, which is `;` by default. |
| csv.escape-character | No | - | The escape character. By default, escape characters are disabled. |
| csv.null-literal | No | - | The string that will be seen as null. |

## WITH parameters for Debezium-json

| Option | Required | Default Value | Description |
|---|---|---|---|
| debezium-json.schema-include | No | false | Whether to include schemas. If you specified `'value.converter.schemas.enable'` when configuring Kafka Connect with Debezium, the JSON data sent by Debezium will include schema information, and you need to set this option to `true` . |

| | | | |
|---|---|---|---|
| debezium-json.ignore-parse-errors | No | false | Whether to ignore parse errors. If this is `true` , fields will be set to null in case of parse failure. |
| debezium-json.timestamp-format.standard | No | SQL | The JSON timestamp format. The default value is `SQL` , in which case the format will be `yyyy-MM-dd HH:mm:ss.s{precision}` . You can also set it to `ISO-8601` , and the format will be `yyyy-MM-ddTHH:mm:ss.s{precision}` . |

### WITH parameters for Canal

| Option | Required | Default Value | Description |
|---|---|---|---|
| canal-json.source.append-mode | No | false | Whether to support append streams. You can set this to `true` when, for example, writing Canal JSON data from Kafka to Hive. This option is only supported for Flink 1.13 clusters. |
| debezium-json.ignore-parse-errors | No | false | Whether to ignore parse errors. If this is `true` , fields will be set to null in case of parse failure. |
| canal-json.* | No | - | See Format Options. |

### Metadata supported by Canal (only Flink 1.13 clusters)

The metadata below can only be used as virtual columns. If a metadata column has the same name as a physical column, you can use the name `meta.` for the metadata column.

| Column | Data Type | Description |
|---|---|---|
| Database | STRING NOT NULL | The name of the database to which the row belongs. |
| table | STRING NOT NULL | The name of the table to which the row belongs. |
| event-timestamp | TIMESTAMP_LTZ(3) NOT NULL | The time when the row was changed in the database. |
| batch-id | BIGINT | The batch ID of the binlog. |
| is-ddl | BOOLEAN | Whether it is a DDL statement. |
| mysql-type | MAP | The database structure. |

| | | |
|---|---|---|
| update-before | ARRAY | The field value before it was modified. |
| pk-names | ARRAY | The primary key field. |
| sql | STRING | Null. |
| sql-type | MAP | The mappings between the sql_type fields and Java data types. |
| ingestion-timestamp | TIMESTAMP_LTZ(3) NOT NULL | The time when the row was received and processed. |
| operation-type | STRING | The operation type, such as `INSERT` or `DELETE` . |

## Example

### JSON

```
CREATE TABLE `kafka_json_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- Replace this with the topic you want to consume data
  'scan.startup.mode' = 'latest-offset', -- Valid values include latest-offset, ear
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your
  'properties.group.id' = 'testGroup',  -- (Required) The group ID.
  -- Define the data format (JSON).
  'format' = 'json',
  'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will oc
  'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors wil
);
CREATE TABLE `kafka_json_sink_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Output',  -- Replace this with the topic you want to write to.
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your

  -- Define the data format (JSON).
  'format' = 'json',
  'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will oc
  'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors wil
);
insert into kafka_json_sink_table select * from kafka_json_source_table;
```

## Canal

```
CREATE TABLE `source`
(
    `aid`            bigint,
    `charname`       string,
    `ts`             timestamp(6),
    `database_name`  string METADATA FROM 'value.database_name',
    `table_name`     string METADATA FROM 'value.table_name',
    `op_ts`          timestamp(3) METADATA FROM 'value.op_ts',
    `op_type` string METADATA FROM 'value.op_type',
    `batch_id` bigint METADATA FROM 'value.batch_id',
    `is_ddl` boolean METADATA FROM 'value.is_ddl',
    `update_before` ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update_before',
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql_type',
    `pk_names` ARRAY<STRING> METADATA FROM 'value.pk_names',
    `sql` STRING METADATA FROM 'value.sql',
    `sql_type` MAP<STRING, INT> METADATA FROM 'value.sql_type',
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ts',
    primary key (`aid`) not enforced
) WITH (
    'connector' = 'mysql-cdc' ,
     'append-mode' = 'true',
    'hostname' = '$IP',
    'port' = '$PORT',
    'username' = '$USERNAME',
    'password' = '$PASSWORD',
    'database-name' = 't_wr',
    'table-name' = 't1',
    'server-time-zone' = 'Asia/Shanghai',
    'server-id' = '5500-5510'
);

CREATE TABLE `kafka_canal_json_sink`
(
    aid                BIGINT COMMENT 'unique id',
    charname              string,
    `ts`             timestamp(6),
    origin_database STRING METADATA FROM 'value.database',
    origin_table STRING METADATA FROM 'value.table',
    origin_ts TIMESTAMP(3) METADATA FROM 'value.event-timestamp',
    `type` STRING METADATA FROM 'value.operation-type',
    `batch_id` bigint METADATA FROM 'value.batch-id',
    `isDdl` BOOLEAN METADATA FROM 'value.is-ddl',
    `old` ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before',
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type',
    `pk_names` ARRAY<STRING> METADATA FROM 'value.pk-names',
    `sql` STRING METADATA FROM 'value.sql',
    `sql_type` MAP<STRING, INT> METADATA FROM 'value.sql-type',
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp'
)
    WITH (
        'connector' = 'kafka',   -- Valid values include 'kafka' and 'kafka-0.11'.
        'topic' = 'TOPIC',  -- Replace this with the topic you want to consume data
        'properties.bootstrap.servers' = '$IP:$PORT', --  Replace this with your Ka
        'properties.group.id' = 'testGroup',  -- (Required) The group ID.
        'format' = 'canal-json'
);

insert into kafka_canal_json_sink select * from source;
```

```
 CREATE TABLE `source`
(
    `aid`           bigint,
    `charname`      string,
    `ts`            timestamp(3),
    origin_database STRING METADATA FROM 'value.database' VIRTUAL,
    origin_table STRING METADATA FROM 'value.table' VIRTUAL,
    origin_es TIMESTAMP(3) METADATA FROM 'value.event-timestamp' VIRTUAL,
    origin_type STRING METADATA FROM 'value.operation-type' VIRTUAL,
    `batch_id` bigint METADATA FROM 'value.batch-id' VIRTUAL,
    `is_ddl` boolean METADATA FROM 'value.is-ddl' VIRTUAL,
    origin_old ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before' VIRTU
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type' VIRTUAL,
    origin_pk_names ARRAY<STRING> METADATA FROM 'value.pk-names' VIRTUAL,
    `sql` STRING METADATA FROM 'value.sql' VIRTUAL,
    origin_sql_type MAP<STRING, INT> METADATA FROM 'value.sql-type' VIRTUAL,
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL,
    WATERMARK FOR `origin_es` AS `origin_es` - INTERVAL '5' SECOND
) WITH (
      'connector' = 'kafka',   -- Make sure you specify the corresponding connector
      'topic' = '$TOPIC', -- Replace this with the topic you want to consume data f
      'properties.bootstrap.servers' = '$IP:PORT', --  Replace this with your Kafka
      'properties.group.id' = 'testGroup',  -- (Required) The group ID.
      'scan.startup.mode' = 'latest-offset',
      'scan.topic-partition-discovery.interval' = '10s',

      'format' = 'canal-json',
      'canal-json.source.append-mode' = 'true', -- This is only supported for Flink
      'canal-json.ignore-parse-errors' = 'false'
);


CREATE TABLE `kafka_canal_json` (
   `aid`            bigint,
   `charname`       string,
   `ts`             timestamp(9),
   origin_database STRING,
   origin_table STRING,
   origin_es TIMESTAMP(9),
   origin_type STRING,
   `batch_id` bigint,
   `is_ddl` boolean,
   origin_old ARRAY<MAP<STRING, STRING>>,
   `mysql_type` MAP<STRING, STRING>,
   origin_pk_names ARRAY<STRING>,
   `sql` STRING,
   origin_sql_type MAP<STRING, INT>,
   `ingestion_ts` TIMESTAMP(9),
   dt STRING,
   hr STRING
) PARTITIONED BY (dt, hr)
with (
     'connector' = 'hive',
     'hive-version' = '3.1.1',
     'hive-database' = 'testdb',
     'partition.time-extractor.timestamp-pattern'='$dt $hr:00:00',
     'sink.partition-commit.trigger'='partition-time',
     'sink.partition-commit.delay'='30 min',
     'sink.partition-commit.policy.kind'='metastore,success-file'
);

insert into kafka_canal_json select *,DATE_FORMAT(`origin_es`,'yyyy-MM-dd'),DATE_FO
from `source`;
```

## SASL authentication

### SASL/PLAIN username and password authentication

1. Follow the instructions in Message Queue CKafka - Configuring ACL Policy to configure username/password-based authentication (SASL_PLAINTEXT) for the topic.
2. Select "SASL_PLAINTEXT" as the access mode when adding a routing policy and access the topic via the address of that mode.
3. Configure WITH parameters for the job.

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLogi
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'PLAIN',
  ...
);
```

**Note**

`username` should be `instance ID` + `#` + the username you set, and `password` is the password you configured.

### SASL/GSSAPI Kerberos authentication

Currently, Tencent Cloud CKafka does not support Kerberos authentication. If your self-built Kafka supports Kerberos authentication, you can follow the steps below to configure it.

1. Get the Kerberos configuration file for your self-built Kafka cluster. If your cluster is built on top of Tencent Cloud EMR, get the files `krb5.conf` and `emr.keytab` in the following paths.

```
/etc/krb5.conf
```

```
/var/krb5kdc/emr.keytab
```

2. Package the files into a JAR file.

```
jar cvf kafka-xxx.jar krb5.conf emr.keytab
```

3. Check the JAR structure (run the Vim command `vim kafka-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
```

4. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
5. Get Kerberos principals to configure advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as follows (use the first): hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- ------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. Configure WITH parameters for the job.

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  'properties.sasl.kerberos.service.name' = 'hadoop',
  ...
);
```

7. Configure advanced job parameters.

```
security.kerberos.login.principal: hadoop/172.28.2.13@EMR-4K3VR5FD
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
security.kerberos.login.contexts: KafkaClient
fs.hdfs.hadoop.security.authentication: kerberos
```

**Note**

Kerberos authentication is not supported for historical Stream Compute Service clusters. To support the feature, please contact us to upgrade the cluster management service.

# Upsert Kafka

Last updated：2023-11-08 11:20:39

## Overview

The Upsert Kafka connector allows for reading data from and writing data into Kafka topics in the upsert fashion.
As a source, the Upsert Kafka connector produces a changelog stream, where each data record represents an update or delete event. More precisely, the value in a data record is interpreted as an UPDATE of the last value for the same key, if any (if a corresponding key doesn't exist yet, the update will be considered an INSERT). Using the table analogy, a data record in a changelog stream is interpreted as an UPSERT (INSERT/UPDATE) because any existing row with the same key is overwritten. Also, a record with a null value represents a DELETE.
As a sink, the Upsert Kafka connector can consume a changelog stream. It will write INSERT/UPDATE_AFTER data as normal Kafka messages, and write DELETE data as Kafka messages with null values (which indicates the messages will be deleted). Flink will guarantee the message ordering on the primary key by partitioning data based on the values of the primary key columns, so the update/delete messages on the same key will fall into the same partition.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported |
| 1.14 | Supported |
| 1.16 | Supported |

## Defining a table in DDL

```
CREATE TABLE kafka_upsert_sink_table (
  id INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  -- Define Upsert Kafka parameters.
  'connector' = 'upsert-kafka',  -- Specify the connector.
  'topic' = 'topic',  -- Replace this with the topic you want to consume data from.
  'properties.bootstrap.servers' = '...',  -- Replace this with your Kafka connecti
  'key.format' = 'json',  -- Define the data format of keys.
  'value.format' = 'json'  -- Define the data format of values.
);
```

**Note**

Make sure you define the primary key in the DDL.

## WITH parameters

| Option | Required | Default Value | Data Type | Description |
|---|---|---|---|---|
| connector | Yes | - | String | The connector to use. For Upsert Kafka, use `'upsert-kafka'` . |
| topic | Yes | - | String | The name of the topic to read from and write |
| properties.bootstrap.servers | Yes | - | String | A list of Kafka brokers, separated with comm |
| properties.* | No | - | String | Arbitrary Kafka configurations. The suffixes n match the configuration key defined in the Ka configuration document.<br>Flink will remove the "properties." key prefix a pass the transformed keys and values to the underlying KafkaClient. For example, you can `'properties.allow.auto.create.t = 'false'`<br>to disable automatic topic creation. However configurations, such as `'key.deseriali` and `'value.deserializer'` , cannot b using this option because Flink will override t |
| key.format | Yes | - | String | The format used to deserialize and serialize t part of Kafka messages. The `key` field is c by PRIMARY KEY. Valid values include `'c` `'json'` , and `'avro'` . |
| key.fields-prefix | No | - | String | A custom prefix for all fields of the key format avoid name conflicts with fields of the value f By default, the prefix is empty. If a custom pr defined, both the table schema and `key.f` will work with prefixed names. When the data the key format is constructed, the prefix will b removed and the non-prefixed names will be within the key format. Please note that this op |

| | | | | |
|---|---|---|---|---|
| | | | | requires that `value.fields-include` to `EXCEPT_KEY` . |
| value.format | Yes | - | String | The format used to deserialize and serialize t value part of Kafka messages. Valid values i `'csv'` , `'json'` , and `'avro'` . |
| value.fields-include | No | 'ALL' | String | A strategy specifying how to deal with key cc in the data type of the value format. Valid val `ALL` : All physical columns of the table sch be included in the value format, including col defined as primary keys. `EXCEPT_KEY` : All physical columns of the schema will be included in the value format e columns defined as primary keys. |
| sink.parallelism | No | - | Integer | The parallelism of the Upsert Kafka sink ope default, the parallelism is determined by the framework using the same parallelism of the upstream chained operator. |
| sink.buffer-flush.max-rows | No | 0 | Integer | The maximum number of buffered records be flush. When the sink receives many updates same key, the buffer will retain the last record same key. This helps reduce data shuffling a possible tombstone messages to Kafka topic disable buffer flushing, set this parameter to (default value). Note both `sink.buffer-flush.max-rows` and `sink.buffer-flush.interval` must be set to greater t zero to enable sink buffer flushing. |
| sink.buffer-flush.interval | No | 0 | Duration | The interval at which asynchronous threads t data. When the sink receives many updates o same key, the buffer will retain the last record same key. This helps reduce data shuffling a possible tombstone messages to Kafka topic To disable buffer flushing, set this parameter (default value). Note both `sink.buffer-flush.max-rows` and `sink.buffer-flush.interval` must be set to greater t zero to enable sink buffer flushing. |

## Example

```
CREATE TABLE `kafka_json_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- Define Kafka parameters.
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- Replace this with the topic you want to consume data
  'scan.startup.mode' = 'latest-offset', -- Valid values include latest-offset, ear
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- Replace this with your
  'properties.group.id' = 'testGroup',  -- (Required) The group ID.
  -- Define the data format (JSON).
  'format' = 'json',
  'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will oc
  'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors wil
);

CREATE TABLE kafka_upsert_sink_table (
  id INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  -- Define Upsert Kafka parameters.
  'connector' = 'upsert-kafka',  -- Specify the connector.
  'topic' = 'topic',  -- Replace this with the topic you want to consume data from.
  'properties.bootstrap.servers' = '...',  -- Replace this with your Kafka connecti
  'key.format' = 'json',  -- Define the data format of keys.
  'value.format' = 'json'  -- Define the data format of values.
);

-- Calculate 'pv' and 'uv' and insert them to 'upsert-kafka sink'.
INSERT INTO kafka_upsert_sink_table
SELECT * FROM kafka_json_source_table;
```

## SASL authentication

### SASL/PLAIN username and password authentication

1. Follow the instructions in Message Queue CKafka - Configuring ACL Policy to configure username/password-based authentication (SASL_PLAINTEXT) for the topic.
2. Select "SASL_PLAINTEXT" as the access mode when adding a routing policy and access the topic via the address of that mode.
3. Configure WITH parameters for the job.

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLogi
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'PLAIN',
  ...
);
```

**Note**

`username` should be `instance ID` + `#` + the username you set, and `password` is the password you configured.

### SASL/GSSAPI Kerberos authentication

Currently, Tencent Cloud CKafka does not support Kerberos authentication. If your self-built Kafka supports Kerberos authentication, you can follow the steps below to configure it.

1. Get the Kerberos configuration file for your self-built Kafka cluster. If your cluster is built on top of Tencent Cloud EMR, get the files `krb5.conf` and `emr.keytab` in the following paths.

```
/etc/krb5.conf
```

```
/var/krb5kdc/emr.keytab
```

2. Package the files into a JAR file.

```
jar cvf kafka-xxx.jar krb5.conf emr.keytab
```

3. Check the JAR structure (run the Vim command `vim kafka-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
```

4. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
5. Get Kerberos principals to configure advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as follows (use the first): hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- ------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. Configure WITH parameters for the job.

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  'properties.sasl.kerberos.service.name' = 'hadoop',
  ...
);
```

**Note**

The value of `properties.sasl.kerberos.service.name` must match the principal you use. For example, if you use `hadoop/${IP}@EMR-OQPO48B9` , the parameter value should be `hadoop` .

7. Configure advanced job parameters.

```
security.kerberos.login.principal: hadoop/172.28.2.13@EMR-4K3VR5FD
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
security.kerberos.login.contexts: KafkaClient
fs.hdfs.hadoop.security.authentication: kerberos
```

# Tencent Cloud Message Queue

Last updated：2023-11-08 14:58:27

## Overview

Cloud Message Queue (CMQ) is a distributed message queue system developed based on Tencent's in-house messaging engine. It can be used as a source or a sink. You can ingest stream data to a CMQ queue, process the data using Flink operators, and output the result to another CMQ queue on the same or a different instance.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Unsupported |

## Limits

CMQ can be used as a source or a sink for tuple streams. It does not support upsert streams currently.

## Defining a table in DDL

### As a source

#### JSON format

```
CREATE TABLE `cmq_source_json_table` (
    `id` INT,
    `name`STRING,
    PRIMARY KEY (`id`) NOT ENFORCED     -- If you want to remove duplicates, specif
) WITH (
    'connector' = 'cmq',                              -- Here, it should be 'cmq'.
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',         -- The name
    'queue' = 'queue_name',                        -- The name of the CMQ queue.
    'secret-id' = 'xxxx',                            -- The account secret ID.
    'secret-key' = 'xxxx',               -- The account secret key.
    'sign-method' = 'HmacSHA1',          -- The signature algorithm.
    'format' = 'json',                    -- The data format (JSON).
    'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will
    'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors w
    'batch-size' = '16',                  -- The number of messages consumed at a time
    'request-timeout' = '5000ms',          -- The request timeout period.
    'polling-wait-timeout'= '10s',         -- The time to wait in case of failure to o
    'key-alive-timeout'= '5min'            -- The valid time period for the deduplicat
);
```

**CSV format**

```
CREATE TABLE `cmq_source_csv_table` (
    `id` int,
    `name` STRING,
    PRIMARY KEY (`id`) NOT ENFORCED     -- If you want to remove duplicates, specif
) WITH (
    'connector' = 'cmq',                         -- Here, it should be 'cmq'.
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',        -- The name
    'queue' = 'queue_name',             -- The name of the CMQ queue.
    'secret-id' = 'xxxx',                  -- The account secret ID.
    'secret-key' = 'xxxx',             -- The account secret key.
    'sign-method' = 'HmacSHA1',        -- The signature algorithm.
    'format' = 'csv',                -- The data format (CSV).
    'batch-size' = '16',               -- The number of messages consumed/sent at a
    'request-timeout' = '5000ms',   -- The request timeout period.
    'polling-wait-timeout'= '10s', -- The time to wait in case of failure to obtain
    'key-alive-timeout'= '5min'    -- The valid time period for the deduplication o
);
```

## As a sink

### JSON format

```
CREATE TABLE `cmq_sink_json_table` (
    `id` int,
    `name` STRING
) WITH (
    'connector' = 'cmq',                             -- Here, it should be 'cmq'.
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',        -- The name
    'queue' = 'queue_name',                -- The name of the CMQ queue.
    'secret-id' = 'xxxx',                     -- The account secret ID.
    'secret-key' = 'xxxx',               -- The account secret key.
    'sign-method' = 'HmacSHA1',          -- The signature algorithm.
    'format' = 'json',                   -- The data format (JSON).
    'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will 
    'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors w
    'batch-size' = '16',                  -- The number of messages sent at a time.
    'request-timeout' = '5000ms',  -- The request timeout period.
    'retry-times' = '3',                           -- The number of retries in case of
    'max-block-timeout' = '0s'                     -- The maximum time to wait to batc
);
```

**CSV format**

```
CREATE TABLE `cmq_sink_csv_table` (
    `id` int,
    `name` STRING
) WITH (
    'connector' = 'cmq',                              -- Here, it should be 'cmq'.
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',         -- The name
    'queue' = 'queue_name',                  -- The name of the CMQ queue.
    'secret-id' = 'xxxx',                       -- The account secret ID.
    'secret-key' = 'xxxx',                  -- The account secret key.
    'sign-method' = 'HmacSHA1',             -- The signature algorithm.
    'format' = 'csv',                    -- The data format (CSV).
    'batch-size' = '16',                    -- The number of messages sent at a time.
    'request-timeout' = '5000ms',   -- The request timeout period.
    'retry-times' = '3',                             -- The number of retries in case of
    'max-block-timeout' = '0s'                       -- The maximum time to wait to batc
);
```

## WITH parameters

### Common WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | - | Here, it should be `'cmq'`. |
| hosts | Yes | - | The name server of the queue's region. For details, see TCP SDK. |
| queue | Yes | - | The CMQ queue name. |
| secret-id | Yes | - | The account secret ID. |
| secret-key | Yes | - | The account secret key. |
| sign-method | No | HmacSHA1 | The signature algorithm. |
| format | Yes | - | The input and output format of CMQ messages. Valid values include `'csv'` and `'json'`. |
| batch-size | No | 16 | The number of messages sent/received at a time. |
| request-timeout | No | 5000ms | The request timeout period. |
| polling-wait-timeout | No | 10s | The time to wait in case of failure to obtain any data. |

| | | | |
|---|---|---|---|
| key-alive-timeout | No | 60s | The valid time period for the deduplication of CMQ messages with primary keys. This option ensures that the same message is consumed only once, but does not guarantee global uniqueness. |
| retry-times | No | 3 | The number of retries in case of failure to send a message. |
| max-block-timeout | No | 0s | The maximum time to wait to batch send data. If it is `'0s'` , data will be sent immediately without waiting. |

### WITH parameters for JSON

| Option | Required | Default Value | Description |
|---|---|---|---|
| json.fail-on-missing-field | No | false | If this is `true` , the job will fail in case of missing parameters. If this is `false` (default), the missing parameters will be set to null and the job will continue to be executed. |
| json.ignore-parse-errors | No | false | If this is `true` , when there is a parse error, the field will be set to null and the job will continue to be executed. If this is `false` , the job will fail in case of a parse error. |
| json.timestamp-format.standard | No | SQL | The JSON timestamp format. The default value is `SQL` , in which case the format will be `yyyy-MM-dd HH:mm:ss.s{precision}` . You can also set it to `ISO-8601` , and the format will be `yyyy-MM-ddTHH:mm:ss.s{precision}` . |

### WITH parameters for CSV

| Option | Required | Default Value | Description |
|---|---|---|---|
| csv.field-delimiter | No | , | The field delimiter, which is comma by default. |
| csv.line-delimiter | No | U&'\\000A' | The line delimiter, which is `\\n` by default (in SQL, you must use `U&'\\000A'` ). You can also set it to `\\r` (in SQL, you need to use `U&'\\000D'` ). |
| csv.disable-quote-character | No | false | Whether to disable quote characters. If this is `true` , 'csv.quote-character' cannot be used. |
| csv.quote-character | No | " | The quote characters. Text inside quotes will be viewed |

| | | | |
|---|---|---|---|
| | | | as a whole. The default value is `''` . |
| csv.ignore-parse-errors | No | false | Whether to ignore parse errors. If this is `true` , fields will be set to null in case of parse failure. |
| csv.allow-comments | No | false | Whether to ignore comment lines that start with # and output them as empty lines (if this is `true` , make sure you set `csv.ignore-parse-errors` to `true` as well). |
| csv.array-element-delimiter | No | ; | The array element delimiter, which is `;` by default. |
| csv.escape-character | No | - | The escape character. By default, escape characters are disabled. |
| csv.null-literal | No | - | The string that will be seen as null. |

## Example

```
CREATE TABLE `cmq_source_json_table` (
    `id` int,
    `name` STRING,
    PRIMARY KEY (`id`) NOT ENFORCED     -- If you want to remove duplicates, specif
) WITH (
    'connector' = 'cmq',                           -- Here, it should be 'cmq'.
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',         -- The name
    'queue' = 'queue_name',                     -- The name of the CMQ queue.
    'secret-id' = 'xxxx',                         -- The account secret ID.
    'secret-key' = 'xxxx',             -- The account secret key.
    'sign-method' = 'HmacSHA1',        -- The signature algorithm.
    'format' = 'json',                  -- The data format (JSON).
    'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will
    'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors w
    'batch-size' = '16',                -- The number of messages consumed at a time
    'request-timeout' = '5000ms',        -- The request timeout period.
    'polling-wait-timeout'= '10s',       -- The time to wait in case of failure to o
    'key-alive-timeout'= '5min'          -- The valid time period for the deduplicat
);
CREATE TABLE `cmq_sink_json_table` (
    `id` int,
    `name` STRING
) WITH (
    'connector' = 'cmq',                          -- Here, it should be 'cmq'.
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',         -- The name
    'queue' = 'queue_name',              -- The name of the CMQ queue.
    'secret-id' = 'xxxx',                   -- The account secret ID.
    'secret-key' = 'xxxx',              -- The account secret key.
    'sign-method' = 'HmacSHA1',         -- The signature algorithm.
    'format' = 'json',                  -- The data format (JSON).
    'json.fail-on-missing-field' = 'false'   -- If this is 'false', no errors will
    'json.ignore-parse-errors' = 'true'    -- If this is 'true', all parse errors w
    'batch-size' = '16',                -- The number of messages sent at a time.
    'request-timeout' = '5000ms',   -- The request timeout period.
    'retry-times' = '3',                        -- The number of retries in case of
    'max-block-timeout' = '0s'                  -- The maximum time to wait to batc
);
insert into cmq_sink_json_table select * from cmq_source_json_table;
```

## Notes

Please pay attention to the following when using CMQ as a source:

1. You can configure the primary key to achieve deduplication over a time period you specify. **The longer this time period is, the higher the memory usage.**
2. We strongly recommend you set the visibility timeout period of CMQ messages to a value larger than the checkpoint interval for your Flink job to prevent consumed messages from being consumed again.

# TDMQ for RabbitMQ

Last updated：2023-11-08 14:55:24

## Overview

TDMQ for RabbitMQ (RMQ) is Tencent's proprietary message queue service. It supports the AMQP 0-9-1 protocol and is compatible with all components of Apache RabbitMQ. It can be used as a sink. You can output stream data processed by Flink operators to an RMQ queue.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Unsupported |

## Limits

RMQ can be used as a sink. It does not support upsert streams.

## Defining a table in DDL

### As a sink

#### JSON format

```
CREATE TABLE `rmq_sink_json_table` (
    `id` int,
    `name` STRING
) WITH (
      'connector' = 'rabbitmq',
    'host' = 'xxxx',
    'port' = 'xxxx',
    'vhost' = '/',
    'username' = 'xxxx',
    'password' = 'xxxx',
    'exchange' = 'exchange',                                    -- The exchange nam
```

```
        'routing-key' = 'Key',
        'format' = 'json',                    -- The data format (JSON).
      'json.fail-on-missing-field' = 'false'    -- If this is 'false', no errors will
      'json.ignore-parse-errors' = 'true'     -- If this is 'true', all parse errors w
);
```

**CSV format**

```
CREATE TABLE `rmq_sink_csv_table` (
    `id` int,
    `name` STRING
) WITH (
      'connector' = 'rabbitmq',
    'host' = 'xxxx',
    'port' = 'xxxx',
    'vhost' = '/',
    'username' = 'xxxx',
    'password' = 'xxxx',
    'exchange' = 'exchange',                                    -- The exchange nam
      'routing-key' = 'Key',
      'format' = 'csv',                -- The data format (CSV).
);
```

## WITH parameters

### Common WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | - | Here, it should be `'rabbitmq'` . |
| host | Yes | - | The host of the queue. |
| port | Yes | 5672 | The RMQ port. |
| vhost | Yes | / | The virtual host. |
| name-server | Yes | guest | The role name. |
| password | Yes | guest | The password of the role. |
| queue | No | - | The queue name. |
| exchange | Yes | - | The exchange name. |
| routing-key | No | - | The key bound by the exchange. |
| delivery-mode | No | 1 | Whether messages will be persistent. `1` : No; `2` : Yes. |
| expiration | No | 86400000 | The message validity period (milliseconds), which is one day by default. |
| network-recovery-interval | No | 30s | The network recovery interval. |
| automatic-recovery | No | true | Whether to connect to RMQ automatically. Automatic |

| | | | |
|---|---|---|---|
| | | | connection is enabled by default. |
| topology-recovery | No | true | Whether to recover RMQ topology automatically. Automatic topology recovery is enabled by default. |
| connection-timeout | No | 30s | The connection timeout period, which is 30 seconds by default. |
| requested-frame-max | No | 0 | The maximum frame size (bytes) for the first request. If this is `0` , no limit will be set. |
| requested-heartbeat | No | 60s | The heartbeat timeout period. |
| prefetch-count | No | 0 | The maximum number of messages the server can send. If this is `0` , no limit will be set. This option is only supported by Flink 1.13. |
| delivery-timeout | No | 30s | The commit timeout period. This option is only supported by Flink 1.13. |
| format | Yes | - | The input and output format of RMQ messages. Valid values include `'csv'` and `'json'` . |

### WITH parameters for JSON

| Option | Required | Default Value | Description |
|---|---|---|---|
| json.fail-on-missing-field | No | false | If this is `true` , the job will fail in case of missing parameters. If this is `false` (default), the missing parameters will be set to null and the job will continue to be executed. |
| json.ignore-parse-errors | No | false | If this is `true` , when there is a parse error, the field will be set to null and the job will continue to be executed. If this is `false` , the job will fail in case of a parse error. |
| json.timestamp-format.standard | No | SQL | The JSON timestamp format. The default value is `SQL` , in which case the format will be `yyyy-MM-dd HH:mm:ss.s{precision}` . You can also set it to `ISO-8601` , and the format will be `yyyy-MM-ddTHH:mm:ss.s{precision}` . |

### WITH parameters for CSV

| Option | Required | Default Value | Description |
|---|---|---|---|
| csv.field-delimiter | No | , | The field delimiter, which is comma by default. |
| csv.line-delimiter | No | U&'\\000A' | The line delimiter, which is `\\n` by default (in SQL, you must use `U&'\\000A'`). You can also set it to `\\r` (in SQL, you need to use `U&'\\000D'`). This option is only supported by Flink 1.11. |
| csv.disable-quote-character | No | false | Whether to disable quote characters. If this is `true`, 'csv.quote-character' cannot be used. |
| csv.quote-character | No | '' | The quote characters. Text inside quotes will be viewed as a whole. The default value is `''`. |
| csv.ignore-parse-errors | No | false | Whether to ignore parse errors. If this is `true`, fields will be set to null in case of parse failure. |
| csv.allow-comments | No | false | Whether to ignore comment lines that start with # and output them as empty lines (if this is `true`, make sure you set `csv.ignore-parse-errors` to `true` as well). |
| csv.array-element-delimiter | No | ; | The array element delimiter, which is `;` by default. |
| csv.escape-character | No | - | The escape character. By default, escape characters are disabled. |
| csv.null-literal | No | - | The string that will be seen as null. |

## Example

```
-- Please replace the parameter values below with the corresponding values for the
CREATE TABLE `rabbitmq_source_json_table` (`id` INT, `name` STRING) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://host:port/database?rewriteBatchedStatements=true&serverTime
  'table-name' = 'source_table_name',
  'username' = 'username',
  'password' = 'password'
);

CREATE TABLE `rabbitmq_sink_json_table` (`id` INT, `name` STRING) WITH (
  'connector' = 'rabbitmq',
  'host' = 'host',
  'port' = 'port',
  'vhost' = 'vhost',
  'username' = 'username',
  'password' = 'password',
  'queue' = 'queue-name',
  'exchange'='exchange',
  'routing-key'='key',
  'format' = 'json'
);
insert into rabbitmq_sink_json_table select * from rabbitmq_source_json_table;
```

**Note**

When RMQ is used as a sink, there is a small possibility of duplication.

# MySQL CDC

Last updated：2023-11-08 16:18:29

## Overview

The MySQL CDC connector (as a source) allows for reading snapshot data and incremental data from MySQL databases and guarantees the exactly-once semantic. This connector uses Debezium as its underlying platform to implement change data capture (CDC).

### How MySQL CDC 1.x works

1. Acquires a global read lock to prohibit writes by other database clients.
2. Starts a repeatable read transaction to ensure that data is always read from the same checkpoint.
3. Reads the current binlog position.
4. Reads the schema of the database and table configured in the connector.
5. Releases the global read lock to allow writes by other databases.
6. Scans the table and, after all data is read, captures the changes made after the binlog position in step 3.

Flink will periodically perform checkpoints to record the binlog position. In case of failover, the job will restart and restore from the checkpointed binlog position. Consequently, it guarantees the exactly-once semantic.

### How MySQL CDC 2.x works

1. A MySQL table must contain a primary key. If a composite key is used, the first field of the key will be selected as the partitioning key (splitKey) to divide the data into chunks during the snapshot phase.
2. A lock-free algorithm is used during the snapshot phase, with no need to lock the table.
3. The sync process consists of two phases. During the snapshot phase, chunks are read concurrently. After this phase ends, the incremental phase starts. The whole process supports checkpoints to guarantee the exactly-once semantic.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | MySQL v5.6 supported |
| 1.13 | MySQL v5.6, v5.7, and v8.x supported<br>Configured by default. The source must contain a primary key. If the source has no primary key, you need to set `'scan.incremental.snapshot.enabled'` to `'false'` in the WITH parameters. |

| | |
|---|---|
| 1.14 | MySQL v5.6, v5.7, and v8.x supported<br>Configured by default. The source must contain a primary key. If the source has no primary key, you need to set `'scan.incremental.snapshot.enabled'` to `'false'` in the WITH parameters. |
| 1.16 | MySQL v5.6, v5.7, and v8.x supported<br>Configured by default. The source must contain a primary key. If the source has no primary key, you need to set `'scan.incremental.snapshot.enabled'` to `'false'` in the WITH parameters. |

## Limits

The MySQL CDC connector can be used only as a source.

## Defining a table in DDL

```
CREATE TABLE `mysql_cdc_source_table` (
  `id` INT,
  `name` STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- Define the primary key here if the database ta
) WITH (
  'connector' = 'mysql-cdc',       -- Here, it should be 'mysql-cdc'.
  'hostname' = '192.168.10.22',    -- IP of the MySQL database server.
  'port' = '3306',                 --  Integer port number of the MySQL database ser
  'username' = 'debezium',         -- Name of the database to use when connecting to
  'password' = 'hello@world!',     -- Password to use when connecting to the MySQL d
  'database-name' = 'YourDatabase',   -- Database name of the MySQL server to monit
```

```
   'table-name' = 'YourTable'       -- Table name of the MySQL database to monitor.
);
```

## WITH parameters

| Option | Description | Required | Remarks |
|---|---|---|---|
| connector | The connector to use. | Yes | Here, it should b |
| hostname | IP address or hostname of the MySQL database server. | Yes | - |
| port | Integer port number of the MySQL database server. | No | Default value: 33 |
| username | Name of the MySQL database to use when connecting to the MySQL database server. | Yes | A MySQL user w permissions (inc RELOAD, SHOV REPLICATION S REPLICATION ( |
| password | Password to use when connecting to the MySQL database server. | Yes | - |
| database-name | Database name of the MySQL server to monitor. | Yes | The table-name regular expressi multiple tables m regular expressi |
| table-name | Table name of the MySQL database to monitor. | Yes | The table-name regular expressi multiple tables m regular expressi |
| server-id | A numeric ID of this database client. | No | It must be unique currently-running processes in the We recommend ID range for eacl database, such a By default, a ran equal to 6400 - Integer.MAX_VA generated. |

| | | | |
|---|---|---|---|
| server-time-zone | Session time zone in the database server. | No | An example is "A controls how the type in MySQL is STRING. |
| append-mode | Whether to enable the append mode. | No | This mode is ava v1.13 or later. It example, syncing CDC data to Hiv |
| filter-duplicate-pair-records | Filters source field change records that are not defined in the Flink DDL statements. | No | For example, a M contains four fiel but only a and b table creation SC this option is use involving only fie ignored and not downstream sys data to be manip |
| scan.lastchunk.optimize.enable | Repartitions the last chunk of the snapshot phase. | No | Numerous writes continuously ma during the snaps cause the last ch large, resulting in restart of TaskM OOM.<br>If this option is e `true` ), Flink a divides the last c large into small c job more stable. |
| debezium.min.row.count.to.stream.results | When the number of records in the table is greater than this value, the records will be read in batches. | No | It defaults to 100 data from a sour following method Full read: The da table is read to th method allows q data, but the me corresponding si consumed. If the extremely large, of OOM.<br>Batch read: Cert read each time u |

| | | | |
|---|---|---|---|
| | | | read. This metho risk of OOM whe large, but the rea |
| debezium.snapshot.fetch.size | Specifies the maximum number of rows that should be read from a MySQL source per time during the snapshot phase. | No | This option appli batch read mode |
| debezium.skipped.operations | Specifies the operations to be skipped, separated by comma. Operations include `c` (insert), `u` (update), and `d` (delete). By default, no operations will be skipped. | No | - |
| scan.incremental.snapshot.enabled | Specifies the incremental snapshot. | No | Default value: `t` |
| scan.incremental.snapshot.chunk.size | The chunk size (number of rows) of table snapshot. Captured tables are split into multiple chunks when the table snapshot is read. | No | Default value: `8` |
| scan.lazy-calculate-splits.enabled | Whether to enable lazy loading for the JobManager during the snapshot phase to avoid JobManager OOM because of too large data and too many chunks. | No | Default value: `t` |
| scan.newly-added-table.enabled | Whether to enable dynamic loading. | No | Default value: `f` |
| scan.split-key.mode | Specifies the primary key as the partitioning key. | No | Two valid values `default` anc `default` indi field of the comp as the partitionin `specific` re `scan.split-` `column` to spe composite key a key. |
| scan.split-key.specific-column | Specifies a field in the composite key as the partitioning key. | No | This option is rec value of `scan.` |

| | | | |
|---|---|---|---|
| | | | `key.mode` is value will be the the composite ke |
| connect.timeout | The maximum time that the connector should wait after trying to connect to the MySQL database server before timing out. | No | Default value: `3` |
| connect.max-retries | Max times that the connector should retry to build a MySQL database server connection. | No | Default value: `3` |
| connection.pool.size | Connection pool size. | No | Default value: `2` |
| jdbc.properties.* | Option to pass custom JDBC URL parameters, such as `'jdbc.properties.useSSL' = 'false'` . | No | Default value: `2` |
| heartbeat.interval | Interval of sending heartbeat event for tracing the latest available binlog offsets, generally used to address slow update of tables. | No | Default value: `2` |
| debezium.* | Debezium properties | No | Specifies Debez fine-grained cont behaviors on the `'debezium.s = 'never'` . F Debezium's MyS properties. |

## Available metadata (to Flink v1.13 or later)

Available metadata columns:

| Key | Data Type | Description |
|---|---|---|
| database_name/meta.database_name | STRING NOT NULL | Name of the table that contains the row. |
| table_name/meta.table_name | STRING NOT NULL | Name of the database that contains the row. |
| op_ts/meta.op_ts | TIMESTAMP_LTZ(3) | When the change was made in the |

| | | |
|---|---|---|
| | NOT NULL | database. |
| meta.batch_id | BIGINT | The batch ID of the binlog. |
| meta.is_ddl | BOOLEAN | Whether the metadata consists of DDL statements. |
| meta.mysql_type | MAP | Table structure. |
| meta.update_before | ARRAY | Field value before it was modified. |
| meta.pk_names | ARRAY | Name of the primary key field. |
| meta.sql | STRING | Null |
| meta.sql_type | MAP | Maps the fields in `sql_type` to the Java data type `ID` . |
| meta.ts | TIMESTAMP_LTZ(3) NOT NULL | When the row was received and processed. |
| meta.op_type | STRING | Operation type, such as `INSERT` or `DELETE` . |
| meta.file | STRING | Null for the snapshot phase, and the name of the binlog file to which the data belongs for the incremental phase, such as `mysql-bin.000101` . |
| meta.pos | BIGINT | `0` for the snapshot phase, and the offset of the binlog file to which the data belongs for the incremental phase, such as `143127802` . |
| meta.gtid | STRING | Null for the snapshot phase, and the `gtid` of the data for the incremental read, such as `3d3c4464-c320-11e9-8b3a-6c92bf62891a:66486240` . |

## Example

```
CREATE TABLE `mysql_cdc_source_table` (
      `id` INT,
      `name` STRING,
      `database_name` string METADATA FROM 'database_name',
      `table_name`    string METADATA FROM 'table_name',
      `op_ts`         timestamp(3) METADATA FROM 'op_ts',
      `op_type` string METADATA FROM 'meta.op_type',
      `batch_id` bigint METADATA FROM 'meta.batch_id',
      `is_ddl` boolean METADATA FROM 'meta.is_ddl',
      `update_before` ARRAY<MAP<STRING, STRING>> METADATA FROM 'meta.update_before'
      `mysql_type` MAP<STRING, STRING> METADATA FROM 'meta.mysql_type',
      `pk_names` ARRAY<STRING> METADATA FROM 'meta.pk_names',
      `sql` STRING METADATA FROM 'meta.sql',
      `sql_type` MAP<STRING, INT> METADATA FROM 'meta.sql_type',
      `ingestion_ts` TIMESTAMP(3) METADATA FROM 'meta.ts',
      PRIMARY KEY (`id`) NOT ENFORCED -- Define the primary key here if the databas
) WITH (
      'connector' = 'mysql-cdc',      -- Here, it should be 'mysql-cdc'.
      'hostname' = '192.168.10.22',   -- IP of the MySQL database server.
      'port' = '3306',                --  Integer port number of the MySQL database
      'username' = 'debezium',        -- Name of the database to use when connectin
      'password' = 'hello@world!',    -- Password to use when connecting to the MyS
      'database-name' = 'YourDatabase',   -- Database name of the MySQL server to m
      'table-name' = 'YourTable'      -- Table name of the MySQL database to monito
);
```

## Reading a MySQL database by sharding

Stream Compute Service supports reading MySQL databases by sharding.
If a MySQL database is already a shard-based database containing multiple tables such as A_1, A_2, A_3 ..., and **each table has the same schema**, you can set the `table-name` option to a regex to match and read multiple tables. For example, you can set `table-name` to `**A_.***` to monitor all tables prefixed with **A_**. `database-name` can also implement this feature.

**Note**

If `database-name` or `table-name` is set to a regex, the regex needs to be placed in `()` .

## Data type mapping

MySQL CDC and Flink SQL data types are mapped as follows:

| MySQL Type | Flink SQL Type | Note |
|---|---|---|
| TINYINT | TINYINT | - |
| SMALLINT<br>TINYINT UNSIGNED<br>TINYINT UNSIGNED ZEROFILL | SMALLINT | - |
| INT<br>MEDIUMINT<br>SMALLINT UNSIGNED<br>SMALLINT UNSIGNED ZEROFILL | INT | - |

| | | |
|---|---|---|
| BIGINT<br>INT UNSIGNED<br>INT UNSIGNED ZEROFILL<br>MEDIUMINT UNSIGNED<br>MEDIUMINT UNSIGNED ZEROFILL | BIGINT | - |
| BIGINT UNSIGNED<br>BIGINT UNSIGNED ZEROFILL<br>SERIAL | DECIMAL(20, 0) | - |
| FLOAT<br>FLOAT UNSIGNED<br>FLOAT UNSIGNED ZEROFILL | FLOAT | - |
| REAL<br>REAL UNSIGNED<br>REAL UNSIGNED ZEROFILL<br>DOUBLE<br>DOUBLE UNSIGNED<br>DOUBLE UNSIGNED ZEROFILL<br>DOUBLE PRECISION<br>DOUBLE PRECISION UNSIGNED<br>DOUBLE PRECISION UNSIGNED ZEROFILL | DOUBLE | - |
| NUMERIC(p, s)<br>NUMERIC(p, s) UNSIGNED<br>NUMERIC(p, s) UNSIGNED ZEROFILL<br>DECIMAL(p, s)<br>DECIMAL(p, s) UNSIGNED<br>DECIMAL(p, s) UNSIGNED ZEROFILL<br>FIXED(p, s)<br>FIXED(p, s) UNSIGNED<br>FIXED(p, s) UNSIGNED ZEROFILL<br>where p <= 38 | DECIMAL(p, s) | - |
| NUMERIC(p, s)<br>NUMERIC(p, s) UNSIGNED<br>NUMERIC(p, s) UNSIGNED ZEROFILL<br>DECIMAL(p, s)<br>DECIMAL(p, s) UNSIGNED<br>DECIMAL(p, s) UNSIGNED ZEROFILL<br>FIXED(p, s)<br>FIXED(p, s) UNSIGNED<br>FIXED(p, s) UNSIGNED ZEROFILL<br>where 38 < p <= 65 | STRING | The precision for DECIMAL data type is up to 65 in MySQL, but the precision for DECIMAL is limited to 38 in Flink. So if you define a decimal column whose precision is greater than 38, you should map it to STRING to avoid precision loss. |

| | | |
|---|---|---|
| BOOLEAN<br>TINYINT(1)<br>BIT(1) | BOOLEAN | - |
| DATE | DATE | - |
| TIME [(p)] | TIME [(p)] | - |
| TIMESTAMP [(p)]<br>DATETIME [(p)] | TIMESTAMP [(p)] | - |
| CHAR(n) | CHAR(n) | - |
| VARCHAR(n) | VARCHAR(n) | - |
| BIT(n) | BINARY(⌈n/8⌉) | - |
| BINARY(n) | BINARY(n) | - |
| VARBINARY(N) | VARBINARY(N) | - |
| TINYTEXT<br>TEXT<br>MEDIUMTEXT<br>LONGTEXT | STRING | - |
| TINYBLOB<br>BLOB<br>MEDIUMBLOB<br>LONGBLOB | BYTES | Currently, for BLOB data type in MySQL, only the blob whose length isn't greater than 2,147,483,647(2 ** 31 - 1) is supported. |
| YEAR | INT | - |
| ENUM | STRING | - |
| JSON | STRING | The JSON data type will be converted into STRING with JSON format in Flink. |
| SET | ARRAY<STRING> | As the SET data type in MySQL is a string object that can have zero or more values, it should always be mapped to an array of string. |
| GEOMETRY<br>POINT<br>LINESTRING<br>POLYGON<br>MULTIPOINT | STRING | The spatial data types in MySQL will be converted into STRING with a fixed JSON format. |

| | | |
|---|---|---|
| MULTILINESTRING<br>MULTIPOLYGON<br>GEOMETRYCOLLECTION | | |

## Example

```
CREATE TABLE `mysql_cdc_source_table` (
  `id` INT,
  `name` STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- Define the primary key here if the database ta
) WITH (
  'connector' = 'mysql-cdc',       -- Here, it should be 'mysql-cdc'.
  'hostname' = '192.168.10.22',    -- IP of the MySQL database server.
  'port' = '3306',                 --  Integer port number of the MySQL database ser
  'username' = 'debezium',         -- Name of the database to use when connecting to
  'password' = 'hello@world!',     -- Password to use when connecting to the MySQL d
  'database-name' = 'YourDatabase',   -- Database name of the MySQL server to monit
  'table-name' = 'YourTable'       -- Table name of the MySQL database to monitor.
);

CREATE TABLE `print_table` (
  `id` INT,
  `name` STRING
) WITH (
 'connector' = 'print'
);
insert into print_table select * from mysql_cdc_source_table;
```

## Notes

### Checkpoints

For MySQL CDC 1.0 ('scan.incremental.snapshot.enabled' = 'false'), you need to make additional configurations to use checkpoints.
When MySQL CDC 1.0 reads snapshot data, it is unable to take a checkpoint. If many tables involving a large volume of data need to be synced, many times of checkpoint failure may occur, resulting in the job failure. You can set the allowed number of checkpoint failures ( `execution.checkpointing.tolerable-failed-checkpoints: 100` ) as instructed in Advanced Job Parameters.
If you use MySQL CDC 2.0 and the default job parallelism is greater than 1, checkpointing must be enabled.
After reading the snapshot data, the CDC connector will wait for the completion of a checkpoint before starting to read the incremental data.

### Risks of using MySQL CDC 1.0

When a table has no primary key, you can only enable the CDC 1.0 mode using `'scan.incremental.snapshot.enabled' = 'false'` , which poses the following risks:
1. The `FLUSH TABLES WITH READ LOCK` (FTWRL) statement will be executed by default.
2. Although FTWRL only lasts for a short period of time, its mechanism may cause the database to be **locked**.
3. FTWRL may also cause the following:
Wait until an UPDATE/SELECT operation to complete.

During this period, the database will be unavailable, blocking new SELECT queries. This is caused by the Query Cache mechanism in MySQL.
If multiple MySQL CDC 1.0 sources run in parallel, the above situation probably will occur.

## User permissions

The user of the source database must have the following permissions: SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT, SELECT, and RELOAD.

```
GRANT SELECT, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'user
FLUSH PRIVILEGES;
```

### Global read lock

In the "How MySQL CDC 1.x works" section above, the first step is to acquire a global read lock to get the schema and binlog position. This will block writes by other database clients and may affect the online business. If the At Least Once semantic is acceptable, set `'debezium.snapshot.locking.mode'` to `'none'` to skip this step.

### Setting a composite key

The following DDL statements set `index1` , `index2` , `index3` , and `index4` as indices of the composite key. They must be defined in the same way in `PRIMARY KEY` , and their order will not affect the sync.

```
CREATE TABLE db_order_dim (
  `index1` STRING,
  `index2` STRING,
  `index3` STRING,
  `index4` STRING,
  `field5` STRING,
  `field6` STRING,
  PRIMARY KEY(`index1`, `index2`, `index3`, `index4`) NOT enforced
) WITH (
  ...
);
```

### Defining a server-id

We do not recommend you specify an explicit `server-id`, because the Stream Compute Service system automatically generates a random `server-id` in the range of `6400 - 2147483647` to avoid a `server-id` conflict that may occur when several jobs read the same database.

If you indeed need to specify a `server-id` in MySQL CDC 2.x, we recommend you set it to a range, such as `5400-5405`. Since each parallel reader must have a unique server ID, `server-id` must be a range like `5400-5405`, and the range must be larger than the parallelism. For MySQL CDC 1.x, only a `server-id` value but not a range can be specified.

Two methods are available for specifying `server-id`:

1. Use the WITH parameters in the `mysql-cdc` DDL statements.
2. Use SQL hints.

```
SELECT * FROM source_table /*+ OPTIONS('server-id'='5401-5404') */ ;
```

## Setting MySQL session timeouts

When an initial consistent snapshot is made for large databases, your established connection could time out while the tables are being read. You can prevent this behavior by configuring `interactive_timeout` and `wait_timeout` in your MySQL configuration file.

`interactive_timeout` indicates the number of seconds the server waits for activity on an interactive connection before closing it. For details, see MySQL documentation.

`wait_timeout` indicates the number of seconds the server waits for activity on a noninteractive connection before closing it. For details, see MySQL documentation.
During the reading of incremental data, heartbeat may be invalid due to an extremely high load on the TaskManagers, and the server disconnects the connection (EOFException). In this case, you can run the following SQL statements in the MySQL server to increase the timeout:

```
SET GLOBAL slave_net_timeout = 120;
SET GLOBAL thread_pool_idle_timeout = 120;
```

### Critical logs of the JobManager

For MySQL CDC 2.x, the sync of each table consists of the following stages: **chunk splitting**, **snapshot reading**, **incremental correction**, and **incremental reading**. Since more resources are occupied for a longer period at the first three stages, Stream Compute Service has made improvements in logs and metrics to help you gain insight into and analyze the running of your jobs.

### 1. Splitting and assigning chunks

**Splitting start**: Search by the `into chunks` keywords, such as `Start splitting table cdc_basic_source.random_source_1 into chunks`, or `Start lazily splitting table cdc_basic_source.random_source_1 into chunks`.
**Splitting end**: Search by the `chunks, time cost` keywords, such as `Split table cdc_basic_source.random_source_1 into 14 chunks, time cost: 994ms`.

### 2. Snapshot reading

**Assignment of chunks from snapshot**: Enable the log of the `DEBUG` level, and search by the `Current assigned splits for` keywords to view the total number of chunks of each table and their assignment.
**Chunk assignment end**: Search by the `finished. Total split number` keywords, such as `Split assignment for cdc_basic_source.random_source_1 finished. Total split number: 14`.
**Snapshot reading end**: Search for the `Assigner status changes from INITIAL_ASSIGNING to INITIAL_ASSIGNING_FINISHED` log.

### 3. Incremental correction

**Incremental correction start**: Search for the `Initial assigning finished as there are no more splits. Creating binlog split` or `Newly added assigning finished as there are no more splits. Waking up binlog reader` log.

### 4. Incremental reading

Refer to the following section, "Critical logs of the TaskManagers".

## Critical logs of the TaskManagers

**Incremental reading start**: Search by the `has entered pure binlog phase` keywords, such as `Table cdc_basic_source.random_source_2 has entered pure binlog phase.`
**Table schema change**: Search by the `Received schema change event` keywords.
**EOFException**: If a job restarts and the prompted exception is `EOFException`, adjust the timeout in the MySQL server as prompted. Alternatively, you can set a smaller total parallelism and increase the spec of each TaskManager to reduce the probability of timeout due to high memory and CPU pressure.

## Monitoring metrics

Stream Compute Service adds many practical metrics in the MySQL CDC connector. You can go to the Flink UI of a job as instructed in Flink UI, and click the MySQL CDC source operator in the execution graph to search for and view these metrics.

**logpos**: Get the current binlog position, which can help identify consumption blocks and other issues.
**numberOfInsertRecords**: Get the number of output +I messages.
**numberOfDeleteRecords**: Get the number of output -D messages.
**numberOfUpdateBeforeRecords**: Get the number of output -U messages.
**numberOfUpdateAfterRecords**: Get the number of output +U messages.

# Pulsar

Last updated：2023-11-08 15:55:14

## Overview

The Pulsar SQL connector allows you to read data from or write data to Pulsar topics using simple SQL queries or Flink Table API.

## Versions

The Flink Pulsar connector, based on StreamNative/Flink, is supported by Flink 1.13 and 1.14. For information about the DataStream API, see the StreamNative document.

| Flink Version | Description | Source Code |
|---|---|---|
| 1.11 | Unsupported | - |
| 1.13 | Supported | develop-flink-1.13 |
| 1.14 | Supported | develop-flink-1.14 |
| 1.16 | Supported | |

## Limits

Pulsar can be used as a source or as a sink for tuple and upsert streams.
It does not support dimension tables.

## Creating a Pulsar table

The example below shows you how to create a Pulsar table:

```
CREATE TABLE PulsarTable (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  `publish_time` TIMESTAMP_LTZ(3) METADATA FROM 'publish_time' VIRTUAL
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.Auth
  -- 'pulsar.client.authParams' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.C
```

```
    'topics' = 'user_behavior',
    'format' = 'json',
    'source.subscription-name' = 'flink',
    'source.start.message-id' = 'earliest'
  );
```

**Note**

If authentication is enabled for your Pulsar cluster, please use a token with admin permissions.

## About tokens

The Flink Pulsar connector uses Pulsar admin APIs to listen for partition changes and subscription creation, so if token authentication is enabled for your Pulsar cluster, a token with admin permissions is needed. You will also need permission to read from and write to the topic.

```
# Checking admin permissions
admin_url=http://172.28.28.46:8080,172.28.28.29:8080,172.28.28.105:8080
token=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-Bi__yGCs1lxEwdUcDKBDuJ
namespace=public/default
pulsar-admin --admin-url ${admin_url} --auth-params token:${token} --auth-plugin or

# Checking topic read and write permissions
service_url=pulsar://172.28.28.46:6650,172.28.28.29:6650,172.28.28.105:6650
token=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-Bi__yGCs1lxEwdUcDKBDuJ
namespace=public/default
topic=xxx
```

```
subscription=yyy
pulsar-client --url ${service_url} --auth-params token:${token} --auth-plugin org.a
```

## Connector options

| Option | Required | Default Value | Data Type | Description |
|---|---|---|---|---|
| connector | Yes | - | String | The connector to us `pulsar'` . |
| admin-url | Yes | - | String | The Pulsar admin U `broker.example.` `broker.example.` |
| service-url | Yes | - | String | The URL of the Puls A Pulsar protocol U cluster. Example: `p` URL for multiple bro `pulsar://local` Production clusters `pulsar://pulsa` URL with TLS authe `west.example.c` |
| topics | Yes | - | String | The names of the A This option can be o such as `topic-1;` You can specify a s `partition-0;to` If both a topic and it example, `some-t` `some-topic1` . |
| pulsar.client.authPluginClassName | No | - | String | The authentication p `org.apache.pu` |
| pulsar.client.authParams | No | - | String | The authentication p `token:xxxx` . |
| explicit | No | true | Boolean | Whether the table is For details, see the |
|  |  |  |  |  |

| | | | | |
|---|---|---|---|---|
| key.fields | No | - | List<String> | The physical fields ir messages. Note tha |
| key.format | No | - | String | The format used to c Valid values include see Formats. |
| format | No | - | String | The format used to c messages. Valid values include see Formats. Between `format` If you specify both, |
| value.format | No | - | String | The format used to c messages. Valid values include see Formats. Between `format` If you specify both, |
| sink.topic-routing-mode | No | round-robin | Enum | The topic routing pol `message-key-ha` also configure a cust `router` option. |
| sink.custom-topic-router | No | - | String | The full class name t option, do not set `s` |
| sink.message-delay-interval | No | 0 | Duration | The delay time for se This allows you to de Pulsar document De |
| pulsar.sink.deliveryGuarantee | No | none | Enum | The message delive `none` , `at-leas` `once` , your Pulsar |
| pulsar.sink.transactionTimeoutMillis | No | 10800000 | Long | The Pulsar transacti than the checkpoint |
| pulsar.producer.batchingEnabled | No | false | Boolean | Whether to enable b |
| pulsar.producer.batchingMaxMessages | No | 1000 | Int | The maximum numb |
| source.start.message-id | No | - | String | The start of source c a specific message I (e.g., "12:2:-1"). |

| | | | | |
|---|---|---|---|---|
| source.start.publish-time | No | - | Long | The publishing time consumption. |
| source.subscription-name | No | flink-sql-connector-pulsar-<RANDOM> | String | The Pulsar subscrip `connector-pulsa` |
| source.subscription-type | No | Exclusive | Enum | The Pulsar subscrip `Shared` . For mor types. |
| source.stop.at-message-id | No | - | String | The end of source co a specific message I (e.g., "12:2:-1"). |
| source.stop.at-publish-time | No | - | Long | The publishing time consumption. |
| source.stop.after-message-id | No | - | String | The ID of the ending `ledgerId:entry` message will be con |
| pulsar.source.partitionDiscoveryIntervalMs | No | 30000 | Long | The interval (millisec partitions. If this is |
| pulsar.admin.requestRetries | No | 5 | Int | The number of retrie |
| pulsar.client.* | No | - | - | An arbitrary Pulsar c |
| pulsar.admin.* | No | - | - | An arbitrary Pulsar a |
| pulsar.sink..* | No | - | - | An arbitrary Pulsar s |
| pulsar.producer.* | No | - | - | An arbitrary Pulsar p |
| pulsar.source..* | No | - | - | An arbitrary Pulsar s |
| pulsar.consumer.* | No | - | - | An arbitrary Pulsar c |

## Available metadata

| Metadata Key | Data Type | R/W | Description |
|---|---|---|---|
| topic | STRING NOT NULL | R | The topic name of a Pulsar |

| | | | |
|---|---|---|---|
| | | | message. |
| message_size | INT NOT NULL | R | The Pulsar message size. |
| producer_name | STRING NOT NULL | R | The producer name of a Pulsar message. |
| message_id | BYTES NOT NULL | R | The ID of a Pulsar message. |
| sequenceId | BIGINT NOT NULL | R | The sequence ID of a Pulsar message. |
| publish_time | TIMESTAMP_LTZ(3) NOT NULL | R | The publishing time of a Pulsar message. |
| event_time | TIMESTAMP_LTZ(3) NOT NULL | R/W | The properties of a Pulsar message. |
| properties | MAP<STRING, STRING> NOT NULL | R/W | The event time of a Pulsar message. |

**Note**
**R**/**W** determines metadata access, which can be read only or write. Read-only columns should be excluded in `VIRTUAL` and `INSERT INTO` operations.
For a list of the fields in Pulsar messages, see the Pulsar document Messages.

## Data type mappings

| Pulsar Schema | Flink Format |
|---|---|
| AVRO | avro |
| JSON | json |
| PROTOBUF | Not supported yet |
| PROTOBUF_NATIVE | Not supported yet |
| AUTO_CONSUME | Not supported yet |
| AUTO_PUBLISH | Not supported yet |
| NONE/BYTES | raw |
| BOOLEAN | raw |

| | |
|---|---|
| STRING | raw |
| DOUBLE | raw |
| FLOAT | raw |
| INT8 | raw |
| INT16 | raw |
| INT32 | raw |
| INT64 | raw |
| LOCAL_DATE | Not supported yet |
| LOCAL_TIME | Not supported yet |
| LOCAL_DATE_TIME | Not supported yet |

## PulsarCatalog

PulsarCatalog can store Pulsar clusters as metadata of Flink tables.

### Explicit and native tables

PulsarCatalog defines two types of tables: explicit tables and native tables.
Explicit tables are tables created explicitly by the CREATE statement or using a table API. They work similarly to the tables in other SQL connectors. You can create an explicit table and read from or write to the table.
Native tables are created automatically by PulsarCatalog. PulsarCatalog scans all the non-system topics in a Pulsar cluster and converts each topic into a Flink table. Such tables are not created using the CREATE statement.

#### Explicit tables

PulsarCatalog uses the `schemaInfo` field in the schema of a topic to store the metadata of an explicit table. For each explicit table, PulsarCatalog creates a place-holding topic. You can specify the tenant of the topic using the `catalog-tenant` option. The default tenant is `__flink_catalog` . Your Flink database maps to a namespace with the same name under this tenant. Then a topic named `table_<FLINK_TABLE_NAME>` is created, whose schema stores the metadata of the Flink table.
For example, if you create a database `testdb` and a Flink table `users` , PulsarCatalog will create a topic `table_users` in the namespace `testdb` under the tenant `__flink_catalog` .
The topic `table_users` is a place-holding topic because it doesn't have any producers or consumers. You can use the schema of this topic to store the metadata of the Flink table.

To get the metadata of a topic, you can use the Pulsar admin command line tool:

```
pulsar-admin schemas get persistent://<tenant>/<namespace>/<topic>
```

### Native tables

Native tables do not have place-holding topics. PulsarCatalog maps topic schema to Flink table schema. For more information about Pulsar schema, see the Pulsar document Understand schema.

| Pulsar Schema | Flink Data Type | Flink Format | Work |
|---|---|---|---|
| | | | |

| | | | |
|---|---|---|---|
| AVRO | It is decided by the Avro format. | avro | Yes |
| JSON | It is decided by the JSON format. | json | Yes |
| PROTOBUF | Not supported yet | / | No |
| PROTOBUF_NATIVE | It is decided by the Protobuf definition. | Not supported yet | No |
| AUTO_CONSUME | Not supported yet | / | No |
| AUTO_PUBLISH | Not supported yet | / | No |
| NONE/BYTES | DataTypes.BYTES() | raw | Yes |
| BOOLEAN | DataTypes.BOOLEAN() | raw | Yes |
| LOCAL_DATE | DataTypes.DATE() | / | No |
| LOCAL_TIME | DataTypes.TIME() | / | No |
| LOCAL_DATE_TIME | DataTypes.TIMESTAMP(3) | / | No |
| STRING | DataTypes.STRING() | raw | Yes |
| DOUBLE | DataTypes.DOUBLE() | raw | Yes |
| FLOAT | DataTypes.FLOAT() | raw | Yes |
| INT8 | DataTypes.TINYINT() | raw | Yes |
| INT16 | DataTypes.SMALLINT() | raw | Yes |
| INT32 | DataTypes.INT() | raw | Yes |
| INT64 | DataTypes.BIGINT() | raw | Yes |

**Note**
Although the Pulsar schema types `LOCAL_DATE` and `LOCAL_TIME` have corresponding Flink data types, Flink cannot parse data based on the two schema types, and automatic schema mapping will fail.

#### Explicit table versus native table

With native tables, you can read data from existing Pulsar topics. PulsarCatalog automatically reads the schema of a topic and determines the format to use for decoding/encoding. However, native tables do not support watermark or primary keys. Therefore, with native tables, you cannot perform data aggregation based on time windows. A native table maps `tenant/namespace` to the Flink database and the topic name to the Flink table name.

If you want full control of a table, create an explicit table, define the watermark, and specify the metadata fields and custom formats. It's similar to creating a Pulsar table in `GenericInMemoryCatalog` . You can bind an explicit table to a Pulsar topic. Each Pulsar topic can be bound with multiple Flink tables.

## PulsarCatalog parameters

| Key | Default | Type | Description |
|---|---|---|---|
| **catalog-admin-url** | "http://localhost:8080&quot; | String | The Pulsar admin URL, such as `http://my-broker.example.com:8080` or `https://my-broker.example.com:8443` . |
| **catalog-auth-params** | - | String | The authentication parameters for accessing the Pulsar clu |
| **catalog-auth-plugin** | - | String | The name of the authentication plugin for accessing the Pu |
| **catalog-service-url** | "pulsar://localhost:6650" | String | The URL of the Pulsar server.<br>A Pulsar protocol URL is needed for a Pulsar client to conr cluster. Example: `pulsar://localhost:6650`<br>URL for multiple brokers:<br>`pulsar://localhost:6550,localhost:6651,lc`<br>Production clusters are usually accessed via domains, suc<br>`pulsar://pulsar.us-west.example.com:6650`<br>URL with TLS authentication enabled: `pulsar+ssl://` `west.example.com:6651` |
| **catalog-tenant** | "__flink_catalog" | String | The Pulsar tenant that stores table information. |
| **default-database** | "default_database" | String | The default database of PulsarCatalog. If a database with not exist, one will be created automatically. |

## PulsarCatalog example

```
CREATE CATALOG pulsar WITH (
  'type' = 'pulsar-catalog',
  -- 'catalog-auth-plugin' = 'org.apache.pulsar.client.impl.auth.AuthenticationToke
  -- 'catalog-auth-params' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2
  'catalog-admin-url' = '<ADMIN_URL>',
  'catalog-service-url' = '<SERVICE_URL>'
);
```

## Full example

### Pulsar source and sink

The example below shows how to create a Pulsar source and a Pulsar sink that guarantee exactly-once, with a transaction timeout period of 2 minutes (note that the transaction timeout period must be longer than the checkpoint interval).

```
CREATE TABLE `pulsar_source` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.Auth
  -- 'pulsar.client.authParams' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.C
  'topics' = 'topic_source',
  'format' = 'json',
  'source.subscription-name' = 'flink',
  'source.start.message-id' = 'earliest'
);

CREATE TABLE `pulsar_sink` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.Auth
  -- 'pulsar.client.authParams' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.C
  'topics' = 'topic_sink',
  'format' = 'json',
  'pulsar.sink.deliveryGuarantee' = 'exactly-once',
  'pulsar.sink.transactionTimeoutMillis' = '120000'
);

INSERT INTO `pulsar_sink` SELECT * FROM `pulsar_source`;
```

## PulsarCatalog example

### Explicit table

```
CREATE CATALOG `pulsar` WITH (
  'type' = 'pulsar-catalog',
  -- 'catalog-auth-plugin' = 'org.apache.pulsar.client.impl.auth.AuthenticationToke
  -- 'catalog-auth-params' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2
  'catalog-admin-url' = 'http://pulsar:8080',
  'catalog-service-url' = 'pulsar://pulsar:6650'
);
INSERT INTO `pulsar`.`default_database`.`pulsar_sink` SELECT * FROM `pulsar`.`defau
```

The `pulsar_source` and `pulsar_sink` tables in the above example are created using the following statements (which can be put in the same SQL job).

```
CREATE TABLE IF NOT EXISTS `pulsar`.`default_database`.`pulsar_source` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.Auth
  -- 'pulsar.client.authParams' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.C
  'topics' = 'topic_source',
  'format' = 'json',
  'source.subscription-name' = 'flink',
  'source.start.message-id' = 'earliest'
);

CREATE TABLE IF NOT EXISTS `pulsar`.`default_database`.`pulsar_sink` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.Auth
  -- 'pulsar.client.authParams' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.C
  'topics' = 'topic_sink',
  'format' = 'json',
  'pulsar.sink.deliveryGuarantee' = 'exactly-once',
  'pulsar.sink.transactionTimeoutMillis' = '120000'
);
```

**Native table**

1. Prepare a JSON file of the topic schema. Name it "schema.json".

```
{
  "schema": "{\\"type\\":\\"record\\",\\"name\\":\\"userBehavior\\",\\"namespace\\"
  "type": "JSON",
  "properties": {}
}
```

2. Use the Pulsar admin command line tool to configure the topic schema.

```
# Configure the schema
bin/pulsar-admin schemas upload -f ./schema.json topic_source
bin/pulsar-admin schemas upload -f ./schema.json topic_sink

# Check the schema
bin/pulsar-admin schemas get topic_source
bin/pulsar-admin schemas get topic_sink
```

3. Below is an example of a job. The format of the Flink table's database `public/default` is `tenant/namespace`, which is the default Pulsar cluster.

```
CREATE CATALOG `pulsar` WITH (
  'type' = 'pulsar-catalog',
  -- 'catalog-auth-plugin' = 'org.apache.pulsar.client.impl.auth.AuthenticationToke
  -- 'catalog-auth-params' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2
  'catalog-admin-url' = 'http://pulsar:8080',
  'catalog-service-url' = 'pulsar://pulsar:6650'
);

INSERT INTO `pulsar`.`public/default`.`topic_sink` SELECT * FROM `pulsar`.`public/d
```

## FAQs

### What should I do if an error occurs saying that I haven't enabled transactions?

```
java.lang.NullPointerException: You haven't enable transaction in Pulsar client.
```

To enable transactions, see How to use transactions?.

### Messages are written to a Pulsar sink with exactly-once guaranteed by default. After the job restarts due to an error, the topic data cannot be consumed. Why?

**Cause**: This may be because you have uncommitted transactions before the job restarts, and the OPEN transactions blocked operations to read data written to the topic after the restart. You can use `pulsar-admin transactions slow-transactions -t 1s` to view transactions in OPEN state. After OPEN transactions are committed or reverted, you will be able to read data written to the topic after the restart.
**Suggestion**: Configure an appropriate transaction timeout period using WITH parameters (the default Pulsar transaction timeout is 3 hours). For example, you can use `'pulsar.sink.transactionTimeoutMillis' = '120000'` to set the timeout period to 2 minutes. Note that the transaction timeout period must be larger than the checkpoint interval.

### Why does the Pulsar source fail to restore data from a checkpoint in the batch messaging scenario?

If the error `java.lang.IllegalArgumentException: We only support normal message id currently` occurs, it's because batch write is enabled for Pulsar write operations. Currently, Pulsar sources do not support restoring batch-write messages. With Stream Compute Service, batch write is disabled by default for Pulsar sinks.

```
Caused by: java.lang.IllegalArgumentException: We only support normal message id cu
```

### What is the relationship between the start of Pulsar source consumption and subscriptions?

If a topic does not have a subscription, a subscription will be created based on the ID of the starting message of source consumption.

If the topic has a subscription, the start of source consumption will be ignored.

If data is not restored from a checkpoint, consumption will start from the subscription cursor. If data is restored from a checkpoint, consumption will start from the message following the message ID recorded by the checkpoint. This is achieved by resetting the subscription cursor (for details, see `PulsarOrderedPartitionSplitReader#beforeCreatingConsumer`).

## Why can't I use the `NonDurable` subscription mode for the Pulsar source?

`PulsarSourceEnumerator#createSubscription` creates a `Durable` subscription first. If `PulsarPartitionSplitReaderBase#createPulsarConsumer` then consumes data in the `NonDurable` mode, the error `Durable subscription with the same name already exists` will occur.

| Option | Required | Default Value | Data Type | Description |
|---|---|---|---|---|
| pulsar.consumer.subscriptionMode | No | Durable | Enum | The Pulsar subscription mode. Valid values include `Durable` and `NonDurable`. In the `Durable` mode, the cursor is durable, which retains messages and persists the current position. If a broker restarts from a failure, it can recover the cursor from the persistent storage (bookie), so that messages can continue to be consumed from the last consumed position. In the `NonDurable` mode, once a broker stops, the cursor is lost and can never be recovered, so messages cannot continue to be consumed from the last consumed position. To learn more, see Subscription modes. |

## About MessageId

See Message Storage and ID Generation Rules. Messages IDs can be compared, for example, `174:1:0 > 174:1:-1`.

## Why does the Pulsar source fail to consume data according to `publish-time`?

**Cause**: If the broker connected does not provide namespace information of the topic, the RESTful API getting the message ID according to `publish-time` will return `HTTP 307 Temporary Redirect`, and the Pulsar client API used in the Flink connector will return `HTTP 500 Server Error`. The job will fail to start. You can use the RESTful API get-message-by-id to view the error.

```
## 1662480195714 is a publishing time accurate to the millisecond.
curl http://${adminUrl}:8080/admin/v2/persistent/public/default/${topic}/messageid/
```

**Note**

Because the RESTful API querying message IDs according to `publish-time` is not used, you can specify `source.stop.at-publish-time` .

**Suggestion**: You can try increasing `pulsar.admin.requestRetries` (the number of retries for RESTful APIs, which is 5 by default) to avoid this issue.

### Why doesn't the job stop at the configured ending position for Pulsar source consumption?

**Solution**: Disable automatic partition detection using `'pulsar.source.partitionDiscoveryIntervalMs' ='0'` .

# TDSQL for MySQL

Last updated：2023-11-08 14:53:32

## Overview

The tdsql-subscribe connector is dedicated to the subscription of TDSQL for MySQL data. It allows integrating the incremental binlog data from TDSQL for MySQL as instructed in Creating TDSQL for MySQL Data Subscription. Before using this connector, make sure a data subscription task has been successfully configured.

**Note**

The tdsql-subscribe connector is under beta testing. If you need to try it, submit a ticket.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Unsupported |

## ## Limits

The tdsql-subscribe connector can be used as a source but not a sink of a stream.

## Defining a table in DDL

When the tdsql-subscribe connector is used as a source, most of its WITH parameters are similar to those of the Kafka connector, and all connection parameters can be found in the subscription task.

Please note that when using the tdsql-subscribe connector, you must set `format` to `protobuf` , because the messages sent to Kafka via the subscription task is in `protobuf` format. Compared with the Kafka connector, the tdsql-subscribe connector contains more authentication information from the subscription task.

## As a source

### Input in protobuf

```
CREATE TABLE `DataInput` (
      `id` INT,
      `name` VARCHAR,
       `age` INT,
) WITH (
    'connector' = 'tdsql-subscribe',   -- Make sure you specify the corresponding c
    'tdsql.database.name' = 'test_case_2022_06_0*', -- Filter subscription messages
    'tdsql.table.name' = 'test_0*', -- Filter subscription messages to consume subs
    'topic' = 'topic-subs-5xop97nffk-tdsqlshard-xxx',  -- Replace it with the topic
    'scan.startup.mode' = 'earliest-offset', -- Valid values: `latest-offset`, `ear
    'properties.bootstrap.servers' = 'guangzhou-kafka-2.cdb-dts.tencentcs.com.cn:32
    'properties.group.id' = 'consumer-grp-subs-xxx-kk',
    'format' = 'protobuf', -- Only protobuf is allowed.
    'properties.security.protocol'='SASL_PLAINTEXT', -- Authentication protocol.
    'properties.sasl.mechanism'='SCRAM-SHA-512', -- Authentication method.
    'properties.sasl.jaas.config'='org.apache.kafka.common.security.scram.ScramLogi
);
CREATE TABLE `jdbc_upsert_sink_table` (
    id INT PRIMARY KEY NOT ENFORCED,
    name STRING,
    age INT
) WITH (
    -- Specify the parameters for database connection.
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://172.28.28.138:3306/testdb', -- Replace it with your MySQL
    'table-name' = 'sink', --The table into which the data will be written.
    'username' = 'user',      -- The username (with the INSERT permission required)
    'password' = 'psw'  -- The password for database access.
);
INSERT INTO jdbc_upsert_sink_table SELECT * FROM DataInput;
```

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | None | Here, it should be `'tdsql-subscribe'` . |
| topic | Yes | None | The name of the Kafka topic to be read. |
| properties.bootstrap.servers | Yes | None | The Kafka bootstrap addresses, separated by comma. |
| properties.group.id | Yes | None | The ID of the Kafka consumer group. |
| format | Yes | None | The input format of a Kafka message. Only `protobuf` supported. |
| scan.startup.mode | No | group-offsets | The Kafka consumer start mode.<br>Valid values: `latest-offset` , `earliest-offset` `specific-offsets` , `group-offsets` , and `timestamp` . If `'specific-offsets'` is used, sp |

| | | | |
|---|---|---|---|
| | | | offset of each partition, such as `'scan.startup.spec offsets' = 'partition:0,offset:42;partition:1,offset` If `'timestamp'` is used, specify the startup timestam such as `'scan.startup.timestamp-miles' = '1631588815000'` . |
| scan.startup.specific-offsets | No | None | If `scan.startup.mode` is set to `'specific-offs` this option must be used to specify the specific offset of th such as `'partition:0,offset:42;partition:1,offse` |
| scan.startup.timestamp-millis | No | None | If `scan.startup.mode` is set to `'timestamp'` , t must be used to specify the time point (Unix timestamp in the startup. |
| tdsql.database.name | No | None | The name of the TDSQL database. If this option is set, thi connector can consume the binlog data of the database s here, provided that the subscription task contains the binl this database. This option supports a regex, such as `test_case_2022_06_0*` . |
| tdsql.table.name | No | None | The name of the TDSQL table. If this option is set, this co can consume the binlog data of the table specified here, p that the subscription task contains the binlog data of this t option supports a regex, such as `test_0*` or `test_1,test_2` . |

**Note**

To use `tdsql.database.name` or `tdsql.table.name` , we recommend you **subscribe to all instances** in the subscription task. If multiple Stream Compute Service tasks consume different TDSQL tables, each task must use a unique consumer group of the subscription task. You can create consumer groups in the subscription task.

## Notes

1. If a subscription task is configured with multiple databases and tables or one database and multiple tables, the tables must be in the same schema to correctly integrate the data of the subscription task.
2. Chinese characters in a source table can be encoded only using `utf8` or `gbk` .

# Redis

Last updated：2023-11-08 14:54:06

## Overview

The Redis connector supports data write to Redis and can be used as a dimension table.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported (write to Redis and use as a dimension table) |
| 1.13 | Supported (write to Redis and use as a dimension table) |
| 1.14 | Supported (write to Redis and use as a dimension table) |
| 1.16 | Supported (write to Redis and use as a dimension table) |

## Use cases

As a dimension table.
As a sink for tuple and upsert streams.

## Defining a table in DDL

**SET command (string key)**

```
-- The first column is "key" and the second column "value". The Redis command is "s
CREATE TABLE `redis_set_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'set',          -- The SET command.
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>',
  'database' = '<database>'
);
```

### LPUSH command (list key)

```
-- The first column is "key" and the second column "value". The Redis command is "I
CREATE TABLE `redis_lpush_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'lpush',        -- The LPUSH command.
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>',
  'database' = '<database>'
);
```

### SADD command (set key)

```
-- The first column is "key" and the second column "value". The Redis command is "s
CREATE TABLE `redis_sadd_sink_table` (
 `key`    STRING,
 `value`  STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'sadd',         -- The SADD command
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>',
  'database' = '<database>'
);
```

### HSET command (hash key)

```
-- The first column is "hash_key" and the second column "hash_value". The Redis com
CREATE TABLE `redis_hset_sink_table` (
 `hash_key`    STRING,
 `hash_value` STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'hset',         -- The HSET command.
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>',
  'database' = '<database>',
  'additional-key' = '<key>' -- The hash key.
);
```

### HSET_WITH_KEY command (hash key)

```
-- The first column is "key", the second column "hash_key", and the third column "h
CREATE TABLE `redis_hset_sink_table` (
`key`         STRING,
`hash_key`    STRING,
`hash_value`  STRING
) WITH (
'connector' = 'redis',      -- The connector (Redis) to use.
'command' = 'hset',         -- The HSET command.
'nodes' = '<host>:<port>', -- The Redis server connection address.
'password' = '<password>',
'database' = '<database>'
);
```

### HMSET command

```
-- The first column is "key", the second column "hash_key", and the third column "h
CREATE TABLE `redis_hmset_sink_table` (
  `key`         STRING,
  `fieldKey`    STRING,
  `fieldValue`  STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'hmset',        -- The HMSET command.
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>'
);
```

### ZADD command (sorted set key)

```
-- The first column is "hash_key" and the second column "hash_value". The Redis com
CREATE TABLE `redis_zadd_sink_table` (
 `value` STRING,
 `score` DOUBLE
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'zadd',         -- The ZADD command.
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>',
  'database' = '<database>',
  'additional-key' = '<key>' -- The sorted set key.
);
```

### ZADD_WITH_KEY command (sorted set key)

```
  -- The first column is "key", the second column "value", and the third column "sc
CREATE TABLE `redis_zadd_sink_table` (
`key` STRING,
`value` STRING,
`score` DOUBLE
) WITH (
'connector' = 'redis',      -- The connector (Redis) to use.
'command' = 'zadd',         -- The ZADD command.
'nodes' = '<host>:<port>', -- The Redis server connection address.
'password' = '<password>',
'database' = '<database>',
);
```

### GET command (dimension table)

```
CREATE TABLE `redis_dimension_table` (
 `key`    STRING,
 `value`  STRING
) WITH (
  'connector' = 'redis',
  'command' = 'get',              -- The GET command.
  'nodes' = '<host>:<port>',      -- The Redis server connection address. For a clus
  'lookup.cache.max-rows' = '500', -- The maximum number of data records allowed in
  'lookup.cache.ttl' = '10 min', -- The maximum cache time of each record.
  'lookup.max-retries' = '5', -- The maximum number of retries after query failure.
  -- 'password' = '<password>',    -- The password (optional).
  -- 'database' = '<database>',    -- The database (optional), which defaults to `0`
  -- 'redis-mode' = 'standalone' -- The Redis mode (optional), which defaults to `s
);
```

## WITH Parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | - | Here, it should be `redis` . |
| command | Yes | - | The operation command. The values and corresponding key types are as follows:<br>`set` : String key<br>`lpush` : List key<br>`sadd` : Set key<br>`hset` : Hash key<br>`hset_with_key` : Hash key<br>`zadd` : Sorted set key<br>`zadd_with_key` : Sorted set key |
| nodes | Yes | - | The Redis server connection address, such as `127.0.0.1:6379` . For cluster architecture, separate nodes by comma. |
| password | No | Null | The Redis password, which defaults to null, meaning that no permission validation is required. |
| database | No | 0 | The DB number of the database to operate, which defaults to `0` . |
| redis-mode | No | standalone | The Redis deployment mode. |

| | | | |
|---|---|---|---|
| | | | `standalone` : Standard architecture<br>`cluster` : Distributed cluster architecture |
| ignore-delete | No | false | Whether to ignore retraction messages. |
| additional-ttl | No | - | The expiration time in seconds. For example, if it is set to `60` , the expiration time is 60 seconds. **Only the SET command allows setting the expiration time.** |
| additional-key | - | - | Specifies the key in the HSET or ZADD command. It is required for running the HSET or ZADD command. |
| lookup.cache.max-rows | No | - | The maximum number of data records allowed in the Lookup Cache. |
| lookup.cache.ttl | No | 10s | The maximum cache time of each record. |
| lookup.max-retries | No | 1 | The maximum number of retries after query failure. |
| lookup.cache.caching-missing-key | No | true | Whether to cache an empty result. |

## Example

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
   'connector' = 'datagen',
   'rows-per-second'='1'  -- The number of data records generated per second.
);

CREATE TABLE `redis_set_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'set',          -- The SET command.
  'nodes' = '127.0.0.1:8121', -- The Redis server connection address.
  'password' = '<password>',
  'database' = '0'
);
insert into redis_set_sink_table select cast(id as string), name from datagen_sourc
```

## Notes

1. A self-built Redis cluster does not support multiple databases; in the cluster mode, the `database` option is invalid.
2. The cluster architecture of TencentDB for Redis supports multiple databases. You can use the `standalone` mode to specify databases (other than the database `0`) whose data will be written to TencentDB for Redis.
Example:

```
CREATE TABLE `redis_set_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- The connector (Redis) to use.
  'command' = 'set',          -- The SET command.
  'nodes' = '<host>:<port>', -- The Redis server connection address.
  'password' = '<password>',
  'redis-mode' = 'standalone',
  'database' = '1'
);
```

# MongoDB CDC

Last updated：2023-11-08 14:54:50

## Overview

The MongoDB CDC source connector automatically tracks MongoDB replica sets or sharded clusters to capture changes in databases and collections.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported |
| 1.14 | Supported |
| 1.16 | Supported |

## Use cases

The MongoDB CDC connector can be used only as a source. It supports MongoDB v4.0, v4.2, and v5.0, and the MongoDB cluster must be a replica set or a sharded cluster.

## Defining a table in DDL

```
-- register a MongoDB table 'products' in Flink SQL
CREATE TABLE mongo_cdc_source_table (
  _id STRING, // must be declared
  name STRING,
  weight DECIMAL(10,3),
  tags ARRAY<STRING>, -- array
  price ROW<amount DECIMAL(10,2), currency STRING>, -- embedded document
  suppliers ARRAY<ROW<name STRING, address STRING>>, -- embedded documents
  PRIMARY KEY(_id) NOT ENFORCED
) WITH (
  'connector' = 'mongodb-cdc',
  'hosts' = 'localhost:27017,localhost:27018,localhost:27019',
  'username' = 'flinkuser',
  'password' = 'flinkpw',
  'database' = 'inventory',
  'collection' = 'products'
);
```

## WITH parameters

| Option | Description | Required | Remarks |
|---|---|---|---|
| connector | The connector to use. | Yes | The value must be 'mysql-cdc |
| hosts | IP and port of the MongoDB database server. | Yes | - |
| username | Name of the database user to use when connecting to MongoDB. | Yes | - |
| password | Password of the database user to use when connecting to MongoDB. | Yes | - |
| database | The name of the MongoDB database to watch for changes. | Yes | - |
| collection | The name of the collection in the MongoDB database to watch for changes. | Yes | - |
| connection.options | The ampersand-separated Connection String Options of MongoDB, such as `relicaSet=test&connectTimeoutMS=300000` . | No | - |
| errors.tolerance | Whether to ignore error records. Valid values: `none` and `all` . If it is set to `all` , all error records will be ignored. | No | none |
| errors.log.enable | Whether to print errors in logs. | No | Default val `true` . |
| copy.existing | Whether to copy the existing data in the database. If changes are made to the data during the copying, they will apply after the copying is completed. | No | Default val `true` . |
| copy.existing.pipeline | This option allows setting filters for copying the existing | No | - |

| | | | |
|---|---|---|---|
| | data. For example, if you set it to `[{"$match": {"closed": "false"}}]` , only records whose value of `closed` is `false` will be copied. For how to use this option, see $match (aggregation). | | |
| copy.existing.max.threads | The number of threads to use when copying data. | No | Default val `Process Count` . |
| copy.existing.queue.size | The maximum size of the queue to use when copying data. | No | Default val `16000` . |
| poll.max.batch.size | The maximum number of change stream documents to include in a single batch when polling for new data. By default, with a check interval of 1.5s, up to 1,000 documents can be included each time. | No | Default val `1000` . |
| poll.await.time.ms | The amount of time to wait before checking for new results on the change stream. By default, with a check interval of 1.5s, up to 1,000 documents can be included each time. | No | Default val `1500` . |
| heartbeat.interval.ms | The length of time in milliseconds between sending heartbeat messages. Set it to `0` to disable the feature. | No | Default val `0` . |

**Note**
If data streams change slowly, we recommend you set `heartbeat.interval.ms` to an appropriate value. A `resumeToken` is included in a heartbeat message to avoid the use of an expired `resumeToken` when a Flink job resumes from checkpoint or savepoint.

## Data type mapping

| MongoDB Type | Flink Type |
|---|---|
| - | TINYINT |
| - | SMALLINT |
| Int | INT |
| Long | BIGINT |
| - | FLOAT |

| | |
|---|---|
| Double | DOUBLE |
| Decimal128 | DECIMAL(p, s) |
| Boolean | BOOLEAN |
| DateTimestamp | DATE |
| DateTimestamp | TIME |
| Date | TIMESTAMP(3) TIMESTAMP_LTZ(3) |
| Timestamp | TIMESTAMP(0) TIMESTAMP_LTZ(0) |
| String<br>ObjectId<br>UUID<br>Symbol<br>MD5<br>JavaScript<br>Regex | STRING |
| BinData | BYTES |
| Object | ROW |
| Array | ARRAY |
| DBPointer | ROW<$ref STRING, $id STRING> |
| GeoJSON | Point : ROW<type STRING, coordinates ARRAY<DOUBLE>><br>Line : ROW<type STRING, coordinates ARRAY<ARRAY< DOUBLE>>><br>... |

## Example

```
CREATE TABLE mongo_cdc_source_table (
  _id STRING, // must be declared
  name STRING,
  weight DECIMAL(10,3),
  tags ARRAY<STRING>, -- array
  price ROW<amount DECIMAL(10,2), currency STRING>, -- embedded document
  suppliers ARRAY<ROW<name STRING, address STRING>>, -- embedded documents
  PRIMARY KEY(_id) NOT ENFORCED
) WITH (
  'connector' = 'mongodb-cdc',
  'hosts' = 'localhost:27017,localhost:27018,localhost:27019',
  'username' = 'flinkuser',
  'password' = 'flinkpw',
  'database' = 'inventory',
  'collection' = 'products'
);
CREATE TABLE `print_table` (
  `id` STRING,
  `name` STRING,
  `currency` STRING
) WITH (
 'connector' = 'print'
);
insert into print_table select _id, name, price.currency from mongo_cdc_source_tabl
```

## Notes

### User permissions

The user of the MongoDB database must have the `changeStream` and `read` permissions.

```
use admin;
db.createUser(
 {
   user: "flinkuser",
   pwd: "flinkpw",
   roles: [
      { role: "read", db: "admin" },
      { role: "readAnyDatabase", db: "admin" }
   ]
 }
);
```

## Parallelism

The task parallelism must be 1.

# MongoDB

Last updated：2023-11-08 14:51:17

## Overview

The Flink connector mongodb allows batch data writing from Flink to MongoDB. It currently supports only append (tuple) streams.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Supported |

## Defining a table in DDL

```
CREATE TABLE mongodb (
  user_id INT,
  item_id INT,
  category_id INT,
  behavior VARCHAR
) WITH (
  'connector' = 'mongodb', -- Here, it should be 'mongodb'.
  'database' = 'test', -- The name of the database.
  'collection' = 'table1',-- The data collection.
  'uri' = 'mongodb://$username:$password@$IP:$PORT,$IP:$PORT,$IP:$PORT/test?authSou
  'batchSize' = '1024' -- The number of records written per batch.
);
```

## Limits

The Flink connector mongodb can be used as a MongoDB sink only. It supports using a TencentDB for MongoDB table as a result table.

## WITH parameters

| Option | Description | Required | Remarks |
|---|---|---|---|
| connector | The result table type. | Yes | Here, it should be `mongodb` . |
| database | The name of the database. | Yes | - |
| collection | The data collection. | Yes | - |
| uri | The MongoDB connection string. | Yes | - |
| batchSize | The number of records written per batch. | No | Default value: 1024. |
| maxConnectionIdleTime | The connection timeout period. | No | Default value: 60000 (ms). |

## Example

```
CREATE TABLE random_source (
  user_id INT,
  item_id INT,
  category_id INT,
  behavior VARCHAR
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100', -- The number of records generated per second.
  'fields.user_id.kind' = 'sequence', -- Whether a bounded sequence (if yes, the ou
  'fields.user_id.start' = '1', -- The start value of the sequence.
  'fields.user_id.end' = '10000', -- The end value of the sequence.
  'fields.item_id.kind' = 'random', -- A random number without range.
  'fields.item_id.min' = '1', -- The minimum random number.
  'fields.item_id.max' = '1000', -- The maximum random number.
  'fields.category_id.kind' = 'random', -- A random number without range.
  'fields.category_id.min' = '1', -- The minimum random number.
  'fields.category_id.max' = '1000', -- The maximum random number.
  'fields.behavior.length' = '5' -- The random string length.
);

CREATE TABLE mongodb (
  user_id INT,
  item_id INT,
  category_id INT,
  behavior VARCHAR
) WITH (
  'connector' = 'mongodb', -- Here, it should be 'mongodb'.
  'database' = 'test', -- The name of the database.
  'collection' = 'table1',-- The data collection.
  'uri' = 'mongodb://$username:$password@$IP:$PORT,$IP:$PORT,$IP:$PORT/test?authSou
  'batchSize' = '1024' -- The number of records written per batch.
);

insert into mongodb select * from random_source;
```

## Notes

### Upsert

The Flink connector mongodb as a sink does not support upsert streams.

### User permissions

The user of the MongoDB database must have the permission to write data to the database.

# PostgreSQL CDC

Last updated：2023-11-08 14:51:44

## Overview

The Postgres CDC source connector allows for reading snapshot data and incremental data from PostgreSQL databases. It can read the data with exactly-once processing even failures occur.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Supported |

## Limits

The Postgres CDC connector can be used only as a source. It supports PostgreSQL v9.6 or later.

## Defining a table in DDL

```
CREATE TABLE postgres_cdc_source_table (
  id INT,
  name STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- Define the primary key here if the database ta
) WITH (
  'connector' = 'postgres-cdc',              -- Here, it should be'postgres-cdc'.
  'hostname' = 'yourHostname',               -- IP of the database server.
  'port' = '5432',                 --  Integer port number of the database server.
  'username' = 'yourUserName',               Name of the PostgreSQL database to use
  'password' = 'psw'  -- Password to use when connecting to the PostgreSQL database
  'database-name' = 'yourDatabaseName',      -- Database name of the PostgreSQL serv
```

```
    'schema-name' = 'yourSchemaName',          -- Schema name of the PostgreSQL databa
    'table-name' = 'yourTableName',            -- Table name of the PostgreSQL databas
    'debezium.slot.name' = 'customslotname'    -- Define a unique slot name, which sup
  );
```

## WITH parameters

| Option | Description | Required | Remarks |
|---|---|---|---|
| connector | The connector to use. | Yes | Here, it should be `postgres-cdc` . |
| hostname | IP address or hostname of the PostgreSQL database server. | Yes | - |
| username | Name of the PostgreSQL database to use when connecting to the PostgreSQL database server. | Yes | The user must have specific permissions (REPLICATION, LOGIN, SCHEMA, DATABASE, and SELECT). |
| password | Password to use when connecting to the PostgreSQL database server. | Yes | - |
| database-name | Database name of the PostgreSQL server to monitor. | Yes | - |
| schema-name | Schema name of the PostgreSQL database to monitor. | Yes | The schema-name supports regular expressions to monitor multiple schemas matching the regular expression. |
| table-name | Table name of the PostgreSQL | Yes | The table-name supports regular expressions to monitor multiple tables matching the regular |

| | | | |
|---|---|---|---|
| | database to monitor. | | expression. |
| port | Integer port number of the PostgreSQL database server. | No | Default value: `5432` . |
| decoding.plugin.name | The name of the Postgres logical decoding plugin. | No | It depends on the Postgres logical decoding plugin installed on the server. Supported values are as follows:<br>decoderbufs (default)<br>wal2json<br>wal2json_rds<br>wal2json_streaming<br>wal2json_rds_streaming<br>pgoutput |
| debezium.* | Debezium properties. | No | Specifies Debezium properties for fine-grained control of the behaviors on the client, such as `'debezium.slot.name' = 'xxxx'` , to prevent `PSQLException: ERROR: replication slot "dl_test" is active for PID 19997` . For details, see Connector configuration properties. |

## Data type mapping

Postgres CDC and Flink data types are mapped as follows:

| Postgres CDC Type | Flink Type |
|---|---|
| SMALLINT | SMALLINT |
| INT2 | |
| SMALLSERIAL | |
| SERIAL2 | |
| INTEGER | INT |
| SERIAL | |
| BIGINT | BIGINT |
| | |

| | |
|---|---|
| BIGSERIAL | |
| REAL | FLOAT |
| FLOAT4 | |
| FLOAT8 | DOUBLE |
| DOUBLE PRECISION | |
| NUMERIC(p, s) | DECIMAL(p, s) |
| DECIMAL(p, s) | |
| BOOLEAN | BOOLEAN |
| DATE | DATE |
| TIME [(p)] [WITHOUT TIMEZONE] | TIME [(p)] [WITHOUT TIMEZONE] |
| TIMESTAMP [(p)] [WITHOUT TIMEZONE] | TIMESTAMP [(p)] [WITHOUT TIMEZONE] |
| CHAR(n) | STRING |
| CHARACTER(n) | |
| VARCHAR(n) | |
| CHARACTER VARYING(n) | |
| TEXT | |
| BYTEA | BYTES |

## Example

```
CREATE TABLE postgres_cdc_source_table (
  id INT,
  name STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- Define the primary key here if the database ta
) WITH (
  'connector' = 'postgres-cdc',              -- Here, it should be'postgres-cdc'.
  'hostname' = 'yourHostname',               -- IP of the database server.
  'port' = '5432',                 --  Integer port number of the database server.
  'username' = 'yourUserName',               Name of the PostgreSQL database to use
  'password' = 'psw'  -- Password to use when connecting to the PostgreSQL database
  'database-name' = 'yourDatabaseName',      -- Database name of the PostgreSQL serv
```

```
    'schema-name' = 'yourSchemaName',          -- Schema name of the PostgreSQL databa
    'table-name' = 'yourTableName',            -- Table name of the PostgreSQL databas
    'debezium.slot.name' = 'customslotname'    -- Define a unique slot name, which sup
);

CREATE TABLE `print_table` (
    `id` INT,
    `name` STRING
) WITH (
  'connector' = 'print'
);
insert into print_table select * from postgres_cdc_source_table;
```

## Notes

### User permissions

The user must at least have the following permissions: REPLICATION, LOGIN, SCHEMA, DATABASE, and SELECT.

```
CREATE ROLE debezium_user REPLICATION LOGIN;
GRANT USAGE ON SCHEMA schema_name TO debezium_user;
GRANT USAGE ON DATABASE schema_name TO debezium_user;
GRANT SELECT ON scheam_name.table_name, scheam_name.table_name TO debezium_user;
```

# HBase

Last updated：2023-11-08 14:53:05

## Overview

The HBase connector supports read and write of an HBase cluster. Stream Compute Service provides the components of the built-in `flink-connector-hbase` connector. For details, see Associating an HBase Dimension Table with ClickHouse Using MySQL.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | HBase v1.4.x supported. |
| 1.13 | HBase v1.4.x, v2.2.x, and 2.3.x supported. |
| 1.14 | HBase v1.4.x and 2.2.x supported. |
| 1.16 | HBase v1.4.x and v2.2.x supported. |

## User cases

The HBase connector can be used as a source or a dimension table, and as a sink of tuple and upsert streams.

## Defining a table in DDL

```
CREATE TABLE hbase_table (
  rowkey INT,
  cf ROW < school_name STRING >,
  PRIMARY KEY (rowkey) NOT ENFORCED
) WITH (
  'connector' = 'hbase-1.4',                          -- Flink v1.13 supports HBase
  'table-name' = 'hbase_sink_table',                         -- The name of the HBa
  'zookeeper.quorum' = 'ip:port,ip:port,ip:port'   -- The ZooKeeper address of the
);
```

## WITH parameters

| Option | Description | Required | Remarks |
|---|---|---|---|
| connector | The table to use. | Yes | `hbase-1.4` or `hbase-2.2` . If HBase 2.3.x is used, `connector` should be set to `hbase-2.2` . |
| table-name | The name of the HBase table. | Yes | - |
| zookeeper.quorum | The ZooKeeper address of the HBase table. | Yes | - |
| zookeeper.znode.parent | The root directory of the HBase table in ZooKeeper. | No | - |
| null-string-literal | When the type of an HBase field is string and the value of the Flink field is null, `null-string-literal` is assigned to this field and written to the HBase table. | No | Default value: `null` . |
| sink.buffer-flush.max-size | The size of data (in bytes) to be cached in the memory before write to HBase. A larger value of this option can help improve the HBase write performance, but the write latency and the memory required will increase, too. **This option applies only when the HBase connector is used as a sink.** | No | This option defaults to 2 MB, and supported units include B, KB, MB, and GB (case-insensitive). Setting it to `0` means no data will be cached. |
| sink.buffer-flush.max-rows | The number of records to be cached in the memory before write to HBase. A larger value of this option can help improve the HBase write performance, but the write latency and the memory required will increase, too. **This option applies only when the HBase connector is used as a sink.** | No | This option defaults to `1000` , and setting it to `0` means no data will be cached. |
| sink.buffer-flush.interval | The interval for writing cached data | No | This option defaults to 1s, |

| | | | |
|---|---|---|---|
| | to HBase. This option allows controlling the delay of data write to HBase. **This option applies only when the HBase connector is used as a sink.** | | and supported units include ms, s, min, h, and d. Setting it to `0` means periodic write is disabled. |

## Data type mapping

HBase stores everything as bytes, so the data needs to be serialized or deserialized during read and write. The Flink and HBase data types are mapped as follows:

| Flink Type | HBase Type |
|---|---|
| CHAR / VARCHAR / STRING | byte[] toBytes(String s) String toString(byte[] b) |
| BOOLEAN | byte[] toBytes(boolean b) boolean toBoolean(byte[] b) |
| BINARY / VARBINARY | byte[] |
| DECIMAL | byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b) |
| TINYINT | new byte[] { val } bytes[0] |
| SMALLINT | byte[] toBytes(short val) short toShort(byte[] bytes) |
| INT | byte[] toBytes(int val) int toInt(byte[] bytes) |
| BIGINT | byte[] toBytes(long val) long toLong(byte[] bytes) |
| FLOAT | byte[] toBytes(float val) float toFloat(byte[] bytes) |
| DOUBLE | byte[] toBytes(double val) double toDouble(byte[] bytes) |
| DATE | Converts the date into the number of days since January 1, 1970, expressed in int, and then converts it into byte using `byte[] toBytes(int val)` . |
| TIME | Converts the time into the number of milliseconds since 00:00:00, expressed in int, and then converts it into byte using `byte[] toBytes(int val)` . |
| TIMESTAMP | Converts the timestamp into the number of milliseconds since 00:00:00 on January 1, 1970, expressed in long, and then converts it into byte using `byte[] toBytes(long val)` . |
| ARRAY | Unsupported |
| | |

| | |
|---|---|
| MAP / MULTISET | Unsupported |
| ROW | Unsupported |

## Example

The following example shows a block of code of a real-time computing job containing an HBase dimension table.

```
CREATE TABLE datagen_source_table (
  id INT,
  name STRING,
  `proc_time` AS PROCTIME()
) with (
  'connector'='datagen',
  'rows-per-second'='1'
);

CREATE TABLE hbase_table (
  rowkey INT,
  cf ROW < school_name STRING >,
  PRIMARY KEY (rowkey) NOT ENFORCED
) WITH (
  'connector' = 'hbase-1.4',                            -- Flink v1.13 supports HBase
  'table-name' = 'hbase_sink_table',                           -- The name of the HBa
  'zookeeper.quorum' = 'ip:port,ip:port,ip:port'   -- The ZooKeeper address of the
);

CREATE TABLE blackhole_sink(
  id INT,
  name STRING
) with (
  'connector' = 'blackhole'
);

INSERT INTO blackhole_sink
     SELECT id, cf.school_name as name FROM datagen_source_table src
JOIN hbase_table FOR SYSTEM_TIME AS OF src.`proc_time` as h ON src.id = h.rowkey;
```

## Notes

The HBase connector generally uses the primary key defined in the DDL statements and works in the `upsert` mode to exchange change logs with the external systems. Therefore, the primary key must be defined in the `rowkey` field (this field must be declared) of the HBase connector. If the PRIMARY KEY clause is not declared, the HBase connector defaults the primary key to `rowkey`.

# Hive

Last updated：2023-11-08 14:50:37

## Overview

The Hive connector can be used as a sink of streams, but it only supports append streams, but not upsert streams. Supported data formats include Text, SequenceFile, ORC, and Parquet.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Hive v1.1.0, v2.3.2, v2.3.5, and v3.1.1 supported<br>Option 'connector.type' = 'hive' |
| 1.13 | Hive v1.0.0 - v1.2.2, v2.0.0 - v2.2.0, v2.3.0 - v2.3.6, and v3.0.0 - 3.1.2 supported<br>Option 'connector.type' = 'hive' |
| 1.14 | Unsupported |
| 1.16 | Hive v2.0.0 - v2.2.0, v2.3.0 - v2.3.6, and v3.0.0 - 3.1.2 supported<br>Option 'connector.type' = 'hive' |

## Defining a table in DDL

**As a sink**

```
CREATE TABLE hive_table (
  `id` INT,
  `name` STRING,
  `dt` STRING,
  `hr` STRING
) PARTITIONED BY (dt, hr)
with (
    'connector' = 'hive',  -- For Flink v1.13, set 'connector' to 'hive'.
    'hive-version' = '3.1.1',
    'hive-database' = 'testdb',
    'partition.time-extractor.timestamp-pattern'='$dt $hr:00:00',
    'sink.partition-commit.trigger'='partition-time',
    'sink.partition-commit.delay'='1 h',
    'sink.partition-commit.policy.kind'='metastore,success-file'
);
```

## Job configurations

Create a Hive table in a Hive database.

```
# Create the table "hive_table" in the Hive database "testdb".
USE testdb;
CREATE TABLE `hive_table` (
  `id` int,
  `name` string)
PARTITIONED BY (`dt` string, `hr` string)
STORED AS ORC;
```

Grant the write access to the HDFS path of the Hive table with one of the following methods.

Method 1: Log in to the EMR Hive cluster as instructed in Basic Hive Operations, and execute the chmod command on `hive_table table` of the target database `testdb` .

```
hdfs dfs -chmod 777 /usr/hive/warehouse/testdb.db/hive_table
```

Method 2: Go to **Jobs > Job parameters** of the target job in the console, and add the following advanced parameters to gain access to the HDFS path using the Hadoop user role.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

**Note**

The Hive table used in the Flink SQL statements is `testdb.hive_table`. Here in the CREATE TABLE statements, the table name used is that in the Hive database (Flink v1.13 supports using the value of the `hive-table` option to overwrite this value), and the database name is specified using the `hive-database` option.

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector.type | Yes | None | Available to Flink v1.11. To use the Hive connector, set it to `hive`. |
| connector | Yes | None | Available to Flink v1.11. To use the Hive connector, set it to `hive`. |
| hive-version | Yes | None | The version of the Hive cluster created in the EMR console. |
| hive-database | Yes | None | The Hive database to write data to. |
| hive-table | No | None | Available to Flink v1.13. Its value is used as the table name of the Hive database. |
| sink.partition-commit.trigger | No | process-time | The partition commit trigger. Valid values:<br>`process-time`: A partition will be committed when a certain time elapses after the partition creation time. The creation time refers to the physical creation time.<br>`partition-time`: A partition will be committed when a certain time elapses after the partition creation time. Here, the creation time is extracted from partition values.<br>`partition-time` requires watermark generation to automatically detect partitions. A partition is committed once `watermark` passes the time extracted from partition values plus `delay`. |
| sink.partition-commit.delay | No | 0s | The time to wait before closing a partition. A partition will not be committed until the specified time elapses after its creation time. |
| sink.partition-commit.policy.kind | Yes | None | The partition committing policy. Valid values (combinations are allowed):<br>`success-file`: When a partition is committed, a `_success` file will be generated in the partition's directory.<br>`metastore`: Add the partition to the Hive Metastore.<br>`custom`: A user-defined partition committing policy. |
| partition.time-extractor.timestamp-pattern | No | None | The partition timestamp extraction format. The timestamps should be `yyyy-mm-dd hh:mm:ss`, with the placeholders replaced with the corresponding partition fields |

| | | | |
|---|---|---|---|
| | | | in the Hive table. By default, `yyyy-mm-dd hh:mm:ss` is extracted from the first field.<br>If timestamps are extracted from a single partition field `dt` , you can set this to `$dt` .<br>If timestamps are extracted from multiple partition fields, such as `year` , `month` , `day` , and `hour` , you can set this to `$year-$month-$day $hour:00:00` .<br>If timestamps are extracted from two partition fields `dt` and `hour` , you can set this to `$dt $hour:00:00` . |
| sink.partition-commit.policy.class | No | None | The partition committing policy class, which needs to be used together with `sink.partition-commit.policy.kind = 'custom'` and must implement `PartitionCommitPolicy` . |
| partition.time-extractor.kind | No | default | The partition time extractor, which applies only when `sink.partition-commit.trigger` is set to `partition-time` . If you have your own time extractor, set this to `custom` . |
| partition.time-extractor.class | No | None | The partition time extractor class. This class should implement the `PartitionTimeExtractor` API. |

## Example

```
CREATE TABLE datagen_source_table (
  id INT,
  name STRING,
  log_ts TIMESTAMP(3),
  WATERMARK FOR log_ts AS log_ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '10'
);

CREATE TABLE hive_table (
  `id` INT,
  `name` STRING,
  `dt` STRING,
  `hr` STRING
) PARTITIONED BY (dt, hr)
with (
    'connector' = 'hive',  -- For Flink v1.13, set 'connector' to 'hive'.
    'hive-version' = '3.1.1',
    'hive-database' = 'testdb',
    'partition.time-extractor.timestamp-pattern'='$dt $hr:00:00',
    'sink.partition-commit.trigger'='partition-time',
    'sink.partition-commit.delay'='1 h',
    'sink.partition-commit.policy.kind'='metastore,success-file'
);

-- streaming sql, insert into hive table
INSERT INTO hive_table
SELECT id, name, DATE_FORMAT(log_ts, 'yyyy-MM-dd'), DATE_FORMAT(log_ts, 'HH')
FROM datagen_source_table;
```

## Hive configurations

### Getting the Hive connection JAR package

To write data to Hive in a Flink SQL task, a JAR package containing Hive and HDFS configurations is required to connect Flink to the target Hive cluster. The steps to get the JAR package and to use it are as follows:

1. Log in to the respective Hive cluster using SSH.
2. Get `hive-site.xml` and `hdfs-site.xml` from the following paths in the EMR Hive cluster.

```
/usr/local/service/hive/conf/hive-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

3. Modifies `hive-site.xml` .

```
Add the following in "hive-site", and set "ip" to the value of "hive.server2.thrift
<property>
 <name>hive.metastore.uris</name>
 <value>thrift://ip:7004</value>
</property>
```

4. Download hivemetastore-site.xml and hiveserver2-site.xml.
5. Package the obtained configuration files in a JAR package.

```
jar -cvf hive-xxx.jar hive-site.xml hdfs-site.xml hivemetastore-site.xml hiveserver
```

6. Check the JAR structure (run the Vi command `vi hive-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
hive-site.xml
hdfs-site.xml
hivemetastore-site.xml
hiveserver2-site.xml
```

### Using a JAR package in a task

Select the Hive connection JAR package as the referenced package. This JAR package is `hive-xxx.jar` obtained in Getting the Hive connection JAR package and must be uploaded in **Dependencies** before use.

## Kerberos authentication

1. Log in to the cluster master node to get the files `krb5.conf` , `emr.keytab` , `core-site.xml` , `hdfs-site.xml` , and `hive-site.xml` in the following paths.

```
/etc/krb5.conf
```

```
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
/usr/local/service/hive/conf/hive-site.xml
```

2. Modify `hive-site.xml`. Add the following in "hive-site", and set "ip" to the value of "hive.server2.thrift.bind.host".

```
<property>
 <name>hive.metastore.uris</name>
 <value>thrift://ip:7004</value>
```

```
</property>
```

3. Download hivemetastore-site.xml and hiveserver2-site.xml.
4. Package the obtained configuration files in a JAR package.

```
jar cvf hive-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml hive-site.xml
```

5. Check the JAR structure (run the Vim command `vim hdfs-xxx.jar` ). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
hive-site.xml
hivemetastore-site.xml
hiveserver2-site.xml
```

6. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
7. Get Kerberos principals to configure advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as follows (use the first): hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- ------------------------------------------------------
  2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
```

```
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

8. Configure advanced job parameters.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: ${krb5.conf.fileName}
```

**Note**

Kerberos authentication is not supported for historical Stream Compute Service clusters. To support the feature, please contact us to upgrade the cluster management service.

## Notes

If the Flink job runs properly and no errors are reported in the logs, but this Hive table cannot be found in the client, fix the table with the following command (replace `hive_table_xxx` with the name of the table to be fixed).

```
msck repair table hive_table_xxx;
```

# ClickHouse

Last updated：2023-11-08 16:16:02

## Overview

The ClickHouse sink connector supports data write to ClickHouse data warehouses. The ClickHouse source connector enables ClickHouse to be used as a batch source and a dimension table.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported (use as sink) |
| 1.13 | Supported (use as source and sink) |
| 1.14 | Supported (use as source and sink) |
| 1.16 | Supported (use as source and sink) |

## ## Limits

The ClickHouse connector does not support standard UPDATE and DELETE operations. For a ClickHouse sink connector, if you need to perform UPDATE and DELETE operations, see CollapsingMergeTree.
For JAR jobs written in Java/Scala, data can be written to ClickHouse using JDBC, which will not be elaborated on here.

## Defining a table in DDL

### As a sink with insert only

```
CREATE TABLE clickhouse_sink_table (
   `id` INT,
   `name` STRING
) WITH (
   -- Specify the parameters for database connection.
   'connector' = 'clickhouse',                      -- Specify the connector to us
   'url' = 'clickhouse://172.28.28.160:8123',       -- Specify the cluster address
   -- You don't need to specify it if the ClickHouse cluster is not configured with
   --'username' = 'root',                           -- The ClickHouse cluster user
   --'password' = 'root',                           -- The ClickHouse cluster pass
   'database-name' = 'db',                          -- The database to write data
```

```
    'table-name' = 'table',                                -- The table to write data to.
    'sink.batch-size' = '1000'                             -- The number of data records 
);
```

### As a sink with upsert

```
CREATE TABLE clickhouse_upsert_sink_table (
    `id` INT,
    `name` STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- Define the primary key here if the database ta
) WITH (
    -- Specify the parameters for database connection.
    'connector' = 'clickhouse',                         -- Specify the connector to use.
    'url' = 'clickhouse://172.28.28.160:8123',          -- Specify the cluster address,
    -- You don't need to specify it if the ClickHouse cluster is not configured with
    --'username' = 'root',                              -- The ClickHouse cluster userna
    --'password' = 'root',                              -- The ClickHouse cluster passwo
    'database-name' = 'db',                             -- The database to write data to
    'table-name' = 'table',                             -- The table to write data to.
    'table.collapsing.field' = 'Sign',                  -- The name of the CollapsingMer
    'sink.batch-size' = '1000'                          -- The number of data records th
);
```

**Note**

You must define a primary key and declare the `table.collapsing.field` field to support upsert. For the ClickHouse table creation statement, see FAQs.

## As a batch source

```
CREATE TABLE `clickhouse_batch_source` (
    `when`      TIMESTAMP,
    `userid`    BIGINT,
    `bytes`     FLOAT
) WITH (
    'connector' = 'clickhouse',
    'url' = 'clickhouse://172.28.1.21:8123',
    'database-name' = 'dts',
    'table-name' = 'download_dist'
--    'scan.by-part.enabled' = 'false', -- Whether to read the ClickHouse table by
--    'scan.part.modification-time.lower-bound' = '2021-09-24 16:00:00', -- The min
--      'scan.part.modification-time.upper-bound' = '2021-09-17 19:16:26', -- The ma
--     'local.read-write' = 'false', -- Whether to read the local table. Default val
--     'table.local-nodes' = '172.28.1.24:8123,172.28.1.226:8123,172.28.1.109:8123,1
);
```

**Note**

Only MergeTree engines support reading by part, and you must stop backend merge and TTL-based data deletions on all nodes with the table being read to avoid inaccurate data reading caused by part changes. Only one replica node address can be configured for each shard for local reading; otherwise, duplicate data may be read.

### As a dimension table

```
CREATE TABLE `clickhouse_dimension` (
    `userid` BIGINT,
    `comment` STRING
) WITH (
    'connector' = 'clickhouse',
    'url' = 'clickhouse://172.28.1.21:8123',
    'database-name' = 'dimension',
    'table-name' = 'download_dist',
    'lookup.cache.max-rows' = '500', -- The maximum number of data records allowed
    'lookup.cache.ttl' = '10min', -- The maximum cache time of each record.
    'lookup.max-retries' = '10' -- The maximum number of retries upon database quer
);
```

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | - | Required if ClickHouse is to be used as the data sink. Here, it should be `clickhouse` . |
| url | Yes | - | The ClickHouse cluster connection URL, such as 'clickhouse://127.1.1.1:8123', which can be viewed on the cluster page. |
| name-server | No | - | The ClickHouse cluster username. |
| password | No | - | The ClickHouse cluster password. |
| database-name | Yes | - | The ClickHouse cluster database. |
| table-name | Yes | - | The ClickHouse cluster table. |
| sink.batch-size | No | 1000 | The number of records to flush to the connector. |
| sink.flush-interval | No | 1000 (unit: ms) | The interval for async threads to flush data to the ClickHouse connector. |
| table.collapsing.field | No | - | The name of the CollapsingMergeTree type column field. |
| sink.max-retries | No | 3 | The number of retries upon write failure. |
| local.read-write | No | false | Whether to enable the feature of writing to the local table. The default value is `false` .<br>Note: This feature is available only to advanced users and must be used together with the parameters described in the "Writing to local table" section of this document. |
| table.local-nodes | No | - | The list of local nodes when `local.read-write` is set to `true` . Example: '127.1.1.10:8123,127.1.2.13:8123' (**the HTTP port number is required**) |
| sink.partition-strategy | No | balanced | The partition strategy when `local.read-write` |

| | | | |
|---|---|---|---|
| | | | is set to `true` . If you want to achieve dynamic updates and the table engine is CollapsingMergeTree, the value must be `hash` , and `sink.partition-key` must be set.<br>Valid values: `balanced` (round-robin), `shuffle` (random), and `hash` (by `sink.partition-key` ). |
| sink.partition-key | No | - | Required when `local.read-write` is set to `true` and `sink.partition-strategy` to `hash` . The value is the primary key defined in the table. If the primary key consists of multiple fields, you need to specify the value as the first field. |
| sink.ignore-delete | No | false | Whether to ignore all DELETE messages that are written to ClickHouse. This option is suitable for scenarios where the ReplacingMergeTree table engine is used and dynamic data updates are expected. |
| sink.backpressure-aware | No | false | If the "Too many parts" error message frequently appears in Flink logs, and this causes job crashes, you can enable this option to significantly reduce the server load and improve overall throughput and stability. |
| sink.reduce-batch-by-key | No | false | Whether to merge data with the same key by retaining only the last record for ClickHouse sink tables that have defined primary keys, within the given flush interval. |
| sink.max-partitions-per-insert | No | 20 | When ClickHouse is a partitioned table and the partitioning function in ClickHouse is intHash32, toYYYYMM, or toYYYYMMDD, Flink writes to ClickHouse by buffering data on the sink side based on partitions. When the number of accumulated partitions reaches the configured value, it triggers the write to downstream ClickHouse (if `sink.flush-interval` or `sink.batch-size` is reached first, they will trigger the write first). This greatly improves the throughput efficiency of writing to ClickHouse. Setting this option to -1 will disable the aggregation writing feature for a partitioned table. |
| scan.fetch-size | No | 100 | The number of rows obtained in batches each time the database is read. |
| scan.by-part.enabled | No | false | Whether to read the ClickHouse table by part. If this is |

| | | | |
|---|---|---|---|
| | | | enabled, you must first stop the backend merges and TTL-based data deletions of the table using the commands 'STOP MERGES' and 'STOP TTL MERGES' on all nodes; otherwise, an error will occur in data reading. |
| scan.part.modification-time.lower-bound | No | - | The minimum `modification_time` (inclusive) for filtering ClickHouse table parts, in the format of `yyyy-MM-dd HH:mm:ss` . |
| scan.part.modification-time.upper-bound | No | - | The maximum `modification_time` (exclusive) for filtering ClickHouse table parts, in the format of `yyyy-MM-dd HH:mm:ss` . |
| lookup.cache.max-rows | No | - | The maximum number of data records cached in the Lookup Cache. |
| lookup.cache.ttl | No | - | The maximum cache time of each data record in the Lookup Cache. |
| lookup.max-retries | No | 3 | The maximum number of retries upon database lookup failure. |

**Note**
When defining `WITH` parameters, you usually only need to specify the required ones. When you enable optional parameters, be sure to understand their meanings and how they may affect data writing.

## Data type mapping

For the data types supported by ClickHouse, see ClickHouse Data Types. The table below lists some of the common data types and their counterparts in Flink. We recommend you map Flink data types to ClickHouse data types according to the table below to avoid unexpected results. Pay special attention to the following:
DateTime: ClickHouse supports a time precision of 1 second and its default configuration of date_time_input_format is `basic` . So it can only parse time in the format of `YYYY-MM-DD HH:MM:SS` or `YYYY-MM-DD` . If your job encounters time parsing exceptions (e.g., `java.sql.SQLException: Code: 6. DB::Exception: Cannot parse string '2023-05-24 14:34:55.166' as DateTime`), refer to the table below to change the corresponding data type in Flink to `TIMESTAMP(0)` , or change the value of date_time_input_format of the ClickHouse cluster to `best_effort` . In addition, ClickHouse supports inserting DateTime data in integer format, so you can also map the type as INTEGER in Flink, but this is not recommended.
DateTime64: ClickHouse supports inserting DateTime64 data in integer and DECIMAL formats, so you may choose to map it as BIGINT or DECIMAL in Flink. If you map it as BIGINT in Flink, ensure that the BIGINT value to write

matches the precision of DateTime64. For example, the default precision for DateTime64 is milliseconds, that is, DateTime64(3) during table creation in ClickHouse, so the BIGINT value to write should also be in milliseconds, for example, 1672542671000 ( `2023-01-01 11:11:11.000` ). To avoid issues, it is recommended to map it as the TIMESTAMP data type according to the table below.

| ClickHouse Type | Flink Type | Java Typ |
|---|---|---|
| String | VARCHAR/STRING | String |
| FixedString(N) | VARCHAR/STRING | String |
| Bool | BOOLEAN | Byte |
| Int8 | TINYINT | Byte |
| UInt8 | SMALLINT | Short |
| Int16 | SMALLINT | Short |
| UInt16 | INTEGER | Integer |
| Int32 | INTEGER | Integer |
| UInt32 | BIGINT | Long |
| Int64 | BIGINT | Long |
| UInt64 | BIGINT | Long |
| Int128 | DECIMAL | BigInteg |
| UInt128 | DECIMAL | BigInteg |
| Int256 | DECIMAL | BigInteg |
| UInt256 | DECIMAL | BigInteg |
| Float32 | FLOAT | Float |
| Float64 | DOUBLE | Double |
| Decimal(P,S)/Decimal32(S)/Decimal64(S)/Decimal128(S)/Decimal256(S) | DECIMAL | BigDecir |
| Date | DATE | LocalDa |
| DateTime([timezone]) | TIMESTAMP(0) | LocalDa |
| DateTime64(precision, [timezone]) | TIMESTAMP(precision) | LocalDa |
| | | |

| | | |
|---|---|---|
| Array(T) | ARRAY<T> | T[] |
| Map(K, V) | MAP<K, V> | Map<?, |
| Tuple(T1, T2, ...) | ROW<f1 T1, f2 T2, ...> | List<Obj |

## Example

```
CREATE TABLE datagen_source_table (
     id INT,
     name STRING
) WITH (
   'connector' = 'datagen',
   'rows-per-second'='1'  -- The number of data records generated per second.
);
CREATE TABLE clickhouse_sink_table (
   `id` INT,
   `name` STRING
) WITH (
   -- Specify the parameters for database connection.
   'connector' = 'clickhouse',                       -- Specify the use of the Clic
   'url' = 'clickhouse://172.28.28.160:8123',        -- Specify the cluster address
   -- You don't need to specify it if the ClickHouse cluster is not configured with
   --'username' = 'root',                            -- The ClickHouse cluster user
   --'password' = 'root',                            -- The ClickHouse cluster pass
   'database-name' = 'db',                           -- The database to write data
   'table-name' = 'table',                           -- The table to write data to.
   'sink.batch-size' = '1000'                        -- The number of data records
);
insert into clickhouse_sink_table select * from datagen_source_table;
```

## FAQs

### Data upsert and delete

ClickHouse does not support upsert. So for scenarios that require dynamic data updates and deletions, ReplacingMergeTree or CollapsingMergeTree is usually used to simulate update or delete operations. These table engines are suitable for different scenarios.

#### ReplacingMergeTree table engine

If the ClickHouse table engine is ReplacingMergeTree, you can set `sink.ignore-delete` to `true` so that Flink will automatically ignore DELETE messages and convert INSERT and UPDATE_AFTER messages into INSERT messages. Then, ClickHouse will automatically use the latest record to overwrite the previous record with the same primary key, achieving data updates.
Note that this mode only supports data insert and update operations, but **not delete operations**. Therefore, if you have simple ETL scenarios like CDC that require precise synchronization, you can use the CollapsingMergeTree table engine described below to achieve better results.

#### CollapsingMergeTree table engine

If the ClickHouse table engine is CollapsingMergeTree, you can specify `Sign` by using the `table.collapsing.field` option. This engine works by sending messages with the same content but opposite `Sign` values to achieve the deletion (cancellation) of old data and the insertion of new data.
In a production environment, ReplicatedCollapsingMergeTree is commonly used. However, the automatic deduplication feature of ReplicatedMergeTree may result in multiple records written to ClickHouse within a short time being identified as duplicate data, leading to data loss. In such cases, you can specify `replicated_deduplication_window=0` when creating or modifying a table to disable the automatic deduplication feature.
Example:

```
CREATE TABLE testdb.testtable on cluster default_cluster (`id` Int32,`name` Nullabl
```

For more information about automatic deduplication, see Data Replication.

### sharding_key of a ClickHouse distributed table

When creating a distributed table, set `sharding_key` in the statement `ENGINE = Distributed(cluster_name, database_name, table_name[, sharding_key]);` to the primary key of the sink table in Flink SQL. **This ensures that records with the same primary key are written to the same node**.

**The parameters in the statement are described as below:**

cluster_name: The cluster name, which corresponds to the custom name specified in the cluster configuration.

database_name: The database name.

table_name: The table name.

sharding_key: (Optional) The key used for sharding. During the data writing process, the distributed table will distribute data to local tables on different nodes based on the rules defined by the sharding key.

## Writing to local tables (feature for advanced users)

If the `local.read-write` option is set to `true` , Flink can directly write to local tables.

**Note**

Enabling write to local tables can significantly improve the throughput. However, if the ClickHouse cluster undergoes scaling or node replacement later, it may result in uneven data distribution, write failure, or data update failure. Therefore, this feature is intended only for advanced users.

If a table has a primary key and uses UPDATE and DELETE semantics, we recommend you use the CollapsingMergeTree table engine when creating the table. You can specify the `Sign` field using the `table.collapsing.field` option and set `sink.partition-strategy` to `hash` to ensure that data with the same primary key is distributed to the same shard. In addition, set `sink.partition-key` to the primary key field (for composite primary keys, set it to the first field).

## Null data

If certain fields in the data can be nullable, you need to set the field declaration in the ClickHouse table creation DDL statements to `Nullable` . Otherwise, an error may occur in data writing.

```
CREATE TABLE testdb.testtable on cluster default_cluster (`id` Int32,`name` Nullabl
```

### Optimization of writing to partitioned tables

When writing to a ClickHouse partitioned table, if the partition definition in ClickHouse uses functions supported by Stream Compute Service (intHash32, toYYYYMM, and toYYYYMMDD), Stream Compute Service enables the buffering of data by partition before writing by default. With this feature enabled, Stream Compute Service tries to include as few partitions as possible in each batch of data (when the throughput of business partition data is large enough, each batch contains only one partition), thereby improving ClickHouse's merge performance. If the data for a single partition has not reached the batch size, data of multiple partitions is merged into one batch and then written to ClickHouse. You can refer to `sink.max-partitions-per-insert` described above for detailed configuration options.

## Monitoring metric description

Stream Compute Service adds many practical metrics in the ClickHouse connector. Click the ClickHouse sink operator in the execution graph in Flink UI and search one of the metrics:
**numberOfInsertRecords**: The number of output +I messages.
**numberOfDeleteRecords**: The number of output -D messages.
**numberOfUpdateBeforeRecords**: The number of output -U messages.
**numberOfUpdateAfterRecords**: The number of output +U messages.

## Example: ClickHouse table creation statements

### CollapsingMergeTree table creation statement supporting UPDATE and DELETE

```
-- Creating a database
CREATE DATABASE test ON cluster default_cluster;

-- Creating a local table
CREATE TABLE test.datagen ON cluster default_cluster (`id` Int32,`name` Nullable(St

-- Creating a distributed table based on a local table
CREATE TABLE test.datagen_all ON CLUSTER default_cluster AS test.datagen ENGINE = D
```

**MergeTree table creation statement including only INSERT**

```
-- Creating a database
CREATE DATABASE test ON cluster default_cluster;

-- Creating a local table
CREATE TABLE test.datagen ON cluster default_cluster (`id` Int32,`name` Nullable(St

-- Creating a distributed table based on a local table
CREATE TABLE test.datagen_all ON CLUSTER default_cluster AS test.datagen ENGINE = D
```

# Data Lake Compute

Last updated：2023-11-08 14:27:50

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported (use as sink) |
| 1.14 | Unsupported |
| 1.16 | Unsupported |

## Use cases

This connector can be used as a sink. It allows for writing data to native tables managed in Data Lake Compute.

## Defining a table in DDL

```
CREATE TABLE `eason_internal_test`(
    `name` STRING,
    `age` INT
) WITH (
    'connector' = 'dlc-inlong',
    'catalog-database' = 'test',
    'catalog-table' = 'eason_internal_test',
    'default-database' = 'test',
    'catalog-name' = 'HYBRIS',
    'catalog-impl' = 'org.apache.inlong.sort.iceberg.catalog.hybris.DlcWrappedHybri
    'qcloud.dlc.secret-id' = '12345asdfghASDFGH',
    'qcloud.dlc.secret-key' = '678910asdfghASDFGH',
    'qcloud.dlc.region' = 'ap-guangzhou',
    'qcloud.dlc.jdbc.url' = 'jdbc:dlc:dlc.internal.tencentcloudapi.com?task_type=Sp
    'qcloud.dlc.managed.account.uid' = '100026378089',
    'request.identity.token' = '100026378089',
    'user.appid' = '1257058945',
    'uri' = 'dlc.internal.tencentcloudapi.com'
);
```

## WITH parameters

### Common parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | None | The connector to use. Here, it should be `dlc-inlo` |
| catalog-database | Yes | None | The name of the database where the Data Lake Com |
| catalog-table | Yes | None | The name of the Data Lake Compute internal table. |
| default-database | Yes | None | The name of the database where the Data Lake Com |
| catalog-name | Yes | None | The name of the catalog. Here, it should be `HYBRIS` |
| catalog-impl | Yes | None | The implementation class of the catalog. Here, it shou `org.apache.inlong.sort.iceberg.catalo` |
| qcloud.dlc.managed.account.uid | Yes | None | The uid of the Data Lake Compute account. Here, it s |
| qcloud.dlc.secret-id | Yes | None | The secretId of the Data Lake Compute user, which c https://console.intl.cloud.tencent.com/cam/capi. |
| qcloud.dlc.secret-key | Yes | None | The secretKey of the Data Lake Compute user, which https://console.intl.cloud.tencent.com/cam/capi. |
| qcloud.dlc.region | Yes | None | The region where the Data Lake Compute instance re region. |
| qcloud.dlc.jdbc.url | Yes | None | The URL for Data Lake Compute JDBC connection. |
| uri | Yes | None | The URI for Data Lake Compute connection. Here, it `dlc.internal.tencentcloudapi.com` . |
| user.appid | Yes | None | The appid of the Data Lake Compute user. |

| | | | |
|---|---|---|---|
| request.identity.token | Yes | None | The token for connecting the Data Lake Compute inte `100026378089` . |
| sink.ignore.changelog | No | Yes | Whether to ignore delete data, which defaults to `fal` append mode is enabled. |

## Configuring a Data Lake Compute table

```
Upsert mode
-- Statements to create a Data Lake Compute table
CREATE TABLE `bi_sensor`(
    `uuid` string,
    `id` string,
    `type` string,
    `project` string,
    `properties` string,
    `sensors_id` string,
    `time` int,
    `hour` int) PARTITIONED BY (`time`);
-- Set the target table as table v2 and allow for upsert operations.
ALTER TABLE `bi_sensor` SET TBLPROPERTIES ('format-version'='2','write.metadata.del

-- oceanus sink DDL. The primary key and partitioning field of the Data Lake Comput
create table bi_sensors (
    `uuid` STRING,
    `id` STRING,
    `type` STRING,
    `project` STRING,
    `properties` STRING,
    `sensors_id` STRING,
    `time` int,
    `hour` int,
    PRIMARY KEY (`uuid`, `time`) NOT ENFORCED
) with (...)
```

# Kudu

Last updated：2023-11-08 16:01:45

## Overview

The Kudu connector allows data reading from and writing to Kudu.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Supported |

## Use cases

The Kudu connector can be used as a source (for a general table and the right table in dimension-table join), a sink for tuple streams, and a source for upsert streams to allow data reading from and writing to Kudu.

## Defining a table in DDL

### As a source

```
CREATE TABLE `kudu_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options to connect Kudu.
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- The connect
    'kudu.table' = 'TableName1', -- Replace it with the table in Kudu, such as "def
    'kudu.hash-columns' = 'id', -- The hash key (optional).
    'kudu.primary-key-columns' = 'id', -- The primary key (optional).
    'kudu.operation-timeout' = '10000', -- The insert timeout period (optional).
    'kudu.max-buffer-size' = '2000', -- The buffer size (optional).
    'kudu.flush-interval' = '1000' -- The interval of data flush to Kudu (optional)
);
```

### As a sink (for tuple streams)

```
CREATE TABLE `kudu_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options to connect Kudu.
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- The connect
    'kudu.table' = 'TableName1', -- Replace it with the table in Kudu, such as "def
    'kudu.igonre-duplicate' = 'true' -- (Optional) If this option is set to `true`,
);
```

### As a sink (for upsert streams)

```
CREATE TABLE `kudu_upsert_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options to connect Kudu.
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- The connect
    'kudu.table' = 'TableName1', -- Replace it with the table in Kudu, such as "def
    'kudu.hash-columns' = 'id', -- The hash key (optional).
    'kudu.primary-key-columns' = 'id', -- The primary key (required). When this con
);
```

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector.type | Yes | None | For connection to a Kudu database, it must be `'kudu'` . |
| kudu.masters | Yes | None | The URL of the Kudu database master server, with a default port of 7051. If the Kudu component provided by Tencent Cloud is used, you can find the master server IP and port this way: Log in to the EMR console, click **ID**/**Name** of the target cluster in the cluster list to go to its details page, and select **Cluster services** > **Kudu** > **Operation** > **View port**. |
| kudu.table | Yes | None | The name of the Kudu table. For example, a Kudu table created through Impala is generally named as `impala::db_name.table_name` , and one created with Java API as `db_name.tablename` . |
| kudu.hash-columns | No | None | The hash key. |
| kudu.primary-key-columns | No | None | The primary key. |
| kudu.replicas | No | None | The number of replicas. |
| kudu.operation-timeout | No | 30000 | The insert timeout period in ms. |
| kudu.max-buffer-size | No | 1000 | Default value: `1000` . |
| kudu.flush-interval | No | 1000 | Default value: `1000` . |

| | | | |
|---|---|---|---|
| kudu.ignore-not-found | No | false | Whether to ignore the data that is not found. |
| kudu.ignore-duplicate | No | false | Whether to ignore the data whose primary key is identical with that of existing data. |

## Data type mapping

| Flink Type | Kudu Type |
|---|---|
| STRING | STRING |
| BOOLEAN | BOOL |
| TINYINT | INT8 |
| SMALLINT | INT16 |
| INT | INT32 |
| BIGINT | INT64 |
| FLOAT | FLOAT |
| DOUBLE | DOUBLE |
| BYTES | BINARY |
| TIMESTAMP(3) | UNIXTIME_MICROS |

## Example

```
CREATE TABLE `kudu_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options to connect Kudu.
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- The connect
    'kudu.table' = 'TableName1', -- Replace it with the table in Kudu, such as "def
    'kudu.hash-columns' = 'id', -- The hash key (optional).
    'kudu.primary-key-columns' = 'id', -- The primary key (optional).
    'kudu.operation-timeout' = '10000', -- The insert timeout period (optional).
    'kudu.max-buffer-size' = '2000', -- The buffer size (optional).
    'kudu.flush-interval' = '1000' -- The interval of data flush to Kudu (optional)
);

CREATE TABLE `kudu_upsert_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options to connect Kudu.
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- The connect
    'kudu.table' = 'TableName1', -- Replace it with the table in Kudu, such as "def
    'kudu.hash-columns' = 'id', -- The hash key (optional).
    'kudu.primary-key-columns' = 'id', -- The primary key (required). When this con
);

insert into kudu_upsert_sink_table select * from kudu_source_table;
```

## Notes

1. To query a Kudu table in Impala, you need to first check whether an external table is created.
2. By default, a table created using a way other than Impala Shell has no external table in Impala, and you need to create an external table for this table to query its records.
3. When the Kudu connector is used as a Stream Compute Service sink, if the sink table does not exist in the Kudu database, an internal table will be created there.

## Kudu authentication with Kerberos

1. Log in to the cluster master node to get the files `krb5.conf` and `emr.keytab` in the following paths.

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
```

2. Package the files into a JAR file.

```
jar cvf kudu-xxx.jar krb5.conf emr.keytab
```

3. Check the JAR structure (run the Vim command `vim kudu-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
```

4. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
5. Get the Kerberos principal and configure it in advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as shown below, and you can just use the first principal: hadoop/17
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   2 07/06/2023 18:50:41 hadoop/172.28.22.43@EMR-E4331BF2
   2 07/06/2023 18:50:41 HTTP/172.28.22.43@EMR-E4331BF2
   2 07/06/2023 18:50:41 kudu/172.28.22.43@EMR-E4331BF2
```

6. Configure the principle in advanced job parameters.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.22.43@EMR-E4331BF2
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
fs.hdfs.hadoop.security.authentication: kerberos
```

**Note**

Kudu authentication with Kerberos may be unavailable to some existing Stream Compute Service clusters. To use this feature, please contact us to upgrade the cluster management service.

# JDBC Connector for Tencent Cloud House-P

Last updated：2023-11-08 14:27:20

## Overview

The JDBC-PG connector allows for data writing to a Tencent Cloud House-P database.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Supported |

## Use cases

The JDBC-PG connector can be used as a sink for tuple and upsert (a primary key is required) streams.

## Defining a table in DDL

### As a sink (for tuple streams)

```
CREATE TABLE `jdbc_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbcPG',
    'url' = 'jdbc:postgresql://10.1.28.93:3306/CDB?reWriteBatchedInserts=true&serve
    'table-name' = 'my-table',  -- The name of JDBC table to connect.
    'username' = 'admin',       -- The username (with the INSERT permission require
    'password' = 'MyPa$$w0rd',  -- The password for database access.
    'sink.buffer-flush.max-rows' = '200',  -- The max size of buffered records befo
    'sink.buffer-flush.interval' = '2s'    -- The flush interval.
);
```

### As a sink (for upsert streams)

```
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
```

```
     'connector' = 'jdbcPG',
     'url' = 'jdbc:postgresql://10.1.28.93:3306/CDB?reWriteBatchedInserts=true&serve
     'table-name' = 'my-upsert-table', -- The name of JDBC table to connect.
     'username' = 'admin',             -- The username (with the INSERT permission r
     'password' = 'MyPa$$w0rd',        -- The password for database access.
     'sink.buffer-flush.max-rows' = '200',  -- The max size of buffered records befo
     'sink.buffer-flush.interval' = '2s'    -- The flush interval.
 );
```

**Note**

For an upsert stream, a primary key is required.

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | None | For connection to a database, it should be `'jdbcPG'` . |
| url | Yes | None | The JDBC database URL. |
| table-name | Yes | None | The name of JDBC table to connect. |
| driver | No | None | The class name of the JDBC driver to use to connect to the above-mentioned URL. If no value is set, it will automatically be derived from the URL. |
| username | No | None | The JDBC username. `'username'` and `'password'` must both be specified if any of them is specified. |
| password | No | None | The JDBC password. |
| scan.partition.column | No | None | The column name used for partitioning the input. The column type must be numeric, date, or timestamp. For details, see Partitioned scan. |
| scan.partition.num | No | None | The number of partitions. |
| scan.partition.lower-bound | No | None | The smallest value of the first partition. |
| scan.partition.upper-bound | No | None | The largest value of the last partition. |

| | | | |
|---|---|---|---|
| scan.fetch-size | No | 0 | The number of rows that should be fetched from the database when reading per round trip. If the value specified is 0, the data is read row by row, indicating low efficiency (low throughput). |
| lookup.cache.max-rows | No | None | The maximum number of data records cached in the Lookup Cache. |
| lookup.cache.ttl | No | None | The maximum cache time of each data record in the Lookup Cache. |
| lookup.max-retries | No | 3 | The maximum number of retries upon database lookup failure. |
| sink.buffer-flush.max-rows | No | 100 | The maximum size of buffered records before flush. If this is set to `0`, the records will not be buffered. |
| sink.buffer-flush.interval | No | 1s | The maximum interval (ms) between flushes. **If `sink.buffer-flush.max-rows` is `0`, and this parameter is not, buffered actions will be processed asynchronously.** |
| sink.max-retries | No | 3 | The maximum retry times if writing records to database failed. |
| write-mode | No | insert | Write in the `copy` mode. Valid values: `insert` and `copy`. |
| copy-delimiter | No | &#x399 | The field delimiter in the `copy` mode. |

## Sample

```
CREATE TABLE jdbc_upsert_sink_table (
  id INT ,
  age INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED -- Define the primary key here if the database tabl
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbcPG',
    'url' = 'jdbc:postgresql://10.0.0.2:5436/postgres',
    'table-name' = 'my-upsert-table', -- The name of JDBC table to connect.
    'username' = 'admin',              -- The username (with the INSERT permission r
     'password' = 'password',         -- The password for database access.
     'sink.buffer-flush.max-rows' = '300',  -- The max size of buffered records befo
     'sink.buffer-flush.interval' = '100s'    -- The flush interval.
);


-- MySQL CDC connector as the source. Use "flink-connector-mysql-cdc" and this conn
For more information, see: https://cloud.tencent.com/document/product/849/52698.
CREATE TABLE mysql_cdc_source_table (
  id INT,
  age INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED -- Define the primary key here if the database tabl
) WITH (
  'connector' = 'mysql-cdc',       -- Here, it should be 'mysql-cdc'.
  'hostname' = '10.0.0.6',    -- IP of the MySQL database server.
  'port' = '3360',                 --  Integer port number of the MySQL database ser
  'username' = 'root',         -- Name of the database to use when connecting to the
  'password' = 'password',     -- Password to use when connecting to the MySQL datab
  -- 'scan.incremental.snapshot.enabled' = 'false' -- This option is required if th
  'database-name' = 'database',    -- Database name of the MySQL server to monitor.
  'table-name' = 'table'       -- Table name of the MySQL database to monitor.
);


INSERT INTO jdbc_upsert_sink_table select * from mysql_cdc_source_table;
```

## Primary key

For append (tuple) data, it is not required to set the primary key option in the DDL statements of the job, nor define a primary key for the table of the JDBC database. We indeed do not recommend defining a primary key (because duplicate data may cause write to fail).

For upsert data, a primary key **must** be defined for the table of the JDBC database, and the `PRIMARY KEY NOT ENFORCED` constraint shall also be set for the corresponding column in the CREATE TABLE clause of the DDL statements.

**Note**

The implementation of the upsert operation does not rely on the `INSERT .. ON CONFLICT .. DO UPDATE SET ..` syntax, but on `upsert +delete` in batches. This syntax applies to PostgreSQL v9.5 or earlier.

## Flush optimization

Setting the first two sink.buffer-flush options allows flush in batches to the data. We recommend you also set relevant underlying database parameters for better flush performance. Otherwise, the data may also be flushed record by record in the underlying system, lowering the efficiency.
For PostgreSQL, it is recommended to add `reWriteBatchedInserts=true` following the URL option as shown below.

```
jdbc:postgresql://10.1.28.93:3306/PG?reWriteBatchedInserts=true&currentSchema=Datab
```

# JDBC

Last updated：2023-11-08 16:06:23

## Overview

The JDBC connector allows for reading data from and writing data to MySQL, PostgreSQL, Oracle, and other common databases.
The `flink-connector-jdbc` connector component provided by Stream Compute Service has built-in MySQL and PostgreSQL drivers. To connect Oracle and other databases, update the JDBC Driver packages as **custom packages**.
**Note**
You can use only the PostgreSQL connector to connect a Tencent Cloud House-P database.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Supported |
| 1.16 | Supported |

## Use cases

The JDBC connector can be used as a source for a table scanned by a fixed column or a joined right table (dimension table); it can also be used as a sink for a tuple or upsert (the primary key is required) table.
To use change records of a JDBC database as stream tables for further consumption, use the built-in CDC source. If the built-in CDC source cannot meet your needs, you can capture and subscribe to the changes of the JBDC database using Debezium, Canal, or other available methods, and Stream Compute Service will further process the captured change events. For more information, see Kafka.

## Defining a table in DDL

### As a source (scanned by a fixed column)

```
CREATE TABLE `jdbc_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true&serverT
    'table-name' = 'my-table',  -- The name of the table to connect.
    'username' = 'admin',      -- The username (with the INSERT permission required
    'password' = 'MyPa$$w0rd',  -- The password for database access.
    'scan.partition.column' = 'id',    -- The name of the partitioned scan column.
    'scan.partition.num' = '2',        -- The number of partitions.
    'scan.partition.lower-bound' = '0', -- The minimum value of the first partition
    'scan.partition.upper-bound' = '100', -- The maximum value of the last partitio
    'scan.fetch-size' = '1' -- The number of rows that should be fetched from the d
);
```

### As a source dimension table

```
CREATE TABLE `jdbc_dim_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true&serverT
    'table-name' = 'my-table',  -- The name of the table to connect.
    'username' = 'admin',       -- The username (with the INSERT permission required
    'password' = 'MyPa$$w0rd',  -- The password for database access.
    'lookup.cache.max-rows' = '100',   -- The maximum number of rows of lookup cach
    'lookup.cache.ttl' = '5000'        -- The max time to live for each row in the L
);
```

### As a sink (for tuple streams)

```
CREATE TABLE `jdbc_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true&serverT
    'table-name' = 'my-table',  -- The name of JDBC table to connect.
    'username' = 'admin',       -- The username (with the INSERT permission require
    'password' = 'MyPa$$w0rd',  -- The password for database access.
    'sink.buffer-flush.max-rows' = '200',  -- The maximum size of buffered records
    'sink.buffer-flush.interval' = '2s'    -- The flush interval.
);
```

### As a sink (for upsert streams)

```
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
```

```
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true&serverT
      'table-name' = 'my-upsert-table', -- The name of JDBC table to connect.
      'username' = 'admin',             -- The username (with the INSERT permission r
      'password' = 'MyPa$$w0rd',        -- The password for database access.
      'sink.buffer-flush.max-rows' = '200',  -- The maximum size of buffered records
      'sink.buffer-flush.interval' = '2s'    -- The flush interval.
);
```

**Note**

For an upsert stream, a primary key is required.

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | None | For connection to a database, it should be `'jdbc'` . |
| url | Yes | None | The JDBC database URL. |
| table-name | Yes | None | The table name. |
| driver | No | None | The class name of the JDBC driver to use to connect to the above-mentioned URL. If no value is set, it will automatically be derived from the URL. |
| name-server | No | None | The JDBC username. `'username'` and `'password'` must both be specified if any of them is specified. |
| password | No | None | The JDBC password. |
| scan.partition.column | No | None | The column name used for partitioning the input. The column type must be numeric, date, or timestamp. For details, see Partitioned scan. |
| scan.partition.num | No | None | The number of partitions. |
| scan.partition.lower-bound | No | None | The smallest value of the first partition. |
| scan.partition.upper-bound | No | None | The largest value of the last partition. |
| | | | |

| | | | |
|---|---|---|---|
| scan.fetch-size | No | 0 | The number of rows that should be fetched from the database when reading per round trip. If the value specified is 0, the data is read row by row, indicating low efficiency (low throughput). |
| lookup.cache.max-rows | No | None | The maximum number of data records cached in the Lookup Cache. |
| lookup.cache.ttl | No | None | The maximum cache time of each data record in the Lookup Cache. |
| lookup.max-retries | No | 3 | The maximum number of retries upon database lookup failure. |
| sink.buffer-flush.max-rows | No | 100 | The maximum size of buffered records before flush. If this is set to `0`, the records will not be buffered. |
| sink.buffer-flush.interval | No | 1s | The maximum interval (ms) between flushes. **If `sink.buffer-flush.max-rows` is `0`, and this option is not, buffered actions will be processed asynchronously.** |
| sink.max-retries | No | 3 | The maximum retry times if writing records to database failed. |
| sink.ignore-delete | No | false | Whether to ignore the DELETE operation. |

## Example

```
CREATE TABLE `jdbc_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true&serverT
    'table-name' = 'my-table',  -- The name of the table to connect.
    'username' = 'admin',       -- The username (with the INSERT permission required
    'password' = 'MyPa$$w0rd',  -- The password for database access.
    'scan.partition.column' = 'id',   -- The name of the partitioned scan column.
    'scan.partition.num' = '2',       -- The number of partitions.
    'scan.partition.lower-bound' = '0', -- The minimum value of the first partition
    'scan.partition.upper-bound' = '100', -- The maximum value of the last partitio
    'scan.fetch-size' = '1' -- The number of rows that should be fetched from the d
);
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
) WITH (
    -- Specify the options for database connection.
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true&serverT
    'table-name' = 'my-upsert-table', -- The name of JDBC table to connect.
    'username' = 'admin',             -- The username (with the INSERT permission r
    'password' = 'MyPa$$w0rd',        -- The password for database access.
    'sink.buffer-flush.max-rows' = '200',  -- The maximum size of buffered records
    'sink.buffer-flush.interval' = '2s'    -- The flush interval.
);
insert into jdbc_upsert_sink_table select * from jdbc_source_table;
```

## Primary key

For append (tuple) data, it is not required to set the primary key option in the DDL statements of the job, nor define a primary key for the table of the JDBC database. We do not recommend defining a primary key (because duplicate data may cause write to fail).

For upsert data, a primary key **must** be defined for the table of the JDBC database, and the `PRIMARY KEY NOT ENFORCED` constraint shall also be set for the corresponding column in the CREATE TABLE clause of the DDL statements.

**Note**

For MySQL tables, the implementation of the upsert operation relies on the `INSERT .. ON DUPLICATE KEY UPDATE ..` syntax, which is supported by all common MySQL versions.

For MySQL tables, the implementation of the upsert operation relies on the `INSERT .. ON CONFLICT .. DO UPDATE SET ..` syntax, which is supported by PostgreSQL v9.5 or later.

For Oracle tables, the implementation of the upsert operation relies on the `MERGE .. INTO .. USING ON .. WHEN UPDATE .. WHEN INSERT ..` syntax, which is supported by Oracle v9i or later.

## Partitioned scan

Partitioned scan can accelerate reading data in parallel source operators (tasks) from a JDBC table. Each task reads the partitions assigned to it only. To use this feature, all the four options starting with `scan.partition` must all be specified. Otherwise, errors will be raised.

**Note**

The maximum value specified by `scan.partition.upper-bound` and the minimum value specified by `scan.partition.lower-bound` refer to the maximum and minimum partition steps. They do not affect the number and accuracy of the read data records.

## Lookup Cache

Flink provides the Lookup Cache feature to improve the performance of lookup in a dimension table. The Lookup Cache is implemented synchronously. It is not enabled by default, which means the data is read from the database for each request and the throughput is very low. To enable it, you must set `lookup.cache.max-rows` and `lookup.cache.ttl`.

**Note**

If the lookup cache TTL is too long, or the number of the data records cached is too large, the Flink job may still read the old data in the cache after the data in the database is updated. Therefore, for a job involving a frequently changed database, please use this feature with caution.

## Flush optimization

Setting the first two sink.buffer-flush options allows flush in batches to the data. We recommend you also set relevant underlying database parameters for better flush performance. Otherwise, the data may also be flushed record by record in the underlying system, lowering the efficiency.

For MySQL, it is recommended to add `rewriteBatchedStatements=true` following the URL option as shown below.

```
jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true
```

For PostgreSQL, it is recommended to add `reWriteBatchedInserts=true` following the URL option as shown below.

```
jdbc:postgresql://10.1.28.93:3306/PG?reWriteBatchedInserts=true&currentSchema=Datab
```

## Descriptions of monitoring metrics

Stream Compute Service adds many practical metrics in the JDBC connector. Click the JDBC sink operator in the execution graph in Flink UI and search for one of the metrics:

**numberOfInsertRecords**: The number of output +I messages.

**numberOfDeleteRecords**: The number of output -D messages.
**numberOfUpdateBeforeRecords**: The number of output -U messages.
**numberOfUpdateAfterRecords**: The number of output +U messages.

# FileSystem

Last updated：2023-11-08 16:00:41

## Overview

The FileSystem connector allows for writing to common file systems such as HDFS and Tencent Cloud Object Storage.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported (supports common compression algorithms such as LZO and Snappy) |
| 1.14 | Supports write to HDFS (does not support the LZO and Snappy compression algorithms) |
| 1.16 | Supports write to HDFS (does not support the LZO and Snappy compression algorithms) |

## Limits

The FileSystem connector can be used as a data sink for append-only data streams. It cannot be used as a sink for upsert data currently. The following data formats are supported:
CSV
JSON
Avro
Parquet
ORC
**Note**
To write data in Avro, Parquet, or ORC format, you need to manually upload a JAR package.

## Defining a table in DDL

**As a sink**

```
CREATE TABLE `hdfs_sink_table` (
    `id` INT,
    `name` STRING,
    `part1` INT,
    `part2` INT
) PARTITIONED BY (part1, part2) WITH (
    'connector' = 'filesystem',
    'path' = 'hdfs://HDFS10000/data/', -- cosn://${buketName}/path/to/store/data
    'format' = 'json',
    'sink.rolling-policy.file-size' = '1M',
    'sink.rolling-policy.rollover-interval' = '10 min',
    'sink.partition-commit.delay' = '1 s',
    'sink.partition-commit.policy.kind' = 'success-file'
);
```

## WITH parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| path | Yes | - | The path to which files are written. |
| sink.rolling-policy.file-size | No | 128MB | The maximum file size. If a file exceeds this size, it will be closed. A new file will be opened and data will be written to this new file. |
| sink.rolling-policy.rollover-interval | No | 30min | The maximum duration a file can stay open. After this duration elapses, the file will be closed. A new file will be opened and data will be written to the new file. |
| sink.rolling-policy.check-interval | No | 1min | The file check interval. The FileSystem connector will check whether a file should be closed at this interval. |
| sink.partition-commit.trigger | No | process-time | The partition commit trigger. Valid values:<br>`process-time` : A partition will be committed when a certain time elapses after the partition creation time. Here, the creation time is the time when the partition is created.<br>`partition-time` : A partition will be committed when a certain time elapses after the partition creation time. Here, the creation time is extracted from partition values. `partition-time` requires watermark generation to automatically detect partitions. A partition is committed once `watermark` passes the time extracted from partition values plus `delay` . |
| sink.partition-commit.delay | No | 0s | The time to wait before committing a partition. A partition will not be committed until the specified time elapses after its creation time. |
| partition.time-extractor.kind | No | default | The partition time extractor. This option is valid only if `sink.partition-commit.trigger` is |

| | | | |
|---|---|---|---|
| | | | `partition-time` . If you have your own time extractor, set this to `custom` . |
| partition.time-extractor.class | No | - | The partition time extractor class. This class should implement the `PartitionTimeExtractor` API. |
| partition.time-extractor.timestamp-pattern | No | - | The partition timestamp extraction format. The timestamps should be `yyyy-mm-dd hh:mm:ss` , with the placeholders replaced with the corresponding partition fields in the Hive table. By default, `yyyy-mm-dd hh:mm:ss` is extracted from the first field.<br>If timestamps are extracted from a single partition field `dt` , you can set this to `$dt` .<br>If timestamps are extracted from multiple partition fields, such as `year` , `month` , `day` , and `hour` , you can set this to `$year-$month-$day $hour:00:00` .<br>If timestamps are extracted from two partition fields `dt` and `hour` , you can set this to `$dt $hour:00:00` . |
| sink.partition-commit.policy.kind | Yes | - | The partition committing policy. Valid values:<br>`success-file` : When a partition is committed, a `_success` file will be generated in the partition's directory.<br>`custom` : A custom committing policy. |
| sink.partition-commit.policy.class | No | - | The partition committing class. This class should implement `PartitionCommitPolicy` . |

## HDFS configuration

After creating a data directory in HDFS, you need to grant write access to the directory. With Stream Compute Service, the user writing data to HDFS is Flink. Before the configuration, you need to log in to your EMR cluster to download the `hdfs-site.xml` file of the Hadoop cluster. The file includes the parameter values needed for the configuration.
The HDFS path is in the format `hdfs://${dfs.nameserivces}/${path}` . You can find the value of `${dfs.nameserivces}` in `hdfs-site.xml` . `${path}` is the path to which data will be written.
If the target Hadoop cluster has only one master node, you only need to pass the HDFS path to `path` . You don't need to set advanced parameters.

If the target Hadoop cluster is a high-availability cluster with two master nodes, after passing the HDFS path, you also need to specify the addresses and ports of the two master nodes by configuring advanced job parameters. Below is an example. You can find the parameter values in `hdfs-site.xml` .

```
fs.hdfs.dfs.nameservices: HDFS12345
fs.hdfs.dfs.ha.namenodes.HDFS12345: nn2,nn1
fs.hdfs.dfs.namenode.http-address.HDFS12345.nn1: 172.27.2.57:4008
fs.hdfs.dfs.namenode.https-address.HDFS12345.nn1: 172.27.2.57:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS12345.nn1: 172.27.2.57:4007
fs.hdfs.dfs.namenode.http-address.HDFS12345.nn2: 172.27.1.218:4008
fs.hdfs.dfs.namenode.https-address.HDFS12345.nn2: 172.27.1.218:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS12345.nn2: 172.27.1.218:4007
fs.hdfs.dfs.client.failover.proxy.provider.HDFS12345: org.apache.hadoop.hdfs.server
```

**Note**

By default, Flink jobs access HDFS via Flink. If the Flink user does not have permission to write to HDFS, you can use advanced job parameters to set the accessing user to a user that has write permission or to the super-user `hadoop`.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

## COS configuration

**Note:** If you write data to COS, the job must run in the same region as the COS bucket.
Use advanced job parameters to specify the COS region. With Stream Compute Service, the user writing data to COS is Flink. You need to configure the following parameters. For the values of regions, see COS - Regions and Access Endpoints.

```
fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: The region of the COS bucket.
fs.cosn.userinfo.appid: The AppID of the account that owns the COS bucket.
```

For JAR jobs, to write data to COS, you need to select the built-in connector `flink-connector-cos` .

**Note**

**If you use Flink 1.16, you can skip this step because** `flink-connector-cos` is built into the image.

## Metadata acceleration-enabled COS buckets and Tencent Cloud HDFS

1. Access **Authorizations**.

Go to **Compute resources** > **Cluster info** > **More** and select **Authorizations**.

2. Grant permission to a metadata acceleration-enabled COS bucket or CHDFS.

3. Download dependencies.

On the **Dependencies** page of the Stream Compute Service console, upload the JAR package you downloaded. For details, see Managing Dependencies.

Metadata acceleration-enabled COS buckets

chdfs_hadoop_plugin_network-2.8.jar

gson-2.9.1.jar

Corresponding Flink connectors

flink-1.13-connector-cos.jar

flink-1.14-connector-cos.jar

CHDFS

chdfs_hadoop_plugin_network-2.8.jar

gson-2.9.1.jar

flink-chdfs-hadoop-1.10.0-0.1.4.jar

4. Configure advanced job parameters.

Metadata acceleration-enabled COS buckets

```
fs.cosn.trsf.fs.AbstractFileSystem.ofs.impl: com.qcloud.chdfs.fs.CHDFSDelegateFSAda
fs.cosn.trsf.fs.ofs.impl: com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter
fs.cosn.trsf.fs.ofs.tmp.cache.dir: /tmp/chdfs/
fs.cosn.trsf.fs.ofs.user.appid: The AppID of the account that owns the COS bucket.
fs.cosn.trsf.fs.ofs.bucket.region: The region of the COS bucket.
fs.cosn.trsf.fs.ofs.upload.flush.flag: true
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop

fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: The region of the COS bucket.
fs.cosn.userinfo.appid: The AppID of the account that owns the COS bucket.
```

**Note**

**Flink 1.16 supports metadata acceleration-enabled buckets by default. You don't need to download JAR packages. Just configure the following two options. The other parameters will be configured automatically.**

containerized.taskmanager.env.HADOOP_USER_NAME: hadoop

containerized.master.env.HADOOP_USER_NAME: hadoop

CHDFS

```
fs.AbstractFileSystem.ofs.impl: com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter
fs.ofs.impl: com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter
fs.ofs.tmp.cache.dir: /tmp/chdfs/
fs.ofs.upload.flush.flag: true
fs.ofs.user.appid: The AppID of the account that owns the file system.
fs.ofs.bucket.region: The region of the file system.
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

## Manually uploading a JAR package

1. Download the required JAR package.
Download addresses for different Flink versions: Flink 1.11, Flink 1.13, Flink 1.14
2. On the **Dependencies** page of the Stream Compute Service console, upload the JAR package you downloaded.
For details, see Managing Dependencies.
3. Go to the debug page of the job and click **Job parameters**.
Find **Referenced package** and click **Add package**. Select the JAR package you upload in step 2, and click **Confirm**.
4. Publish the job.

## HDFS Kerberos authentication

1. Log in to the cluster master node to get the files `krb5.conf` , `emr.keytab` , `core-site.xml` , and `hdfs-site.xml` in the following paths.

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

2. Package the files into a JAR file.

```
jar cvf hdfs-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml
```

3. Check the JAR structure (run the Vim command `vim hdfs-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
```

4. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
5. Get Kerberos principals to configure advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as follows (use the first): hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- ------------------------------------------------------
  2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. Configure advanced job parameters.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

If you use Flink 1.13, you need to add the following configurations to advanced parameters. The parameter values should be the same as those in `hdfs-site.xml` .

```
fs.hdfs.dfs.nameservices: HDFS17995
fs.hdfs.dfs.ha.namenodes.HDFS17995: nn2,nn1
fs.hdfs.dfs.namenode.http-address.HDFS17995.nn1: 172.28.28.214:4008
fs.hdfs.dfs.namenode.https-address.HDFS17995.nn1: 172.28.28.214:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS17995.nn1: 172.28.28.214:4007
fs.hdfs.dfs.namenode.http-address.HDFS17995.nn2: 172.28.28.224:4008
fs.hdfs.dfs.namenode.https-address.HDFS17995.nn2: 172.28.28.224:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS17995.nn2: 172.28.28.224:4007
fs.hdfs.dfs.client.failover.proxy.provider.HDFS17995: org.apache.hadoop.hdfs.server
fs.hdfs.hadoop.security.authentication: kerberos
```

**Note**

Kerberos authentication is not supported for historical Stream Compute Service clusters. To support the feature, please contact us to upgrade the cluster management service.

## Example

```
CREATE TABLE datagen_source_table (
  id INT,
  name STRING,
  part1 INT,
  part2 INT Block
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1',  -- The number of records generated per second.
  'fields.part1.min'='1',
  'fields.part1.max'='2',
  'fields.part2.min'='1',
  'fields.part2.max'='2'
);

CREATE TABLE hdfs_sink_table (
    id INT,
    name STRING,
    part1 INT,
    part2 INT Block
) PARTITIONED BY (part1, part2) WITH (
    'connector' = 'filesystem',
    'path' = 'hdfs://HDFS10000/data/',
    'format' = 'json',
    'sink.rolling-policy.file-size' = '1M',
    'sink.rolling-policy.rollover-interval' = '10 min',
    'sink.partition-commit.delay' = '1 s',
    'sink.partition-commit.policy.kind' = 'success-file'
);

INSERT INTO hdfs_sink_table
SELECT id, name, part1, part2
FROM datagen_source_table;
```

## About compressible-fs

Can only be used with Flink 1.13.
Supports writing data in the CSV or JSON format. Other formats such as Avro, Parquet, and ORC are built-in with compression capabilities.
Supports the compression algorithms LzopCodec and OceanusSnappyCodec.
Supports writing data to HDFS and COS files. The method is the same as that for FileSystem.

**As a sink**

```
CREATE TABLE `hdfs_sink_table` (
    `id` INT,
    `name` STRING,
    `part1` INT,
    `part2` INT
) PARTITIONED BY (part1, part2) WITH (
    'connector' = 'compressible-fs',
    'hadoop.compression.codec' = 'LzopCodec',
    'path' = 'hdfs://HDFS10000/data/',
    'format' = 'json',
    'sink.rolling-policy.file-size' = '1M',
    'sink.rolling-policy.rollover-interval' = '10 min',
    'sink.partition-commit.delay' = '1 s',
    'sink.partition-commit.policy.kind' = 'success-file'
);
```

## WITH parameters

In addition to the above parameters supported by the `filesystem` connector, `compressible-fs` also supports the following three parameters:

| Option | Required | Default Value | Description |
|---|---|---|---|
| hadoop.compression.codec | No | - | The compression algorithm to use. Valid values include `LzopCodec` and `OceanusSnappyCodec` . If this is not specified, data will be written in the default file format. `OceanusSnappyCodec` is used due to Snappy library version restrictions. It works the same as `SnappyCodec` . |
| filename.suffix | No | - | The name of the file to which data is written. If this is not specified, a suffix will be generated according to the compression algorithm used. If neither lzop nor Snappy is used, and this option is not specified, the filename suffix will be empty. |
| filepath.contain.partition-key | No | false | Whether to include the partition field in the path when data is written into partition files. The partition field is not included by default. For example, if partitioning is based on the date `dt=12` and hour `ht=24` , the default partition path will be `12/24` instead of `dt=12/ht=24` . |

# Testing Connectors

Last updated：2024-04-19 12:23:54

## Testing sources and sinks

If you want to check whether a job can run successfully or whether the logic is correct, you can use connectors specially designed for testing. This saves you the need to deploy external systems and can reduce interferences.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Supported |
| 1.16 | Supported |

## Datagen source

Datagen is a random data generator built into Flink. It can be used directly as a data source. For details, see the Flink document.

Below is an example of a Datagen source. It generates two fields: `id` and `name` . `id` is a random number, and `name` is a random string.

### Defining a table in DDL

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1'  -- The number of data records generated per second.
);
```

### WITH parameters

| Option | Required | Default Value | Data Type | Description |
|---|---|---|---|---|
| connector | Yes | - | String | The connector to use. Here, it should be `datagen`. |
| rows-per-second | No | 10000 | Long | The number of rows generated per second. This determines the data send rate. |
| fields.#.kind | No | random | String | The value generator for the `#` field. Valid values include `sequence` and `random`. |
| fields.#.min | No | (Minimum value of type) | (Type of field) | The minimum value the random generator can generate. This is used for numeric data types. |
| fields.#.max | No | (Maximum value of type) | (Type of field) | The maximum value the random generator can generate. This is used for numeric data types. |
| fields.#.length | No | 100 | Integer | The length of strings generated by the random generator. This is used for data types including `char`, `varchar`, and `string`. |
| fields.#.start | No | - | (Type of field) | The start value of the sequence generator. |
| fields.#.end | No | - | (Type of field) | The end value of the sequence generator. |

## Logger sink

The Logger sink is a custom logger example provided by Stream Compute Service. It can write final data to the log file of TaskManager so that you can view the log output via Flink UI or the logging section of the Stream Compute Service console.

1. To use the Logger sink, download the JAR package. **If you want to customize the output logic, you can modify the package and rebuild it.**
2. Upload the package to the Stream Compute Service console. For details, see Managing Dependencies.
3. In your SQL job, reference the package.

### Defining a table in DDL

```
CREATE TABLE logger_sink_table (
    id INT,
    name STRING
) WITH (
    'connector' = 'logger',
    'print-identifier' = 'DebugData'
);
```

### WITH parameters

| Option | Required | Data Type | Description |
|---|---|---|---|
| connector | Yes | String | The connector to use. Here, it should be `logger` . |
| print-identifier | No | String | The prefix for log printing. |
| all-changelog-mode | No | Boolean | If you enable this, -U data will not be filtered out. This allows you to simulate ClickHouse Collapsing data streams. |
| records-per-second | No | Integer | The number of records output per second. You can use this to achieve throttling. |
| mute-output | No | Boolean | Whether to drop all output data and only calculate the number of records (an enhanced version of a BlackHole sink). |

### Monitoring metrics

Stream Compute Service offers a series of handy metrics for the Logger sink. Click the Logger sink operator in the execution graph in Flink UI and search for one of the metrics:

**numberOfInsertRecords**: The number of output +I messages.
**numberOfDeleteRecords**: The number of output -D messages.
**numberOfUpdateBeforeRecords**: The number of output -U messages.
**numberOfUpdateAfterRecords**: The number of output +U messages.

## Print sink (not recommended)

Flink is built in with the standard-output Print sink. However, because the printing format does not comply with the rules of Stream Compute Service's log collector, the data is not well displayed in the console. We recommend you use the Logger sink instead.

### Defining a table in DDL

```
CREATE TABLE `print_table` (
  `id` INT,
  `name` STRING
) WITH (
 'connector' = 'print'
);
```

### WITH parameters

| Option | Required | Default | Data Type | Description |
|---|---|---|---|---|

| | | Value | | |
|---|---|---|---|---|
| connector | Yes | - | String | The connector to use. Here, it should be `print` . |
| print-identifier | No | - | String | The identifier (prefix) for the output data. |
| standard-error | No | false | Boolean | Whether to print as standard error instead of standard output. To print as standard error, set this to `True` . |

# Custom Connectors

Last updated：2023-11-08 14:25:19

## Overview

If the built-in connectors cannot meet your needs, you can **customize your own connectors** by uploading source and sink API classes and dynamically loading and calling them when the job is executed.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Supported |
| 1.13 | Supported |
| 1.14 | Supported |
| 1.16 | Supported |

## Prepare your connector package

You can either use a third-party connector package (such as Bahir) or build your own package.

### Apache Bahir

Apache Bahir offers common source and sink extensions for Flink.

Currently, the following third-party components are provided:

ActiveMQ

Akka

Flume

InfluxDB

Kudu

Redis

Netty

### Building your own package

See Flink API.

## Build and upload the connector package

### Step 1. Compile the source code

We recommend you refer to existing connector projects to modify the `pom.xml` file to import dependencies and create a JAR package in Maven.

**Note**

We recommend you use `maven-shade-plugin` to shade common dependencies such as Apache Commons and Guava to avoid conflicts between the imported dependencies and classes of Stream Compute Service.

### Step 2. Upload the package

Go to the Dependencies page of the Stream Compute Service console and upload the connector package. For the first upload, the package will be v1. If you upload a new version, it will be v2, and so on.

### ### Step 3. Reference the package

On the details page of your job, under the **Job parameters** tab, select the package version you uploaded.

**Note**

To avoid unexpected errors, make sure you select the correct package version.

### ### Step 4. Save/Publish the job

Click **Save** to save the settings or **Publish draft** to publish the job.

# Hudi

Last updated：2023-11-08 15:56:45

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported (use as source and sink) |
| 1.14 | Unsupported |
| 1.16 | Unsupported |

## Use cases

This connector can be used as a source or a sink.

## Defining a table in DDL

As a sink:

```
CREATE TABLE hudi_sink
(
    uuid         VARCHAR(20) PRIMARY KEY NOT ENFORCED,
    name         VARCHAR(10),
    age          INT,
    ts           TIMESTAMP(3),
    `partition` VARCHAR(20)
) WITH (
    'connector' = 'hudi'
    , 'path' = 'hdfs://HDFS1000/data/hudi/mor'
    , 'table.type' = 'MERGE_ON_READ'  --  The MERGE_ON_READ table, which defaults t
```

```
      , 'write.tasks' = '3' -- Default value: 4.
      , 'compaction.tasks' = '4' -- Default value: 4.
      --  , 'hive_sync.enable' = 'true'  -- Default value: false.
      --  , 'hive_sync.db' = 'default'
      --  , 'hive_sync.table' = 'datagen_mor_1'
      --  , 'hive_sync.mode' = 'jdbc'
      --  , 'hive_sync.username' = ''
      --  , 'hive_sync.password' = ''
      --  , 'hive_sync.jdbc_url' = 'jdbc:hive2://172.28.1.185:7001'
      --  , 'hive_sync.metastore.uris' = 'thrift://172.28.1.185:7004'
  );
```

As a source:

```
CREATE TABLE `source`
(
    uuid        VARCHAR(20) PRIMARY KEY NOT ENFORCED,
    name        VARCHAR(10),
    age         INT,
    ts          TIMESTAMP(3),
    `partition` VARCHAR(20)
) WITH (
      'connector' = 'hudi'
      , 'path' = 'hdfs://172.28.28.202:4007/path/hudidata'
      , 'table.type' = 'MERGE_ON_READ'  -- The MOR table. Its incremental data cann
```

```
        , 'read.tasks' = '1'  -- The parallelism of the read tasks, which defaults to
        , 'hoodie.datasource.query.type' = 'snapshot' -- Default value: snapshot; oth
        , 'read.streaming.enabled' = 'true'  -- This option enables the streaming rea
        , 'read.start-commit' = 'earliest' -- Specifies the start commit instant time
        , 'read.streaming.check-interval' = '4'
        );
```

## WITH parameters

### Common parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | None | Here, it should be `hudi` . |
| path | Yes | None | The data storage path, in the format of `hdfs://` for data storage in HDFS and `COSN://$bucket/$path` for data storage in COS. |

### Parameters when as a sink

| Option | Required | Default Value | Description |
|---|---|---|---|
| table.type | No | COPY_ON_WRITE | The Hudi table type. Valid values: `COPY_ON_WRITE` and `MERGE_ON_READ` . |

### `HoodieRecord` parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| hoodie.datasource.write.recordkey.field | No | uuid | The key field. If the Flink table has a primary key, use it. |
| hoodie.datasource.write.partitionpath.field | No | "" | The partition path field. Null indicates the table is not partitioned. |
| write.precombine.field | No | ts | Field used in pre-combining before actual write. When two records have the same key value, the one with the larger value will be picked for the pre-combine field, determined by Object.compareTo(..). |

### Parallelism parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| write.tasks | No | 4 | The parallelism of tasks that do actual write. |
| write.index_bootstrap.tasks | No | None | The parallelism of tasks that do index bootstrap, which defaults to the parallelism of the execution environment. |
| write.bucket_assign.tasks | No | None | The parallelism of tasks that do bucket assign, which defaults to the parallelism of the execution environment. |
| compaction.tasks | No | 4 | The parallelism of tasks that do actual compaction. |

### Compaction parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| compaction.schedule.enabled | No | true | Whether to enable compaction. |
| compaction.async.enabled | No | true | Whether to use async compaction. |
| compaction.trigger.strategy | No | num_commits | num_commits / time_elapsed / num_and_time / num_or_time |

### Hive metadata sync parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| hive_sync.enable | No | false | - |
| hive_sync.db | No | - | - |
| hive_sync.table | No | - | - |
| hive_sync.mode | No | jdbc | Valid values: `hms` , `jdbc` , and `hiveql` . |
| hive_sync.username | No | - | - |
| hive_sync.password | No | - | - |
| hive_sync.jdbc_url | No | - | - |

#### More parameters

For more parameters, see Flink Options.

### Parameters when as a source

| Option | Required | Default Value | Description |
|---|---|---|---|
| read.tasks | No | 4 | The parallelism of tasks that do actual read. |
| hoodie.datasource.query.type | No | snapshot | Valid values: `snapshot` , `read_optimized` , and `incremental` . |
| read.streaming.enabled | No | false | - |
| read.streaming.check-interval | No | 60 | The check interval for streaming read in seconds. |
| read.streaming.skip_compaction | No | false | - |
| read.start-commit | No | None | The start commit instant for reading in the format of 'yyyyMMddHHmmss'. It can be set to `earliest` for reading from the earliest instant for streaming read. |
| read.end-commit | No | None | The end commit instant for reading, which does not need to be specifically set. |

### More parameters

For more parameters, see Flink Options.

## COS configurations

No additional configurations are required. You just need to set `path` to the respective cosn path.

## HDFS configurations

### Getting the HDFS JAR package

To write data to Hudi in a Flink SQL task, if the data is stored in HDFS, a JAR package containing HDFS configurations is required to connect Flink to the target HDFS cluster. The steps to get the JAR package and to use it are as follows:

1. Log in to the respective Hive cluster using SSH.
2. Get `hive-site.xml` and `hdfs-site.xml` from the following paths in the EMR Hive cluster.

```
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

3. Package the obtained configuration files in a JAR package.

```
jar -cvf hdfs-xxx.jar hdfs-site.xml
```

4. Check the JAR structure (you can run a Vim command to view it). Make sure the JAR file includes the following information and has the correct structure.

```
vi hdfs-xxx.jar
```

```
META-INF/
META-INF/MANIFEST.MF
hdfs-site.xml
```

## Setting the HDFS user

**Note**

By default, Flink jobs access HDFS with a Flink user. If the Flink user does not have permission to write to HDFS, you can use advanced job parameters to set the accessing user to a user that has write permission or to the super-user `hadoop` .

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

## Kerberos authentication

1. Log in to the cluster master node to get the files `krb5.conf` , `emr.keytab` , `core-site.xml` , and `hdfs-site.xml` in the following paths.

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

2. Package the obtained configuration files in a JAR package.

```
jar cvf hdfs-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml
```

3. Check the JAR structure (run the Vim command `vim hdfs-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
```

4. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
5. Get the Kerberos principal and configure it in advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as follows (use the first): hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- --------------------------------------------------------
  2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. Configure the principal in advanced job parameters.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

## FAQs

### Hive sync

The Hive table sync failed, with an error raised.

```
java.lang.ClassNotFoundException: org.apache.hudi.hadoop.HoodieParquetInputFormat
```

Check whether the Hive environment contains the JAR package required for Hudi. For details, see Hive.

hudi-hadoop-mr-bundle-x.y.z.jar Download.

# Iceberg

Last updated：2023-11-08 16:02:26

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported (use as source and sink) |
| 1.14 | Supported (use as source and sink) |
| 1.16 | Unsupported |

## ## Use cases

This connector can be used as a source or a sink. When used as a source, it does not support an Iceberg source to which data is written with the upsert operations.

## Defining a table in DDL

As a sink:

```
CREATE TABLE `sink` (
  `id` bigint,
  `YCSB_KEY` string,
  `FIELD0` string,
  `FIELD1` string,
  `FIELD2` string,
  `database_name` string,
  `table_name` string,
  `op_ts` timestamp(3),
  `date` string
) PARTITIONED BY (`date`) WITH (
  'connector' = 'iceberg',
  'hdfs://HDFS14979/usr/hive/warehouse',
  'write.upsert.enabled'='false', -- Whether to enable "upsert".
  'catalog-type' = 'hive',
  'catalog-name'='xxx',
  'catalog-database'='xxx',
  'catalog-table'='xxx',
  -- The thrift URI of the Hive metastore, which can be obtained from the configura
  'uri'='thrift://ip:port',
  'engine.hive.enabled' = 'true',
  'format-version' = '2'
);
```

As a source:

```
CREATE TABLE `icesource` (
  `id` bigint,
  `YCSB_KEY` string,
  `FIELD0` string,
  `FIELD1` string,
  `FIELD2` string,
  `database_name` string,
  `table_name` string,
  `op_ts` timestamp(3),
PRIMARY KEY(id) NOT ENFORCED
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = 'hive_catalog',
  'catalog-type' = 'hive',
  'catalog-database' = 'database_ta',
  'catalog-table' = 't_p1_hive3_avro_3',
  'warehouse'='hdfs://HDFS14979/usr/hive/warehouse',
  'engine.hive.enabled' = 'true',
  'format-version' = '2',
  'streaming'='true',
  'monitor-interval'='10',
   -- The thrift URI of the Hive metastore, which can be obtained from the configur
   'uri'='thrift://ip:port'
);
```

## WITH parameters

### Common parameters

| Option | Required | Default Value | Description |
|---|---|---|---|
| connector | Yes | None | Here, it should be `iceberg` . |
| location | Yes | None | The data storage path, in the format of `hdfs://` for data storage in HDFS and `COSN://$bucket/$path` for data storage in COS. |
| catalog-name | Yes | None | A custom catalog name. |
| catalog-type | Yes | None | The catalog type. Valid values: `hadoop` , `hive` , and `custom` . |
| catalog-database | Yes | None | The name of the Iceberg database. |
| catalog-table | Yes | None | The name of the Iceberg table. |
| catalog-impl | No | None | This option is required when `catalog-type` is set to `custom` . |
| uri | No | None | -- The thrift URI of the Hive metastore, which can be obtained from the configuration file hive-site.xm and whose key is "hive-metastore-uris; Eg. thrift://172.28.1.149:7004". |
| format-version | No | 1 | For more Iceberg formats, see Iceberg Table Spec. |

For more options, see Configuration.

## COS configuration

No additional configurations are required. You just need to set `path` to the respective cosn path.

## HDFS configuration

### Getting the HDFS JAR package

To write data to Iceberg in a Flink SQL task, if the data is stored in HDFS, a JAR package containing HDFS configurations is required to connect Flink to the target HDFS cluster. The steps to get the JAR package and to use it are as follows:

1. Log in to the respective Hive cluster using the SSH method.
2. Get `hive-site.xml` and `hdfs-site.xml` from the following paths in the EMR Hive cluster.

```
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

3. Package the obtained configuration files in a JAR package.

```
jar -cvf hdfs-xxx.jar hdfs-site.xml
```

4. Check the JAR structure (run a Vim command to view it). Make sure the JAR file includes the following information and has the correct structure.

```
vi hdfs-xxx.jar
```

```
META-INF/
META-INF/MANIFEST.MF
hdfs-site.xml
```

### Setting the HDFS user

**Note**

By default, Flink jobs access HDFS with a Flink user. If the Flink user does not have permission to write to HDFS, you can use advanced job parameters to set the accessing user to a user that has write permission or to the super-user `hadoop` .

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

## Kerberos authentication

1. Log in to the cluster master node to get the files `krb5.conf` , `emr.keytab` , `core-site.xml` , and `hdfs-site.xml` in the following paths.

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

2. Package the obtained configuration files in a JAR package.

```
jar cvf hdfs-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml
```

3. Check the JAR structure (run the Vim command `vim hdfs-xxx.jar`). Make sure the JAR file includes the following information and has the correct structure.

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
```

4. Upload the JAR file to the Dependencies page of the Stream Compute Service console, and reference the package when configuring job parameters.
5. Get the Kerberos principal and configure it in advanced job parameters.

```
klist -kt /var/krb5kdc/emr.keytab

# The output is as follows (use the first): hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- ------------------------------------------------------
 2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
 2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
 2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
 2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. Configure the principle in advanced job parameters.

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

**Note**

The values of `security.kerberos.login.keytab` and `security.kerberos.login.conf` are the respective file names.

# SQLServer CDC

Last updated：2023-11-08 14:23:39

## Overview

The SQLServer CDC source connector allows for reading snapshot data and incremental data from SQLServer database. This document describes how to set up the SQLServer CDC connector.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported |
| 1.14 | Unsupported |
| 1.16 | Unsupported |

## Limits

The SQLServer CDC connector can be used only as a source.

## Setting up a SQLServer database

Change data capture (CDC) must be enabled on the SQLServer source tables to capture data changes.

1. Enable CDC on a database as instructed in Setting Change Data Capture (CDC).
2. Enable CDC on a SQLServer source table as instructed in sys.sp_cdc_enable_table.

```
USE MyDB
GO

EXEC sys.sp_cdc_enable_table
@source_schema = N'dbo',     -- Specifies the schema of the source table.
@source_name   = N'MyTable', -- Specifies the name of the table that you want to ca
@role_name     = NULL,  -- Specifies a role MyRole to which you can add users to wh
@filegroup_name = NULL, -- Specifies the filegroup where SQL Server places the chan
@supports_net_changes = 0 -- Whether to support querying net changes for the captur
GO
```

3. Check whether CDC is enabled on the source table as instructed in sys.sp_cdc_help_change_data_capture.

```
USE MyDB
GO

EXEC sys.sp_cdc_help_change_data_capture
GO
```

## Defining a table in DDL

```
-- register a SqlServer table 'orders' in Flink SQL
CREATE TABLE orders (
    id INT,
    order_date DATE,
    purchaser INT,
    quantity INT,
    product_id INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'sqlserver-cdc',
    'hostname' = 'localhost',
    'port' = '1433',
    'username' = 'sa',
    'password' = 'Password!',
    'database-name' = 'inventory',
    'schema-name' = 'dbo',
    'table-name' = 'orders'
);

-- read snapshot and binlogs from orders table
SELECT * FROM orders;
```

## WITH parameters

| Option | Required | Default Value | Type | Description |
|---|---|---|---|---|
| connector | Yes | None | String | The value must be `sqlserver-cd` . |
| hostname | Yes | None | String | IP address or hostname of the SQLServer database. |
| username | Yes | None | String | Username to use when connecting to the SQLServer database. |
| password | Yes | None | String | Password to use when connecting to the SQLServer database. |
| database-name | Yes | None | String | Database name of the SQLServer database to monitor. |
| schema-name | Yes | None | String | Schema name of the source table. Regexes in Java are supported. For example, `dbo.*` can match `dbo` , `dbo1` , and `dbo_test` . We recommend you place a regex in brackets to prevent errors when it is combined with `table-name` . |
| table-name | Yes | None | String | Table name of the SQLServer database to monitor. Regexes in Java are supported. We recommend you place a regex in brackets to prevent errors when it is combined with `schema-name` . |
| port | No | 1433 | Integer | Integer port number of the SQLServer database. |
| server-time-zone | No | UTC | String | The session time zone in database server, such as "Asia/Shanghai". |

| | | | | |
|---|---|---|---|---|
| debezium.* | No | None | String | Specifies Debezium properties for fine-grained control of the behaviors on the client, such as `'debezium.snapshot.mode' = 'initial_only'` . For details, see Debezium's SQLServer Connector properties. |

## Available metadata

| Key | Data Type | Description |
|---|---|---|
| table_name | STRING NOT NULL | Name of the database that contains the row. |
| schema_name | STRING NOT NULL | Name of the schema that contains the row. |
| database_name | STRING NOT NULL | Name of the table that contains the row. |
| op_ts | TIMESTAMP_LTZ(3) NOT NULL | It indicates the time that the change was made in the database. For the data during the snapshot phase, the value of this option is `0` . |

Example:

```
CREATE TABLE products (
    table_name STRING METADATA  FROM 'table_name' VIRTUAL,
    schema_name STRING METADATA  FROM 'schema_name' VIRTUAL,
    db_name STRING METADATA FROM 'database_name' VIRTUAL,
    operation_ts TIMESTAMP_LTZ(3) METADATA FROM 'op_ts' VIRTUAL,
    id INT NOT NULL,
    name STRING,
    description STRING,
    weight DECIMAL(10,3)
) WITH (
    'connector' = 'sqlserver-cdc',
    'hostname' = 'localhost',
    'port' = '1433',
    'username' = 'sa',
    'password' = 'Password!',
    'database-name' = 'inventory',
    'schema-name' = 'dbo',
    'table-name' = 'products'
);
```

## Data type mapping

| SQLServer Type | Flink SQL Type |
|---|---|
| char(n) | CHAR(n) |
| varchar(n) nvarchar(n) nchar(n) | VARCHAR(n) |
| text ntext xml | STRING |
| decimal(p, s) money smallmoney | DECIMAL(p, s) |
| numeric | NUMERIC |
| float real | DOUBLE |
| bit | BOOLEAN |
| int | INT |
| tinyint | SMALLINT |
| smallint | SMALLINT |
| bigint | BIGINT |
| date | DATE |
| time(n) | TIME(n) |
| datetime2 datetime smalldatetime | TIMESTAMP(n) |
| datetimeoffset | TIMESTAMP_LTZ(3) |

## Notes

### Can't perform checkpoint during scanning snapshot of tables

During scanning snapshot of database tables, since there is no recoverable position, we can’t perform checkpoints. In order not to perform checkpoints, SQLServer CDC source will keep the checkpoint waiting to timeout. The timeout checkpoint will be recognized as a failed checkpoint, and by default, this will trigger a failover for the Flink job. So, if the database table is large, it is recommended to add the following Flink configurations to avoid failover because of the timeout checkpoints:

```
execution.checkpointing.interval: 10min
execution.checkpointing.tolerable-failed-checkpoints: 100
restart-strategy: fixed-delay
```

```
restart-strategy.fixed-delay.attempts: 2147483647
```

## Single thread reading

The SQLServer CDC source can't work in parallel reading, because there is only one task that can receive change events.

# StarRocks

Last updated：2023-11-08 14:24:26

## Overview

The flink-connector-starrocks connector is based on the open-source starrocks-connector-for-apache-flink v1.2.4. Data can be read from and written to StarRocks via Flink.

## Versions

| Flink Version | Description |
|---|---|
| 1.11 | Unsupported |
| 1.13 | Supported (as a batch source, sink, and dimension table) |
| 1.14 | Supported (as a batch source, sink, and dimension table) |
| 1.16 | Unsupported |

## As a sink

The flink-connector-starrocks connector can be used as a sink to load data into StarRocks. It has a higher and more stable performance than flink-connector-jdbc provided by Apache Flink. It caches data and batch loads it into StarRocks using the Stream Load transaction interface. Two data formats are supported: CSV and JSON.
The following example shows how to load data from a MySQL-CDC connector to StarRocks.

```
CREATE TABLE `mysql_cdc` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',       -- Here, it should be 'mysql-cdc'.
  'hostname' = '9.134.34.15',    -- IP of the database server.
  'port' = '3306',                 --  Integer port number of the database server.
  'username' = 'root',         -- Name of the database to use when connecting to the
  'password' = 'xxx',     -- Password to use when connecting to the MySQL database s
  'database-name' = 'test',    -- Database name of the MySQL server to monitor.
  'table-name' = 'user_behavior'       -- Table name of the MySQL database to monito
);

CREATE TABLE `pk_starrocks`(
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'starrocks',
  'jdbc-url' = 'jdbc:mysql://172.28.28.98:9030',
  'load-url' = '172.28.28.98:8030',
  'database-name' = 'oceanus',
  'table-name' = 'pk_user_behavior',
  'username' = 'root',
  'password' = 'xxx',
  'sink.buffer-flush.interval-ms' = '15000',
  'sink.properties.format' = 'json',
  'sink.properties.strip_outer_array' = 'true', -- Here, it should be "true".
  -- 'sink.parallelism' = '1',
  'sink.max-retries' = '3',
  'sink.semantic' = 'exactly-once'
);

INSERT INTO `pk_starrocks` SELECT * FROM `mysql_cdc`;
```

**Note**

The StarRocks table needs to use the Primary Key model. Otherwise, data deletion from the source table cannot be synced to StarRocks.

You can use the StarRocks Migration Tools (SMT) provided by StarRocks to sync the database and table schema to StarRocks as instructed in Synchronize database & table schema.

## WITH parameters

| Option | Required | Default Value | Data Type | Description |
|---|---|---|---|---|
| connector | Yes | None | String | Here, it should be "starrocks". |
| jdbc-url | Yes | None | String | The address that is used to connect t in the format of `jdbc:mysql://fe_ip1:query_` such as `jdbc:mysql://172.28.28.98:` |
| | | | | |

| | | | | |
|---|---|---|---|---|
| load-url | Yes | None | String | The HTTP URL of the FE in your Star in the format of `fe_ip1:http_por` such as `172.28.28.98:8030`. |
| database-name | Yes | None | String | The name of the StarRocks database |
| table-name | Yes | None | String | The name of the table that you want t |
| username | Yes | None | String | The username of the account that you StarRocks. |
| password | Yes | None | String | The password of the preceding accou |
| sink.semantic | No | at-least-once | String | Valid values: `at-least-once` `exactly-once`, which means dat checkpoint. In this case, `sink.buf` |
| sink.version | No | AUTO | String | Valid values: `V1` : Use `Stream Load` interfac processing. It has a relatively low perf versions. `V2` : Use `Stream Load transa` processing. It requires StarRocks to b `AUTO` : Automatically choose the sir version of StarRocks supports Strean |
| sink.buffer-flush.max-bytes | No | 94371840 (90M) | String | A flush option, which is available only `at-least-once`. When the size c value set here, the flush to StarRocks `10GB]`. |
| sink.buffer-flush.max-rows | No | 500000 | String | A flush option, which is available only `at-least-once`. When the numb the value set here, the flush to StarRo `[64,000, 5000,000]`. |
| sink.buffer-flush.interval-ms | No | 300000 | String | A flush option, which is available only `at-least-once`. This option spe between two flushes to StarRocks. Va |
| sink.max-retries | No | 3 | String | The number of times that the system r job. Value range: `[0, 1000]`. |
| sink.parallelism | No | NULL | String | The parallelism of the connector. If no will be used. |

| | | | | |
|---|---|---|---|---|
| sink.connect.timeout-ms | No | 1000 | String | The timeout (ms) for waiting response `[100, 60000]` . |
| sink.label-prefix | No | No | String | The label prefix used by Stream Load `z0-9]` . For more details of labels, s Load. |
| sink.properties.format | No | CSV | String | The format of data to be loaded to Sta (default) and `JSON` . |
| sink.properties.column_separator | No | \\t | String | The column separator for CSV-format parameters. |
| sink.properties.row_delimiter | No | \\n | String | The row delimiter for CSV-formatted c parameters. |
| sink.properties.strip_outer_array | No | false | String | Specifies whether to strip the outermo formatted data. Valid values: `true` load from Flink, the JSON data may h indicated by a pair of square brackets **recommend that you set this para** StarRocks removes the outermost sq array as a separate data record. If you StarRocks parses the entire JSON da array as a single data record. For exa `[ {"category" : 1, "author` `"author" : 4} ]` . If you set this parses `{"category" : 1, "aut` `: 3, "author" : 4}` into separa separate StarRocks table rows. For d |
| sink.properties.* | No | None | String | The parameters that are used to contr `'sink.properties.columns'` onwards, Primary Key model support more parameters, see STREAM LOA |

**Note**

If you set `sink.semantic` to `at-least-once` , flush to StarRocks will be triggered when `sink.buffer-flush.max-bytes` , `sink.buffer-flush.max-rows` , or `sink.buffer-flush.interval-ms` is met. If you set `sink.semantic` to `exactly-once` , depending on Flink checkpointing, the data and its labels will be saved during each checkpoint. After the checkpointing is completed, more writes to the database are blocked, and all the data cached in state will be flushed to the sink in the first invocation to guarantee the exactly-once semantic. In this situation, `sink.buffer-flush.*` options are invalid.

### Data type mapping (as a sink)

| Flink Type | StarRocks Type |
|---|---|
| BOOLEAN | BOOLEAN |
| TINYINT | TINYINT |
| SMALLINT | SMALLINT |
| INTEGER | INTEGER |
| BIGINT | BIGINT |
| FLOAT | FLOAT |
| DOUBLE | DOUBLE |
| DECIMAL | DECIMAL |
| BINARY | INT |
| CHAR | STRING |
| VARCHAR | STRING |
| STRING | STRING |
| DATE | DATE |
| TIMESTAMP_WITHOUT_TIME_ZONE(N) | DATETIME |
| TIMESTAMP_WITH_LOCAL_TIME_ZONE(N) | DATETIME |
| ARRAY<T> | ARRAY<T> |
| MAP<KT,VT> | JSON STRING |
| ROW<arg T...> | JSON STRING |

**Note**: BYTES, VARBINARY, TIME, INTERVAL, MULTISET, and RAW in Flink are not supported. For details, see Data Types.

## Notes

### StarRocks data models

StarRocks provides four data models as shown in Data Models: Duplicate Key, Aggregate Key, Unique Key, and Primary Key. The Primary Key model supports the pushdown of predicates and indexes. As such, the Primary Key model can deliver high query performance despite real-time and frequent data updates. Therefore, if you have no special needs, the Primary Key model is recommended.
For upsert streams, the Primary Key model must be used. Otherwise, DELETE messages cannot be flushed to StarRocks.
Compared with the Unique Key model based on the Merge-On-Read policy, the Primary Key model improves the query performance by 3 to 10 times.
The Primary Key model can join multiple streams by performing update operations on individual columns.

## As a source

### WITH parameters

| Option | Required | Default Value | Data Type | Description |
|---|---|---|---|---|
| connector | Yes | None | String | Here, it should be "starrocks". |
| scan-url | Yes | None | String | The HTTP URL of the FE in your StarRocks `fe_ip1:http_port,fe_ip2:http_po` `172.28.28.98:8030` . |
| jdbc-url | Yes | None | String | The address that is used to connect the My `jdbc:mysql://fe_ip1:query_port,` such as `jdbc:mysql://172.28.28.9` |
| username | Yes | None | String | The username of your StarRocks cluster ac |
| password | Yes | None | String | The password of your StarRocks cluster acc |
| database-name | Yes | None | String | The name of the StarRocks database to whi want to read belongs. |
| table-name | Yes | None | String | The name of the StarRocks table you want t |
| scan.connect.timeout-ms | No | 1000 | String | The maximum amount of time (in millisecond from the Flink connector to your StarRocks |
| scan.params.keep-alive-min | No | 10 | String | The maximum amount of time (in minutes) d keeps alive. |
| scan.params.query-timeout-s | No | 600(5min) | String | The maximum amount of time (in seconds) a out. |

| | | | | |
|---|---|---|---|---|
| scan.params.mem-limit-byte | No | 102410241024 (1G) | String | The maximum amount of memory allowed p |
| scan.max-retries | No | 1 | String | The maximum number of times that the reac failures. |
| lookup.cache.ttl-ms | No | 5000 | Long | The maximum amount of time (in milliseconc cache of the dimension table times out. |

### As a batch source

```
CREATE TABLE `starrocks` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'starrocks' ,
  'jdbc-url' = 'jdbc:mysql://172.28.28.98:9030', -- query_port, FE mysql server por
  'scan-url' = '172.28.28.98:8030', -- http_port
  'database-name' = 'oceanus',
  'table-name' = 'pk_user_behavior',
  'username' = 'root',
  'password' = 'xxx'
);

CREATE TABLE `print_sink` (
  `user_id` BIGINT,
  `item_id` BIGINT,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'logger'
);

INSERT INTO `print_sink`
SELECT * FROM starrocks;
```

### As a dimension table

```
CREATE TABLE `starrocks` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'starrocks' ,
  'jdbc-url' = 'jdbc:mysql://172.28.28.98:9030', -- query_port, FE mysql server por
  'scan-url' = '172.28.28.98:8030', -- http_port
  'database-name' = 'oceanus',
  'table-name' = 'pk_user_behavior',
  'username' = 'root',
  'password' = 'xxx'
);

CREATE TABLE `datagen` (
  `user_id` BIGINT,
  `ts` as PROCTIME(),
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1',
  'fields.user_id.min' = '1',
  'fields.user_id.max' = '20'
);

CREATE TABLE `print_sink` (
  `user_id` BIGINT,
  `item_id` BIGINT,
  `behavior` STRING,
  `ts` TIMESTAMP,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'logger'
);

INSERT INTO `print_sink`
SELECT a.user_id,b.item_id,b.behavior,a.ts
FROM `datagen` a LEFT JOIN `starrocks` FOR SYSTEM_TIME AS OF a.ts as b
ON a.user_id = b.user_id;
```

## Data type mapping (as a source)

| StarRocks | Flink |
|---|---|
| NULL | NULL |
| BOOLEAN | BOOLEAN |
| TINYINT | TINYINT |
| SMALLINT | SMALLINT |
| INT | INT |
| BIGINT | BIGINT |
| LARGEINT | STRING |

| | |
|---|---|
| FLOAT | FLOAT |
| DOUBLE | DOUBLE |
| DATE | DATE |
| DATETIME | TIMESTAMP |
| DECIMAL | DECIMAL |
| DECIMALV2 | DECIMAL |
| DECIMAL32 | DECIMAL |
| DECIMAL64 | DECIMAL |
| DECIMAL128 | DECIMAL |
| CHAR | CHAR |
| VARCHAR | STRING |

# SET Statement

# Flink Configuration Items

Last updated：2023-11-08 14:23:08

## Overview

The `SET` statement can be used to adjust some key job parameters. Currently, you can configure most parameters in advanced job parameters. Only a few need to be set using the SET statement.

**Note**

SET is an advanced statement. Improper configuration may cause your job to fail, so use it only when necessary.

Comments ( `--` ) are not supported for the SET statement.

A semicolon is required at the end of a SET clause.

## Syntax

In a SET clause, string-type values must be quoted with single quotation marks. This is not necessary for Boolean or numeric values.

```
SET option = 'Value';
```

## Example

### Idle State Retention Time

For statements with large state, such as GROUP BY and JOIN, Flink offers the Idle State Retention Time mechanism. You can specify the minimum and maximum state retention time to avoid OOM caused by the continuous increase of states. The clause below sets the minimum retention time to 5 hours and the maximum retention time to 6 hours.

**Note**

The minimum and maximum retention time must be at least 5 minutes apart; otherwise, an error will occur and Flink will ignore the settings.

The format of time units should follow Flink's requirements. For example, you can set the time to `10min` , `3h` , `3hour` , `7day` , or `7d` . Spaces between numbers and time units are optional.

```
SET execution.min-idle-state-retention = '5 h';
SET execution.max-idle-state-retention = '6 h';
```

### Checkpoint timeout for SQL jobs

By default, the minimum timeout period for SQL jobs is 10 minutes and the maximum period is twice the checkpoint interval. That is to say, the timeout period for a job whose checkpoint interval is 60 seconds must be at least 10 minutes, and the timeout period for a job whose checkpoint interval is 10 minutes cannot exceed 20 minutes.
To customize a checkpoint timeout period, use the SET clause below.

```
SET CHECKPOINT_TIMEOUT = '300 s';
```

**Note**

The Flink option (advanced parameter) `execution.checkpointing.timeout` may not take effect for SQL jobs. Please use the SET clause above to configure the timeout period.

### Mini-batch

Flink SQL supports mini-batch aggregation, which can significantly increase throughput. Mini-batch may increase delay and is therefore disabled by default. To enable mini-batch, add the following clauses to the edit page of your SQL job (the batch size and latency can be modified but cannot be omitted):

```
set table.exec.mini-batch.enabled = true;
set table.exec.mini-batch.size = 5000;
set table.exec.mini-batch.allow-latency = '200 ms';
```

# Operators and Built-in Functions

## Overview

Last updated：2023-11-08 11:32:33

This section introduces the built-in operators and functions of Stream Compute Service. Based on open-source Flink functions, Stream Compute Service has supplemented and optimized some functions. For more information, see Comparison with Flink Built-in Functions.

**Function naming conventions are described as follows**:

`{ }` means an optional parameter and any option can be used. For example, `{BOTH|LEADING|TRAILING}` means that you can leave it blank (use the default behavior), or choose any one of BOTH, LEADING, or TRAILING. The feature varies depending on the option.

`[ ]` means an optional parameter, and `*` means it can be repeated zero or more times. For example, `[, value]*` means there can be zero or more `value` following it. This is often used to represent an uncertain number of parameters, such as `value, value, value ... value`.

A string consisting of lowercase letters represents a variable, for example, value1, value, boolean, or numeric. Uppercase letters (such as IS NULL) or symbols (such as =, <) represent operators or built-in functions.

**Categories of built-in functions**:

Comparison Functions
Logical Functions
Arithmetic Functions
String Functions
Conditional Functions
Type Conversion Functions
Date and Time Functions
Aggregate Functions
Time Window Functions
Other Functions

# Comparison with Flink Built-in Functions

Last updated：2023-11-08 14:20:43

Based on the open-source Flink functions, Stream Compute Service has supplemented and optimized some functions. The following table lists some built-in functions.

| Built-in Function | Stream Compute Service Definition | Flink Definition | Remar |
|---|---|---|---|
| unbase64 | unbase64(string) | - | Strean Base6 string |
| decode | DECODE(binary,charset), DECODE(binary) | DECODE(binary, string) | Strean DECO 8” by Flink: is NUL |
| split | SPLIT(string, separator) | - | Strean `stri` returns |
| get_row_field_str | GET_ROW_FIELD_STR(row, index) | - | Strean value object The re |
| get_row_arity | GET_ROW_ARITY(row) | - | Strean numbe `row` |
| can_cast_to | CAN_CAST_TO(str, type) | - | Strean whethe conver `type` The re conditi examp `CAN_` `'INTE` `CAN_` `'DOUE` |
| date_format_simple | DATE_FORMAT_SIMPLE(timestamp, simple_format) | - | Strean field of |

| | | | |
|---|---|---|---|
| | | | repres millise format Simple `DATE` `'yyyy` returns 12:13: the out |
| to_timestamp | TO_TIMESTAMP(string, simple_format) | TO_TIMESTAMP(string1[, string2]) | Stream `stri` `simp` compa Simple default `TO_T HH:mr and th Flink: S |
| timestamp_to_long | TIMESTAMP_TO_LONG(timestamp), TIMESTAMP_TO_LONG(timestamp, mode) | - | Stream TIMES value c `mode` Unix ti examp any ot conver millise 15484 |
| get_tumble_start | get_tumble_start(timestamp,windowSize), get_tumble_start(timestamp,offset,windowSize) | - | Stream start ti offset i |
| if_null_str | IF_NULL_STR(str, defaultValue) | - | Stream value c otherw `defa` |
| get_json_object | GET_JSON_OBJECT(json_str, path_str) | - | Stream nested view a |
| split_index | SPLIT_INDEX(string, separator, index) | SPLIT_INDEX(string1, | Splits |

| | | | |
|---|---|---|---|
| | | string2, integer1) | and re (VARC `inde` Strean `stri` returns Flink: I left em |
| first_str | FIRST_VALUE(expression) | FIRST_VALUE(expression) | Strean first va Flink: S |
| last_str | LAST_VALUE(expression) | LAST_VALUE(expression) | Strean last va Flink: S |
| explode | EXPLODE(inputStr, separator) | - | Strean string i multipl functio keywo referer tempo JOIN. |

# Comparison Functions

Last updated：2023-11-08 14:21:14

Comparison functions are described as follows:

| Function | Description |
|---|---|
| value1 = value2 | Returns TRUE if `value1` is equal to `value2` and `FALSE` otherwise.<br>Returns NULL if `value1` or `value2` is NULL. In WHERE conditions, NULL is treated as FALSE. Therefore, it is recommended to use IS NULL instead of = NULL when comparing a value with NULL.<br>The difference between = and IS NOT DISTINCT FROM lies in how they handle NULL values. |
| value1 <> value2 | Returns TRUE if `value1` is not equal to `value2` ; otherwise FALSE. |
| value1 > value2 | Returns TRUE if `value1` is greater than `value2` ; otherwise FALSE. |
| value1 >= value2 | Returns TRUE if `value1` is greater than or equal to `value2` ; otherwise FALSE. |
| value1 < value2 | Returns TRUE if `value1` is less than `value2` ; otherwise FALSE. |
| value1 <= value2 | Returns TRUE if `value1` is less than or equal to `value2` ; otherwise FALSE. |
| value IS NULL | Returns TRUE if `value` is NULL; otherwise FALSE. |
| value IS NOT NULL | Returns TRUE if `value` is not NULL; otherwise FALSE. |
| value1 IS DISTINCT FROM value2 | Returns TRUE if two values are different; otherwise FALSE. NULL values are treated as identical. |
| value1 IS NOT DISTINCT FROM value2 | Returns TRUE if two values are equal; otherwise FALSE. NULL values are treated as identical. |
| value1 BETWEEN [ ASYMMETRIC \| SYMMETRIC ] value2 AND value3 | Returns TRUE if `value1` is greater than or equal to `value2` and less than or equal to `value3` ; otherwise FALSE.<br><br>[ ] represents an optional parameter. The default value is ASYMMETRIC. When the SYMMETRIC keyword is explicitly |

| | |
|---|---|
| | declared, the values of `value2` and `value3` can be interchanged without affecting the result. |
| value1 NOT BETWEEN [ ASYMMETRIC \| SYMMETRIC ] value2 AND value3 | Returns TRUE if `value1` is less than `value2` or greater than `value3` ; otherwise FALSE.<br><br>[ ] represents an optional parameter. The default value is ASYMMETRIC. When the SYMMETRIC keyword is explicitly declared, the values of `value2` and `value3` can be interchanged without affecting the result. |
| string1 LIKE string2 | Returns TRUE if `string1` matches pattern `string2` ; otherwise FALSE. |
| string1 NOT LIKE string2 | Returns TRUE if `string1` does not match pattern `string2` ; otherwise FALSE. |
| string1 SIMILAR TO string2 | Returns TRUE if `string1` matches regular expression `string2` ; otherwise FALSE. |
| string1 NOT SIMILAR TO string2 | Returns TRUE if `string1` does not match regular expression `string2` ; otherwise FALSE. |
| value IN (listItem [, listItem]* ) | Returns TRUE if `value` exists in the given list. This statement is equivalent to connecting multiple OR expressions.<br>When the list contains NULL, returns NULL if `value` cannot be found and FALSE otherwise.<br>Always returns NULL if `value` is NULL. |
| value NOT IN (listItem, [, listItem]*) | Returns TRUE if `value` does not exist in the given list; otherwise FALSE. |
| EXISTS (sub-query) | Returns TRUE if sub-query returns at least one row; otherwise FALSE.<br>This query can cause memory pressure. |
| value IN (sub-query) | Returns TRUE if `value` is equal to a row returned by sub-query.<br>This query can cause memory pressure. |
| value NOT IN (sub-query) | Returns TRUE if `value` is not equal to a row returned by sub-query.<br>This query can cause memory pressure. |

# Logic Functions

Last updated：2023-11-08 14:20:13

Logical functions are used to perform logical operations, and the result is a Boolean value.
Three logical statuses are available: TRUE, FALSE, and UNKNOWN (represented by NULL). Therefore, NOT TRUE is not necessarily FALSE; it can also be UNKNOWN.

| Function | Description |
|---|---|
| boolean1 OR boolean2 | Returns TRUE if `boolean1` or `boolean2` is TRUE. |
| boolean1 AND boolean2 | Returns TRUE if `boolean1` and `boolean2` are both TRUE. |
| NOT boolean | Returns TRUE if `boolean` is FALSE; returns FALSE if `boolean` is TRUE; returns UNKNOWN if `boolean` is UNKNOWN. |
| boolean IS FALSE | Returns TRUE if `boolean` is FALSE; returns FALSE if `boolean` is UNKNOWN. |
| boolean IS NOT FALSE | Returns TRUE if `boolean` is not FALSE; returns TRUE if `boolean` is UNKNOWN. |
| boolean IS UNKNOWN | Returns TRUE if `boolean` is UNKNOWN; otherwise FALSE. |
| boolean IS NOT UNKNOWN | Returns TRUE if `boolean` is not UNKNOWN; otherwise FALSE. |

# Arithmetic Functions

Last updated：2023-11-08 14:19:38

Arithmetic functions are described as follows:

| Function | Description |
|---|---|
| +numeric | Returns the value of `numeric` itself. |
| -numeric | Returns the negative value of `numeric` . |
| numeric1 + numeric2 | Returns `numeric1` plus `numeric2` . |
| numeric1 - numeric2 | Returns `numeric1` minus `numeric2` . |
| numeric1 * numeric2 | Returns `numeric1` multiplied by `numeric2` . |
| numeric1 / numeric2 | Returns `numeric1` divided by `numeric2` . |
| POWER(numeric1, numeric2) | Returns `numeric1` raised to the power of `numeric2` . |
| ABS(numeric) | Returns the absolute value of `numeric` . |
| MOD(numeric1, numeric2) | Returns the remainder of `numeric1` divided by `numeric2` . The result is negative if `numeric1` is negative. |
| SQRT(numeric) | Returns the square root of `numeric` . |
| LN(numeric) | Returns the natural logarithm (base e) of `numeric` . |
| LOG10(numeric) | Returns the base 10 logarithm of `numeric` . |
| LOG2(numeric) | Returns the base 2 logarithm of `numeric` . |
| LOG(numeric2) LOG(numeric1, numeric2) | When called with one parameter, returns the natural logarithm of `numeric2` (equivalent to `LN` ). When called with two parameters, returns the logarithm of `numeric2` to the base `numeric1` . |
| EXP(numeric) | Returns e raised to the power of `numeric` . |
| CEIL(numeric) CEILING(numeric) | Returns the smallest integer value that is larger than or equal to `numeric` . |
| FLOOR(numeric) | Returns the largest integer value that is less than or equal to `numeric` . |
| TRUNCATE(numeric1, numeric2) | Returns `numeric1` truncated to `numeric2` decimal places. |

| | |
|---|---|
| | Returns NULL if any parameter is NULL.<br>If `numeric2` is 0 or left empty, the result has no decimal point or fractional part. `numeric2` can be negative to cause numeric2 digits left of the decimal point of the value to become zero. For example, `TRUNCATE(42.345, 2)` returns `42.34` , `TRUNCATE(42.345)` returns `42.0` , and `TRUNCATE(42.345, -1)` returns `40.0` . |
| SIN(numeric) | Returns the sine of `numeric` . |
| SINH(numeric) | Returns the hyperbolic sine of `numeric` . The return value is of DOUBLE type. |
| COS(numeric) | Returns the cosine of `numeric` . |
| COSH(numeric) | Returns the hyperbolic cosine of `numeric` . The return value is of DOUBLE type. |
| TAN(numeric) | Returns the tangent of `numeric` . |
| TANH(numeric) | Returns the hyperbolic tangent of `numeric` . The return value is of DOUBLE type. |
| COT(numeric) | Returns the cotangent of `numeric` . |
| ASIN(numeric) | Returns the arc sine of `numeric` . |
| ACOS(numeric) | Returns the arc cosine of `numeric` . |
| ATAN(numeric) | Returns the arc tangent of `numeric` . |
| ATAN2(numeric1, numeric2) | Returns the arc tangent of a coordinate (numeric1, numeric2). |
| DEGREES(numeric) | Returns the degree representation of a radian `numeric` . |
| RADIANS(numeric) | Returns the radian representation of a degree `numeric` . |
| SIGN(numeric) | Returns the signum of `numeric` . Returns `-1` for a negative number, `0` for 0, and `1` for a positive number. |
| ROUND(numeric, int) | Returns a number rounded to `int` decimal places for `numeric` . |
| PI() | Returns a value that is closer than any other values to pi. |
| E() | Returns a value that is closer than any other values to e. |
| RAND() RAND(seed value) | Returns a pseudorandom value in the range [0.0, 1.0). You can specify an integer as the seed value. |

| | |
|---|---|
| RAND_INTEGER(upper limit) RAND_INTEGER(seed value, upper limit) | Returns a pseudorandom value in the range [0.0, upper limit). |
| UUID() | Returns a Universally Unique Identifier (UUID) string according to type 4 (pseudo randomly generated) UUID. |
| LOG(numeric) LOG(base, numeric) | Returns the natural logarithm of `numeric` or the logarithm of `numeric` ·to `base` . |
| BIN(numeric) | Returns a string representation of `numeric` in binary format. For example, `BIN(4)` returns "100". |
| HEX(numeric) HEX(string) | Returns a string representation of `numeric` or `string` in hex format. For example, `HEX(15)` or `HEX("15")` returns "F". |

# Condition Functions

Last updated：2023-11-08 14:19:01

Conditional functions are described as follows:

| Function | Description |
|---|---|
| CASE valueWHEN value1 [, value11 ] *THEN result1[ WHEN valueN [, valueN1 ]* THEN resultN ]* [ ELSE resultZ ] END | Returns result1 when the value is contained in the range of value1 to value11.<br>Returns resultN when the value is contained in the range of valueN to valueN1.<br>Returns resultZ when no value matches. |
| CASEWHEN condition1 THEN result1[ WHEN conditionN THEN resultN ] * [ ELSE resultZ ] END | Returns result1 when condition1 is met.<br>Returns resultN when conditionN is met.<br>Returns resultZ when no condition is met. |
| NULLIF(value1, value2) | Returns NULL if `value1` is equal to `value2` ; returns `value1` otherwise. For example, `NULLIF(5, 5)` returns `NULL` , and `NULLIF(5, 0)` returns `5` . |
| COALESCE(value, value [, value ]* ) | Returns the first parameter that is not NULL. For example, `COALESCE(NULL, 5)` returns `5` . |
| IF(condition, true_value, false_value) | Returns the true_value if `condition` is met, otherwise the false_value. For example, `IF(2 > 1, 2, 1)` returns `2` , and `IF (1 > 2, 99, 100)` returns `100` . |
| IS_ALPHA(string) | Checks whether `string` contains only letters. Returns `true` if so, otherwise `false` . |
| IS_DECIMAL(string) | Checks whether `string` is a valid number (integer, decimal, or negative number). Returns `true` if so, otherwise `false` . |
| IS_DIGIT(string) | Checks whether `string` contains only digits (i.e., an unsigned integer). Returns `true` if so, otherwise `false` . |
| IF_NULL_STR(str, defaultValue) | Returns the value of `str` itself if it is not NULL; otherwise, returns the value of `defaultValue` . |

# String Functions

Last updated：2023-11-08 14:17:35

## Functions

| Function | Description |
|---|---|
| string1 \|\| string2 | Returns the concatenation of `string1` and `string2` . This is equivalent to `CONCAT(string1, string2)` . |
| CHAR_LENGTH(string) | Returns the number of characters in `string` . |
| CHARACTER_LENGTH(string) | Same as `CHAR_LENGTH(string)` . |
| UPPER(string) | Returns `string` in uppercase. |
| LOWER(string) | Returns `string` in lowercase. |
| POSITION(string1 IN string2) | Returns the position (**starts from 1**) of the first occurrence of `string1` in `string2` ; returns 0 if `string1` cannot be found in `string2` . |
| TRIM({BOTH \|LEADING \|TRAILING }string1 FROM string2 ) | Returns a string that removes leading and/or trailing characters `string1` from `string2` . By default, spaces at both sides are removed. |
| LTRIM(string) | Returns a string that removes the left spaces from `string` . For example, `LTRIM(' Hello')` returns `'Hello'` . |
| RTRIM(string) | Returns a string that removes the right spaces from `string` . For example, `RTRIM(' World ')` returns `' World'` . |
| REPEAT(string, integer) | Returns a string that repeats the base string `integer` times. For example, `REPEAT('Meow', 3)` returns `'MeowMeowMeow'` . |
| REGEXP_REPLACE(string1, string2, string3) | Returns a string from `string1` with all the substrings that match a regular expression `string2` being replaced with `string3` . For example, `REGEXP_REPLACE('banana', 'a\|n', 'A')` returns `'bAAAAA'` . |
| REPLACE(string1, string2, string3) | Returns a string from `string1` with all the substrings that match `string2` being replaced with `string3` . For example, `REPLACE('banana', 'a', 'A')` returns `'bAnAnA'` . |

| | |
|---|---|
| OVERLAY(string1 PLACING string2 FROM start_pos [ FOR length ]) | Returns a string that replaces `length` characters of `string1` with `string2` from position `start_pos` (**starts from 1**). |
| SUBSTRING(string from pos [ FOR length]) | Returns a substring of `string` starting from position `pos` with `length` (to the end by default). **pos starts from 1** instead of 0. |
| REGEXP_EXTRACT(string1, string2[, integer]) | Returns a string from `string1` which extracted with a specified regular expression `string2` and a regex match group index integer. You can specify the group index (starts from 1) using the third parameter `integer` . If you do not specify the group index or specify it as 0, the string that matches the whole regex is returned. For example, `REGEXP_EXTRACT('foothebar', 'foo(.*?)(bar)', 2)` returns `'bar'` . |
| INITCAP(string) | Returns a new form of `string` with the first character of each word converted to uppercase and the remaining characters to lowercase. For example, `INITCAP('i have a dream')` returns `'I Have A Dream'` . |
| CONCAT(string1, string2 ...) | Returns a string that concatenates multiple strings (string1, string2, ...). Returns NULL if any string is NULL. |
| CONCAT_WS(separator, string1, string2, ...) | Returns a string that concatenates multiple strings (string1, string2, ...) with `separator` . Returns NULL if `separator` is NULL. Automatically skips NULL strings, but not empty strings. For example, `CONCAT_WS('~', 'AA','BB', '', 'CC')` returns `AA~BB~~CC` . |
| LPAD(text, length, padding) | Returns a new string from `text` left-padded with `padding` to `length` characters. If the length of `text` is longer than `length` , returns `text` shortened to `length` characters. |
| RPAD(text, length, padding) | Returns a new string from `text` right-padded with `padding` to `length` characters. If the length of `text` is longer than `length` , returns `text` shortened to `length` characters. |
| FROM_BASE64(string) | Returns the Base64-decoded result from `string` . Returns NULL if `string` is NULL. |
| TO_BASE64(string) | Returns the Base64-encoded result from `string` . |
| ASCII(string) | Returns the ASCII code of the first character in `string` . Returns NULL if `string` is NULL. For example, `ASCII('an apple')` returns `97` (the first character `a` corresponds to `97` ). |

| | |
|---|---|
| CHR(integer) | Returns the ASCII character corresponding to `integer` . For example, `CHR(97)` returns `a` . |
| ENCODE(string, charset) | Encodes `string` into a BINARY using the provided character set `charset` . Example: `ENCODE(hello, 'GBK')` |
| DECODE(binary, charset) | Decodes `binary` into a string using the provided character set `charset` . Example: `DECODE(binary_field, 'UTF-16LE')` |
| INSTR(string1, string2) | Returns the position of the first occurrence of `string2` in `string1` . Returns NULL if any parameter is NULL. |
| LEFT(string, n) | Returns the leftmost `n` characters from `string` . Returns an empty string if `n` is negative. Returns NULL if any parameter is NULL. |
| RIGHT(string, n) | Returns the rightmost `n` characters from `string` . Returns an empty string if `n` is negative. Returns NULL if any parameter is NULL. |
| LOCATE(string1, string2[, integer]) | Returns the position of the first occurrence of `string1` in `string2` after position `integer` (**parameters are in reverse order compared with the INSTR function**). Returns 0 if `string1` is not found. Returns NULL if any parameter is NULL. |
| REGEXP(string, regex) | Returns TRUE if any substring of `string` matches the regular expression `regex` , otherwise FALSE. Returns NULL if any parameter is NULL. |
| REVERSE(string) | Returns the reversed string. Returns NULL if any parameter is NULL. |
| SPLIT_INDEX(string, separator, index) | Splits `string` by `separator` , and returns the indexth string (VARCHAR) of the split strings. **index starts from 0**. |
| SPLIT(string, separator) | Splits `string` by `separator` , and returns a Row object. |
| STR_TO_MAP(string1[, string2, string3]) | Returns `MAP<string, string>` after splitting `string1` into key-value pairs using separators. `string2` is the pair separator (default: `,` ), and `string3` is the key-value separator (default: `=` ). For example, `STR_TO_MAP('k1=v1,k2=v2,k3=v3')` returns key-value pairs {'k1': 'v1', 'k2': 'v2', 'k3': 'v3'}. |
| SUBSTR(string[, pos[, length]]) | Returns a substring of `string` starting from position `pos` with `length` (to the end by default). |
| EXPLODE(inputStr, separator) | Splits a string into a temporary table with multiple rows. This function is a table function, and you need to use the keyword `LATERAL TABLE ( )` to reference this dynamically generated temporary table as the right table for JOIN. |

| | |
|---|---|
| GET_ROW_ARITY(row) | Gets the number of columns for Row object `row` . |
| GET_ROW_FIELD_STR(row, index) | Gets the value of the indexth column in Row object `row` . **index starts from 0**. The return value is of VARCHAR type. |
| GET_JSON_OBJECT(json_str, path_str) | Gets the elements of a JSON string `json_str` at the JSON path specified by `path_str` , which can be arbitrarily nested. Supported JSONPath syntax: `$` for root object, `.` for a child element, `[]` for an array index, and `*` as a wildcard for array index []. |
| IS_ALPHA(content) | Checks whether a string contains only letters. |
| IS_DIGITS(content) | Checks whether a string contains only digits. |
| MD5(string) | Returns the MD5 hash of `string` . |
| | |
| SHA1 | Returns the SHA-1 hash of `expr` . |
| SHA256 | Returns the SHA-256 hash of `expr` . |

## Examples

### ||

**Feature description**: Returns the concatenation of `string1` and `string2` . This is equivalent to `CONCAT(string1, string2)` .

**Syntax**: string1 || string2

**Example statement**: SELECT string1 || string2 FROM Test;

**Test data and result**:

| Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Result (VARCHAR) |
|---|---|---|
| Oce | anus | Oceanus |

### CHAR_LENGTH

**Feature description**: Returns the number of characters in `string` .

**Syntax**: CHAR_LENGTH(string)

**Example statement**: SELECT CHAR_LENGTH(var1) AS length FROM Test;

**Test data and result**:

| | |
|---|---|

| Test Data (VARCHAR var1) | Test Result (INT length) |
|---|---|
| Oceanus | 7 |

## CHARACTER_LENGTH

**Feature description**: Same as `CHAR_LENGTH(string)` .

**Syntax**: CHARACTER_LENGTH(string)

**Example statement**: SELECT CHAR_LENGTH(var1) AS length FROM Test;

**Test data and result**:

| Test Data (VARCHAR var1) | Test Result (INT length) |
|---|---|
| Oceanus | 7 |

## LOWER

**Feature description**: Returns `string` in lowercase.

**Syntax**: LOWER(string)

**Example statement**: SELECT LOWER(var1) AS lower FROM Test;

**Test data and result**:

| Test Data (VARCHAR var1) | Test Result (VARCHAR lower) |
|---|---|
| OCeanus | oceanus |

## UPPER

**Feature description**: Returns `string` in uppercase.

**Syntax**: UPPER( string)

**Example statement**: SELECT UPPER(var1) AS upper FROM Test;

**Test data and result**:

| Test Data (VARCHAR var1) | Test Result (VARCHAR upper) |
|---|---|
| OCeanus | OCEANUS |

## TRIM

**Feature description**: Returns a string that removes leading and/or trailing characters `string1` from `string2` . By default, spaces at both sides are removed.

**Syntax**: TRIM({BOTH | LEADING | TRAILING } string1 FROM string2 )

**Example statement**: SELECT TRIM(BOTH string1 FROM string2) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Result (VARCHAR res) |
|---|---|---|
| a | aoceanusa | oceanus |

## CONCAT

**Feature description**: Concatenates two or more strings. Automatically skips NULL parameters.

**Syntax**: CONCAT( string1, string2 ...)

**Example statement**: SELECT CONCAT('123', '456', 'abc', 'def') AS res FROM Test;

**Test data and result**: '123456abcdef'

| Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Data (VARCHAR string3) | Test Data (VARCHAR string4) | Test Result (VARCHAR res) |
|---|---|---|---|---|
| 123 | 456 | abc | def | 123456abcdef |

## CONCAT_WS

**Feature description**: Returns a string that concatenates multiple strings (string1, string2, ...) with `separator`. Returns NULL if `separator` is NULL. Automatically skips NULL strings, but not empty strings.

**Syntax**: CONCAT_WS(separator, string1, string2, ...)

**Example statement**: SELECT CONCAT_WS(separator, string1,string2, string3) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR separator) | Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Data (VARCHAR string3) | Test Result (VARCHAR res) |
|---|---|---|---|---|
| - | AA | BB | CC | AA-BB-CC |

## INITCAP

**Feature description**: Returns a new form of `string` with the first character of each word converted to uppercase and the remaining characters to lowercase.

**Syntax**: INITCAP(string)

**Example statement**: SELECT INITCAP(var1) AS str FROM Test;

**Test data and result**:

| Test Data (VARCHAR var1) | Test Result (VARCHAR str) |
|---|---|
| i have a dream | I Have A Dream |

## IS_ALPHA

**Feature description**: Checks whether a string contains only letters.
**Syntax**: IS_ALPHA(content)
**Example statement**: SELECT IS_ALPHA(content) AS result FROM Test;
**Test data and result**:

| Test Data (VARCHAR content) | Test Result (BOOLEAN result) |
|---|---|
| Oceanus | true |
| oceanus123 | false |
| '' | false |
| null | false |

## IS_DIGITS

**Feature description**: Checks whether a string contains only digits.
**Syntax**: IS_DIGITS(content)
**Example statement**: SELECT IS_DIGITS(content) AS result FROM Test;
**Test data and result**:

| Test Data (VARCHAR content) | Test Result (BOOLEAN case_result) |
|---|---|
| 58.0 | true |
| 58 | true |
| 58pl | false |
| '' | false |
| null | false |

## LPAD

**Feature description**: Returns a new string from `text` left-padded with `padding` to `length` characters. If the length of `text` is longer than `length` , returns `text` shortened to `length` characters.
**Syntax**: LPAD(text , length , padding)
**Example statement**: SELECT LPAD(text, length, padding) AS res FROM Test;
**Test data and result**:

| Test Data (VARCHAR text) | Test Data (INT length) | Test Data (VARCHAR padding) | Test Result (VARCHAR res) |
|---|---|---|---|
| oceanus | 3 | hello | hel |

| | | | |
|---|---|---|---|
| oceanus | -1 | hello | '' |
| oceanus | 12 | hello | hellooceanus |

## RPAD

**Feature description**: Returns a new string from `text` right-padded with `padding` to `length` characters. If the length of `text` is longer than `length` , returns `text` shortened to `length` characters.

**Syntax**: RPAD(text , length , padding)

**Example statement**: SELECT RPAD(text, length, padding) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR text) | Test Data (INT length) | Test Data (VARCHAR padding) | Test Result (VARCHAR res) |
|---|---|---|---|
| oceanus | 3 | hello | oce |
| oceanus | -1 | hello | '' |
| oceanus | 12 | hello | oceanushello |

## MD5

**Feature description**: Returns the MD5 hash of `string` .

**Syntax**: MD5(string)

**Example statement**: SELECT MD5(content) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR content) | Test Result (VARCHAR res) |
|---|---|
| abc | 900150983cd24fb0d6963f7d28e17f72 |

## OVERLAY

**Feature description**: Returns a string that replaces `length` characters of `string1` with `string2` from position `start_pos` (**starts from 1**).

**Syntax**: SELECT OVERLAY(string1 PLACING string2 FROM start_pos [ FOR length ])

**Example statement**: SELECT OVERLAY(string1 PLACING string2 FROM start_pos FOR length) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Data (INT start_pos) | Test Data (INT length) | Test Result (VARCHAR res) |
|---|---|---|---|---|
| | | | | |

| | | | | |
|---|---|---|---|---|
| oceanus | abc | 2 | 2 | oabcanus |

## POSITION

**Feature description**: Returns the position of the first occurrence of `string1` in `string2` ; returns 0 if `string1` cannot be found in `string2` .

**Syntax**: POSITION(string1 IN string2)

**Example statement**: SELECT POSITION(string1 IN string2) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Result (VARCHAR res) |
|---|---|---|
| nu | oceanus | 5 |

## GET_JSON_OBJECT(json_str, path_str)

**Feature description**: Gets the elements of a JSON string `json_str` at the JSON path specified by `path_str` , which can be arbitrarily nested. Supported JSONPath syntax: `$` for root object, `.` for a child element, `[]` for an array index, and `*` as a wildcard for array index [].

**Syntax**: GET_JSON_OBJECT(json_str, path_str)

**Example statement**: SELECT GET_JSON_OBJECT(json_str, path_str) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR json_str) |
|---|
| {"school": {"student":[{"num":8,"type":"A"},{"num":9,"type":"B"}],"teacher":{"num":200,"type":"A"} },"headmaster":"mark" } |

| Test Data (VARCHAR path_str) | Test Result (VARCHAR res) |
|---|---|
| $.school | {\\"student\\":[{\\"num\\":8,\\"type\\":\\"A\\"},{\\"num\\":9,\\"type\\":\\"B\\"}], \\"teacher\\":{\\"num\\":200,\\"type\\":\\"A\\"}} |
| $.school.student[1] | {\\"num\\":9,\\"type\\":\\"B\\"} |
| $.school.teacher | {\\"num\\":200,\\"type\\":\\"A\\"} |
| $.headmaster | mark |

## REPLACE

**Feature description**: Returns a string from `string1` with all the substrings that match `string2` being replaced with `string3` .

**Syntax**: REPLACE(string1, string2, string3)

**Example statement**: SELECT REPLACE( string1, string2, string3) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR string1) | Test Data (VARCHAR string2) | Test Data (VARCHAR string3) | Test Result (VARCHAR res) |
|---|---|---|---|
| banana | a | A | bAnAnA |

## SHA1

**Feature description**: Returns the SHA-1 hash of `expr` .

**Syntax**: SHA1(expr)

**Example statement**: SELECT SHA1(expr) AS res FROM Test;

**Test data and result**:

| Test Data (VARCHAR expr) | Test Result (VARCHAR res) |
|---|---|
| abc | a9993e364706816aba3e25717850c26c9cd0d89d |

## SHA256

**Feature description**: Returns the SHA-256 hash of `expr` .

**Syntax**: SHA256(expr)

**Example statement**: SELECT SHA256(expr) FROM Test;

**Test data and result**:

| Test Data (VARCHAR expr) | Test Result (VARCHAR res) |
|---|---|
| abc | ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad |

# Type Conversion Functions

Last updated：2023-11-08 14:18:31

## Functions

**Note**

For the CAST() function, if you convert INTERVAL to a number, the result will be a literal value (which may not be what you expect). For example, `CAST(INTERVAL '1234' MINUTE AS BIGINT)` returns `1234` instead of the number of milliseconds represented by INTERVAL.

| Function | Description |
|---|---|
| CAST(value AS type) | Returns a new value being cast to the type specified by `type`. For example, `CAST(` hello `AS VARCHAR)` returns `hello` converted to VARCHAR type. |
| CAN_CAST_TO(str, type) | Checks whether the string `str` can be converted to the type specified by `type`, and returns a Boolean value. The return value can be used as a condition in a CASE statement. For example, `CAN_CAST_TO('123456', 'INTEGER')` returns True, and `CAN_CAST_TO('a145', 'DOUBLE')` returns False. |
| TO_TIMESTAMP(string, simple_format) | Converts `string` to a timestamp in the `simple_format`, which is compatible with Java's SimpleDateFormat. UTC+8 is used by default. Example: `TO_TIMESTAMP(ts, 'yyyy-MM-dd HH:mm:ss')` |
| DATE_FORMAT_SIMPLE(timestamp, simple_format) | Converts a field of **BIGINT (Long)** type (which represents a Unix timestamp in milliseconds) to a string in the `simple_format`, which is compatible with Java's SimpleDateFormat. For example, `DATE_FORMAT_SIMPLE(unix_ts, 'yyyy-MM-dd HH:mm:ss')` returns a string similar to "2020-01-01 12:13:14". |
| DATE_FORMAT(timestamp, format) | Converts a field of **Timestamp** type to a string in the `format`, which is compatible with Java's SimpleDateFormat. For example, `DATE_FORMAT(ts, 'yyyy-MM-dd HH:mm:ss')` returns a string similar to "2020-01-01 12:13:14". |
| TIMESTAMP_TO_LONG(timestamp) or TIMESTAMP_TO_LONG(timestamp, mode) | Converts a TIMESTAMP type parameter to a value of BIGINT (Long) type. If `mode` is 'SECOND', converts to a Unix timestamp in seconds, for example, 1548403425. If `mode` is |

| | |
|---|---|
| | any other value or not specified, converts to a Unix timestamp in milliseconds, for example, 1548403425512. |

## Examples

### CAST

**Feature description**: Returns a new value being cast to the type specified by `type` .
**Syntax**: CAST(value AS type)
**Example statement**: SELECT CAST(var1 AS VARCHAR) FROM TEST;
**Test data and result**:

| Test Data (INT var1) | Test Result (VARCHAR) |
|---|---|
| 58 | '58' |

### CAN_CAST_TO

**Feature description**: Checks whether the string `str` can be converted to the type specified by `type` , and returns a Boolean value. The return value can be used as a condition in a CASE statement.
**Syntax**: CAN_CAST_TO(str, type)
**Example statement**: SELECT CAN_CAST_TO(var1,type) FROM Test;
**Test data and result**:

| Test Data (VARCHAR var1) | Test Data (VARCHAR type) | Test Result (BOOLEAN) |
|---|---|---|
| 123456 | INTEGER | true |

### DATE_FORMAT_SIMPLE

**Feature description**: Converts a field of **BIGINT (Long)** type (which represents a Unix timestamp in milliseconds) to a string (GMT+8 time zone) using format that is compatible with Java's SimpleDateFormat.
**Syntax**: DATE_FORMAT_SIMPLE(timestamp, simple_format)
**Example statement**: SELECT DATE_FORMAT_SIMPLE(unix_ts,'yyyy-MM-dd HH:mm:ss') FROM Test;
**Test data and result**:

| Test Data (unix_ts) | Test Result (VARCHAR) |
|---|---|
| 1627997937000 | 2021-08-03 21:38:57 |

### DATE_FORMAT

**Feature description**: Converts a field of **Timestamp** type to a string using format that is compatible with Java's SimpleDateFormat.

**Syntax**: DATE_FORMAT(timestamp, format)

**Example statement**: SELECT DATE_FORMAT(timestamp,format) FROM Test;

**Test data and result**:

| Test Data (timestamp) | Test Data (format) | Test Result (VARCHAR) |
|---|---|---|
| 2021-01-01 12:13:14 | yyMMdd | 210101 |
| 2021-01-01 12:13:14 | yyyyMMdd | 20210101 |

# Date and Time Functions

Last updated：2023-11-08 14:18:06

Date and time functions are described as follows:

| Function | Description |
|---|---|
| DATE string | Returns a SQL date parsed from `string` in the format of "yyyy-MM-dd". |
| TIME string | Returns a SQL time parsed from `string` in the format of "HH:mm:ss". |
| TIMESTAMP string | Returns a SQL timestamp parsed from `string` in the format of "yyyy-MM-dd HH:mm:ss[.SSS]". |
| INTERVAL string range | Parses an interval string in the format "dd hh:mm:ss.fff" for intervals of milliseconds or "yyyy-MM" for intervals of months.<br>An interval range can be DAY, MINUTE, DAY TO HOUR, or DAY TO SECOND for intervals of milliseconds;<br>and YEAR or YEAR TO MONTH for intervals of months. Examples: `INTERVAL '10 00:00:00.004' DAY TO SECOND` , `INTERVAL '10' DAY` , and `INTERVAL '2-10' YEAR TO MONTH` . |
| CURRENT_DATE | Returns the current SQL date (UTC). |
| CURRENT_TIME | Returns the current SQL time (UTC). |
| CURRENT_TIMESTAMP | Returns the current SQL timestamp (UTC). |
| LOCALTIME | Returns the current SQL time in the local time zone. |
| LOCALTIMESTAMP | Returns the current SQL timestamp in the local time zone. |
| EXTRACT(timeintervalunit FROM temporal) | Returns an item in a time point or time interval string. For example, `EXTRACT(DAY FROM DATE '2006-06-05')` returns `5` , and `EXTRACT(YEAR FROM DATE '2018-06-12')` returns `2018` . |
| YEAR(date) | Returns the year from `date` . Equivalent to `EXTRACT(YEAR FROM date)` . For example, `YEAR(DATE '2020-08-12')` returns `2020` . |
| QUARTER(date) | Returns the quarter of a year from `date` . Equivalent to `EXTRACT(QUARTER FROM date)` . For example, `QUARTER(DATE '2012-09-10')` returns `3` . |

| | |
|---|---|
| MONTH(date) | Returns the month from `date`. Equivalent to `EXTRACT(MONTH FROM date)`. For example, `MONTH(DATE '2012-09-10')` returns `9`. |
| WEEK(date) | Returns the week of a year from `date`. Equivalent to `EXTRACT(WEEK FROM date)`. For example, `WEEK(DATE '1994-09-27')` returns `39`. |
| DAYOFYEAR(date) | Returns the day of a year (an integer between 1 and 366) from `date`. Equivalent to `EXTRACT(DOY FROM date)`. For example, `DAYOFYEAR(DATE '1994-09-27')` returns `270`. |
| DAYOFMONTH(date) | Returns the day of a month (an integer between 1 and 31) from `date`. Equivalent to `EXTRACT(DAY FROM date)`. For example, `DAYOFMONTH(DATE '1994-09-27')` returns `27`. |
| DAYOFWEEK(date) | Returns the day of a week (an integer between 1 and 7) from `date`. Equivalent to `EXTRACT(DOW FROM date)`. For example, `DAYOFWEEK(DATE '1994-09-27')` returns `3`. |
| HOUR(timestamp) | Returns the hour of a day (an integer between 0 and 23) from `timestamp`. Equivalent to `EXTRACT(HOUR FROM timestamp)`. For example, `HOUR('2017-10-02 12:25:44')` returns `12`. |
| MINUTE(timestamp) | Returns the minute of an hour (an integer between 0 and 59) from `timestamp`. Equivalent to `EXTRACT(MINUTE FROM timestamp)`. For example, `MINUTE('2017-10-02 12:25:44')` returns `25`. |
| SECOND(timestamp) | Returns the second of a minute (an integer between 0 and 59) from `timestamp`. Equivalent to `EXTRACT(SECOND FROM timestamp)`. For example, `SECOND('2017-10-02 12:25:44')` returns `44`. |
| FLOOR(timepoint TO timeintervalunit) | Returns a value that rounds `timepoint` down to the time unit `timeintervalunit`. For example, `FLOOR(TIME '12:44:31' TO MINUTE)` returns `12:44:00`. |
| CEIL(timepoint TO timeintervalunit) | Returns a value that rounds `timepoint` up to the time unit `timeintervalunit`. For example, `CEIL(TIME '12:44:31' TO MINUTE)` returns `12:45:00`. |
| (timepoint, temporal) OVERLAPS (timepoint, temporal) | Checks whether two time intervals overlap (returns TRUE if yes). For example, `(TIME'2:55:00', INTERVAL '1' HOUR)` |

| | |
|---|---|
| | `OVERLAPS (TIME'3:30:00', INTERVAL '2' HOUR)` returns TRUE, and `(TIME'9:00:00', TIME '10:00:00') OVERLAPS (TIME'10:15:00', INTERVAL '3' HOUR)` returns FALSE. |
| TO_TIMESTAMP(string, simple_format) | Converts a string-formatted timestamp to Timestamp type. For more information, see Type Conversion Functions. |
| DATE_FORMAT_SIMPLE (timestamp, simple_format) | Converts a BIGINT type Unix timestamp in milliseconds to a string. For more information, see Type Conversion Functions. |
| DATE_FORMAT(timestamp, format) | Converts `timestamp` to a value of string in the `format` . For more information, see Type Conversion Functions. |
| TIMESTAMP_TO_LONG(timestamp) | Converts a Timestamp type timestamp to a BIGINT type Unix timestamp in milliseconds or seconds (configurable). For more information, see Type Conversion Functions. |
| TIMESTAMPADD(unit, interval, timestamp) | Adds `interval` (which can be negative) to the specified `timestamp` . The unit of the interval must be one of SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR. For example, `TIMESTAMPADD(WEEK, 1, '2013-01-02')` returns '2013-01-09'. |
| TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2) | Returns the number of timepointunit between timepoint1 and timepoint2, which can be negative. `timepointunit` must be one of SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR. For example, `TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00')` returns `1` (that is, 1 day). |
| CONVERT_TZ(string1, string2, string3) | Converts `string1` (in the format of 'yyyy-MM-dd HH:mm:ss') from time zone `string2` to time zone `string3` . The format of the time zone should be either an abbreviation such as "PST", a full name such as "Asia/Shanghai", or a custom ID such as "GMT-8:00". For example, `CONVERT('1970-01-01 00:00:00', 'UTC', 'America/Los_Angeles')` returns '1969-12-31 16:00:00'. |
| FROM_UNIXTIME(numeric[, string]) | Returns a representation of the `numeric` parameter as a value in string format ('YYYY-MM-DD hh:mm:ss'). `numeric` represents the number of seconds since 1970-01-01 00:00:00 UTC. The default time zone is UTC+8, which is Beijing time (Asia/Shanghai). |
| UNIX_TIMESTAMP() | Returns the current Unix timestamp in seconds (number of seconds since 1970-01-01 00:00:00 UTC). The default time zone is UTC+8, which is Beijing time (Asia/Shanghai). |
| | |

| | |
|---|---|
| UNIX_TIMESTAMP(string1[, string2]) | Converts `string1` with format `string2` (default: 'yyyy-MM-dd HH:mm:ss'; configurable) to Unix timestamp (BIGINT). |
| TO_DATE(string1[, string2]) | Converts `string1` with format `string2` (default: 'yyyy-MM-dd HH:mm:ss'; configurable) to a date. |
| TO_TIMESTAMP(string1[, string2]) | Converts `string1` with format `string2` (default: 'yyyy-MM-dd HH:mm:ss'; configurable) to a timestamp. The default time zone is UTC+8, which is Beijing time (Asia/Shanghai). |
| NOW() | Returns the current SQL timestamp (UTC). |

# Aggregate Functions

Last updated：2023-11-08 14:16:28

## Functions

| Function | Description |
|---|---|
| COUNT([ ALL ] expression \| DISTINCT expression1 [, expression2]*) | By default or with ALL, returns the number of input rows for which expression is not NULL. Use DISTINCT for one unique instance of each value. |
| COUNT(*) COUNT(1) | Returns the number of input rows (including those for which expression is NULL). |
| AVG([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the average (arithmetic mean) of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| SUM([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the sum of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| MAX([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the maximum value of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| MIN([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the minimum value of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| STDDEV_POP([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the population standard deviation of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| STDDEV_SAMP([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the sample standard deviation of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| VAR_POP([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the population variance of expression across all input rows. Use DISTINCT for one unique instance of each value. |
| VAR_SAMP([ ALL \| DISTINCT ] expression) VARIANCE([ ALL \| DISTINCT ] expression) | By default or with ALL, returns the sample variance of expression across all input rows. Use DISTINCT for one unique instance of each value. The two expressions are equivalent. |

| | |
|---|---|
| COLLECT([ ALL \| DISTINCT ] expression) | By default or with ALL, returns a multiset of expression across all input rows. NULL values will be ignored, and duplicate values are allowed. If all values are NULL, an empty set is returned. |
| RANK() | Returns the rank of a value in a group of values. The values may produce gaps in the sequence. For example, if there are five values, two of which rank the second place, the result of `RANK()` will be 1, 2, 2, 4, 5. |
| DENSE_RANK() | Returns the rank of a value in a group of values. The values will not produce gaps in the sequence. For example, if there are five values, two of which rank the second place, the result of `RANK()` will be 1, 2, 2, 3, 4. |
| ROW_NUMBER() | Assigns a unique, sequential number to each row, starting from 1. For example, if there are five values, two of which rank the second place, the result of `RANK()` will be 1, 2, 3, 4, 5. |
| LEAD(expression [, offset] [, default] ) | Returns the value of expression at the offsetth row after the current row in the window. The default value of `offset` is 1 and the default value of `default` is NULL. |
| LAG(expression [, offset] [, default]) | Returns the value of expression at the offsetth row before the current row in the window. The default value of `offset` is 1 and the default value of `default` is NULL. |
| FIRST_VALUE(expression) | Returns the first value in a set of values. |
| LAST_VALUE(expression) | Returns the last value in a set of values. |
| LISTAGG(expression [, separator]) | Concatenates the values of string expressions and places separator values between them. The default value of `separator` is `,` . This function is similar to the `String.join()` method in other programming languages. |

## Examples

For better illustration, an example test data table Test is created as follows:

| id | site_id | count | date |
|---|---|---|---|
| 1 | 1 | 45 | 2021-07-10 |
| 2 | 3 | 100 | 2021-07-13 |

| | | | |
|---|---|---|---|
| 3 | 1 | 230 | 2021-07-14 |
| 4 | 2 | 10 | 2021-07-14 |
| 5 | 5 | 205 | 2021-07-14 |
| 6 | 4 | 13 | 2021-07-15 |
| 7 | 3 | 220 | 2021-07-15 |
| 8 | 5 | 545 | 2021-07-16 |
| 9 | 3 | 201 | 2021-07-17 |

### COUNT

**Feature description**: Counts the total number of rows.

**Syntax**: COUNT([ ALL ] expression | DISTINCT expression1 [, expression2] * )

**Example statement**: `SELECT  COUNT(*) FROM Test;` **Test data and result**:

| Test Statement | Test Result (nums) |
|---|---|
| SELECT  COUNT(*)  AS nums FROM Test; | 9 |
| SELECT  COUNT(DISTINCT site_id)  AS nums FROM Test; | 5 |

### AVG

**Feature description**: Returns the average value.

**Syntax**: AVG([ ALL | DISTINCT ] expression)

**Example statement**: SELECT AVG(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (nums) |
|---|---|
| SELECT AVG(count) AS nums FROM Test; | 176.3 |

### SUM

**Feature description**: Returns the sum.

**Syntax**: SUM([ ALL | DISTINCT ] expression)

**Example statement**: SELECT SUM(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (nums) |
|---|---|

| | |
|---|---|
| SELECT SUM(count) AS nums FROM Test; | 1569 |

## MAX

**Feature description**: Returns the maximum value.

**Syntax**: MAX([ ALL | DISTINCT ] expression)

**Example statement**: SELECT MAX(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (nums) |
|---|---|
| SELECT MAX(count) AS nums FROM Test; | 545 |

## MIN

**Feature description**: Returns the minimum value.

**Syntax**: MIN([ ALL | DISTINCT ] expression)

**Example statement**: SELECT MIN(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (nums) |
|---|---|
| SELECT MIN(count) AS nums FROM Test; | 13 |

## STDDEV_POP

**Feature description**: Returns the population standard deviation.

**Syntax**: STDDEV_POP([ ALL | DISTINCT ] expression)

**Example statement**: SELECT STDDEV_POP(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (pop) |
|---|---|
| SELECT STDDEV_POP(count) AS pop FROM Test; | 156.18 |

## VAR_POP

**Feature description**: Returns the population variance.

**Syntax**: VAR_POP([ ALL | DISTINCT ] expression)

**Example statement**: SELECT VAR_POP(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (pop) |
|---|---|
| SELECT VAR_POP(count) AS pop FROM Test; | 24390.7 |

## FIRST_VALUE

**Feature description**: Returns the first value in a set of values.

**Syntax**: FIRST_VALUE(expression)

**Example statement**: SELECT FIRST_VALUE(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (first) |
|---|---|
| SELECT FIRST_VALUE(count) AS first FROM Test; | 45 |

## LAST_VALUE

**Feature description**: Returns the last value in a set of values.

**Syntax**: LAST_VALUE(expression)

**Example statement**: SELECT LAST_VALUE(count) FROM Test;

**Test data and result**:

| Test Statement | Test Result (last) |
|---|---|
| SELECT LAST_VALUE(count) AS last FROM Test; | 201 |

# Time Window Functions

Last updated：2023-11-08 11:33:35

In stream processing, streams are unbounded, and we don't know when the data source will continue/stop sending data. Therefore, the way to do aggregations (e.g., count, sum) on streams differs from that in batch processing. In stream processing, windows are used to limit the scope of aggregation, such as "counting website clicks in the last 2 minutes" and "counting the total number of people who liked this video among the recent 100 people". Windows are equivalent to collecting a dynamic table of bounded data so that we can perform aggregations on the data in the table.
Window functions are a special type of functions that are not used in the projection list of SELECT, but in the GROUP BY clause. Stream Compute Service supports three types of window functions: TUMBLE, HOP, and SESSION.
To learn about Flink stream processing, see Timely Stream Processing.

## TUMBLE

A tumbling window assigns data to non-overlapping windows with a fixed size that is customizable. We can perform aggregations on the data within the window.

### Syntax

```
TUMBLE(time_attr, interval)
```

`time_attr` is a timestamp parameter that specifies the time when a record is processed. **If specified as** `PROCTIME`, it is an automatically generated timestamp that records the moment when the data is processed by Flink. It is generally used in Processing Time mode.

`interval` specifies the window size. For example, use `INTERVAL '1' DAY` to set a 1-day window size and `INTERVAL '2' HOUR` a 2-hour window size. For other usage, see Date and Time Functions.

**Note**

In Event Time mode (the WATERMARK FOR statement is used to define the timestamp field), the first parameter of the TUMBLE, HOP, and SESSION window functions must be the timestamp field.
In Processing Time mode, the first parameter of the TUMBLE, HOP, and SESSION window functions must be the calculated column generated by the `proctime()` function. In the following example, we use `PROCTIME`, but you should replace it with the actual column name in your job.

### Identification functions

Identification functions are used to identify the start and end timestamps of windows.

| Function | Description |
|---|---|
| TUMBLE_START(time-attr, size-interval) | Returns the start timestamp of the window. |
| TUMBLE_END(time-attr, size-interval) | Returns the end timestamp of the window. |

### Example

This example helps you better understand a tumbling window. It uses Event Time to count the hourly income of each user.

**Example data**:

| username (VARCHAR) | income (BIGINT) | times (TIMESTAMP) |
|---|---|---|
| Tom | 20 | 2021-11-11 10:30:00.0 |
| Jack | 10 | 2021-11-11 10:35:00.0 |
| Tom | 10 | 2021-11-11 10:35:00.0 |
| Tom | 10 | 2021-11-11 10:40:00.0 |
| Tom | 15 | 2021-11-11 11:30:00.0 |
| Jack | 10 | 2021-11-11 11:30:00.0 |
| Jack | 15 | 2021-11-11 11:40:00.0 |

**SQL statements**:

```
CREATE TABLE user_income (
      username VARCHAR,
      Income INT,
      times TIMESTAMP(3),
      WATERMARK FOR times AS times - INTERVAL '3' SECOND
) WITH (
      ...
  );

CREATE TABLE output (
            win_start TIMESTAMP,
            win_end TIMESTAMP,
            username VARCHAR,
            hour_income BIGINT
)WITH(
      ...
);

INSERT INTO output
SELECT
TUMBLE_START(times,INTERVAL '1' HOUR),
TUMBLE_END(times,INTERVAL '1' HOUR),
username,
SUM(Income)
FROM user_income
GROUP BY TUMBLE(times,INTERVAL '1' HOUR),username;
```

**Output result**:

| win_start (TIMESTAMP) | win_end (TIMESTAMP) | username (VARCHAR) | hour_income (BIGINT) |
|---|---|---|---|
| 2021-11-11 10:00:00.0 | 2021-11-11 11:00:00.0 | Tom | 40 |
| 2021-11-11 10:00:00.0 | 2021-11-11 11:00:00.0 | Jack | 10 |
| 2021-11-11 11:00:00.0 | 2021-11-11 12:00:00.0 | Tom | 15 |
| 2021-11-11 11:00:00.0 | 2021-11-11 12:00:00.0 | Jack | 25 |

## HOP

A hopping window assigns elements to fixed-size windows. Similar to a tumbling window, a hopping window supports window size customization. The other parameter controls how frequently a window is started.
A hopping window maintains a fixed window size and slides at a specified hop interval, allowing overlapping windows. The hop interval determines the frequency at which Flink creates new windows.
If the hop interval is smaller than the window size, hopping windows are overlapping. In this case, elements are assigned to multiple windows.
If the hop interval is greater than the window size, some events may be discarded.
If the hop interval is equal to the window size, this window is equivalent to a tumbling window.

### Syntax

```
HOP(time_attr, sliding_interval, window_size_interval)
```

`time_attr` is a timestamp parameter that specifies the time when a record is processed. If specified as `PROCTIME`, it is an automatically generated timestamp that records the moment when the data is processed by Flink. It is generally used in Processing Time mode.

`window_size_interval` specifies the window size. For example, use `INTERVAL '1' DAY` to set a 1-day window size and `INTERVAL '2' HOUR` a 2-hour window size. For other usage, see Date and Time Functions.

`sliding_interval` specifies the hop interval. For example, use `INTERVAL '1' DAY` to set a 1-day window size and `INTERVAL '2' HOUR` a 2-hour window size. For other usage, see Date and Time Functions.

### Identification functions

Identification functions are used to identify the start and end timestamps of windows.

| Function | Description |
|---|---|
| HOP_START(time-attr, slide-interval,size-interval) | Returns the start timestamp of the window. |
| HOP_END(time-attr, slide-interval,size-interval) | Returns the end timestamp of the window. |

### Example

This example helps you better understand a hopping window. It uses Event Time to count the hourly income of each user, which is updated every 30 minutes. **The window size is 1 hour, with a hop interval of 10 minutes.**

**Example data**:

| username (VARCHAR) | income (BIGINT) | times (TIMESTAMP) |
|---|---|---|
| Tom | 20 | 2021-11-11 10:30:00.0 |
| Jack | 10 | 2021-11-11 10:35:00.0 |
| Tom | 10 | 2021-11-11 10:35:00.0 |
| Tom | 10 | 2021-11-11 10:40:00.0 |
| Tom | 15 | 2021-11-11 11:35:00.0 |
| Jack | 10 | 2021-11-11 11:30:00.0 |
| Jack | 15 | 2021-11-11 11:40:00.0 |

**SQL statements**:

```
CREATE TABLE user_income (
      username VARCHAR,
      Income INT,
      times TIMESTAMP(3),
      WATERMARK FOR times AS times - INTERVAL '3' MINUTE
)WITH(
      ...
  );

CREATE TABLE output (
            win_start TIMESTAMP,
              win_end TIMESTAMP,
              username VARCHAR,
              hour_income BIGINT
)WITH(
      ...
  );

INSERT INTO output
SELECT
HOP_START(times,INTERVAL '30' MINUTE,INTERVAL '1' HOUR),
HOP_END(times,INTERVAL '30' MINUTE,INTERVAL '1' HOUR),
username,
SUM(income)
FROM user_income
GROUP BY HOP(times,INTERVAL '30' MINUTE,INTERVAL '1' HOUR),username;
```

**Output result**:

| win_start (TIMESTAMP) | win_end (TIMESTAMP) | username (VARCHAR) | hour_income (BIGINT) |
|---|---|---|---|
| 2021-11-11 10:00:00.0 | 2021-11-11 11:00:00.0 | Tom | 40 |
| 2021-11-11 10:00:00.0 | 2021-11-11 11:00:00.0 | Jack | 10 |
| 2021-11-11 10:30:00.0 | 2021-11-11 11:30:00.0 | Jack | 10 |
| 2021-11-11 10:30:00.0 | 2021-11-11 11:30:00.0 | Tom | 40 |
| 2021-11-11 11:00:00.0 | 2021-11-11 12:00:00.0 | Tom | 15 |
| 2021-11-11 11:00:00.0 | 2021-11-11 12:00:00.0 | Jack | 25 |
| 2021-11-11 11:30:00.0 | 2021-11-11 12:30:00.0 | Jack | 25 |
| 2021-11-11 11:30:00.0 | 2021-11-11 12:30:00.0 | Tom | 15 |

## SESSION

A session window groups elements by session activity. Unlike a tumbling or hopping window, a session window does not have overlapping windows or fixed start and end timestamps. Instead, the window is closed when it no longer receives elements within a fixed time period, which is called an inactive interval. A session window is configured using a session interval. The session interval defines the length of the inactive period. When this period elapses, the current session is closed and subsequent elements are assigned to a new session window.
Session windows are based on inactivity rather than size. For example, if there is no activity (no new data) for more than 30 minutes, the existing window is closed, and a new window starts when new data is observed later.

### Syntax

```
SESSION(time_attr, interval)
```

`time_attr` is a timestamp parameter that specifies the time when a record is processed. If specified as `PROCTIME`, it is an automatically generated timestamp that records the moment when the data is processed by Flink. It is generally used in Processing Time mode.

`interval` specifies the window size. For example, use `INTERVAL '1' DAY` to set a 1-day window size and `INTERVAL '2' HOUR` a 2-hour window size. For other usage, see Date and Time Functions.

## Identification functions

Identification functions are used to identify the start and end timestamps of windows.

| Function | Description |
|---|---|
| SESSION_START(time-attr, size-interval) | Returns the start timestamp of the window. |
| SESSION_END(time-attr, size-interval) | Returns the end timestamp of the window. |

## Example

This example helps you better understand a session window. It uses Event Time to count the hourly income of each user, with a session timeout period of 30 minutes.

**Example data**:

| username (VARCHAR) | income (BIGINT) | times (TIMESTAMP) |
|---|---|---|
| Tom | 20 | 2021-11-11 10:30:00.0 |
| Jack | 10 | 2021-11-11 10:35:00.0 |
| Tom | 10 | 2021-11-11 10:35:00.0 |
| Tom | 10 | 2021-11-11 10:40:00.0 |
| Tom | 15 | 2021-11-11 11:50:00.0 |
| Jack | 10 | 2021-11-11 11:40:00.0 |
| Jack | 15 | 2021-11-11 11:45:00.0 |

**SQL statements**:

```
CREATE TABLE user_income (
      username VARCHAR,
      Income INT,
      times TIMESTAMP(3),
      WATERMARK FOR times AS times - INTERVAL '3' MINUTE
)WITH(
      ...
  );

CREATE TABLE output (
            win_start TIMESTAMP,
              win_end TIMESTAMP,
              username VARCHAR,
              hour_income BIGINT
)WITH(
      ...
  );

INSERT INTO output
SELECT
SESSION_START(times,INTERVAL '30' MINUTE),
SESSION_END(times,INTERVAL '30' MINUTE),
username,
SUM(Income)
FORM user_income
GROUP BY SESSION(times,INTERVAL '30' MINUTE),username;
```

**Output result**:

| win_start (TIMESTAMP) | win_end (TIMESTAMP) | username (VARCHAR) | hour_income (BIGINT) |
|---|---|---|---|
| 2021-11-11 10:30:00.0 | 2021-11-11 11:10:00.0 | Tom | 40 |
| 2021-11-11 10:35:00.0 | 2021-11-11 11:05:00.0 | Jack | 10 |
| 2021-11-11 11:30:00.0 | 2021-11-11 12:00:00.0 | Tom | 15 |
| 2021-11-11 11:30:00.0 | 2021-11-11 12:10:00.0 | Jack | 25 |

## More notes

The preceding windows all have corresponding auxiliary functions, which are the same except for the prefix. Here, the auxiliary functions for tumbling windows are used as examples.
**TUMBLE_ROWTIME**: (Event Time mode) Returns the timestamp of the inclusive upper bound of the tumbling window. The resulting attribute can be used in subsequent time-based operations such as interval joins and group window or over window aggregations. Example:

```
SELECT user,
TUMBLE_START(rowtime, INTERVAL '12' HOUR) AS sStart,
TUMBLE_ROWTIME(rowtime, INTERVAL '12' HOUR) AS snd,
SUM(amount)
FROM Orders
GROUP BY TUMBLE(rowtime, INTERVAL '12' HOUR), user
```

**TUMBLE_PROCTIME**: (Processing Time mode) Returns the timestamp of the inclusive upper bound of the tumbling window. The resulting attribute can be used in subsequent time-based operations such as interval joins and group window or over window aggregations. Example:

```
SELECT user,
TUMBLE_START(PROCTIME, INTERVAL '12' HOUR) AS sStart,
TUMBLE_PROCTIME(PROCTIME, INTERVAL '12' HOUR) AS snd,
SUM(amount)
FROM Orders
GROUP BY TUMBLE(PROCTIME, INTERVAL '12' HOUR), user
```

# Other Functions

Last updated：2023-11-08 11:31:56

## Hash functions

Hash functions are described as follows:

| Function | Description |
|---|---|
| MD5(string) | Returns the MD5 hash (a string of 32 hexadecimal digits) of `string` . Returns NULL if `string` is NULL. |
| SHA1(string) | Returns the SHA-1 hash (a string of 40 hexadecimal digits) of `string` . Returns NULL if `string` is NULL. |
| SHA256(string) | Returns the SHA-256 hash (a string of 64 hexadecimal digits) of `string` . Returns NULL if `string` is NULL. |
| SHA224(string) | Returns the SHA-224 hash (a string of 56 hexadecimal digits) of `string` . Returns NULL if `string` is NULL. |
| SHA384(string) | Returns the SHA-384 hash (a string of 96 hexadecimal digits) of `string` . Returns NULL if `string` is NULL. |
| SHA512(string) | Returns the SHA-512 hash (a string of 128 hexadecimal digits) of `string` . Returns NULL if `string` is NULL. |
| SHA2(string, hashLength) | Returns the hash of `string` using the SHA-2 family of hash functions. For example, when `hashLength` is 256, it is equivalent to `SHA256(string)` . |

## Value access functions

Value access functions are described as follows:

| Function | Description |
|---|---|
| tableName.compositeType.field | Returns the value of a field from a composite type (e.g., Tuple, POJO) by name. |
| tableName.compositeType.* | Returns the values of all fields from a composite type (e.g., Tuple, POJO). |

## Value construction functions

Value construction functions are described as follows:

| Function | Description |
|---|---|
| (value, [, value]*)<br>ROW(value, [, value]*) | Returns a row created from a list of values. The two expressions are equivalent. |
| ARRAY '[' value [, value ] * ']' | Returns an array created from a list of values. |
| MAP '[' key, value [, key, value ] * ']' | Returns a map created from a list of key-value pairs. |

## Collection functions

Collection functions include array and map operations and are described as follows:

| Function | Description |
|---|---|
| CARDINALITY(array) | Returns the number of elements in `array` . |
| array '[' index ']' | Returns the element at position index in `array` . The index starts from 1. |
| ELEMENT(array) | Returns the sole element of `array` (whose cardinality should be one). Returns NULL if `array` is left empty. Throws an exception if `array` has more than one element. |
| CARDINALITY(map) | Returns the number of entries in `map` . |
| map '[' key ']' | Returns the value specified by `key` value in `map` . |

## Grouping functions

Grouping functions are used to represent the results of a GROUP BY grouping. They are described as follows:

| Function | Description |
|---|---|
| GROUP_ID() | Returns an integer that uniquely identifies the combination of grouping keys. |
| GROUPING(expression1 [, expression2]* )<br>GROUPING_ID(expression1 [, expression2]* ) | Returns the group ID (a bit vector) of the given grouping expressions. |

# Identifiers and Reserved Words

## Naming Rules

Last updated：2023-11-08 11:30:13

A table or column name uniquely identifies a table or column. Stream Compute Service has the following rules regarding the naming of tables and columns:

Do not use fields reserved for data types. If you have to, include them in backquotes, for example, `time` .

For column names, do not use `PROCTIME` or `SOURCETIME` . This is to avoid conflicts with system-generated timestamps.

Do not start with `_DataStreamTable_` .

If a table or column name includes spaces or special characters, include them in backquotes, such as ` HELLO WORLD` .

The length cannot exceed 128 half-width characters (one Chinese character or full-width character is equal to two English letters or two half-width characters).

# Reserved Words

Last updated：2023-11-08 11:30:37

This document lists the words reserved for use by Stream Compute Service. If you need to use a reserved word for the name of a table or column, include it in backquotes (``); otherwise, errors will be detected by syntax check.

**Note**

In the Processing Time mode (that is, if you did not use `WATERMARK FOR` to define a timestamp field for the data source), to avoid conflicts with system-generated timestamps, do not use `PROCTIME` as the name of a column.

**A**

```
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, AUTHORIZATION, AVG
```

**B**

```
BEFORE, BEGIN, BERNOULLI, BETWEEN, BIGINT, BINARY, BIT, BLOB, BOOLEAN, BOTH BREADTH
```

### C

```
C, CALL, CALLED, CARDINALITY, CASCADE, CASCADED, CASE, CAST, CATALOG, CATALOG_NAME,
CEIL, CEILING, CENTURY, CHAIN, CHAR, CHARACTER, CHARACTERISTICTS, CHARACTERS,
CHARACTER_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_NAME, CHARACTER_SET_SCHEMA,
CHAR_LENGTH, CHECK, CLASS_ORIGIN, CLOB, CLOSE, COALESCE, COBOL, COLLATE, COLLATION,
COLLATION_CATALOG, COLLATION_NAME, COLLATION_SCHEMA, COLLECT, COLUMN, COLUMN_NAME,
COMMAND_FUNCTION, COMMAND_FUNCTION_CODE, COMMIT, COMMITTED,CONDITION, CONDITION_NUM
CONNECT, CONNECTION, CONNECTION_NAME, CONSTRAINT, CONSTRAINTS, CONSTRAINT_CATALOG,
CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORR,
CORRESPONDING, COUNT, COVAR_POP, COVAR_SAMP, CREATE, CROSS, CUBE, CUME_DIST, CURREN
CURRENT_CATALOG, CURRENT_DATE, CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRE
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_TRANSFORM_GROUP_FOR_TYPE,
CURRENT_USER, CURSOR, CURSOR_NAME, CYCLE, DATA, DATABASE, DATE, DATETIME_INTERVAL_C
```

### D

```
DATETIME_INTERVAL_PRECISION, DAY, DEALLOCATE, DEC, DECADE, DECIMAL, DECLARE, DEFAUL
DEFAULTS, DEFERRABLE, DEFERRED, DEFINED, DEFINER, DEGREE, DELETE, DENSE_RANK, DEPTH
DERIVED, DESC, DESCRIBE, DESCRIPTION, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISAL
DISPATCH, DISTINCT, DOMAIN, DOUBLE, DOW, DOY, DROP, DYNAMIC, DYNAMIC_FUNCTION, DYNA
```

### E

```
EACH, ELEMENT, ELSE, END, END-EXEC, EPOCH, EQUALS, ESCAPE, EVERY, EXCEPT, EXCEPTION
EXCLUDE, EXCLUDING, EXEC, EXECUTE, EXISTS, EXP, EXPLAIN, EXTEND, EXTERNAL, EXTRACT
```

**F**

```
FALSE, FETCH, FILTER, FINAL, FIRST, FIRST_VALUE, FLOAT, FLOOR, FOLLOWING, FOR, FORE
FORTRAN, FOUND, FRAC_SECOND, FREE, FROM, FULL, FUNCTION, FUSION
```

### G

```
G, GENERAL, GENERATED, GET, GLOBAL, GO, GOTO, GRANT, GRANTED, GROUP, GROUPING
```

**H**

```
HAVING, HIERARCHY, HOLD, HOUR
```

## I

```
IDENTITY, IMMEDIATE, IMPLEMENTATION, IMPORT, IN, INCLUDING, INCREMENT, INDICATOR,
INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTANCE, INSTANTIABLE, INT,
INTEGER, INTERSECT, INTERSECTION, INTERVAL, INTO, INVOKER, IS, ISOLATION
```

**J**

```
JAVA, JOIN
```

**K**

```
K, KEY, KEY_MEMBER, KEY_TYPE
```

**L**

```
LABEL, LANGUAGE, LARGE, LAST, LAST_VALUE, LATERAL, LEADING, LEFT, LENGTH, LEVEL,
LIBRARY, LIKE, LIMIT, LN, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, LOWER
```

**M**

```
M, MAP, MATCH, MATCHED, MAX, MAXVALUE, MEMBER, MERGE, MESSAGE_LENGTH, MESSAGE_OCTET
MESSAGE_TEXT, METHOD, MICROSECOND, MILLENNIUM, MIN, MINUTE, MINVALUE, MOD, MODIFIES
MONTH, MORE, MULTISET, MUMPS
```

**N**

```
NAME, NAMES, NATIONAL, NATURAL, NCHAR, NCLOB, NESTING, NEW, NEXT, NO, NONE, NORMALI
NORMALIZED, NOT, NULL, NULLABLE, NULLIF, NULLS, NUMBER, NUMERIC
```

### O

```
OBJECT, OCTETS, OCTET_LENGTH, OF, OFFSET, OLD, ON, ONLY, OPEN, OPTION, OPTIONS, OR,
ORDER, ORDERING, ORDINALITY, OTHERS, OUT, OUTER, OUTPUT, OVER, OVERLAPS, OVERLAY, O
```

### P

```
PAD, PARAMETER, PARAMETER_MODE, PARAMETER_NAME, PARAMETER_ORDINAL_POSITION,
PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_NAME, PARAMETER_SPECIFIC_SCHEMA, PAR
PARTITION, PASCAL, PASSTHROUGH, PATH, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RAN
PLAN, PLI, POSITION, POWER, PRECEDING, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR
PROCEDURE, PUBLIC
```

**Q**

```
QUARTER
```

**R**

```
RANGE, RANK, READ, READS, REAL, RECURSIVE, REF, REFERENCES, REFERENCING, REGR_AVGX,
REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REG
RELATIVE, RELEASE, REPEATABLE, RESET, RESTART, RESTRICT, RESULT, RETURN, RETURNED_C
RETURNED_LENGTH, RETURNED_OCTET_LENGTH, RETURNED_SQLSTATE, RETURNS, REVOKE, RIGHT,
ROLLUP, ROUTINE, ROUTINE_CATALOG, ROUTINE_NAME, ROUTINE_SCHEMA, ROW, ROWS, ROW_COUN
```

## S

```
SAVEPOINT, SCALE, SCHEMA, SCHEMA_NAME, SCOPE, SCOPE_CATALOGS, SCOPE_NAME, SCOPE_SCH
SCROLL, SEARCH, SECOND, SECTION, SECURITY, SELECT, SELF, SENSITIVE, SEQUENCE, SERIA
SERVER, SERVER_NAME, SESSION, SESSION_USER, SET, SETS, SIMILAR, SIMPLE, SIZE, SMALL
SOME, SOURCE, SPACE, SPECIFIC, SPECIFICTYPE, SPECIFIC_NAME, SQL, SQLEXCEPTION, SQLS
SQLWARNING, SQL_TSI_DAY, SQL_TSI_FRAC_SECOND, SQL_TSI_HOUR, SQL_TSI_MICROSECOND, SQ
SQL_TSI_MONTH, SQL_TSI_QUARTER, SQL_TSI_SECOND, SQL_TSI_WEEK, SQL_TSI_YEAR, SQRT, S
STATE, STATEMENT, STATIC, STDDEV_POP, STDDEV_SAMP, STREAM, STRUCTURE, STYLE, SUBCLA
SUBMULTISET, SUBSTITUTE, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER
```

**T**

```
TABLE, TABLESAMPLE, TABLE_NAME, TEMPORARY, THEN, TIES, TIME, TIMESTAMP, TIMESTAMPAD
TIMESTAMPDIFF, TIMEZONE_HOUR, TIMEZONE_MINUTE, TINYINT, TO, TOP_LEVEL_COUNT, TRAILI
TRANSACTION, TRANSACTIONS_ACTIVE, TRANSACTIONS_COMMITTED, TRANSACTIONS_ROLLED_BACK,
TRANSFORM, TRANSFORMS, TRANSLATE, TRANSLATION, TREAT, TRIGGER, TRIGGER_CATALOG,
TRIGGER_NAME, TRIGGER_SCHEMA, TRIM, TRUE, TYPE
```

**U**

```
UESCAPE, UNBOUNDED, UNCOMMITTED, UNDER, UNION, UNIQUE, UNKNOWN, UNNAMED, UNNEST,
UPDATE, UPPER, UPSERT, USAGE, USER, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_CO
USER_DEFINED_TYPE_NAME, USER_DEFINED_TYPE_SCHEMA, USING
```

**V**

```
VALUE, VALUES, VARBINARY, VARCHAR, VARYING, VAR_POP, VAR_SAMP, VERSION, VIEW
```

**W**

```
WATERMARK, WEEK, WHEN, WHENEVER, WHERE, WIDTH_BUCKET, WINDOW, WITH, WITHIN, WITHOUT
WORK, WRAPPER, WRITE
```

**X**

```
XML
```

**Y**

```
YEAR
```

## Z

```
ZONE
```