

Data Lake Insight

Flink SQL Syntax

Issue 01
Date 2021-12-28



Copyright © Huawei Technologies Co., Ltd. 2023. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Flink OpenSource SQL 1.12 Syntax Reference.....	1
1.1 Constraints and Definitions.....	1
1.1.1 Supported Data Types.....	1
1.1.2 Syntax.....	1
1.1.2.1 Data Definition Language (DDL).....	1
1.1.2.1.1 CREATE TABLE.....	1
1.1.2.1.2 CREATE VIEW.....	4
1.1.2.1.3 CREATE FUNCTION.....	4
1.1.2.2 Data Manipulation Language (DML).....	5
1.2 Overview.....	7
1.3 DDL Syntax.....	8
1.3.1 Creating Source Tables.....	9
1.3.1.1 DataGen Source Table.....	9
1.3.1.2 GaussDB(DWS) Source Table.....	12
1.3.1.3 HBase Source Table.....	16
1.3.1.4 JDBC Source Table.....	21
1.3.1.5 Kafka Source Table.....	27
1.3.1.6 MySQL CDC Source Table.....	40
1.3.1.7 Postgres CDC Source Table.....	44
1.3.1.8 Redis Source Table.....	49
1.3.1.9 Upsert Kafka Source Table.....	55
1.3.2 Creating Result Tables.....	60
1.3.2.1 BlackHole Result Table.....	60
1.3.2.2 ClickHouse Result Table.....	61
1.3.2.3 GaussDB(DWS) Result Table.....	66
1.3.2.4 Elasticsearch Result Table.....	72
1.3.2.5 HBase Result Table.....	79
1.3.2.6 JDBC Result Table.....	85
1.3.2.7 Kafka Result Table.....	90
1.3.2.8 Print Result Table.....	100
1.3.2.9 Redis Result Table.....	103
1.3.2.10 Upsert Kafka Result Table.....	113
1.3.2.11 FileSystem Result Table.....	117

1.3.3 Creating Dimension Tables.....	122
1.3.3.1 GaussDB(DWS) Dimension Table.....	122
1.3.3.2 HBase Dimension Table.....	128
1.3.3.3 JDBC Dimension Table.....	133
1.3.3.4 Redis Dimension Table.....	139
1.3.4 Format.....	145
1.3.4.1 Avro.....	145
1.3.4.2 Canal.....	148
1.3.4.3 Confluent Avro.....	152
1.3.4.4 CSV.....	155
1.3.4.5 Debezium.....	157
1.3.4.6 JSON.....	160
1.3.4.7 Maxwell.....	163
1.3.4.8 Raw.....	166
1.4 DML Syntax.....	167
1.4.1 SELECT.....	168
1.4.2 Set Operations.....	172
1.4.3 Window.....	173
1.4.4 JOIN.....	181
1.4.5 OrderBy & Limit.....	183
1.4.6 Top-N.....	184
1.4.7 Deduplication.....	185
1.5 Functions.....	186
1.5.1 User-Defined Functions (UDFs).....	186
1.5.2 Built-In Functions.....	189
1.5.2.1 Mathematical Operation Functions.....	189
1.5.2.2 String Functions.....	198
1.5.2.3 Temporal Functions.....	204
1.5.2.4 Conditional Functions.....	227
1.5.2.5 Type Conversion Functions.....	228
1.5.2.6 Collection Functions.....	231
1.5.2.7 Value Construction Functions.....	231
1.5.2.8 Value Access Functions.....	232
1.5.2.9 Hash Functions.....	232
1.5.2.10 Aggregate Functions.....	233
1.5.2.11 Table-Valued Functions.....	234
1.5.2.11.1 string_split.....	234
2 Flink OpenSource SQL 1.10 Syntax Reference.....	237
2.1 Constraints and Definitions.....	237
2.1.1 Supported Data Types.....	237
2.1.2 Syntax Definition.....	237
2.1.2.1 Data Definition Language (DDL).....	237

2.1.2.1.1 CREATE TABLE.....	237
2.1.2.1.2 CREATE VIEW.....	240
2.1.2.1.3 CREATE FUNCTION.....	240
2.1.2.2 Data Manipulation Language (DML).....	241
2.2 Flink OpenSource SQL 1.10 Syntax.....	243
2.3 Data Definition Language (DDL).....	244
2.3.1 Creating a Source Table.....	244
2.3.1.1 Kafka Source Table.....	244
2.3.1.2 DIS Source Table.....	247
2.3.1.3 JDBC Source Table.....	250
2.3.1.4 GaussDB(DWS) Source Table.....	252
2.3.1.5 Redis Source Table.....	255
2.3.1.6 HBase Source Table.....	256
2.3.1.7 userDefined Source Table.....	258
2.3.2 Creating a Result Table.....	259
2.3.2.1 ClickHouse Result Table.....	260
2.3.2.2 Kafka Result Table.....	263
2.3.2.3 Upsert Kafka Result Table.....	264
2.3.2.4 DIS Result Table.....	266
2.3.2.5 JDBC Result Table.....	268
2.3.2.6 GaussDB(DWS) Result Table.....	269
2.3.2.7 Redis Result Table.....	272
2.3.2.8 SMN Result Table.....	276
2.3.2.9 HBase Result Table.....	277
2.3.2.10 Elasticsearch Result Table.....	279
2.3.2.11 OpenTSDB Result Table.....	282
2.3.2.12 User-defined Result Table.....	284
2.3.2.13 Print Result Table.....	286
2.3.2.14 File System Result Table.....	288
2.3.3 Creating a Dimension Table.....	290
2.3.3.1 JDBC Dimension Table.....	291
2.3.3.2 GaussDB(DWS) Dimension Table.....	293
2.3.3.3 HBase Dimension Table.....	296
2.4 Data Manipulation Language (DML).....	297
2.4.1 SELECT.....	297
2.4.2 Set Operations.....	301
2.4.3 Window.....	302
2.4.4 JOIN.....	307
2.4.5 OrderBy & Limit.....	310
2.4.6 Top-N.....	311
2.4.7 Deduplication.....	312
2.5 Functions.....	313

2.5.1 User-Defined Functions.....	313
2.5.2 Built-In Functions.....	317
2.5.2.1 Mathematical Operation Functions.....	317
2.5.2.2 String Functions.....	325
2.5.2.3 Temporal Functions.....	331
2.5.2.4 Conditional Functions.....	355
2.5.2.5 Type Conversion Function.....	356
2.5.2.6 Collection Functions.....	358
2.5.2.7 Value Construction Functions.....	358
2.5.2.8 Value Access Functions.....	359
2.5.2.9 Hash Functions.....	359
2.5.2.10 Aggregate Function.....	360
2.5.2.11 Table-Valued Functions.....	361
2.5.2.11.1 split_cursor.....	361
2.5.2.11.2 string_split.....	362
3 Historical Version.....	364
3.1 Flink SQL Syntax (This Syntax Will Not Evolve. Use FlinkOpenSource SQL Instead.).....	364
3.1.1 Constraints and Definitions.....	364
3.1.2 Overview.....	365
3.1.3 Creating a Source Stream.....	366
3.1.3.1 CloudTable HBase Source Stream.....	366
3.1.3.2 DIS Source Stream.....	368
3.1.3.3 DMS Source Stream.....	373
3.1.3.4 MRS Kafka Source Stream.....	373
3.1.3.5 Open-Source Kafka Source Stream.....	377
3.1.3.6 OBS Source Stream.....	381
3.1.4 Creating a Sink Stream.....	383
3.1.4.1 CloudTable HBase Sink Stream.....	383
3.1.4.2 CloudTable OpenTSDB Sink Stream.....	386
3.1.4.3 MRS OpenTSDB Sink Stream.....	388
3.1.4.4 CSS Elasticsearch Sink Stream.....	390
3.1.4.5 DCS Sink Stream.....	392
3.1.4.6 DDS Sink Stream.....	395
3.1.4.7 DIS Sink Stream.....	396
3.1.4.8 DMS Sink Stream.....	399
3.1.4.9 DWS Sink Stream (JDBC Mode).....	399
3.1.4.10 DWS Sink Stream (OBS-based Dumping).....	402
3.1.4.11 MRS HBase Sink Stream.....	405
3.1.4.12 MRS Kafka Sink Stream.....	407
3.1.4.13 Open-Source Kafka Sink Stream.....	410
3.1.4.14 File System Sink Stream (Recommended).....	412
3.1.4.15 OBS Sink Stream.....	415

3.1.4.16 RDS Sink Stream.....	419
3.1.4.17 SMN Sink Stream.....	422
3.1.5 Creating a Temporary Stream.....	424
3.1.6 Creating a Dimension Table.....	424
3.1.6.1 Creating a Redis Table.....	424
3.1.6.2 Creating an RDS Table.....	425
3.1.7 Custom Stream Ecosystem.....	428
3.1.7.1 Custom Source Stream.....	428
3.1.7.2 Custom Sink Stream.....	429
3.1.8 Data Manipulation Language (DML).....	430
3.1.8.1 SELECT.....	430
3.1.8.2 Condition Expression.....	434
3.1.8.3 Window.....	436
3.1.8.4 JOIN Between Stream Data and Table Data.....	438
3.1.9 Data Types.....	440
3.1.10 User-Defined Functions.....	444
3.1.11 Built-In Functions.....	449
3.1.11.1 Mathematical Operation Functions.....	449
3.1.11.2 String Functions.....	454
3.1.11.3 Temporal Functions.....	468
3.1.11.4 Type Conversion Functions.....	472
3.1.11.5 Aggregate Functions.....	474
3.1.11.6 Table-Valued Functions.....	479
3.1.11.7 Other Functions.....	479
3.1.12 Geographical Functions.....	480
3.1.13 Configuring Time Models.....	488
3.1.14 Pattern Matching.....	490
3.1.15 StreamingML.....	494
3.1.15.1 Anomaly Detection.....	494
3.1.15.2 Time Series Forecasting.....	496
3.1.15.3 Real-Time Clustering.....	498
3.1.15.4 Deep Learning Model Prediction.....	499
3.1.16 Reserved Keywords.....	501

1 Flink OpenSource SQL 1.12 Syntax Reference

1.1 Constraints and Definitions

1.1.1 Supported Data Types

STRING, BOOLEAN, BYTES, DECIMAL, TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, DATE, TIME, TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL, ARRAY, MULTISSET, MAP, ROW

1.1.2 Syntax

1.1.2.1 Data Definition Language (DDL)

1.1.2.1.1 CREATE TABLE

Syntax

```
CREATE TABLE table_name
(
  { <column_definition> | <computed_column_definition> }[, ...n]
  [ <watermark_definition> ]
  [ <table_constraint> ][, ...n]
)
[COMMENT table_comment]
[PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
WITH (key1=val1, key2=val2, ...)
```

<column_definition>:
column_name column_type [<column_constraint>] [COMMENT column_comment]

<column_constraint>:
[CONSTRAINT constraint_name] PRIMARY KEY NOT ENFORCED

<table_constraint>:
[CONSTRAINT constraint_name] PRIMARY KEY (column_name, ...) NOT ENFORCED

<computed_column_definition>:


```
column_name AS computed_column_expression [COMMENT column_comment]

<watermark_definition>:
  WATERMARK FOR rowtime_column_name AS watermark_strategy_expression

<source_table>:
  [catalog_name.][db_name.]table_name
```

Function

Create a table with a specified name.

Description

COMPUTED COLUMN

A computed column is a virtual column generated using **column_name AS computed_column_expression**. A computed column evaluates an expression that can reference other columns declared in the same table. The column itself is not physically stored within the table. A computed column could be defined using **cost AS price * quantity**. This expression can contain any combination of physical columns, constants, functions, or variables, but cannot contain any subquery.

In Flink, a computed column is used to define the time attribute in **CREATE TABLE** statements. A processing time attribute can be defined easily via **proc AS PROCTIME()** using the system's **PROCTIME()** function. The event time column may be obtained from an existing field. In this case, you can use the computed column to obtain event time. For example, if the original field is not of the **TIMESTAMP(3)** type or is nested in a JSON string, you can use computed columns.

Note:

- An expression that defines a computed column in a source table is calculated after data is read from the data source. The column can be used in the **SELECT** statement.
- A computed column cannot be the target of an **INSERT** statement. In an **INSERT** statement, the schema of the **SELECT** statement must be the same as that of the target table that does not have a computed column.

WATERMARK

The **WATERMARK** clause defines the event time attribute of a table and takes the form **WATERMARK FOR rowtime_column_name AS watermark_strategy_expression**.

rowtime_column_name defines an existing column that is marked as the event time attribute of the table. The column must be of the **TIMESTAMP(3)** type and must be the top-level column in the schema. It can also be a computed column.

watermark_strategy_expression defines the watermark generation strategy. It allows arbitrary non-query expressions, including computed columns, to calculate the watermark. The expression return type must be **TIMESTAMP(3)**, which represents the timestamp since the Epoch. The returned watermark will be emitted only if it is non-null and its value is greater than the previously emitted local watermark (to preserve the contract of ascending watermarks). The watermark generation expression is evaluated by the framework for every record.

The framework will periodically emit the largest generated watermark. If the current watermark is still identical to the previous one, or is null, or the value of the returned watermark is smaller than that of the last emitted one, then no new watermark will be emitted. A watermark is emitted in an interval defined by **pipeline.auto-watermark-interval**. If the watermark interval is 0 ms, a watermark will be emitted per record if it is not null and greater than the last emitted one.

When using event time semantics, tables must contain an event time attribute and watermark strategy.

Flink provides several commonly used watermark strategies.

- Strictly ascending timestamps: **WATERMARK FOR rowtime_column AS rowtime_column**
Emits a watermark of the maximum observed timestamp so far. Rows that have a timestamp bigger than the maximum timestamp are not late.
- Ascending timestamps: **WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SECOND**
Emits a watermark of the maximum observed timestamp so far minus 1. Rows that have a timestamp bigger than or equal to the maximum timestamp are not late.
- Bounded out-of-order timestamps: **WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'string' timeUnit**
Emits a watermark, which is the maximum observed timestamp minus the specified delay, for example, **WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '5' SECOND** is a 5-second delayed watermark strategy.

```
CREATE TABLE Orders (  
  user BIGINT,  
  product STRING,  
  order_time TIMESTAMP(3),  
  WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND  
) WITH (...);
```

PRIMARY KEY

The primary key constraint is a hint for Flink to leverage for optimizations. It tells that a column or a set of columns of a table or a view are unique and they do not contain null. Neither of columns in a primary can be nullable. The primary key therefore uniquely identifies a row in a table.

The primary key constraint can be either declared along with a column definition (a column constraint) or as a single line (a table constraint). For both cases, it should only be declared as a singleton. If you define multiple primary key constraints at the same time, an exception would be thrown.

Validity Check

SQL standard specifies that a constraint can either be **ENFORCED** or **NOT ENFORCED**. This controls if the constraint checks are performed on the incoming/outgoing data. Flink does not own the data and therefore the only mode we want to support is the **NOT ENFORCED** mode. It is up to the user to ensure that the query enforces key integrity.

Flink will assume correctness of the primary key by assuming that the columns nullability is aligned with the columns in the primary key. Connectors should ensure those are aligned.

Note: In a **CREATE TABLE** statement, creating a primary key constraint will alter the columns nullability, which means, a column with a primary key constraint is not nullable.

PARTITIONED BY

Partition the created table by the specified columns. A directory is created for each partition if this table is used as a file system sink.

WITH OPTIONS

Table properties used to create a table source/sink. The properties are usually used to find and create the underlying connector.

The key and value of expression **key1=val1** should both be string literal.

Note: The table registered with the **CREATE TABLE** statement can be used as both the table source and table sink. We cannot decide if it is used as a source or sink until it is referenced in the DMLs.

1.1.2.1.2 CREATE VIEW

Syntax

```
CREATE VIEW [IF NOT EXISTS] view_name  
  [{columnName [, columnName ]* }] [COMMENT view_comment]  
  AS query_expression
```

Function

Create a view with multiple layers nested in it to simplify the development process.

Description

IF NOT EXISTS

If the view already exists, nothing happens.

Example

Create a view named **viewName**.

```
create view viewName as select * from dataSource
```

1.1.2.1.3 CREATE FUNCTION

Syntax

```
CREATE FUNCTION  
  [IF NOT EXISTS] function_name  
  AS identifier [LANGUAGE JAVA|SCALA]
```

Function

Create a user-defined function.

For details about how to create a user-defined function, see [User-Defined Functions \(UDFs\)](#).

Description

IF NOT EXISTS

If the function already exists, nothing happens.

LANGUAGE JAVA|SCALA

The language tag is used to instruct Flink runtime how to execute the function. Currently, only **JAVA** and **SCALA** language tags are supported, the default language for a function is **JAVA**.

Example

Create a function named **STRINGBACK**.

```
create function STRINGBACK as 'com.dli.StringBack'
```

1.1.2.2 Data Manipulation Language (DML)

DML Statements

Syntax

```
INSERT INTO table_name [PARTITION part_spec] query
part_spec: (part_col_name1=val1 [, part_col_name2=val2, ...])
query:
values
| {
  select
  | selectWithoutFrom
  | query UNION [ ALL ] query
  | query EXCEPT query
  | query INTERSECT query
  }
  [ ORDER BY orderItem [, orderItem ]* ]
  [ LIMIT { count | ALL } ]
  [ OFFSET start { ROW | ROWS } ]
  [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]
orderItem:
expression [ ASC | DESC ]
select:
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ]
[ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]
selectWithoutFrom:
SELECT [ ALL | DISTINCT ]
```

```

{ * | projectItem [, projectItem ]* }

projectItem:
  expression [ [ AS ] columnAlias ]
  | tableAlias . *

tableExpression:
  tableReference [, tableReference ]*
  | tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression [ joinCondition ]

joinCondition:
  ON booleanExpression
  | USING '(' column [, column ]* ')'

tableReference:
  tablePrimary
  [ matchRecognize ]
  [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]

tablePrimary:
  [ TABLE ] [ [ catalogName . ] schemaName . ] tableName
  | LATERAL TABLE '(' functionName '(' expression [, expression ]* ')' ')'
  | UNNEST '(' expression ')'

values:
  VALUES expression [, expression ]*

groupByItem:
  expression
  | '(' ')'
  | '(' expression [, expression ]* ')'
  | CUBE '(' expression [, expression ]* ')'
  | ROLLUP '(' expression [, expression ]* ')'
  | GROUPING SETS '(' groupItem [, groupItem ]* ')'

windowRef:
  windowName
  | windowSpec

windowSpec:
  [ windowName ]
  '('
  [ ORDER BY orderItem [, orderItem ]* ]
  [ PARTITION BY expression [, expression ]* ]
  [
    RANGE numericOrIntervalExpression {PRECEDING}
    | ROWS numericExpression {PRECEDING}
  ]
  ')'

matchRecognize:
  MATCH_RECOGNIZE '('
  [ PARTITION BY expression [, expression ]* ]
  [ ORDER BY orderItem [, orderItem ]* ]
  [ MEASURES measureColumn [, measureColumn ]* ]
  [ ONE ROW PER MATCH ]
  [ AFTER MATCH
    ( SKIP TO NEXT ROW
    | SKIP PAST LAST ROW
    | SKIP TO FIRST variable
    | SKIP TO LAST variable
    | SKIP TO variable )
  ]
  PATTERN '(' pattern ')'
  [ WITHIN intervalLiteral ]
  DEFINE variable AS condition [, variable AS condition ]*
  ')'

measureColumn:

```

```

expression AS alias
pattern:
  patternTerm [ '|' patternTerm ]*
patternTerm:
  patternFactor [ patternFactor ]*
patternFactor:
  variable [ patternQuantifier ]
patternQuantifier:
  '*'
  | '*?'
  | '+'
  | '+?'
  | '?'
  | '??'
  | '{ [ minRepeat ], [ maxRepeat ] }' ['?']
  | '{ repeat }'

```

Precautions

Flink SQL uses a lexical policy for identifier (table, attribute, function names) similar to Java:

- The case of identifiers is preserved whether they are quoted.
- Identifiers are matched case-sensitively.
- Unlike Java, back-ticks allow identifiers to contain non-alphanumeric characters (for example, **SELECT a AS `my field` FROM t**).

String literals must be enclosed in single quotes (for example, **SELECT'Hello World'**). Duplicate a single quote for escaping (for example, **SELECT 'It''s me.'**). Unicode characters are supported in string literals. If explicit Unicode points are required, use the following syntax:

- Use the backslash (\) as an escaping character (default): **SELECT U&'\263A'**
- Use a custom escaping character: **SELECT U&'#263A' UESCAPE '#'**

1.2 Overview

This section describes the Flink open source SQL 1.12 syntax supported by DLI. For details about the parameters and examples, see the syntax description.

Creating Tables

Table 1-1 Syntax for creating tables

Classification	Function
Creating a source table	DataGen Source Table
	GaussDB(DWS) Source Table
	HBase Source Table
	JDBC Source Table

Classification	Function
	Kafka Source Table
	MySQL CDC Source Table
	Postgres CDC Source Table
	Redis Source Table
	Upsert Kafka Source Table
Creating a result table	BlackHole Result Table
	ClickHouse Result Table
	GaussDB(DWS) Result Table
	Elasticsearch Result Table
	HBase Result Table
	JDBC Result Table
	Kafka Result Table
	Print Result Table
	Redis Result Table
	Upsert Kafka Result Table
Creating a dimension table	GaussDB(DWS) Dimension Table
	HBase Dimension Table
	JDBC Dimension Table
	Redis Dimension Table
Format	Avro
	Canal
	Confluent Avro
	CSV
	Debezium
	JSON
	Maxwell
	Raw

1.3 DDL Syntax

1.3.1 Creating Source Tables

1.3.1.1 DataGen Source Table

Function

DataGen is used to generate random data for debugging and testing.

Prerequisites

None

Precautions

- When you create a DataGen table, the table field type cannot be Array, Map, or Row. You can use **COMPUTED COLUMN** in **CREATE TABLE** to construct similar functions.
- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.

Syntax

```
create table dataGenSource(
  attr_name attr_type
  ('; attr_name attr_type)*
  ('; WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
)
with (
  'connector' = 'datagen'
);
```

Parameters

Table 1-2 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to datagen .
rows-per-second	No	10000	Long	Number of rows generated per second, which is used to control the emit rate.

Parameter	Mandatory	Default Value	Data Type	Description
fields.# .kind	No	random	String	<p>Generator of the # field. The # field must be an actual field in the DataGen table. Replace # with the corresponding field name. The meanings of the # field for other parameters are the same.</p> <p>The value can be sequence or random.</p> <ul style="list-style-type: none"> random is the default generator. You can use the fields.#.max and fields.#.min parameters to specify the maximum and minimum values that are randomly generated. If the specified field type is char, varchar, or string, you can also use the fields.#.length field to specify the length. A random generator is an unbounded generator. Sequence generator. You can use fields.#.start and fields.#.end to specify the start and end values of a sequence. A sequence generator is a bounded generator. When the sequence number reaches the end value, the reading ends.
fields.# .min	No	Minimum value of the field type specified by #	Field type specified by #	<p>This parameter is valid only when fields.#.kind is set to random.</p> <p>Minimum value of the random generator. It applies only to numeric field types specified by #.</p>
fields.# .max	No	Maximum value of the field type specified by #	Field type specified by #	<p>This parameter is valid only when fields.#.kind is set to random.</p> <p>Maximum value of the random generator. It applies only to numeric field types specified by #.</p>
fields.# .length	No	100	Integer	<p>This parameter is valid only when fields.#.kind is set to random.</p> <p>Length of the characters generated by the random generator. It applies only to char, varchar, and string types specified by #.</p>

Parameter	Mandatory	Default Value	Data Type	Description
fields.#.start	No	None	Field type specified by #	This parameter is valid only when fields.#.kind is set to sequence . Start value of a sequence generator.
fields.#.end	No	None	Field type specified by #	This parameter is valid only when fields.#.kind is set to sequence . End value of a sequence generator.

Example

Create a Flink OpenSource SQL job. Run the following script to generate random data through the DataGen table and output the data to the Print result table.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

```
create table dataGenSource(
  user_id string,
  amount int
) with (
  'connector' = 'datagen',
  'rows-per-second' = '1', --Generates a piece of data per second.
  'fields.user_id.kind' = 'random', --Specifies a random generator for the user_id field.
  'fields.user_id.length' = '3' --Limits the length of user_id to 3.
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print'
);

insert into printSink select * from dataGenSource;
```

After the job is submitted, the job status changes to **Running**. You can perform the following operations to view the output result:

- Method 1:
 - a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - b. Locate the row that contains the target Flink job, and choose **More > FlinkUI** in the **Operation** column.
 - c. On the Flink UI, choose **Task Managers**, click the task name, and select **Stdout** to view job logs.
- Method 2: If you select **Save Job Log** on the **Running Parameters** tab before submitting the job, perform the following operations:

- a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
- b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
- c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

1.3.1.2 GaussDB(DWS) Source Table

Function

DLI reads data of Flink jobs from GaussDB(DWS). GaussDB(DWS) database kernel is compliant with PostgreSQL. The PostgreSQL database can store data of more complex types and deliver space information services, multi-version concurrent control (MVCC), and high concurrency. It applies to location applications, financial insurance, and e-Commerce.

GaussDB(DWS) is an online data processing database based on the cloud infrastructure and platform and helps you mine and analyze massive sets of data.

For more information about DWS, see [Data Warehouse Service \(DWS\)](#)

Prerequisites

- You have created a GaussDB(DWS) cluster.
For details about how to create a GaussDB(DWS) cluster, see **Creating a Cluster** in the *Data Warehouse Service Management Guide*.
- You have created a GaussDB(DWS) database table.
- An enhanced datasource connection has been created for DLI to connect to GaussDB(DWS) clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Precautions

When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.

Syntax

```
create table dwsSource (  
  attr_name attr_type  
  (' attr_name attr_type)*  
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
  (' watermark for rowtime_column_name as watermark-strategy_expression)  
)  
with (  
  'connector' = 'gaussdb',  
  'url' = "
```

```
'table-name' = "",
'username' = "",
'password' = ""
);
```

Parameters

Table 1-3 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to gaussdb .
url	Yes	None	String	JDBC connection address. Set the IP address in this parameter to the internal IP address of GaussDB(DWS). If you use the gsjdbc4 driver, set the value in jdbc:postgresql://\${ip}:\${port}/\${dbName} format. If you use the gsjdbc200 driver, set the value in jdbc:gaussdb://\${ip}:\${port}/\${dbName} format.
table-name	Yes	None	String	Name of the GaussDB(DWS) table to be operated. If the GaussDB(DWS) table is in a schema, refer to the description of GaussDB(DWS) table in a schema .
driver	No	org.postgresql.Driver	String	JDBC connection driver. The default value is org.postgresql.Driver . <ul style="list-style-type: none"> If you use the gsjdbc4 driver for connection, set this parameter to org.postgresql.Driver. If you use the gsjdbc200 driver for connection, set this parameter to com.huawei.gauss200.jdbc.Driver.
username	No	None	String	Username for GaussDB(DWS) database authentication. This parameter must be configured in pair with password .
password	No	None	String	Password for GaussDB(DWS) database authentication. This parameter must be configured in pair with username .

Parameter	Mandatory	Default Value	Data Type	Description
scan.partition.column	No	None	String	Name of the column used to partition the input. Note: This parameter must be used together with scan.partition.lower-bound , scan.partition.upper-bound , and scan.partition.num .
scan.partition.lower-bound	No	None	Integer	Lower bound of values to be fetched for the first partition. This parameter must be used together with scan.partition.column , scan.partition.upper-bound , and scan.partition.num .
scan.partition.upper-bound	No	None	Integer	Upper bound of values to be fetched for the last partition. This parameter must be used together with scan.partition.column , scan.partition.lower-bound , and scan.partition.num .
scan.partition.num	No	None	Integer	Number of partitions to be created. This parameter must be used together with scan.partition.column , scan.partition.upper-bound , and scan.partition.upper-bound .
scan.fetch-size	No	0	Integer	Number of rows fetched from the database each time. The default value is 0 , indicating that the number of rows is not limited.
pwd_auth_name	No	None	String	Name of password datasource authentication created on DLI. If this parameter is set, you do not need to set the password in SQL statements.

Example

In this example, data is read from the GaussDB(DWS) data source and written to the Print result table. The procedure is as follows:

1. Create a table named **dws_order** in GaussDB(DWS).

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR);
```

Insert data into the **dws_order** table.

```
insert into public.dws_order
(order_id,
order_channel,
order_time,
pay_amount,
real_pay,
pay_time,
user_id,
user_name,
area_id) values
('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');
```

2. Create an enhanced datasource connection in the VPC and subnet where GaussDB(DWS) locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
3. Set GaussDB(DWS) security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the GaussDB(DWS) address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job. The job script uses the GaussDB(DWS) data source and the Print result table.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE dwsSource (
order_id string,
order_channel string,
order_time string,
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'gaussdb',
'url' = 'jdbc:postgresql://DWSIP:DWSPort/DWSdbName',
'table-name' = 'dws_order',
'driver' = 'org.postgresql.Driver',
'username' = 'DWSUserName',
'password' = 'DWSPassword'
);

CREATE TABLE printSink (
order_id string,
order_channel string,
order_time string,
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'print'
);

insert into printSink select * from dwsSource;
```

5. Perform the following operations to view the data result in the **taskmanager.out** file:
 - a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
+I(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,330110)
```

FAQ

- Q: What should I do if the job execution fails and the log contains the following error information?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

A: The datasource connection is not bound or the binding fails.

- To reconfigure datasource connections, refer to [Enhanced Datasource Connection](#). Rectify the fault by referring to [DLI Failed to Connect to GaussDB\(DWS\) Through an Enhanced Datasource Connection](#).
- Q: How can I configure a GaussDB(DWS) table that is in a schema?
- A: The following provides an example of configuring the **dws_order** table in the **dbuser2** schema:

```
CREATE TABLE dwsSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DWSIP:DWSPort/DWSdbName',
  'table-name' = 'dbuser2"."dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword'
);
```

1.3.1.3 HBase Source Table

Function

Create a source stream to obtain data from HBase as input for jobs. HBase is a column-oriented distributed cloud storage system that features enhanced

reliability, excellent performance, and elastic scalability. It applies to the storage of massive amounts of data and distributed computing. You can use HBase to build a storage system capable of storing TB- or even PB-level data. With HBase, you can filter and analyze data with ease and get responses in milliseconds, rapidly mining data value. DLI can read data from HBase for filtering, analysis, and data dumping.

Prerequisites

- An enhanced datasource connection has been created for DLI to connect to HBase, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).
- If MRS HBase is used, IP addresses of all hosts in the MRS cluster have been added to host information of the enhanced datasource connection.

For details, see section "Modifying the Host Information" in the *Data Lake Insight User Guide*.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- The column families in created HBase source table must be declared as the ROW type, the field names map the column family names, and the nested field names map the column qualifier names.

There is no need to declare all the families and qualifiers in the schema. Users can declare what is used in the query. Except the ROW type fields, the single atomic type field (for example, STRING or BIGINT) will be recognized as the HBase rowkey. The rowkey field can be an arbitrary name, but should be quoted using backticks if it is a reserved keyword.

Syntax

```
create table hbaseSource (  
  attr_name attr_type  
  (',' attr_name attr_type)*  
  (',' watermark for rowtime_column_name as watermark_strategy_expression)  
  ','PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
)  
with (  
  'connector' = 'hbase-2.2',  
  'table-name' = "",  
  'zookeeper.quorum' = ""  
);
```


Parameters

Table 1-4 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to hbase-2.2 .
table-name	Yes	None	String	Name of the HBase table to connect.
zookeeper.quorum	Yes	None	String	HBase ZooKeeper quorum, in the format of "ZookeeperAddress:ZookeeperPort". The following uses an MRS HBase cluster as an example to describe how to obtain the IP address and port number of ZooKeeper used by this parameter: <ul style="list-style-type: none"> On MRS Manager, choose Cluster and click the name of the desired cluster. Choose Services > ZooKeeper > Instance, and obtain the IP address of the ZooKeeper instance. On MRS Manager, choose Cluster and click the name of the desired cluster. Choose Services > ZooKeeper > Configurations > All Configurations, search for the clientPort parameter, and obtain its value, that is, the ZooKeeper port number.
zookeeper.znode.parent	No	/hbase	String	Root directory in ZooKeeper. The default value is /hbase .
null-string-literal	No	None	String	Representation for null values for string fields. HBase source encodes/decodes empty bytes as null values for all types except the string type.

Data Type Mapping

HBase stores all data as byte arrays. The data needs to be serialized and deserialized during read and write operations.

When serializing and de-serializing, Flink HBase connector uses utility class **org.apache.hadoop.hbase.util.Bytes** provided by HBase (Hadoop) to convert Flink data types to and from byte arrays.

Flink HBase connector encodes null values to empty bytes, and decode empty bytes to null values for all data types except the string type. For the string type, the null literal is determined by the **null-string-literal** option.

Table 1-5 Data type mapping

Flink SQL Type	HBase Conversion
CHAR/VARCHAR/STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY/VARBINARY	Returns byte[] as is.
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
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	Stores the number of days since epoch as an int value.
TIME	Stores the number of milliseconds of the day as an int value.
TIMESTAMP	Stores the milliseconds since epoch as a long value.
ARRAY	Not supported
MAP/MULTISET	Not supported
ROW	Not supported

Example

In this example, data is read from the HBase data source and written to the Print result table. The procedure is as follows (the HBase versions used in this example are 1.3.1, 2.1.1, and 2.2.3):

1. Create an enhanced datasource connection in the VPC and subnet where HBase locates, and bind the connection to the required Flink queue. For details, see [Enhanced Datasource Connections](#).
2. Set HBase cluster security groups and add inbound rules to allow access from the Flink job queue. Test the connectivity using the HBase address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

3. Use the HBase shell to create HBase table **order** that has only one column family **detail**. The creation statement is as follows:

```
create 'order', {NAME => 'detail'}
```

4. Run the following command in the HBase shell to insert a data record:

```
put 'order', '202103241000000001', 'detail:order_channel','webShop'  
put 'order', '202103241000000001', 'detail:order_time','2021-03-24 10:00:00'  
put 'order', '202103241000000001', 'detail:pay_amount','100.00'  
put 'order', '202103241000000001', 'detail:real_pay','100.00'  
put 'order', '202103241000000001', 'detail:pay_time','2021-03-24 10:02:03'  
put 'order', '202103241000000001', 'detail:user_id','0001'  
put 'order', '202103241000000001', 'detail:user_name','Alice'  
put 'order', '202103241000000001', 'detail:area_id','330106'
```

5. Create a Flink OpenSource SQL job. Enter the following job script and submit the job. The job script uses the HBase data source and the Print result table.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
create table hbaseSource (  
  order_id string,-- Indicates the unique rowkey.  
  detail Row( -- Indicates the column family.  
    order_channel string,  
    order_time string,  
    pay_amount string,  
    real_pay string,  
    pay_time string,  
    user_id string,  
    user_name string,  
    area_id string),  
  primary key (order_id) not enforced  
) with (  
  'connector' = 'hbase-2.2',  
  'table-name' = 'order',  
  'zookeeper.quorum' = 'ZookeeperAddress.ZookeeperPort'  
) ;  
  
create table printSink (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount string,  
  real_pay string,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) with (  
  'connector' = 'print'  
) ;
```

```
insert into printSink select order_id,  
detail.order_channel,detail.order_time,detail.pay_amount,detail.real_pay,  
detail.pay_time,detail.user_id,detail.user_name,detail.area_id from hbaseSource;
```

6. Perform the following operations to view the data result in the **taskmanager.out** file:
 - a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.00,100.00,2021-03-24 10:02:03,0001,Alice,  
330106)
```

FAQ

- Q: What should I do if the Flink job execution fails and the log contains the following error information?
java.lang.IllegalArgumentException: offset (0) + length (8) exceed the capacity of the array: 6
A: If data in the HBase table is imported in other modes, the data is represented in the string format. Therefore, this error is reported when other data formats are used. Change the type of the non-string fields in the HBase source table created by Flink to the string format.
- Q: What should I do if the Flink job execution fails and the log contains the following error information?
org.apache.zookeeper.ClientCnxn\$SessionTimeoutException: Client session timed out, have not heard from server in 90069ms for connection id 0x0
A: The datasource connection is not bound, the binding fails, or the security group of the HBase cluster is not configured to allow access from the network segment of the DLI queue. Configure the datasource connection by referring to [Enhanced Datasource Connection](#) or configure the security group of the HBase cluster to allow access from the DLI queue.

1.3.1.4 JDBC Source Table

Function

The JDBC connector is a Flink's built-in connector to read data from a database.

Prerequisites

- An enhanced datasource connection with the instances has been established, so that you can configure security group rules as required.
- For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
- For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Precautions

When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.

Syntax

```
create table jdbcSource (
  attr_name attr_type
  ('; attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  ('; watermark for rowtime_column_name as watermark-strategy_expression)
) with (
  'connector' = 'jdbc',
  'url' = "",
  'table-name' = "",
  'username' = "",
  'password' = ""
);
```

Parameters

Table 1-6 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	No	String	Connector to be used. Set this parameter to jdbc .
url	Yes	No	String	Database URL.
table-name	Yes	No	String	Name of the table where the data will be read from the database.
driver	No	No	String	Driver required for connecting to the database. If you do not set this parameter, it will be automatically derived from the URL.
username	No	No	String	Database authentication username. This parameter must be configured in pair with password .
password	No	No	String	Database authentication password. This parameter must be configured in pair with username .
scan.partition.column	No	No	String	Name of the column used to partition the input. For details, see Partitioned Scan .
scan.partition.num	No	No	Integer	Number of partitions to be created. For details, see Partitioned Scan .

Parameter	Mandatory	Default Value	Data Type	Description
scan.partition.lower-bound	No	None	Integer	Lower bound of values to be fetched for the first partition. For details, see Partitioned Scan .
scan.partition.upper-bound	No	None	Integer	Upper bound of values to be fetched for the last partition. For details, see Partitioned Scan .
scan.fetch-size	No	0	Integer	Number of rows fetched from the database each time. If this parameter is set to 0, the SQL hint is ignored.
scan.auto-commit	No	true	Boolean	Whether each statement is committed in a transaction automatically.

Partitioned Scan

To accelerate reading data in parallel Source task instances, Flink provides the partitioned scan feature for the JDBC table. The following parameters describe how to partition the table when reading in parallel from multiple tasks.

- **scan.partition.column**: name of the column used to partition the input. The data type of the column must be number, date, or timestamp.
- **scan.partition.num**: number of partitions.
- **scan.partition.lower-bound**: minimum value of the first partition.
- **scan.partition.upper-bound**: maximum value of the last partition.

NOTE

- When a table is created, the preceding partitioned scan parameters must all be specified if any of them is specified.
- The **scan.partition.lower-bound** and **scan.partition.upper-bound** parameters are used to decide the partition stride instead of filtering rows in the table. All rows in the table are partitioned and returned.

Data Type Mapping

Table 1-7 Data type mapping

MySQL Type	PostgreSQL Type	Flink SQL Type
TINYINT	-	TINYINT

MySQL Type	PostgreSQL Type	Flink SQL Type
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

MySQL Type	PostgreSQL Type	Flink SQL Type
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

Example

This example uses JDBC as the data source and Print as the sink to read data from the RDS MySQL database and write the data to the Print result table.

1. Create an enhanced datasource connection in the VPC and subnet where RDS MySQL locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set RDS MySQL security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the RDS address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Log in to the RDS MySQL database, create table **orders** in the Flink database, and insert data.

Create table **orders** in the Flink database.

```
CREATE TABLE `flink`.`orders` (
  `order_id` VARCHAR(32) NOT NULL,
  `order_channel` VARCHAR(32) NULL,
  `order_time` VARCHAR(32) NULL,
  `pay_amount` DOUBLE UNSIGNED NOT NULL,
  `real_pay` DOUBLE UNSIGNED NULL,
  `pay_time` VARCHAR(32) NULL,
  `user_id` VARCHAR(32) NULL,
  `user_name` VARCHAR(32) NULL,
  `area_id` VARCHAR(32) NULL,
  PRIMARY KEY (`order_id`)
) ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_general_ci;
```

Insert data into the table.

```
insert into orders(
  order_id,
  order_channel,
  order_time,
  pay_amount,
  real_pay,
```



```
pay_time,  
user_id,  
user_name,  
area_id) values  
(('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',  
'0001', 'Alice', '330106'),  
(('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',  
'0002', 'Bob', '330110'));
```

4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE jdbcSource (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'jdbc',  
  'url' = 'jdbc:mysql://MySQLAddress.MySQLPort/flink',--flink is the database name created in RDS  
  MySQL.  
  'table-name' = 'orders',  
  'username' = 'MySQLUsername',  
  'password' = 'MySQLPassword'  
);
```

```
CREATE TABLE printSink (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'print'  
);
```

```
insert into printSink select * from jdbcSource;
```

5. Perform the following operations to view the data result in the **taskmanager.out** file:
 - a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,  
330106)  
+I(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,  
330110)
```

FAQ

None

1.3.1.5 Kafka Source Table

Function

Create a source stream to obtain data from Kafka as input data for jobs.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

- You have created a Kafka cluster.
- An enhanced datasource connection has been created for DLI to connect to Kafka clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- For details about how to use data types when creating tables, see [Format](#).

Syntax

```
create table kafkaSource(  
  attr_name attr_type  
  (' attr_name attr_type)*  
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
  (' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)  
)  
with (  
  'connector' = 'kafka',  
  'topic' = "",  
  'properties.bootstrap.servers' = "",  
  'properties.group.id' = "",  
  'scan.startup.mode' = "",  
  'format' = ""  
);
```

Parameters

Table 1-8 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to kafka .
topic	Yes	None	String	Topic name of the Kafka record. Note: <ul style="list-style-type: none"> Only one of topic and topic-pattern can be specified. If there are multiple topics, separate them with semicolons (;), for example, topic-1;topic-2.
topic-pattern	No	None	String	Regular expression for a pattern of topic names to read from. Only one of topic and topic-pattern can be specified. For example: 'topic.*' '(topic-c topic-d)' '(topic-a topic-b topic-\\d*)' '(topic-a topic-b topic-[0-9]*)'
properties.bootstrap.servers	Yes	None	String	Comma separated list of Kafka brokers.
properties.group.id	Yes	None	String	ID of the consumer group for the Kafka source.

Parameter	Mandatory	Default Value	Data Type	Description
properties.*	No	None	String	<p>This parameter can set and pass arbitrary Kafka configurations.</p> <p>Note:</p> <ul style="list-style-type: none"> The suffix to properties. must match the configuration key in Apache Kafka. For example, you can disable automatic topic creation via 'properties.allow.auto.create.topics' = 'false'. Some configurations are not supported, for example, 'key.deserializer' and 'value.deserializer'.
format	Yes	None	String	<p>Format used to deserialize and serialize the value part of Kafka messages. Note: Either this parameter or the value.format parameter is required. Refer to Format for more details and format parameters.</p>
key.format	No	None	String	<p>Format used to deserialize and serialize the key part of Kafka messages.</p> <p>Note:</p> <ul style="list-style-type: none"> If a key format is defined, the key.fields parameter is required as well. Otherwise, the Kafka records will have an empty key. Refer to Format for more details and format parameters.
key.fields	No	[]	List<String>	<p>Defines the columns in the table as the list of keys. This parameter must be configured in pair with key.format. This parameter is left empty by default. Therefore, no key is defined. The format is like field1;field2.</p>
key.fields-prefix	No	None	String	<p>Defines a custom prefix for all fields of the key format to avoid name clashes with fields of the value format.</p>

Parameter	Mandatory	Default Value	Data Type	Description
value.format	Yes	None	String	<p>Format used to deserialize and serialize the value part of Kafka messages.</p> <p>Note:</p> <ul style="list-style-type: none"> • Either this parameter or the format parameter is required. If two parameters are configured, a conflict occurs. • Refer to Format for more details and format parameters.
value.fields-include	No	ALL	Enum Possible values: [ALL, EXCEPT_KEY]	<p>Whether to contain the key field when parsing the message body.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • ALL (default): All defined fields are included in the value of Kafka messages. • EXCEPT_KEY: All the fields except those defined by key.fields are included in the value of Kafka messages.
scan.startup.mode	No	group-offsets	String	<p>Start position for Kafka to read data.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • earliest-offset: Data is read from the earliest Kafka offset. • latest-offset: Data is read from the latest Kafka offset. • group-offsets (default): Data is read based on the consumer group. • timestamp: Data is read from a user-supplied timestamp. When setting this option, you also need to specify scan.startup.timestamp-millis in WITH. • specific-offsets: Data is read from user-supplied specific offsets for each partition. When setting this option, you also need to specify scan.startup.specific-offsets in WITH.

Parameter	Mandatory	Default Value	Data Type	Description
scan.startup-specific-offsets	No	None	String	This parameter takes effect only when scan.startup.mode is set to specific-offsets . It specifies the offsets for each partition, for example, partition:0,offset:42;partition:1,offset:300 .
scan.startup.timestamp-millis	No	None	Long	Startup timestamp. This parameter takes effect when scan.startup.mode is set to timestamp .
scan.topic-partition-discovery.interval	No	None	Duration	Interval for a consumer to periodically discover dynamically created Kafka topics and partitions.

Metadata Column

You can define metadata columns in the source table to obtain the metadata of Kafka messages. For example, if multiple topics are defined in the **WITH** parameter and the metadata column is defined in the Kafka source table, the data read by Flink is labeled with the topic from which the data is read.

Table 1-9 Metadata column

Key	Data Type	R/W	Description
topic	STRING NOT NULL	R	Topic name of the Kafka record.
partition	INT NOT NULL	R	Partition ID of the Kafka record.
headers	MAP<STRING, BYTES> NOT NULL	R/W	Headers of Kafka messages.
leader-epoch	INT NULL	R	Leader epoch of the Kafka record. For details, see example 1.
offset	BIGINT NOT NULL	R	Offset of the Kafka record.
timestamp	TIMESTAMP(3) WITH LOCAL TIME ZONE NOT NULL	R/W	Timestamp of the Kafka record.

Key	Data Type	R/W	Description
timestamp-type	STRING NOT NULL	R	<p>Timestamp type of the Kafka record. The options are as follows:</p> <ul style="list-style-type: none"> • NoTimestampType: No timestamp is defined in the message. • CreateTime: time when the message is generated. • LogAppendTime: time when the message is added to the Kafka broker. <p>For details, see example 1.</p>

Example (SASL_SSL Disabled for the Kafka Cluster)

- **Example 1: Read data from the Kafka metadata column and write it to the Print sink.**
 - Create an enhanced datasource connection in the VPC and subnet where Kafka locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
 - Set Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
 - Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Change the values of the parameters in bold as needed in the following script.**

```
CREATE TABLE orders (
  `topic` String metadata,
  `partition` int metadata,
  `headers` MAP<STRING, BYTES> metadata,
  `leaderEpoch` INT metadata from 'leader-epoch',
  `offset` bigint metadata,
  `timestamp` TIMESTAMP(3) metadata,
  `timestampType` string metadata from 'timestamp-type',
  `message` string
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  "format" = "csv",
```

```
"csv.field-delimiter" = "\u0001",
"csv.quote-character" = ""
);

CREATE TABLE printSink (
  `topic` String,
  `partition` int,
  `headers` MAP<STRING, BYTES>,
  `leaderEpoch` INT,
  `offset` bigint,
  `timestamp` TIMESTAMP(3),
  `timestampType` string,
  `message` string -- Indicates that data written by users is read from Kafka.
) WITH (
  'connector' = 'print'
);

insert into printSink select * from orders;
```

If you need to read the value of each field instead of the entire message, use the following statements:

```
CREATE TABLE orders (
  `topic` String metadata,
  `partition` int metadata,
  `headers` MAP<STRING, BYTES> metadata,
  `leaderEpoch` INT metadata from 'leader-epoch',
  `offset` bigint metadata,
  `timestamp` TIMESTAMP(3) metadata,
  `timestampType` string metadata from 'timestamp-type',
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupID',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE printSink (
  `topic` String,
  `partition` int,
  `headers` MAP<STRING, BYTES>,
  `leaderEpoch` INT,
  `offset` bigint,
  `timestamp` TIMESTAMP(3),
  `timestampType` string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from orders;
```


- d. Send the following data to the corresponding topics in Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

- e. Perform the following operations to view the output:

- i. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
- ii. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
- iii. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+l(fz-source-json,0,{}),0,243,2021-12-27T09:23:32.253,CreateTime,
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"})
+l(fz-source-json,0,{}),0,244,2021-12-27T09:23:39.655,CreateTime,
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"})
+l(fz-source-json,0,{}),0,245,2021-12-27T09:23:48.405,CreateTime,
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"})
```

- **Example 2: Use the Kafka source table and Print result table to read JSON data from Kafka and output it to the log file.**

- a. Create an enhanced datasource connection in the VPC and subnet where Kafka locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
- b. Set Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
- c. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Change the values of the parameters in bold as needed in the following script.**

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
```

```
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = '<yourTopic>',
'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
'properties.group.id' = 'GroupId',
'scan.startup.mode' = 'latest-offset',
'format' = 'json'
);

CREATE TABLE printSink (
order_id string,
order_channel string,
order_time timestamp(3),
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'print'
);

insert into printSink select * from orders;
```

- d. Send the following test data to the corresponding topics in Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24
10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03",
"user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24
16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06",
"user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

- e. Perform the following operations to view the output:
- Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+|(202103241000000001,webShop,2021-03-24T10:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,
330106)
+|(202103241606060001,appShop,
2021-03-24T16:06:06,200.0,180.0,2021-03-2416:10:06,0001,Alice,330106)
+|(202103251202020001,miniAppShop,
2021-03-25T12:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
```

Example (SASL_SSL Enabled for the Kafka Cluster)

- **Example 1: Enable SASL_SSL authentication for the DMS cluster.**

Create a Kafka cluster for DMS, enable SASL_SSL, download the SSL certificate, and upload the downloaded certificate **client.jks** to an OBS bucket.

```
CREATE TABLE ordersSource (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',  
  'properties.group.id' = 'Groupld',  
  'scan.startup.mode' = 'latest-offset',  
  'properties.connector.auth.open' = 'true',  
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks', -- Location where the user uploads the  
certificate to  
  'properties.sasl.mechanism' = 'PLAIN', -- Value format: SASL_PLAINTEXT  
  'properties.security.protocol' = 'SASL_SSL',  
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required  
username=\"xx\" password=\"xx\";', -- Account and password set when the Kafka cluster is created  
  "format" = "json"  
);  
  
CREATE TABLE ordersSink (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',  
  'properties.connector.auth.open' = 'true',  
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks',  
  'properties.sasl.mechanism' = 'PLAIN',  
  'properties.security.protocol' = 'SASL_SSL',  
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required  
username=\"xx\" password=\"xx\";',  
  "format" = "json"  
);  
  
insert into ordersSink select * from ordersSource;
```

- **Example 2: Enable Kafka SASL_SSL authentication for the MRS cluster.**
 - Enable Kerberos authentication for the MRS cluster.
 - Click the **Components** tab and click **Kafka**. In the displayed page, click the **Service Configuration** tab, locate the **security.protocol**, and set it to **SASL_SSL**.
 - Log in to the FusionInsight Manager of the MRS cluster and download the user credential. Choose **System > Permission > User**. Locate the row that contains the target user, choose **More > Download Authentication Credential**.
Obtain the **truststore.jks** file using the authentication credential and store the credential and **truststore.jks** file in OBS.
 - If "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier

than `8u_242` or delete the `renew_lifetime = 0m` configuration item from the `krb5.conf` configuration file.

- Set the port to the `ssl.port` configured in the Kafka service configuration.
- In the following statements, set `security.protocol` to `SASL_SSL`.

```
CREATE TABLE ordersSource (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',  
  'properties.group.id' = 'Group1d',  
  'scan.startup.mode' = 'latest-offset',  
  'properties.sasl.kerberos.service.name' = 'kafka',  
  'properties.connector.auth.open' = 'true',  
  'properties.connector.kerberos.principal' = 'xx', --Username  
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',  
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',  
  'properties.security.protocol' = 'SASL_SSL',  
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',  
  'properties.ssl.truststore.password' = 'xx', -- Password set for generating truststore.jks  
  'properties.sasl.mechanism' = 'GSSAPI',  
  "format" = "json"  
)  
);  
  
CREATE TABLE ordersSink (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',  
  'properties.sasl.kerberos.service.name' = 'kafka',  
  'properties.connector.auth.open' = 'true',  
  'properties.connector.kerberos.principal' = 'xx',  
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',  
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',  
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',  
  'properties.ssl.truststore.password' = 'xx',  
  'properties.security.protocol' = 'SASL_SSL',  
  'properties.sasl.mechanism' = 'GSSAPI',  
  "format" = "json"  
)  
);  
  
insert into ordersSink select * from ordersSource;
```

- **Example 3: Enable Kerberos SASL_PAINTTEXT authentication for the MRS cluster**

- Enable Kerberos authentication for the MRS cluster.

- Click the **Components** tab and click **Kafka**. In the displayed page, click the **Service Configuration** tab, locate the **security.protocol**, and set it to **SASL_PLAINTEXT**.
- Log in to the FusionInsight Manager of the MRS cluster and download the user credential. Choose **System > Permission > User**. Locate the row that contains the target user, choose **More > Download Authentication Credential**. Upload the credential to OBS.
- If error message "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier than 8u_242 or delete the **renew_lifetime = 0m** configuration item from the **krb5.conf** configuration file.
- Set the port to the **sasl.port** configured in the Kafka service configuration.
- In the following statements, set **security.protocol** to **SASL_PLAINTEXT**.

```
CREATE TABLE ordersSources (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:21007,xx:21007',  
  'properties.group.id' = 'Groupld',  
  'scan.startup.mode' = 'latest-offset',  
  'properties.sasl.kerberos.service.name' = 'kafka',  
  'properties.connector.auth.open' = 'true',  
  'properties.connector.kerberos.principal' = 'xx',  
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',  
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',  
  'properties.security.protocol' = 'SASL_PLAINTEXT',  
  'properties.sasl.mechanism' = 'GSSAPI',  
  "format" = "json"  
);  
  
CREATE TABLE ordersSink (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:21007,xx:21007',  
  'properties.sasl.kerberos.service.name' = 'kafka',  
  'properties.connector.auth.open' = 'true',  
  'properties.connector.kerberos.principal' = 'xx',  
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',  
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',  
  'properties.security.protocol' = 'SASL_PLAINTEXT',  
  'properties.sasl.mechanism' = 'GSSAPI',  
  "format" = "json"  
);
```

```
insert into ordersSink select * from ordersSource;
```

- **Example 4: Use SSL for the MRS cluster**

- Do not enable Kerberos authentication for the MRS cluster.
- Log in to the FusionInsight Manager of the MRS cluster and download the user credential. Choose **System > Permission > User**. Locate the row that contains the target user, choose **More > Download Authentication Credential**.

Obtain the **truststore.jks** file using the authentication credential and store the credential and **truststore.jks** file in OBS.

- Set the port to the **ssl.port** configured in the Kafka service configuration.
- In the following statements, set **security.protocol** to **SSL**.
- Set **ssl.mode.enable** to **true**.

```
CREATE TABLE ordersSource (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',  
  'properties.group.id' = 'Groupld',  
  'scan.startup.mode' = 'latest-offset',  
  'properties.connector.auth.open' = 'true',  
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',  
  'properties.ssl.truststore.password' = 'xx', -- Password set for generating truststore.jks  
  'properties.security.protocol' = 'SSL',  
  "format" = "json"  
);
```

```
CREATE TABLE ordersSink (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'print'  
);
```

```
insert into ordersSink select * from ordersSource;
```

FAQ

- **Q: What should I do if the Flink job execution fails and the log contains the following error information?**

```
org.apache.kafka.common.errors.TimeoutException: Timeout expired while fetching topic metadata
```

A: The datasource connection is not bound, the binding fails, or the security group of the Kafka cluster is not configured to allow access from the network segment of the DLI queue. Configure the datasource connection by referring

to [Enhanced Datasource Connection](#) or configure the security group of the Kafka cluster to allow access from the DLI queue.

- **Q: What should I do if the Flink job execution fails and the log contains the following error information?**

Caused by: java.lang.RuntimeException: RealLine:45;Table 'default_catalog.default_database.printSink' declares persistable metadata columns, but the underlying DynamicTableSink doesn't implement the SupportsWritingMetadata interface. If the column should not be persisted, it can be declared with the VIRTUAL keyword.

A: The metadata type is defined in the sink table, but the Print connector does not support deletion of metadata from the sink table.

1.3.1.6 MySQL CDC Source Table

Function

The MySQL CDC source table, that is, the MySQL streaming source table, reads all historical data in the database first and then smoothly switches data read to the Binlog to ensure data integrity.

Prerequisites

- MySQL CDC requires MySQL 5.7 or 8.0.x.
- An enhanced datasource connection has been created for DLI to connect to the MySQL database, so that you can configure security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).
- Binlog is enabled for MySQL, and `binlog_row_image` is set to **FULL**.
- A MySQL user has been created and granted the **SELECT**, **SHOW DATABASES**, **REPLICATION SLAVE**, and **REPLICATION CLIENT** permissions.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- Each client that synchronizes database data has a unique ID, that is, the server ID. You are advised to configure a unique server ID for each MySQL CDC job in the same database.

Main reasons are as follows:

 - The MySQL server maintains the network connection and Binlog location based on the ID. Therefore, if a large number of clients with the same server ID connect to the MySQL server, the CPU usage of the MySQL server may increase sharply, affecting the stability of online services.
 - If multiple jobs share the same server ID, Binlog locations will be disordered, making data read inaccurate. Therefore, you are advised to configure different server IDs for each MySQL CDC job.
- Watermarks cannot be defined for MySQL CDC source tables. For details about window aggregation, see [FAQ](#).

- If you connect to a sink source that supports upsert, such as GaussDB(DWS) and MySQL, you need to define the primary key in the statement for creating the sink table. For details, see the printSink table creation statement in [Example](#).

Syntax

```
create table mySqlCdcSource (
  attr_name attr_type
  (' attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'mysql-cdc',
  'hostname' = 'mysqlHostname',
  'username' = 'mysqlUsername',
  'password' = 'mysqlPassword',
  'database-name' = 'mysqlDatabaseName',
  'table-name' = 'mysqlTableName'
);
```

Parameters

Table 1-10 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to mysql-cdc .
hostname	Yes	None	String	IP address or hostname of the MySQL database.
username	Yes	None	String	Username of the MySQL database.
password	Yes	None	String	Password of the MySQL database.
database-name	Yes	None	String	Name of the database to connect. The database name supports regular expressions to read data from multiple databases. For example, flink(.) * indicates all database names starting with flink .
table-name	Yes	None	String	Name of the table to read data from. The table name supports regular expressions to read data from multiple tables. For example, cdc_order(.) * indicates all table names starting with cdc_order .
port	No	3306	Integer	Port number of the MySQL database.

Parameter	Mandatory	Default Value	Data Type	Description
server-id	No	A random value from 5400 to 6000	String	A numeric ID of the database client, which must be globally unique in the MySQL cluster. You are advised to set a unique ID for each job in the same database. By default, a random value ranging from 5400 to 6400 is generated.
scan.startup.mode	No	initial	String	Startup mode for consuming data. <ul style="list-style-type: none"> ● initial (default): In the first startup, the database scans all historical data and then reads the latest Binlog data. ● latest-offset: In the first startup, the database reads data directly from the end of the Binlog (the latest Binlog) instead of scanning all historical data. That is, it reads only the latest changes after the connector is started.
server-time-zone	No	None	String	Time zone of the session used by the database. For example, Asia/Shanghai .

Example

In this example, MySQL-CDC is used to read data from RDS for MySQL in real time and write the data to the Print result table. The procedure is as follows (MySQL 5.7.32 is used in this example):

1. Create an enhanced datasource connection in the VPC and subnet where MySQL locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set MySQL security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the MySQL address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Create a table named **cdc_order** in database **flink** of the MySQL database.

```
CREATE TABLE `flink`.`cdc_order` (
  `order_id` VARCHAR(32) NOT NULL,
  `order_channel` VARCHAR(32) NULL,
  `order_time` VARCHAR(32) NULL,
  `pay_amount` DOUBLE NULL,
  `real_pay` DOUBLE NULL,
```

```
`pay_time` VARCHAR(32) NULL,  
`user_id` VARCHAR(32) NULL,  
`user_name` VARCHAR(32) NULL,  
`area_id` VARCHAR(32) NULL,  
PRIMARY KEY (`order_id`)  
) ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8mb4  
COLLATE = utf8mb4_general_ci;
```

4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
create table mysqlCdcSource(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id STRING  
) with (  
  'connector' = 'mysql-cdc',  
  'hostname' = 'mysqlHostname',  
  'username' = 'mysqlUsername',  
  'password' = 'mysqlPassword',  
  'database-name' = 'mysqlDatabaseName',  
  'table-name' = 'mysqlTableName'  
)  
);  
  
create table printSink(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id STRING,  
  primary key(order_id) not enforced  
) with (  
  'connector' = 'print'  
)  
);  
  
insert into printSink select * from mysqlCdcSource;
```

5. Insert test data in MySQL.

```
insert into cdc_order values  
('202103241000000001','webShop','2021-03-24 10:00:00','100.00','100.00','2021-03-24 10:02:03','0001','Alice','330106'),  
('202103241606060001','appShop','2021-03-24 16:06:06','200.00','180.00','2021-03-24 16:10:06','0001','Alice','330106');  
  
delete from cdc_order where order_channel = 'webShop';  
  
insert into cdc_order values('202103251202020001','miniAppShop','2021-03-25 12:02:02','60.00','60.00','2021-03-25 12:03:00','0002','Bob','330110');
```

6. Perform the following operations to view the data result in the **taskmanager.out** file:

- a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.

- b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
- c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+I(202103241000000001,webShop,2021-03-2410:00:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330106)
+I(202103241606060001,appShop,2021-03-2416:06:06,200.0,180.0,2021-03-2416:10:06,0001,Alice,330106)
-D(202103241000000001,webShop,2021-03-2410:00:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330106)
+I(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
```

FAQ

Q: How do I perform window aggregation if the MySQL CDC source table does not support definition of watermarks?

A: You can use the non-window aggregation method. That is, convert the time field into a window value, and then use **GROUP BY** to perform aggregation based on the window value.

For example, you can use the following script to collect statistics on the number of orders per minute (**order_time** indicates the order time, in the string format):

```
insert into printSink select DATE_FORMAT(order_time, 'yyyy-MM-dd HH:mm'), count(*) from mysqlCdcSource group by DATE_FORMAT(order_time, 'yyyy-MM-dd HH:mm');
```

1.3.1.7 Postgres CDC Source Table

Function

The Postgres CDC source table, that is, Postgres streaming source table, is used to read the full snapshot data and changed data of the PostgreSQL database in sequence. The exactly-once processing semantics is used to ensure data accuracy even if a failure occurs.

Prerequisites

- The PostgreSQL version be 9.6, 10, 11, or 12.
- An enhanced datasource connection with the database has been established, so that you can configure security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When you create a Flink OpenSource SQL job, set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.

- The PostgreSQL version cannot be earlier than PostgreSQL 11.
- If operations such as update will be performed on the Postgres table, you need to run the following statement in PostgreSQL. Note: Replace **test.cdc_order** with the actual database and table.
ALTER TABLE *test.cdc_order* REPLICA IDENTITY FULL
- Before creating the PostgreSQL CDC source table, check whether the current PostgreSQL contains the default plug-in. You can run the following statement in PostgreSQL to query the current plug-ins:
SELECT name FROM pg_available_extensions;
If the default plug-in **decoderbufs** is not available, you need to set the **decoding.plugin.name** parameter to specify an existing plug-in in PostgreSQL when creating the PostgreSQL CDC source table.

Syntax

```
create table postgresCdcSource (
  attr_name attr_type
  (,' attr_name attr_type)*
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'postgres-cdc',
  'hostname' = 'PostgresHostname',
  'username' = 'PostgresUsername',
  'password' = 'PostgresPassword',
  'database-name' = 'PostgresDatabaseName',
  'schema-name' = 'PostgresSchemaName',
  'table-name' = 'PostgresTableName'
);
```

Parameters

Table 1-11 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to postgres-cdc .
hostname	Yes	None	String	IP address or hostname of the Postgres database.
username	Yes	None	String	Username of the Postgres database.
password	Yes	None	String	Password of the Postgres database.
database-name	Yes	None	String	Database name.

Parameter	Mandatory	Default Value	Data Type	Description
schema-name	Yes	None	String	Postgres schema name. The schema name supports regular expressions to read data from multiple schemas. For example, test(.)* indicates all schema names starting with test .
table-name	Yes	None	String	Postgres table name. The table name supports regular expressions to read data from multiple tables. For example, cdc_order(.)* indicates all table names starting with cdc_order .
port	No	5432	Integer	Port number of the Postgres database.
decoding.plugin.name	No	decoderbufs	String	Determined based on the plug-in that is installed in the PostgreSQL database. The value can be: <ul style="list-style-type: none"> • decoderbufs (default) • wal2json • wal2json_rds • wal2json_streaming • wal2json_rds_streaming • pgoutput
debezium.*	No	None	String	Fine-grained control over the behavior of Debezium clients, for example, 'debezium.snapshot.mode' = 'never' . For details, see Connector configuration properties . You are advised to set the debezium.slot.name parameter for each table to avoid the following error: "PSQLException: ERROR: replication slot "debezium" is active for PID 974"

Example

In this example, Postgres-CDC is used to read data from RDS for PostgreSQL in real time and write the data to the Print result table. The procedure is as follows (PostgreSQL 11.11 is used in this example):

1. Create an enhanced datasource connection in the VPC and subnet where PostgreSQL locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set PostgreSQL security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the PostgreSQL address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. In PostgreSQL, create database **flink** and schema **test**.
4. Create table **cdc_order** in the schema **test** of database **flink** in PostgreSQL.

```
create table test.cdc_order(  
  order_id VARCHAR,  
  order_channel VARCHAR,  
  order_time VARCHAR,  
  pay_amount FLOAT8,  
  real_pay FLOAT8,  
  pay_time VARCHAR,  
  user_id VARCHAR,  
  user_name VARCHAR,  
  area_id VARCHAR,  
  primary key(order_id)  
);
```

5. Run the following SQL statement in PostgreSQL. If you do not run this statement, an error will be reported when the Flink job is executed. For details, see the error message in [FAQ](#).

```
ALTER TABLE test.cdc_order REPLICA IDENTITY FULL
```

6. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
create table postgresCdcSource(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id STRING,  
  primary key (order_id) not enforced  
) with (  
  'connector' = 'postgres-cdc',  
  'hostname' = 'PostgresHostname',  
  'username' = 'PostgresUsername',  
  'password' = 'PostgresPassword',  
  'database-name' = 'flink',  
  'schema-name' = 'test',  
  'table-name' = 'cdc_order'  
);
```

```
create table printSink(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id STRING,
```

```
primary key(order_id) not enforced
) with (
  'connector' = 'print'
);

insert into printSink select * from postgresCdcSource;
```

7. Run the following command in PostgreSQL:

```
insert into test.cdc_order
(order_id,
order_channel,
order_time,
pay_amount,
real_pay,
pay_time,
user_id,
user_name,
area_id) values
('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');

update test.cdc_order set order_channel = 'webShop' where order_id = '202103251202020001';

delete from test.cdc_order where order_id = '202103241000000001';
```

8. Perform the following operations to view the data result in the **taskmanager.out** file:

- a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
- b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
- c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
+I(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,330110)
-U(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,330110)
+U(202103251202020001,webShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,330110)
-D(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
```

FAQ

- Q: What should I do if the Flink job execution fails and the log contains the following error information?
org.postgresql.util.PSQLException: ERROR: logical decoding requires wal_level >= logical
- A: Change the value of **wal_level** to **logical** and restart the PostgreSQL database.
After modifying the PostgreSQL parameter, restart the RDS PostgreSQL instance for the modification to take effect.
- Q: What should I do if the Flink job execution fails and the log contains the following error information?

```
java.lang.IllegalStateException: The "before" field of UPDATE/DELETE message is null, please check the Postgres table has been set REPLICIA IDENTITY to FULL level. You can update the setting by running the command in Postgres 'ALTER TABLE test.cdc_order REPLICIA IDENTITY FULL'.
```

A: If a similar error is reported in the run log, run the **ALTER TABLE test.cdc_order REPLICIA IDENTITY FULL** statement in PostgreSQL.

1.3.1.8 Redis Source Table

Function

Create a source stream to obtain data from Redis as input for jobs.

Prerequisites

An enhanced datasource connection has been created for DLI to connect to the Redis database, so that you can configure security group rules as required.

- For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
- For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- To obtain the key values, you can set the primary key in Flink. The primary key maps to the Redis key.
- The primary key cannot be a composite primary key, and only can be one field.
- Constraints on **schema-syntax**:
 - If **schema-syntax** is **map** or **array**, there can be only one non-primary key and it must be of the same **map** or **array** type.
 - If **schema-syntax** is **fields-scores**, the number of non-primary keys must be an even number, and the second key of every two keys except the primary key must be of the **double** type. The **double** value is the score of the previous key. The following is an example:

```
CREATE TABLE redisSource (  
  redisKey string,  
  order_id string,  
  score1 double,  
  order_channel string,  
  score2 double,  
  order_time string,  
  score3 double,  
  pay_amount double,  
  score4 double,  
  real_pay double,  
  score5 double,  
  pay_time string,  
  score6 double,  
  user_id string,  
  score7 double,  
  user_name string,  
  score8 double,
```



```
area_id string,  
score9 double,  
primary key (redisKey) not enforced  
) WITH (  
'connector' = 'redis',  
'host' = 'RedisIP',  
'password' = 'RedisPassword',  
'data-type' = 'sorted-set',  
'deploy-mode' = 'master-replica',  
'schema-syntax' = 'fields-scores'  
);
```

- Restrictions on **data-type**:
 - When **data-type** is **set**, the types of non-primary keys defined in Flink must be the same.
 - If **data-type** is **sorted-set** and **schema-syntax** is **fields** or **array**, only **sorted-set** values can be read from Redis, and the **score** value cannot be read.
 - If **data-type** is **string**, only one non-primary key field is allowed.
 - If **data-type** is **sorted-set** and **schema-syntax** is **map**, only one non-primary key field is allowed besides the primary key field.
- If **data-type** is **sorted-set** and **schema-syntax** is **array-scores**, only two non-primary keys are allowed and must be of the **array** type.

This non-primary key field must be of the **map** type. The map value of the field must be of the **double** type, indicating the score. The map key of the field indicates the value in the Redis set.

The first key indicates values in the Redis set. The second key is of the **array<double>** type, indicating index scores. The following is an example:

```
CREATE TABLE redisSink (  
  order_id string,  
  arrayField Array<String>,  
  arrayScore array<double>,  
  primary key (order_id) not enforced  
) WITH (  
'connector' = 'redis',  
'host' = 'RedisIP',  
'password' = 'RedisPassword',  
'data-type' = 'sorted-set',  
"default-score" = '3',  
'deploy-mode' = 'master-replica',  
'schema-syntax' = 'array-scores'  
);
```

Syntax

```
create table dwsSource (  
  attr_name attr_type  
  (,' attr_name attr_type)*  
  (,' watermark for rowtime_column_name as watermark-strategy_expression)  
  ,PRIMARY KEY (attr_name, ...) NOT ENFORCED  
)  
with (  
  'connector' = 'redis',  
  'host' = "  
);
```

Parameters

Table 1-12 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to redis .
host	Yes	None	String	Redis connector address.
port	No	6379	Integer	Redis connector port.
password	No	None	String	Redis authentication password.
namespace	No	None	String	Redis key namespace.
delimiter	No	:	String	Delimiter between the Redis key and namespace.
data-type	No	hash	String	Redis data type. Available values are as follows: <ul style="list-style-type: none"> • hash • list • set • sorted-set • string For details about the constraints, see Constraints on data-type .
schema-syntax	No	fields	String	Redis schema semantics. Available values are as follows (for details, see Precautions and FAQ): <ul style="list-style-type: none"> • fields: applicable to all data types • fields-scores: applicable to sorted-set data • array: applicable to list, set, and sorted-set data • array-scores: applicable to sorted-set data • map: applicable to hash and sorted-set data For details about the constraints, see Constraints on schema-syntax .

Parameter	Mandatory	Default Value	Data Type	Description
deploy-mode	No	standalone	String	Deployment mode of the Redis cluster. The value can be standalone , master-replica , or cluster . The default value is standalone .
retry-count	No	5	Integer	Number of attempts to connect to the Redis cluster.
connection-timeout-millis	No	10000	Integer	Maximum timeout for connecting to the Redis cluster.
commands-timeout-millis	No	2000	Integer	Maximum time for waiting for a completion response.
rebalancing-timeout-millis	No	15000	Integer	Sleep time when the Redis cluster fails.
scan-keys-count	No	1000	Integer	Number of data records read in each scan.
default-score	No	0	Double	Default score when data-type is sorted-set .
deserialize-error-policy	No	fail-job	Enum	Policy of how to process a data parsing failure. Available values are as follows: <ul style="list-style-type: none">• fail-job: Fail the job.• skip-row: Skip the current data.• null-field: Set the current data to null.
skip-null-values	No	true	Boolean	Whether null values will be skipped.

Example

In this example, data is read from the DCS Redis data source and written to the Print result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where Redis locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set Redis security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Redis address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

3. Run the following commands on the Redis client to insert data into different keys and store the data in hash format:

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time "2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24 10:02:03" user_id 0001 user_name Alice area_id 330106
```

```
HMSET redisSource1 order_id 202103241606060001 order_channel appShop order_time "2021-03-24 16:06:06" pay_amount 200.00 real_pay 180.00 pay_time "2021-03-24 16:10:06" user_id 0001 user_name Alice area_id 330106
```

```
HMSET redisSource2 order_id 202103251202020001 order_channel miniAppShop order_time "2021-03-25 12:02:02" pay_amount 60.00 real_pay 60.00 pay_time "2021-03-25 12:03:00" user_id 0002 user_name Bob area_id 330110
```

4. Create a Flink OpenSource SQL job. Enter the following job script to read data in hash format from Redis.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE redisSource (  
  redisKey string,  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  primary key (redisKey) not enforced --Obtains the key value from Redis.  
) WITH (  
  'connector' = 'redis',  
  'host' = 'RedisIP',  
  'password' = 'RedisPassword',  
  'data-type' = 'hash',  
  'deploy-mode' = 'master-replica'  
)  
;  
  
CREATE TABLE printSink (  
  redisKey string,  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'print'  
)  
;
```

```
insert into printSink select * from redisSource;
```

5. Perform the following operations to view the data result in the **taskmanager.out** file:
 - a. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - b. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.

- c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+l(redisSource1,202103241606060001,appShop,2021-03-24 16:06:06,200.0,180.0,2021-03-24
16:10:06,0001,Alice,330106)
+l(redisSource,202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24
10:02:03,0001,Alice,330106)
+l(redisSource2,202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
```

FAQ

- Q: What should I do if the Flink job execution fails and the log contains the following error information?

Caused by: org.apache.flink.client.program.ProgramInvocationException: The main method caused an error: RealLine:36;Usage of 'set' data-type and 'fields' schema syntax in source Redis connector with multiple non-key column types. As 'set' in Redis is not sorted, it's not possible to map 'set's values to table schema with different types.

A: If **data-type** is **set**, the data types of non-primary key fields in Flink are different. As a result, this error is reported. When **data-type** is **set**, the types of non-primary keys defined in Flink must be the same.

- Q: If **data-type** is **hash**, what are the differences between **schema-syntax** set to **fields** and that to **map**?

A: When **schema-syntax** is set to **fields**, the hash value in the Redis key is assigned to the field with the same name in Flink. When **schema-syntax** is set to **map**, the hash key and hash value of each hash in Redis are put into a map, which represents the value of the corresponding Flink field. Specifically, this map contains all hash keys and hash values of a key in Redis.

– For **fields**:

- i. Insert the following data into Redis:

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time
"2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24
10:02:03" user_id 0001 user_name Alice area_id 330106
```

- ii. When **schema-syntax** is set to **fields**, use the following job script:

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'hash',
  'deploy-mode' = 'master-replica'
);

CREATE TABLE printSink (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
```

```
real_pay double,  
pay_time string,  
user_id string,  
user_name string,  
area_id string  
) WITH (  
  'connector' = 'print'  
);  
  
insert into printSink select * from redisSource;
```

iii. The job execution result is as follows:

```
+l(redisSource,202103241000000001,webShop,2021-03-24  
10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
```

– For **map**:

i. Insert the following data into Redis:

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time  
"2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24  
10:02:03" user_id 0001 user_name Alice area_id 330106
```

ii. When **schema-syntax** is set to **map**, use the following job script:

```
CREATE TABLE redisSource (  
  redisKey string,  
  order_result map<string, string>,  
  primary key (redisKey) not enforced  
) WITH (  
  'connector' = 'redis',  
  'host' = 'RedisIP',  
  'password' = 'RedisPassword',  
  'data-type' = 'hash',  
  'deploy-mode' = 'master-replica',  
  'schema-syntax' = 'map'  
);  
  
CREATE TABLE printSink (  
  redisKey string,  
  order_result map<string, string>  
) WITH (  
  'connector' = 'print'  
);  
  
insert into printSink select * from redisSource;
```

iii. The job execution result is as follows:

```
+l(redisSource,{user_id=0001, user_name=Alice, pay_amount=100.00, real_pay=100.00,  
order_time=2021-03-24 10:00:00, area_id=330106, order_id=202103241000000001,  
order_channel=webShop, pay_time=2021-03-24 10:02:03})
```

1.3.1.9 Upsert Kafka Source Table

Function

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

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 does not 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, also known as INSERT/UPDATE, because any existing

row with the same key is overwritten. Also, null values are interpreted in a special way: A record with a null value represents a DELETE.

Prerequisites

An enhanced datasource connection has been created for DLI to connect to Kafka clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.

- For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
- For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- The Upsert Kafka always works in the upsert fashion and requires to define the primary key in the DDL. With the assumption that records with the same key should be ordered in the same partition, the primary key semantic on the changelog source means the materialized changelog is unique on the primary keys. The primary key definition will also control which fields should end up in Kafka's key.
- Because the connector is working in upsert mode, the last record on the same key will take effect when reading back as a source.
- For details about how to use data types, see section [Format](#).

Syntax

```
create table kafkaSource(
  attr_name attr_type
  (' attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'upsert-kafka',
  'topic' = "",
  'properties.bootstrap.servers' = "",
  'key.format' = "",
  'value.format' = ""
);
```

Parameters

Table 1-13 Parameter description

Parameter	Man dato ry	Defa ult Valu e	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to upsert-kafka .

Parameter	Mandatory	Default Value	Data Type	Description
topic	Yes	None	String	Kafka topic name.
properties.bootstrap.servers	Yes	None	String	Comma separated list of Kafka brokers.
key.format	Yes	None	String	Format used to deserialize and serialize the key part of Kafka messages. The key fields are specified by the PRIMARY KEY syntax. The following formats are supported: <ul style="list-style-type: none"> • csv • json • avro Refer to Format for more details and format parameters.
key.fields-prefix	No	None	String	Defines a custom prefix for all fields of the key format to avoid name clashes with fields of the value format. By default, the prefix is empty. If a custom prefix is defined, both the table schema and key.fields will work with prefixed names. When constructing the data type of the key format, the prefix will be removed and the non-prefixed names will be used within the key format. Note that this option requires that value.fields-include must be set to EXCEPT_KEY .
value.format	Yes	None	String	Format used to deserialize and serialize the value part of Kafka messages. The following formats are supported: <ul style="list-style-type: none"> • csv • json • avro Refer to Format for more details and format parameters.

Parameter	Mandatory	Default Value	Data Type	Description
value.fields-include	Yes	ALL	String	Controls which fields should appear in the value part. Possible values are: <ul style="list-style-type: none"> • ALL: All fields in the schema, including the primary key field, are included in the value part. • EXCEPT_KEY: All the fields of the table schema are included, except the primary key field.
properties.*	No	None	String	This option can set and pass arbitrary Kafka configurations. The suffix to properties. must match the parameter defined in Kafka Configuration documentation . Flink will remove the properties. key prefix and pass the transformed key and value to the underlying KafkaClient. For example, you can disable automatic topic creation via 'properties.allow.auto.create.topics' = 'false' . But there are some configurations that do not support to set, because Flink will override them, for example, 'key.deserializer' and 'value.deserializer' .

Example

In this example, data is read from the Kafka data source and written to the Print result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where Kafka locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.
Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE upsertKafkaSource (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  PRIMARY KEY (order_id) NOT ENFORCED  
) WITH (  
  'connector' = 'upsert-kafka',  
  'topic' = 'KafkaTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'key.format' = 'csv',  
  'value.format' = 'json'  
);  
  
CREATE TABLE printSink (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  PRIMARY KEY (order_id) NOT ENFORCED  
) WITH (  
  'connector' = 'print'  
);  
  
INSERT INTO printSink  
SELECT * FROM upsertKafkaSource;
```

4. Insert the following data to the specified topics in Kafka. (Note: Specify the key when inserting data to Kafka.)

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25  
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",  
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",  
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",  
"user_name":"Cindy", "area_id":"330108"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25  
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",  
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

5. Perform the following operations to view the output:
- Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The data result is as follows:

```
+I(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,  
330110)  
+I(202103251505050001,qqShop,2021-03-2515:05:05,500.0,400.0,2021-03-2515:10:00,0003,Cindy,  
330108)
```

```
-U(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
+U(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
```

FAQ

None

1.3.2 Creating Result Tables

1.3.2.1 BlackHole Result Table

Function

The BlackHole connector allows for swallowing all input records. It is designed for high-performance testing and UDF output. It is not a substantive sink. The BlackHole result table is a built-in connector.

For example, if an error is reported when you register a result table of another type, but you are not sure whether it is caused by a system fault or an invalid setting of the **WITH** parameter for the result table, you can change the value of **connector** to **blackhole** and click **Run**. If no error is reported, the system is normal. You must check the settings of the **WITH** parameter.

Prerequisites

None

Precautions

When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.

Syntax

```
create table blackhole_table (
  attr_name attr_type (' attr_name attr_type) *
) with (
  'connector' = 'blackhole'
);
```

Parameters

Table 1-14

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to blackhole .

Example

The DataGen source table generates data, and the BlackHole result table receives the data.

```
create table datagenSource (  
  user_id string,  
  user_name string,  
  user_age int  
) with (  
  'connector' = 'datagen',  
  'rows-per-second'=1'  
);  
create table blackholeSink (  
  user_id string,  
  user_name string,  
  user_age int  
) with (  
  'connector' = 'blackhole'  
);  
insert into blackholeSink select * from datagenSource;
```

1.3.2.2 ClickHouse Result Table

Function

DLI can output Flink job data to the ClickHouse database. ClickHouse is a column-based database oriented to online analysis and processing. It supports SQL query and provides good query performance. The aggregation analysis and query performance based on large and wide tables is excellent, which is one order of magnitude faster than other analytical databases.

Prerequisites

- Your jobs are running on a dedicated queue (non-shared queue) of DLI.
- You have established an enhanced datasource connection to ClickHouse and set the port in the security group rule of the ClickHouse cluster as needed.
- For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
- For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- When you create a ClickHouse cluster for MRS, set the cluster version to MRS 3.1.0 or later and do not enable Kerberos authentication.
- The ClickHouse result table does not support table data deletion.
- Flink supports the following data types: string, tinyint, smallint, int, long, float, double, date, timestamp, decimal, and array.

The array supports only the int, bigint, string, float, and double data types.

Syntax

```
create table clickhouseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = clickhouse,
  'connector.url' = "",
  'connector.table' = ""
);
```

Parameters

Table 1-15 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector.type	Yes	None	String	Result table type. Set this parameter to clickhouse .

Parameter	Mandatory	Default Value	Data Type	Description
connector.url	Yes	None	String	<p>ClickHouse URL.</p> <p>Parameter format: jdbc:clickhouse://ClickHouseBalancer instance IP address:HTTP port number for ClickHouseBalancer instances/Database name</p> <ul style="list-style-type: none"> IP address of a ClickHouseBalancer instance: Log in to the MRS console and choose Clusters > Active Clusters in the navigation pane. Click a cluster name, and choose Components > ClickHouse > Instances to obtain the business IP address of the ClickHouseBalancer instance. HTTP port of a ClickHouseBalancer instance: Log in to the MRS console and choose Clusters > Active Clusters in the navigation pane. Click a cluster name, and choose Components > ClickHouse > Service Configuration. On the Service Configuration page, select ClickHouseBalancer from the All Roles drop-down list, search for lb_http_port, and obtain the parameter value. The default value is 21425. The database name is the name of the database created for the ClickHouse cluster.
connector.table	Yes	None	String	Name of the ClickHouse table to be created.

Parameter	Mandatory	Default Value	Data Type	Description
connector.driver	No	ru.yandex.clickhouse.ClickHouseDriver	String	Driver required for connecting to the database. <ul style="list-style-type: none"> If this parameter is not specified during table creation, the driver automatically extracts the value from the ClickHouse URL. If this parameter is specified during table creation, the value must be ru.yandex.clickhouse.ClickHouseDriver.
connector.username	No	None	String	Username for connecting to the ClickHouse database.
connector.password	No	None	String	Password for connecting to the ClickHouse database.
connector.write.flush.max-rows	No	5000	Integer	Maximum number of rows to be updated when data is written. The default value is 5000 .
connector.write.flush.interval	No	0	Duration	Interval for data update. The unit can be ms, milli, millisecond/s, sec, second/min, or minute. Value 0 indicates that data is not updated.
connector.write.max-retries	No	3	Integer	Maximum number of retries for writing data to the result table. The default value is 3 .

Example

In this example, data is from Kafka and inserted to table **order** in ClickHouse database **flink**. The procedure is as follows (the ClickHouse version is 21.3.4.25 in MRS):

1. Create an enhanced datasource connection in the VPC and subnet where ClickHouse and Kafka clusters locate, and bind the connection to the required Flink queue. For details, see [Enhanced Datasource Connections](#).
2. Set ClickHouse and Kafka cluster security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the ClickHouse address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Use the ClickHouse client to connect to the ClickHouse server and run the following command to query other environment parameters such as the cluster ID:

```
select cluster,shard_num,replica_num,host_name from system.clusters;
```

The following information is displayed:

cluster	shard_num
default_cluster	1
default_cluster	2

- Run the following command to create database **flink** on a node of the ClickHouse cluster based on the obtained cluster ID, for example, **default_cluster**:

```
CREATE DATABASE flink ON CLUSTER default_cluster;
```

- Run the following command to create the ReplicatedMergeTree table named **order** on the node of cluster **default_cluster** and on database **flink**:

```
CREATE TABLE flink.order ON CLUSTER default_cluster(order_id String,order_channel String,order_time String,pay_amount Float64,real_pay Float64,pay_time String,user_id String,user_name String,area_id String) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/flink/order', '{replica}')ORDER BY order_id;
```

- Create a Flink OpenSource SQL job. Enter the following job script and submit the job. The job script uses the Kafka data source and the ClickHouse result table.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
```

```
create table clickhouseSink(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) with (
  'connector.type' = 'clickhouse',
  'connector.url' = 'jdbc:clickhouse://ClickhouseAddress:ClickhousePort/flink',
  'connector.table' = 'order',
  'connector.write.flush.max-rows' = '1'
);
```

```
insert into clickhouseSink select * from orders;
```

- Connect to the Kafka cluster and insert the following test data into Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
```



```
"user_name":"Alice", "area_id":"330106"}
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

8. Use the ClickHouse client to connect to the ClickHouse and run the following command to query the data written to table **order** in database **flink**:

```
select * from flink.order;
```

The query result is as follows:

```
202103241000000001 webShop 2021-03-24 10:00:00 100 100 2021-03-24 10:02:03 0001 Alice 330106
202103241606060001 appShop 2021-03-24 16:06:06 200 180 2021-03-24 16:10:06 0001 Alice 330106
202103251202020001 miniAppShop 2021-03-25 12:02:02 60 60 2021-03-25 12:03:00 0002 Bob
330110
```

FAQ

None

1.3.2.3 GaussDB(DWS) Result Table

Function

DLI outputs the Flink job output data to GaussDB(DWS). GaussDB(DWS) database kernel is compliant with PostgreSQL. The PostgreSQL database can store data of more complex types and deliver space information services, multi-version concurrent control (MVCC), and high concurrency. It applies to location applications, financial insurance, and e-Commerce.

GaussDB(DWS) is an online data processing database based on the cloud infrastructure and platform and helps you mine and analyze massive sets of data.

Prerequisites

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- You have created a GaussDB(DWS) cluster. For details about how to create a GaussDB(DWS) cluster, see **Creating a Cluster** in the *Data Warehouse Service Management Guide*.
- You have created a GaussDB(DWS) database table.
- An enhanced datasource connection has been created for DLI to connect to GaussDB(DWS) clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see **Enhanced Datasource Connections** in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Precautions

- To use the upsert mode, you must define the primary key for both the GaussDB(DWS) result table and the GaussDB(DWS) table connected to the result table.
- If tables with the same name exist in different GaussDB(DWS) schemas, you need to specify the schemas in the Flink open source SQL statements.
- Before submitting a Flink job, you are advised to select **Save Job Log** and set the OBS bucket for saving job logs. This helps you view logs and locate faults when the job fails to be submitted or runs abnormally.
- If you use the gsjdbc4 driver for connection, set **driver** to **org.postgresql.Driver**. You can omit this parameter because the gsjdbc4 driver is the default one.

For example, run the following statements to use the gsjdbc4 driver to write data to GaussDB(DWS) in upsert mode:

```
create table dwsSink(  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_speed INT  
) with (  
  'connector' = 'gaussdb',  
  'url' = 'jdbc:postgresql://DwsAddress:DwsPort/DwsDatabase',  
  'table-name' = 'car_info',  
  'username' = 'DwsUserName',  
  'password' = 'DwsPasswrod',  
  'write.mode' = 'upsert'  
);
```

- If you use the gsjdbc200 driver for connection, set **driver** to **com.huawei.gauss200.jdbc.Driver**.

For example, run the following statements to write data to GaussDB(DWS) result table **test** that is in schema **ads_game_sdk_base**:

```
create table dwsSink(  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_speed INT  
) with (  
  'connector' = 'gaussdb',  
  'table-name' = 'ads_game_sdk_base\".\"test',  
  'driver' = 'com.huawei.gauss200.jdbc.Driver',  
  'url' = 'jdbc:gaussdb://DwsAddress:DwsPort/DwsDatabase',  
  'username' = 'DwsUserName',  
  'password' = 'DwsPasswrod',  
  'write.mode' = 'upsert'  
);
```

Syntax

NOTE

Do not set all attributes in a GaussDB(DWS) result table to **PRIMARY KEY**.

```
create table dwsSink (  
  attr_name attr_type  
  (,' attr_name attr_type)*  
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
)  
with (  
  'connector' = 'gaussdb',  
  'url' = ",
```

```
'table-name' = "",
'driver' = "",
'username' = "",
'password' = ""
);
```

Parameters

Table 1-16 Parameter description

Parameter	Man dato ry	Defau lt Value	Data Type	Description
connector	Yes	None	Strin g	Connector to be used. Set this parameter to gaussdb .
url	Yes	None	Strin g	JDBC connection address. If you use the gsjdbc4 driver, set the value in jdbc:postgresql://\${ip}:\${port}/\${dbName} format. If you use the gsjdbc200 driver, set the value in jdbc:gaussdb://\${ip}:\${port}/\${dbName} format.
table-name	Yes	None	Strin g	Name of the table to be operated. If the GaussDB(DWS) table is in a schema, the format is schema\." <i>Table name</i> . For details, see FAQ .
driver	No	org.po stgres ql.Driv er	Strin g	JDBC connection driver. The default value is org.postgresql.Driver . <ul style="list-style-type: none"> If you use the gsjdbc4 driver for connection, set this parameter to org.postgresql.Driver. If you use the gsjdbc200 driver for connection, set this parameter to com.huawei.gauss200.jdbc.Driver.
username	No	None	Strin g	Username for GaussDB(DWS) database authentication. This parameter must be configured in pair with password .
password	No	None	Strin g	Password for GaussDB(DWS) database authentication. This parameter must be configured in pair with username .

Parameter	Mandatory	Default Value	Data Type	Description
write.mode	No	None	String	<p>Data write mode. The value can be copy, insert, or upsert. The default value is upsert.</p> <p>This parameter must be configured depending on primary key.</p> <ul style="list-style-type: none"> If primary key is not configured, data can be appended in copy and insert modes. If primary key is configured, all the three modes are available. <p>Note: GaussDB(DWS) does not support the update of distribution columns. The primary keys of columns to be updated must cover all distribution columns defined in the GaussDB(DWS) table.</p>
sink.buffer-flush.max-rows	No	100	Integer	<p>Maximum number of rows to buffer for each write request.</p> <p>It can improve the performance of writing data, but may increase the latency.</p> <p>You can set this parameter to 0 to disable it.</p>
sink.buffer-flush.interval	No	1s	Duration	<p>Interval for refreshing the buffer, during which data is refreshed by asynchronous threads.</p> <p>It can improve the performance of writing data to the database, but may increase the latency.</p> <p>You can set this parameter to 0 to disable it.</p> <p>Note: If sink.buffer-flush.max-size and sink.buffer-flush.max-rows are both set to 0 and the buffer refresh interval is configured, the buffer is asynchronously refreshed.</p> <p>The format is {length value}{time unit label}, for example, 123ms, 321s. The supported time units include d, h, min, s, and ms (default unit).</p>
sink.max-retries	No	3	Integer	<p>Maximum number of write retries.</p>

Parameter	Man dato ry	Defau lt Value	Data Type	Description
write.escape-string-value	No	false	Boolean	Whether to escape values of the string type. This parameter is used only when write.mode is set to copy .

Example

In this example, data is read from the Kafka data source and written to the GaussDB(DWS) result table in insert mode. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where GaussDB(DWS) and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set GaussDB(DWS) and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the GaussDB(DWS) and Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Connect to the GaussDB(DWS) database and create a table named **dws_order**.

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR);
```

4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job. The job script uses the Kafka data source and the GaussDB(DWS) result table.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
```

```
'format' = 'json'
);

CREATE TABLE dwsSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DWSAddress:DWSPort/DWSdbName',
  'table-name' = 'dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword',
  'write.mode' = 'insert'
);

insert into dwsSink select * from kafkaSource;
```

5. Connect to the Kafka cluster and enter the following test data to Kafka:
{
"order_id": "202103241000000001",
"order_channel": "webShop",
"order_time": "2021-03-24 10:00:00",
"pay_amount": "100.00",
"real_pay": "100.00",
"pay_time": "2021-03-24 10:02:03",
"user_id": "0001",
"user_name": "Alice",
"area_id": "330106"
}
6. Run the following SQL statement in GaussDB(DWS) to view the data result:
select * from dws_order

The data result is as follows:

```
202103241000000001 webShop 2021-03-24 10:00:00 100.0 100.0 2021-03-24 10:02:03
0001 Alice 330106
```

FAQ

- Q: What should I do if the Flink job execution fails and the log contains the following error information?

```
java.io.IOException: unable to open JDBC writer
```

```
...
```

```
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
```

```
...
```

```
Caused by: java.net.SocketTimeoutException: connect timed out
```

A: The datasource connection is not bound or the binding fails.

- To reconfigure datasource connections, refer to [Enhanced Datasource Connection](#). Rectify the fault by referring to [DLI Failed to Connect to GaussDB\(DWS\) Through an Enhanced Datasource Connection](#).

- Q: How can I configure a GaussDB(DWS) table that is in a schema?

A: When GaussDB(DWS) table **test** is in schema **ads_game_sdk_base**, refer to the '**table-name**' parameter setting in the following example:

```
CREATE TABLE ads_rpt_game_sdk_realtime_ada_reg_user_pay_mm (
  ddate DATE,
  dmin TIMESTAMP(3),
  game_appkey VARCHAR,
  channel_id VARCHAR,
  pay_user_num_1m bigint,
  pay_amt_1m bigint,
  PRIMARY KEY (ddate, dmin, game_appkey, channel_id) NOT ENFORCED
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://<yourDwsAddress>:<yourDwsPort>/dws_bigdata_db',
  'table-name' = 'ads_game_sdk_base\.'\'test',
  'username' = '<yourUsername>',
```

```
'password' = '<yourPassword>',  
'write.mode' = 'upsert'  
);
```

- Q: What can I do if a job is running properly but there is no data in GaussDB(DWS)?
A: Check the following items:
 - Check whether the JobManager and TaskManager logs contain error information. To view logs, perform the following steps:
 - i. Log in to the DLI console. In the navigation pane, choose **Job Management** > **Flink Jobs**.
 - ii. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - iii. Go to the folder of the date, find the folder whose name contains **taskmanager** or **jobmanager**, download the **taskmanager.out** or **jobmanager.out** file, and view result logs.
 - Check whether the datasource connection is correctly bound and whether a security group rule allows access of the queue.
 - Check whether the GaussDB(DWS) table to which data is to be written exists in multiple schemas. If it does, specify the schemas in the Flink job.

1.3.2.4 Elasticsearch Result Table

Function

DLI outputs Flink job output data to Elasticsearch of Cloud Search Service (CSS). Elasticsearch is a popular enterprise-class Lucene-powered search server and provides the distributed multi-user capabilities. It delivers multiple functions, including full-text retrieval, structured search, analytics, aggregation, and highlighting. With Elasticsearch, you can achieve stable, reliable, real-time search. Elasticsearch applies to diversified scenarios, such as log analysis and site search.

CSS is a fully managed, distributed search service. It is fully compatible with open-source Elasticsearch and provides DLI with structured and unstructured data search, statistics, and report capabilities.

Prerequisites

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- You have created a cluster on CSS.
If you need to access Elasticsearch using the cluster username and password, enable the security mode and disable HTTPS for the created CSS cluster.
- An enhanced datasource connection has been created for DLI to connect to CSS, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

- For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Precautions

- Currently, only CSS 7.X and later versions are supported. Version 7.6.2 is recommended.
- If the username and password are not used, the security mode must be enabled and HTTPS must be disabled for the created CSS cluster.
- ICMP must be enabled for the security group inbound rules of the CSS cluster.
- For details about how to use data types, see section [Format](#).
- Before submitting a Flink job, you are advised to select **Save Job Log** and set the OBS bucket for saving job logs. This helps you view logs and locate faults when the job fails to be submitted or runs abnormally.
- The Elasticsearch sink can work in either upsert mode or append mode, depending on whether a primary key is defined.
 - If a primary key is defined, the Elasticsearch sink works in upsert mode, which can consume queries containing UPDATE and DELETE messages.
 - If a primary key is not defined, the Elasticsearch sink works in append mode which can only consume queries containing INSERT messages.

In the Elasticsearch result table, the primary key is used to calculate the Elasticsearch document ID. The document ID is a string of up to 512 bytes. It cannot have spaces. The Elasticsearch result table generates a document ID string for every row by concatenating all primary key fields in the order defined in the DDL using a key delimiter specified by **document-id.key-delimiter**. Certain types are not allowed as a primary key field as they do not have a good string representation, for example, BYTES, ROW, ARRAY, and MAP. If no primary key is specified, Elasticsearch will generate a document ID automatically.

- The Elasticsearch result table supports both static index and dynamic index.
 - If you want to have a static index, the index option value should be a plain string, such as **myusers**, all the records will be consistently written into the **myusers** index.
 - If you want to have a dynamic index, you can use **{field_name}** to reference a field value in the record to dynamically generate a target index. You can also use **{field_name|date_format_string}** to convert a field value of the TIMESTAMP, DATE, or TIME type into the format specified by **date_format_string**. **date_format_string** is compatible with Java's **DateTimeFormatter**. For example, if the option value is **myusers-{log_ts|yyyy-MM-dd}**, then a record with **log_ts** field value **2020-03-27 12:25:55** will be written into the **myusers-2020-03-27** index.

Syntax

```
create table esSink (  
  attr_name attr_type  
  (,' attr_name attr_type)*  
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
)  
with (  
  'connector' = 'elasticsearch-7',  
  'hosts' = '',
```



```
'index' = "  
);
```

Parameters

Table 1-17 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to elasticsearch-7 , indicating to connect to a cluster of Elasticsearch 7.x or later.
hosts	Yes	None	String	Host name of the cluster where Elasticsearch locates. Use semicolons (;) to separate multiple host names. Ensure that the host name starts with http , for example, http://x.x.x.x:9200 .
index	Yes	None	String	Elasticsearch index for every record. The index can be a static index (for example, 'myIndex') or a dynamic index (for example, 'index-{log_ts yyyy-MM-dd}').
username	No	None	String	Username of the cluster where Elasticsearch locates. This parameter must be configured in pair with password . If the username and password are used, the security mode must be enabled and HTTPS must be disabled for the created CSS cluster.
password	No	None	String	Password of the cluster where Elasticsearch locates. This parameter must be configured in pair with username .
document-id.key-delimiter	No	_	String	Delimiter of composite primary keys. The default value is _.

Parameter	Mandatory	Default Value	Data Type	Description
failure-handler	No	fail	String	<p>Failure handling strategy in case a request to Elasticsearch fails. Valid strategies are:</p> <ul style="list-style-type: none"> • fail: throws an exception if a request fails and thus causes a job failure. • ignore: ignores failures and drops the request. • retry-rejected: re-adds requests that have failed due to queue capacity saturation. • Custom class name: for failure handling with an ActionRequestFailureHandler subclass.
sink.flush-on-checkpoint	No	true	Boolean	<p>Whether to flush on checkpoint. If this parameter is set to false, the connector will not wait for all pending action requests to be acknowledged by Elasticsearch on checkpoints. Therefore, the connector does not provide any strong guarantees for at-least-once delivery of action requests.</p>
sink.bulk-flush.max-actions	No	1000	Integer	<p>Maximum number of buffered actions per bulk request. You can set this parameter to 0 to disable it.</p>
sink.bulk-flush.max-size	No	2mb	MemorySize	<p>Maximum size in memory of buffered actions per bulk request. It must be in MB granularity. You can set this parameter to 0 to disable it.</p>
sink.bulk-flush.interval	No	1s	Duration	<p>Interval for flushing buffered actions. You can set this parameter to 0 to disable it.</p> <p>Note:</p> <p>Both sink.bulk-flush.max-size and sink.bulk-flush.max-actions can be set to 0 with the flush interval set allowing for complete asynchronous processing of buffered actions.</p>

Parameter	Mandatory	Default Value	Data Type	Description
sink.bulk-flush.backoff.strategy	No	DISABLED	String	Specifies how to perform retries if any flush actions failed due to a temporary request error. Valid strategies are: <ul style="list-style-type: none"> • DISABLED: no retry performed, that is, fail after the first request error. • CONSTANT: wait for backoff delay between retries. • EXPONENTIAL: initially wait for backoff delay and increase exponentially between retries.
sink.bulk-flush.backoff.max-retries	No	8	Integer	Maximum number of backoff retries.
sink.bulk-flush.backoff.delay	No	50ms	Duration	Delay between each backoff attempt. For CONSTANT backoff, this is simply the delay between each retry. For EXPONENTIAL backoff, this is the initial base delay.
connection.max-retry-timeout	No	None	Duration	Maximum timeout between retries.
connection.path-prefix	No	None	String	Prefix string to be added to every REST communication, for example, '/v1'.
format	No	json	String	The Elasticsearch connector supports to specify a format. The format must produce a valid JSON document. By default, the built-in JSON format is used. Refer to Format for more details and format parameters.

Example

In this example, data is read from the Kafka data source and written to the Elasticsearch result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where Elasticsearch and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).

2. Set Elasticsearch and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Elasticsearch and Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Log in to Kibana of the Elasticsearch cluster, select Dev Tools, enter and execute the following statement to create an index whose value is **orders**:

```
PUT /orders
{
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "properties": {
      "order_id": {
        "type": "text"
      },
      "order_channel": {
        "type": "text"
      },
      "order_time": {
        "type": "text"
      },
      "pay_amount": {
        "type": "double"
      },
      "real_pay": {
        "type": "double"
      },
      "pay_time": {
        "type": "text"
      },
      "user_id": {
        "type": "text"
      },
      "user_name": {
        "type": "text"
      },
      "area_id": {
        "type": "text"
      }
    }
  }
}
```

4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort,
```

```
'properties.group.id' = 'GroupId',
'scan.startup.mode' = 'latest-offset',
'format' = "json"
);

CREATE TABLE elasticsearchSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'elasticsearch-7',
  'hosts' = 'ElasticsearchAddress:ElasticsearchPort',
  'index' = 'orders'
);

insert into elasticsearchSink select * from kafkaSource;
```

5. Connect to the Kafka cluster and insert the following test data into Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

6. Enter the following statement in Kibana of the Elasticsearch cluster and view the result:

```
GET orders/_search
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "orders",
        "_type" : "_doc",
        "_id" : "ae7wpH4B1dV9conjpXeB",
        "_score" : 1.0,
        "_source" : {
          "order_id" : "202103241000000001",
          "order_channel" : "webShop",
          "order_time" : "2021-03-24 10:00:00",
          "pay_amount" : 100.0,
          "real_pay" : 100.0,
          "pay_time" : "2021-03-24 10:02:03",
          "user_id" : "0001",
          "user_name" : "Alice",
          "area_id" : "330106"
        }
      },
      {
        "_index" : "orders",
```

```
"_type" : "_doc",
"_id" : "au7xpH4B1dV9conjn3er",
"_score" : 1.0,
"_source" : {
  "order_id" : "202103241606060001",
  "order_channel" : "appShop",
  "order_time" : "2021-03-24 16:06:06",
  "pay_amount" : 200.0,
  "real_pay" : 180.0,
  "pay_time" : "2021-03-24 16:10:06",
  "user_id" : "0001",
  "user_name" : "Alice",
  "area_id" : "330106"
}
}
]
```

1.3.2.5 HBase Result Table

Function

DLI outputs the job data to HBase. HBase is a column-oriented distributed cloud storage system that features enhanced reliability, excellent performance, and elastic scalability. It applies to the storage of massive amounts of data and distributed computing. You can use HBase to build a storage system capable of storing TB- or even PB-level data. With HBase, you can filter and analyze data with ease and get responses in milliseconds, rapidly mining data value. Structured and semi-structured key-value data can be stored, including messages, reports, recommendation data, risk control data, logs, and orders. With DLI, you can write massive volumes of data to HBase at a high speed and with low latency.

Prerequisites

- An enhanced datasource connection has been created for DLI to connect to HBase, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.
- If MRS HBase is used, IP addresses of all hosts in the MRS cluster have been added to host information of the enhanced datasource connection.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- The column families in created HBase result table must be declared as the ROW type, the field names map the column family names, and the nested field names map the column qualifier names. There is no need to declare all the families and qualifiers in the schema. Users can declare what is used in the query. Except the ROW type fields, the single atomic type field (for

example, STRING or BIGINT) will be recognized as the HBase rowkey. The rowkey field can be an arbitrary name, but should be quoted using backticks if it is a reserved keyword.

Syntax

```
create table hbaseSink (
  attr_name attr_type
  (' attr_name attr_type)*
  ',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
) with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```

Parameters

Table 1-18 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to hbase-2.2 .
table-name	Yes	None	String	Name of the HBase table to connect.
zookeeper.quorum	Yes	None	String	HBase ZooKeeper instance information, in the format of ZookeeperAddress:ZookeeperPort. The following uses an MRS HBase cluster as an example to describe how to obtain the IP address and port number of ZooKeeper used by this parameter: <ul style="list-style-type: none"> On MRS Manager, choose Cluster and click the name of the desired cluster. Choose Services > ZooKeeper > Instance, and obtain the IP address of the ZooKeeper instance. On MRS Manager, choose Cluster and click the name of the desired cluster. Choose Services > ZooKeeper > Configurations > All Configurations, search for the clientPort parameter, and obtain its value, that is, the ZooKeeper port number.
zookeeper.znode.parent	No	/hbase	String	Root directory in ZooKeeper. The default value is /hbase .

Parameter	Mandatory	Default Value	Data Type	Description
null-string-literal	No	null	String	Representation for null values for string fields. The HBase sink encodes/decodes empty bytes as null values for all types except the string type.
sink.buffer-flush.max-size	No	2mb	MemorySize	Maximum size in memory of buffered rows for each write request. This can improve performance for writing data to the HBase database, but may increase the latency. You can set this parameter to 0 to disable it.
sink.buffer-flush.max-rows	No	1000	Integer	Maximum number of rows to buffer for each write request. This can improve performance for writing data to the HBase database, but may increase the latency. You can set this parameter to 0 to disable it.
sink.buffer-flush.interval	No	1s	Duration	Interval for refreshing the buffer, during which data is refreshed by asynchronous threads. This can improve performance for writing data to the HBase database, but may increase the latency. You can set this parameter to 0 to disable it. Note: If sink.buffer-flush.max-size and sink.buffer-flush.max-rows are both set to 0 and the buffer refresh interval is configured, the buffer is asynchronously refreshed. The format is <i>{length value}{time unit label}</i> , for example, 123ms , 321s . The supported time units include d , h , min , s , and ms (default unit).

Parameter	Mandatory	Default Value	Data Type	Description
sink.parallelism	No	None	Integer	Defines the parallelism of the HBase sink operator. By default, the parallelism is determined by the framework using the same parallelism of the upstream chained operator.

Data Type Mapping

HBase stores all data as byte arrays. The data needs to be serialized and deserialized during read and write operations.

When serializing and de-serializing, Flink HBase connector uses utility class **org.apache.hadoop.hbase.util.Bytes** provided by HBase (Hadoop) to convert Flink data types to and from byte arrays.

Flink HBase connector encodes null values to empty bytes, and decode empty bytes to null values for all data types except the string type. For the string type, the null literal is determined by the **null-string-literal** option.

Table 1-19 Data type mapping

Flink SQL Type	HBase Conversion
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY / VARBINARY	Returns byte[] as is.
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)

Flink SQL Type	HBase Conversion
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	Stores the number of days since epoch as an int value.
TIME	Stores the number of milliseconds of the day as an int value.
TIMESTAMP	Stores the milliseconds since epoch as a long value.
ARRAY	Not supported
MAP / MULTISSET	Not supported
ROW	Not supported

Example

In this example, data is read from the Kafka data source and written to the HBase result table. The procedure is as follows (the HBase versions used in this example are 1.3.1 and 2.2.3):

1. Create an enhanced datasource connection in the VPC and subnet where HBase and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set HBase and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the HBase and Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Use the HBase shell to create HBase table **order** that has only one column family **detail**.

```
create 'order', {NAME => 'detail'}
```
4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job. The job script uses Kafka as the data source and HBase as the result table (the Rowkey is **order_id** and the column family name is **detail**).

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
```

```
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = 'KafkaTopic',
'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
'properties.group.id' = 'GroupId',
'scan.startup.mode' = 'latest-offset',
'format' = 'json'
);

create table hbaseSink(
order_id string,
detail Row(
order_channel string,
order_time string,
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string)
) with (
'connector' = 'hbase-2.2',
'table-name' = 'order',
'zookeeper.quorum' = 'ZookeeperAddress:ZookeeperPort',
'sink.buffer-flush.max-rows' = '1'
);

insert into hbaseSink select order_id,
Row(order_channel,order_time,pay_amount,real_pay,pay_time,user_id,user_name,area_id) from orders;
```

5. Connect to the Kafka cluster and enter the following data to Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

6. Run the following statement on the HBase shell to view the data result:

```
scan 'order'
```

The data result is as follows:

```
202103241000000001 column=detail:area_id, timestamp=2021-12-16T21:30:37.954, value=330106
```

```
202103241000000001 column=detail:order_channel, timestamp=2021-12-16T21:30:37.954,
value=webShop
```

```
202103241000000001 column=detail:order_time, timestamp=2021-12-16T21:30:37.954,
value=2021-03-24 10:00:00
```

```
202103241000000001 column=detail:pay_amount, timestamp=2021-12-16T21:30:37.954, value=@Y
\x00\x00\x00\x00\x00\x00
```

```
202103241000000001 column=detail:pay_time, timestamp=2021-12-16T21:30:37.954,
value=2021-03-24 10:02:03
```

```
202103241000000001 column=detail:real_pay, timestamp=2021-12-16T21:30:37.954, value=@Y
\x00\x00\x00\x00\x00\x00
```

```
202103241000000001 column=detail:user_id, timestamp=2021-12-16T21:30:37.954, value=0001
```

```
202103241000000001 column=detail:user_name, timestamp=2021-12-16T21:30:37.954, value=Alice
202103241606060001 column=detail:area_id, timestamp=2021-12-16T21:30:44.842, value=330106
202103241606060001 column=detail:order_channel, timestamp=2021-12-16T21:30:44.842,
value=appShop
202103241606060001 column=detail:order_time, timestamp=2021-12-16T21:30:44.842,
value=2021-03-24 16:06:06
202103241606060001 column=detail:pay_amount, timestamp=2021-12-16T21:30:44.842, value=@i
\x00\x00\x00\x00\x00\x00
202103241606060001 column=detail:pay_time, timestamp=2021-12-16T21:30:44.842,
value=2021-03-24 16:10:06
202103241606060001 column=detail:real_pay, timestamp=2021-12-16T21:30:44.842, value=@f
\x80\x00\x00\x00\x00\x00
202103241606060001 column=detail:user_id, timestamp=2021-12-16T21:30:44.842, value=0001
202103241606060001 column=detail:user_name, timestamp=2021-12-16T21:30:44.842, value=Alice
202103251202020001 column=detail:area_id, timestamp=2021-12-16T21:30:52.181, value=330110
202103251202020001 column=detail:order_channel, timestamp=2021-12-16T21:30:52.181,
value=miniAppShop
202103251202020001 column=detail:order_time, timestamp=2021-12-16T21:30:52.181,
value=2021-03-25 12:02:02
202103251202020001 column=detail:pay_amount, timestamp=2021-12-16T21:30:52.181, value=@N
\x00\x00\x00\x00\x00\x00
202103251202020001 column=detail:pay_time, timestamp=2021-12-16T21:30:52.181,
value=2021-03-25 12:03:00
202103251202020001 column=detail:real_pay, timestamp=2021-12-16T21:30:52.181, value=@N
\x00\x00\x00\x00\x00\x00
202103251202020001 column=detail:user_id, timestamp=2021-12-16T21:30:52.181, value=0002
202103251202020001 column=detail:user_name, timestamp=2021-12-16T21:30:52.181, value=Bob
```

FAQ

Q: What should I do if the Flink job execution fails and the log contains the following error information?

```
org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from
server in 90069ms for connection id 0x0
```

A: The datasource connection is not bound or the binding fails. Configure the datasource connection by referring to [Enhanced Datasource Connection](#) or configure the security group of the Kafka cluster to allow access from the DLI queue.

1.3.2.6 JDBC Result Table

Function

DLI outputs the Flink job output data to RDS through the JDBC result table.

Prerequisites

An enhanced datasource connection with the instances has been established, so that you can configure security group rules as required.

- For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
- For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- The connector operates in upsert mode if the primary key was defined; otherwise, the connector operates in append mode.
 - In upsert mode, Flink will insert a new row or update the existing row according to the primary key. Flink can ensure the idempotence in this way. To guarantee the output result is as expected, it is recommended to define a primary key for the table.
 - In append mode, Flink will interpret all records as INSERT messages. The INSERT operation may fail if a primary key or unique constraint violation happens in the underlying database.

Syntax

```
create table jdbcSink (  
  attr_name attr_type  
  (,' attr_name attr_type)*  
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
)  
with (  
  'connector' = 'jdbc',  
  'url' = "",  
  'table-name' = "",  
  'driver' = "",  
  'username' = "",  
  'password' = ""  
);
```

Parameters

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to jdbc .
url	Yes	None	String	Database URL.
table-name	Yes	None	String	Name of the table where the data will be read from the database.

Parameter	Mandatory	Default Value	Data Type	Description
driver	No	None	String	Driver required for connecting to the database. If you do not set this parameter, it will be automatically derived from the URL.
username	No	None	String	Database authentication username. This parameter must be configured in pair with password .
password	No	None	String	Database authentication password. This parameter must be configured in pair with username .
sink.buffer-flush.max-rows	No	100	Integer	Maximum number of rows to buffer for each write request. It can improve the performance of writing data, but may increase the latency. You can set this parameter to 0 to disable it.
sink.buffer-flush.interval	No	1s	Duration	Interval for refreshing the buffer, during which data is refreshed by asynchronous threads. It can improve the performance of writing data, but may increase the latency. You can set this parameter to 0 to disable it. Note: If sink.buffer-flush.max-rows is set to 0 and the buffer refresh interval is configured, the buffer is asynchronously refreshed. The format is <i>{length value}{time unit label}</i> , for example, 123ms , 321s . The supported time units include d , h , min , s , and ms (default unit).
sink.max-retries	No	3	Integer	Maximum number of retries if writing records to the database failed.

Data Type Mapping

Table 1-20 Data type mapping

MySQL Type	PostgreSQL Type	Flink SQL Type
TINYINT	-	TINYINT
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

MySQL Type	PostgreSQL Type	Flink SQL Type
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

Example

In this example, Kafka is used to send data, and Kafka data is written to the MySQL database through the JDBC result table.

1. Create an enhanced datasource connection in the VPC and subnet where MySQL and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set MySQL and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the MySQL and Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Log in to the MySQL database and create table **orders** in database **flink**.

```
CREATE TABLE `flink`.`orders` (
  `order_id` VARCHAR(32) NOT NULL,
  `order_channel` VARCHAR(32) NULL,
  `order_time` VARCHAR(32) NULL,
  `pay_amount` DOUBLE UNSIGNED NOT NULL,
  `real_pay` DOUBLE UNSIGNED NULL,
  `pay_time` VARCHAR(32) NULL,
  `user_id` VARCHAR(32) NULL,
  `user_name` VARCHAR(32) NULL,
  `area_id` VARCHAR(32) NULL,
  PRIMARY KEY (`order_id`)
) ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_general_ci;
```

4. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
```



```
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = 'KafkaTopic',
'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
'properties.group.id' = 'GroupId',
'scan.startup.mode' = 'latest-offset',
'format' = 'json'
);

CREATE TABLE jdbcSink (
order_id string,
order_channel string,
order_time string,
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'jdbc',
'url? = 'jdbc:mysql://MySQLAddress:MySQLPort/flink',-- flink is the MySQL database where the
orders table locates.
'table-name' = 'orders',
'username' = 'MySQLUsername',
'password' = 'MySQLPassword',
'sink.buffer-flush.max-rows' = '1'
);
```

```
insert into jdbcSink select * from kafkaSource;
```

5. Connect to the Kafka cluster and send the following test data to the Kafka topics:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

6. Run the SQL statement in the MySQL database to view data in the table:
select * from orders;

The following is an example of the result (note that the following data is replicated from the MySQL database but not the data style in the MySQL database):

```
202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,
330106
202103241606060001,appShop,2021-03-24 16:06:06,200.0,180.0,2021-03-24 16:10:06,0001,Alice,
330106
```

FAQ

None

1.3.2.7 Kafka Result Table

Function

DLI outputs the Flink job output data to Kafka through the Kafka result table.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

- You have created a Kafka cluster.
- An enhanced datasource connection has been created for DLI to connect to Kafka clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- For details about how to use data types, see section [Format](#).

Syntax

```
create table kafkaSink(
  attr_name attr_type
  (' attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'kafka',
  'topic' = "",
  'properties.bootstrap.servers' = "",
  'format' = ""
);
```

Parameters

Table 1-21 Parameter description

Parameter	Man dato ry	Defaul t Value	Data Type	Description
connector	Yes	None	string	Connector to be used. Set this parameter to kafka .
topic	Yes	None	string	Topic name of the Kafka result table.

Parameter	Mandatory	Default Value	Data Type	Description
properties.bootstrap.servers	Yes	None	string	Kafka broker address. The value is in the format of host:port,host:port,host:port . Multiple host:port pairs are separated with commas (,).
format	Yes	None	string	Format used by the Flink Kafka connector to serialize Kafka messages. Either this parameter or the value.format parameter is required. The following formats are supported: <ul style="list-style-type: none"> • csv • json • avro Refer to Format for more details and format parameters.
topic-pattern	No	None	String	Regular expression for matching the Kafka topic name. Only one of topic and topic-pattern can be specified. Example: 'topic.*' '(topic-c topic-d)' '(topic-a topic-b topic-\\d*)' '(topic-a topic-b topic-[0-9]*)'
properties.*	No	None	String	This parameter can set and pass arbitrary Kafka configurations. Note: <ul style="list-style-type: none"> • Suffix names must match the configuration key defined in Apache Kafka. For example, you can disable automatic topic creation via 'properties.allow.auto.create.topics' = 'false'. • Some configurations are not supported, for example, 'key.deserializer' and 'value.deserializer'.

Parameter	Mandatory	Default Value	Data Type	Description
key.format	No	None	String	<p>Format used to deserialize and serialize the key part of Kafka messages.</p> <p>Note:</p> <ul style="list-style-type: none"> If a key format is defined, the key.fields parameter is required as well. Otherwise, the Kafka records will have an empty key. Possible values are: <ul style="list-style-type: none"> csv json avro debezium-json canal-json maxwell-json avro-confluent raw <p>Refer to Format for more details and format parameters.</p>
key.fields	No	[]	List<String>	<p>Defines the columns in the table as the list of keys. This parameter must be configured in pair with key.format.</p> <p>This parameter is left empty by default. Therefore, no key is defined.</p> <p>The format is like field1;field2.</p>
key.fields-prefix	No	None	String	<p>Defines a custom prefix for all fields of the key format to avoid name clashes with fields of the value format.</p>

Parameter	Mandatory	Default Value	Data Type	Description
value.format	Yes	None	String	<p>Format used to deserialize and serialize the value part of Kafka messages.</p> <p>Note:</p> <ul style="list-style-type: none"> • Either this parameter or the format parameter is required. If two parameters are configured, a conflict occurs. • Refer to Format for more details and format parameters.
value.fields-include	No	ALL	Enum Possible values: [ALL, EXCEPT_KEY]	<p>Whether to contain the key field when parsing the message body.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • ALL (default): All defined fields are included in the value of Kafka messages. • EXCEPT_KEY: All the fields except those defined by key.fields are included in the value of Kafka messages.
sink.partitione r	No	None	string	<p>Mapping from Flink's partitions into Kafka's partitions. Valid values are as follows:</p> <ul style="list-style-type: none"> • fixed (default): Each Flink partition ends up in at most one Kafka partition. • round-robin: A Flink partition is distributed to Kafka partitions in a round-robin manner. • Custom FlinkKafkaPartitioner subclass: If fixed and round-robin do not meet your requirements, you can create subclass FlinkKafkaPartitioner to customize the partition mapping, for example, org.mycompany.MyPartitioner.

Parameter	Mandatory	Default Value	Data Type	Description
sink.semantic	No	at-least-once	String	Defines the delivery semantic for the Kafka sink. Valid values are as follows: <ul style="list-style-type: none"> at-least-once exactly-once none
sink.parallelism	No	None	Integer	Defines the parallelism of the Kafka sink operator. By default, the parallelism is determined by the framework using the same parallelism of the upstream chained operator.

Example (SASL_SSL Disabled for the Kafka Cluster)

In this example, data is read from a Kafka topic and written to another using a Kafka result table.

1. Create an enhanced datasource connection in the VPC and subnet where Kafka locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);
```

```
CREATE TABLE kafkaSink (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'KafkaSinkTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'format' = 'json'  
);
```

```
insert into kafkaSink select * from kafkaSource;
```

4. Connect to the Kafka cluster and insert the following test data into the source topic in Kafka:

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24  
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24  
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24  
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24  
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

5. Connect to the Kafka cluster and read data from the sink topic of Kafka.

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24  
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24  
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24  
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24  
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

Example (SASL_SSL Enabled for the Kafka Cluster)

- **Example 1: Enable SASL_SSL authentication for the DMS cluster.**

Create a Kafka cluster for DMS, enable SASL_SSL, download the SSL certificate, and upload the downloaded certificate **client.jks** to an OBS bucket.

```
CREATE TABLE ordersSource (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'xx',  
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',  
  'properties.group.id' = 'GroupId',  
  'scan.startup.mode' = 'latest-offset',  
  'properties.connector.auth.open' = 'true',  
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks', -- Location where the user uploads the  
certificate to  
  'properties.sasl.mechanism' = 'PLAIN', -- Value format: SASL_PLAINTEXT  
  'properties.security.protocol' = 'SASL_SSL',  
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required  
username=\"xx\" password=\"xx\";', -- Account and password set when the Kafka cluster is created
```

```
"format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks',
  'properties.sasl.mechanism' = 'PLAIN',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required
  username=\"xx\" password=\"xx\";',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **Example 2: Enable Kafka SASL_SSL authentication for the MRS cluster.**

- Enable Kerberos authentication for the MRS cluster.
- Click the **Components** tab and click **Kafka**. In the displayed page, click the **Service Configuration** tab, locate the **security.protocol**, and set it to **SASL_SSL**.
- Log in to the FusionInsight Manager of the MRS cluster and download the user credential. Choose **System > Permission > User**. Locate the row that contains the target user, choose **More > Download Authentication Credential**.
Obtain the **truststore.jks** file using the authentication credential and store the credential and **truststore.jks** file in OBS.
- If "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier than 8u_242 or delete the **renew_lifetime = 0m** configuration item from the **krb5.conf** configuration file.
- Set the port to the **sasl_ssl.port** configured in the Kafka service configuration.
- In the following statements, set **security.protocol** to **SASL_SSL**.

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
```



```
'properties.sasl.kerberos.service.name' = 'kafka',
'properties.connector.auth.open' = 'true',
'properties.connector.kerberos.principal' = 'xx', --Username
'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
'properties.security.protocol' = 'SASL_SSL',
'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
'properties.ssl.truststore.password' = 'xx', -- Password set for generating truststore.jks
'properties.sasl.mechanism' = 'GSSAPI',
"format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **Example 3: Enable Kerberos SASL_PAINTTEXT authentication for the MRS cluster**
 - Enable Kerberos authentication for the MRS cluster.
 - Click the **Components** tab and click **Kafka**. In the displayed page, click the **Service Configuration** tab, locate the **security.protocol**, and set it to **SASL_PLAINTEXT**.
 - Log in to the FusionInsight Manager of the MRS cluster and download the user credential. Choose **System > Permission > User**. Locate the row that contains the target user, choose **More > Download Authentication Credential**. Upload the credential to OBS.
 - If error message "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier than 8u_242 or delete the **renew_lifetime = 0m** configuration item from the **krb5.conf** configuration file.
 - Set the port to the **sasl.port** configured in the Kafka service configuration.
 - In the following statements, set **security.protocol** to **SASL_PLAINTEXT**.

```
CREATE TABLE ordersSources (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
```

```
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = 'xx',
'properties.bootstrap.servers' = 'xx:21007,xx:21007',
'properties.group.id' = 'Groupld',
'scan.startup.mode' = 'latest-offset',
'properties.sasl.kerberos.service.name' = 'kafka',
'properties.connector.auth.open' = 'true',
'properties.connector.kerberos.principal' = 'xx',
'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
'properties.security.protocol' = 'SASL_PLAINTEXT',
'properties.sasl.mechanism' = 'GSSAPI',
"format" = "json"
);

CREATE TABLE ordersSink (
order_id string,
order_channel string,
order_time timestamp(3),
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = 'xx',
'properties.bootstrap.servers' = 'xx:21007,xx:21007',
'properties.sasl.kerberos.service.name' = 'kafka',
'properties.connector.auth.open' = 'true',
'properties.connector.kerberos.principal' = 'xx',
'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
'properties.security.protocol' = 'SASL_PLAINTEXT',
'properties.sasl.mechanism' = 'GSSAPI',
"format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **Example 4: Use SSL for the MRS cluster**

- Do not enable Kerberos authentication for the MRS cluster.
- Log in to the FusionInsight Manager of the MRS cluster and download the user credential. Choose **System > Permission > User**. Locate the row that contains the target user, choose **More > Download Authentication Credential**.
Obtain the **truststore.jks** file using the authentication credential and store the credential and **truststore.jks** file in OBS.
- Set the port to the **ssl.port** configured in the Kafka service configuration.
- In the following statements, set **security.protocol** to **SSL**.
- Set **ssl.mode.enable** to **true**.

```
CREATE TABLE ordersSource (
order_id string,
order_channel string,
order_time timestamp(3),
pay_amount double,
real_pay double,
pay_time string,
```

```

user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = 'xx',
'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
'properties.group.id' = 'Groupld',
'scan.startup.mode' = 'latest-offset',
'properties.connector.auth.open' = 'true',
'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
'properties.ssl.truststore.password' = 'xx', -- Password set for generating truststore.jks
'properties.security.protocol' = 'SSL',
"format" = "json"
);

CREATE TABLE ordersSink (
order_id string,
order_channel string,
order_time timestamp(3),
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'print'
);

insert into ordersSink select * from ordersSource;

```

1.3.2.8 Print Result Table

Function

The Print connector is used to print output data to the error file or TaskManager file, making it easier for you to view the result in code debugging.

Prerequisites

None

Precautions

- The Print result table supports the following output formats:

Print	Condition 1	Condition 2
Identifier:Task ID> Output data	A print identifier prefix must be provided. That is, you must specify print-identifier in the WITH parameter when creating the Print result table.	parallelism > 1

Print	Condition 1	Condition 2
Identifier> Output data	A print identifier prefix must be provided. That is, you must specify print-identifier in the WITH parameter when creating the Print result table.	parallelism == 1
Task ID> Output data	A print identifier prefix is not needed. That is, you do not specify print-identifier in the WITH parameter when creating the Print result table.	parallelism > 1
Output data	A print identifier prefix is not needed. That is, you do not specify print-identifier in the WITH parameter when creating the Print result table.	parallelism == 1

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.

Syntax

```
create table printSink (
  attr_name attr_type
  (' attr_name attr_type) *
  (' PRIMARY KEY (attr_name,...) NOT ENFORCED)
) with (
  'connector' = 'print',
  'print-identifier' = "",
  'standard-error' = ""
);
```

Parameters

Table 1-22 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to print .
print-identifier	No	None	String	Message that identifies print and is prefixed to the output of the value.

Parameter	Mandatory	Default Value	Data Type	Description
standard-error	No	false	Boolean	<p>The value can be only true or false. The default value is false.</p> <ul style="list-style-type: none"> If the value is true, data is output to the error file of the TaskManager. If the value is false, data is output to the out file of the TaskManager.

Example

Create a Flink OpenSource SQL job. Run the following script to generate random data through the DataGen table and output the data to the Print result table.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

```
create table dataGenSource(
  user_id string,
  amount int
) with (
  'connector' = 'datagen',
  'rows-per-second' = '1', --Generate a piece of data per second.
  'fields.user_id.kind' = 'random', --Specify a random generator for the user_id field.
  'fields.user_id.length' = '3' --Limit the length of user_id to 3.
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print'
);

insert into printSink select * from dataGenSource;
```

After the job is submitted, the job status changes to **Running**. You can perform the following operations of either method to view the output result:

- Method 1:
 - Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - Locate the row that contains the target Flink job, and choose **More > FlinkUI** in the **Operation** column.
 - On the Flink UI, choose **Task Managers**, click the task name, and select **Stdout** to view job logs.
- Method 2: If you select **Save Job Log** on the **Running Parameters** tab before submitting the job, perform the following operations:
 - Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.

- c. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

1.3.2.9 Redis Result Table

Function

DLI outputs the Flink job output data to Redis. Redis is a key-value storage system that supports multiple types of data structures. It can be used in scenarios such as caching, event publish/subscribe, and high-speed queuing. Redis supports direct read/write of strings, hashes, lists, queues, and sets. Redis works with in-memory datasets and provides persistence. For more information about Redis, visit <https://redis.io/>.

Prerequisites

- An enhanced datasource connection with Redis has been established, so that you can configure security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- If the Redis key field is not defined in the statement for creating the Redis result table, the generated UUID is used as the key.
- To specify a key in Redis, you need to define a primary key in the Redis result table of Flink. The value of the primary key is the Redis key.
- If the primary key defined for the Redis result table, it cannot be a composite primary key and only can be one field.
- Constraints on **schema-syntax**:
 - If **schema-syntax** is **map** or **array**, there can be only one non-primary key and it must be of the same **map** or **array** type.
 - If **schema-syntax** is **fields-scores**, the number of non-primary keys must be an even number, and the second key of every two keys except the primary key must be of the **double** type. The **double** value is the score of the previous key. The following is an example:

```
CREATE TABLE redisSink (  
  order_id string,  
  order_channel string,  
  order_time double,  
  pay_amount STRING,  
  real_pay double,  
  pay_time string,  
  user_id double,  
  user_name string,  
  area_id double,  
  primary key (order_id) not enforced  
) WITH (  
  'connector' = 'redis',
```

```
'host' = 'RedisIP',
'password' = 'RedisPassword',
'data-type' = 'sorted-set',
'deploy-mode' = 'master-replica',
'schema-syntax' = 'fields-scores'
);
```

- Restrictions on **data-type**:
 - If **data-type** is **string**, only one non-primary key field is allowed.
 - If **data-type** is **sorted-set** and **schema-syntax** is **fields** or **array**, **default-score** is used as the score.
 - If **data-type** is **sorted-set** and **schema-syntax** is **map**, there can be only one non-primary key in addition to the primary key and the non-primary key must be of the **map** type. The **map** values of the non-primary key must be of the **double** type, indicating the score. The keys in the map are the values in the Redis set.
 - If **data-type** is **sorted-set** and **schema-syntax** is **array-scores**, only two non-primary keys are allowed and must be of the **array** type.

The first key indicates values in the Redis set. The second key is of the **array<double>** type, indicating index scores. The following is an example:

```
CREATE TABLE redisSink (
  order_id string,
  arrayField Array<String>,
  arrayScore array<double>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  "default-score" = '3',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'array-scores'
);
```

Syntax

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'redis',
  'host' = "
);
```

Parameters

Table 1-23 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector to be used. Set this parameter to redis .
host	Yes	None	String	Redis connector address.

Parameter	Mandatory	Default Value	Data Type	Description
port	No	6379	Integer	Redis connector port.
password	No	None	String	Redis authentication password.
namespace	No	None	String	Redis key namespace. For example, if the value is set to "person" and the key is "jack", the value in the Redis is person:jack.
delimiter	No	:	String	Delimiter between the Redis key and namespace.
data-type	No	hash	String	Redis data type. Available values are as follows: <ul style="list-style-type: none"> • hash • list • set • sorted-set • string For details about the constraints, see Constraints on data-type .
schema-syntax	No	fields	String	Redis schema semantics. Available values are as follows: <ul style="list-style-type: none"> • fields: applicable to all data types. This value indicates that multiple fields can be set and the value of each field is read when data is written. • fields-scores: applicable to sorted-set data, indicating that each field is read as an independent score. • array: applicable to list, set, and sorted-set data. • array-scores: applicable to sorted-set data. • map: applicable to hash and sorted-set data. For details about the constraints, see Constraints on schema-syntax .

Parameter	Mandatory	Default Value	Data Type	Description
deploy-mode	No	standalone	String	Deployment mode of the Redis cluster. The value can be standalone , master-replica , or cluster . The default value is standalone . For details about the setting, see the instance type description of the Redis cluster.
retry-count	No	5	Integer	Number of attempts to connect to the Redis cluster.
connection-timeout-millis	No	10000	Integer	Maximum timeout for connecting to the Redis cluster.
commands-timeout-millis	No	2000	Integer	Maximum time for waiting for a completion response.
rebalancing-timeout-millis	No	15000	Integer	Sleep time when the Redis cluster fails.
default-score	No	0	Double	Default score when data-type is sorted-set .
ignore-retraction	No	false	Boolean	Whether to ignore Retract messages.
skip-null-values	No	true	Boolean	Whether null values will be skipped. If this parameter is false , null will be assigned for null values.
key-ttl-mode	No	no-ttl	String	Whether the Redis sink TTL function will be enabled. The value can be no-ttl , expire-msec , expire-at-date or expire-at-timestamp . <ul style="list-style-type: none"> • no-ttl: No expiration time is set. • expire-msec: validity period of the key. The parameter is a long string, in milliseconds. • expire-at-date: Date and time when the key expires. The value is in UTC time format. • expire-at-timestamp: Timestamp when the key expires.

Parameter	Mandatory	Default Value	Data Type	Description
key-ttl	No	None	String	<p>Supplementary parameter of key-ttl-mode. Available values are as follows:</p> <ul style="list-style-type: none"> • If key-ttl-mode is no-ttl, this parameter does not need to be configured. • If key-ttl-mode is expire-msec, set this parameter to a string that can be parsed into the Long type. For example, 5000 indicates that the key will expire in 5000 ms. • If key-ttl-mode is expire-at-date, set this parameter to a date. For example, 2011-12-03T10:15:30 indicates that the expiration time is 2011-12-03 18:15:30 (UTC+8). • If key-ttl-mode is expire-at-timestamp, set this parameter to a timestamp, in milliseconds. For example, 1679385600000 indicates that the expiration time is 2023-03-21 16:00:00.

Example

In this example, data is read from the Kafka data source and written to the Redis result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where Redis locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set Redis security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Redis address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Change the values of the parameters in bold as needed in the following script.**

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
```

```
order_time string,
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string
) WITH (
'connector' = 'kafka',
'topic' = '<yourTopic>',
'properties.bootstrap.servers' = '<yourKafka>:<port>',
'properties.group.id' = '<yourGroupID>',
'scan.startup.mode' = 'latest-offset',
'format' = 'json'
);
--In the following redisSink table, data-type is set to default value hash, schema-syntax is fields, and
order_id is defined as the primary key. Therefore, the value of this field is used as the Redis key.
CREATE TABLE redisSink (
order_id string,
order_channel string,
order_time string,
pay_amount double,
real_pay double,
pay_time string,
user_id string,
user_name string,
area_id string,
primary key (order_id) not enforced
) WITH (
'connector' = 'redis',
'host' = '<yourRedis>',
'password' = '<yourPassword>',
'deploy-mode' = 'master-replica',
'schema-syntax' = 'fields'
);
insert into redisSink select * from orders;
```

4. Connect to the Kafka cluster and insert the following test data into Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

5. Run the following commands in Redis and view the result:

- Obtain the result whose key is **202103241606060001**.

Run following command:

```
HGETALL 202103241606060001
```

Command output:

```
1) "user_id"
2) "0001"
3) "user_name"
4) "Alice"
5) "pay_amount"
6) "200.0"
7) "real_pay"
8) "180.0"
9) "order_time"
10) "2021-03-24 16:06:06"
11) "area_id"
12) "330106"
13) "order_channel"
14) "appShop"
15) "pay_time"
16) "2021-03-24 16:10:06"
```

- Obtain the result whose key is **202103241000000001**.

Run following command:

```
HGETALL 202103241000000001
```

Command output:

```
1) "user_id"  
2) "0001"  
3) "user_name"  
4) "Alice"  
5) "pay_amount"  
6) "100.0"  
7) "real_pay"  
8) "100.0"  
9) "order_time"  
10) "2021-03-24 10:00:00"  
11) "area_id"  
12) "330106"  
13) "order_channel"  
14) "webShop"  
15) "pay_time"  
16) "2021-03-24 10:02:03"
```

FAQ

- Q: When data-type is **set**, why is the final result data less than the input data?
A: This is because the input data contains duplicate data. Deduplication is performed in the Redis set, and the number of records in the result decreases.
- Q: What should I do if Flink job logs contain the following error information?
org.apache.flink.table.api.ValidationException: SQL validation failed. From line 1, column 40 to line 1, column 105: Parameters must be of the same type
A: The array type is used. However, the types of fields in the array are different. You need to ensure that the types of fields in the array in Redis are the same.
- Q: What should I do if Flink job logs contain the following error information?
org.apache.flink.addons.redis.core.exception.RedisConnectorException: Wrong Redis schema for 'map' syntax: There should be a key (possibly) and 1 MAP non-key column.
A: When **schema-syntax** is **map**, the table creation statement in Flink can contain only one non-primary key column, and the column type must be **map**.
- Q: What should I do if Flink job logs contain the following error information?
org.apache.flink.addons.redis.core.exception.RedisConnectorException: Wrong Redis schema for 'array' syntax: There should be a key (possibly) and 1 ARRAY non-key column.
A: When **schema-syntax** is **array**, the table creation statement in Flink can contain only one non-primary key column, and the column type must be **array**.
- Q: What is the function of **schema-syntax** since **data-type** has been set?
A: **schema-syntax** is used to process special types, such as **map** and **array**.
 - If it is set to **fields**, the value of each field is processed. If it is set to **array** or **map**, each element in the field is processed. For **fields**, the field value of the **map** or **array** type is directly used as a value in Redis.
 - For **array** or **map**, each value in the array is used as a Redis value, and the field value of the map is used as the Redis value. **array-scores** is used to process the **sorted-set** data type. It indicates that two array fields are used, the first one is the value in the set, and the second one is the score. **fields-scores** is used to process the **sorted-set** data type, indicating that

the score is derived from the defined field. The field of an odd number except the primary key indicates the value in the set, and its next field indicates its score. Therefore, its next field must be of the **double** type.

- Q: If **data-type** is **hash**, what are the differences between **schema-syntax** set to **fields** and that to **map**?

A: When **fields** is used, the field name in Flink is used as the Redis field of the hash data type, and the value of that field is used as the value of the hash data type in Redis. When **map** is used, the field key in Flink is used as the Redis field of the hash data type, and the value of that field is used as the value of the hash data type in Redis. The following is an example:

- For **fields**:

- i. The execution script of the Flink job is as follows:

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'kafkaTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'GroupId',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
)  
);  
  
CREATE TABLE redisSink (  
  order_id string,  
  maptest Map<string, String>,  
  primary key (order_id) not enforced  
) WITH (  
  'connector' = 'redis',  
  'host' = 'RedisIP',  
  'password' = 'RedisPassword',  
  'deploy-mode' = 'master-replica',  
  'schema-syntax' = 'fields'  
)  
);  
  
insert into redisSink select order_id, Map[user_id, area_id] from orders;
```

- ii. Connect to the Kafka cluster and insert the following test data into the Kafka topic:

```
{"order_id":"202103241000000001", "order_channel":"webShop",  
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",  
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",  
"area_id":"330106"}
```

- iii. In the Redis, the result is as follows:

```
1) "maptest"  
2) "{0001=330106}"
```

- For **map**:

- i. The execution script of the Flink job is as follows:

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,
```

```
pay_time string,  
user_id string,  
user_name string,  
area_id string  
) WITH (  
'connector' = 'kafka',  
'topic' = 'kafkaTopic',  
'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
'properties.group.id' = 'GroupId',  
'scan.startup.mode' = 'latest-offset',  
'format' = 'json'  
);  
  
CREATE TABLE redisSink (  
order_id string,  
maptest Map<string, String>,  
primary key (order_id) not enforced  
) WITH (  
'connector' = 'redis',  
'host' = 'RedisIP',  
'password' = 'RedisPassword',  
'deploy-mode' = 'master-replica',  
'schema-syntax' = 'map'  
);  
  
insert into redisSink select order_id, Map[user_id, area_id] from orders;
```

- ii. Connect to the Kafka cluster and insert the following test data into the Kafka topic:

```
{"order_id":"202103241000000001", "order_channel":"webShop",  
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",  
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",  
"area_id":"330106"}
```

- iii. In the Redis, the result is as follows:

```
1) "0001"  
2) "330106"
```

- Q: If **data-type** is **list**, what are the differences between **schema-syntax** set to **fields** and that to **array**?

A: The setting to **fields** or **array** does not result in different results. The only difference is that in the Flink table creation statement. **fields** can be multiple fields. However, **array** requires that the field is of the **array** type and the data types in the array must be the same. Therefore, **fields** are more flexible.

- For **fields**:

- i. The execution script of the Flink job is as follows:

```
CREATE TABLE orders (  
order_id string,  
order_channel string,  
order_time string,  
pay_amount double,  
real_pay double,  
pay_time string,  
user_id string,  
user_name string,  
area_id string  
) WITH (  
'connector' = 'kafka',  
'topic' = 'kafkaTopic',  
'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
'properties.group.id' = 'GroupId',  
'scan.startup.mode' = 'latest-offset',  
'format' = 'json'  
);  
  
CREATE TABLE redisSink (  

```

```
order_id string,  
order_channel string,  
order_time string,  
pay_amount double,  
real_pay double,  
pay_time string,  
user_id string,  
user_name string,  
area_id string,  
primary key (order_id) not enforced  
) WITH (  
  'connector' = 'redis',  
  'host' = 'RedisIP',  
  'password' = 'RedisPassword',  
  'data-type' = 'list',  
  'deploy-mode' = 'master-replica',  
  'schema-syntax' = 'fields'  
);
```

```
insert into redisSink select * from orders;
```

- ii. Connect to the Kafka cluster and insert the following test data into the Kafka topic:

```
{"order_id":"202103241000000001", "order_channel":"webShop",  
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",  
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",  
"area_id":"330106"}
```

- iii. View the result.

Run the following command in Redis:

```
LRANGE 202103241000000001 0 8
```

The command output is as follows:

```
1) "webShop"  
2) "2021-03-24 10:00:00"  
3) "100.0"  
4) "100.0"  
5) "2021-03-24 10:02:03"  
6) "0001"  
7) "Alice"  
8) "330106"
```

- For **array**:

- i. The execution script of the Flink job is as follows:

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'kafkaTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'GroupId',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
);  
  
CREATE TABLE redisSink (  
  order_id string,  
  arraytest Array<String>,  
  primary key (order_id) not enforced  
) WITH (  
  'connector' = 'redis',
```

```
'host' = 'RedisIP',  
'password' = 'RedisPassword',  
'data-type' = 'list',  
'deploy-mode' = 'master-replica',  
'schema-syntax' = 'array'  
);
```

```
insert into redisSink select order_id,  
array[order_channel,order_time,pay_time,user_id,user_name,area_id] from orders;
```

- ii. Connect to the Kafka cluster and insert the following test data into the Kafka topic:

```
{"order_id":"202103241000000001", "order_channel":"webShop",  
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",  
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",  
"area_id":"330106"}
```

- iii. In Redis, view the result. (The result is different from that of **fields** because data of the **double** type is not added to the table creation statement of the sink in Flink. Therefore, two values are missing. This is not caused by the difference between **fields** and **array**.)

```
1) "webShop"  
2) "2021-03-24 10:00:00"  
3) "2021-03-24 10:02:03"  
4) "0001"  
5) "Alice"  
6) "330106"
```

1.3.2.10 Upsert Kafka Result Table

Function

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages. DLI outputs the Flink job output data to Kafka in upsert mode.

The Upsert Kafka connector allows for reading data from and writing data into Kafka topics in the upsert fashion.

As a sink, the Upsert Kafka connector can consume a changelog stream. It will write INSERT/UPDATE_AFTER data as normal Kafka messages value, and write DELETE data as Kafka messages with null values (indicate tombstone for the key). Flink will guarantee the message ordering on the primary key by partition data on the values of the primary key columns, so the UPDATE/DELETE messages on the same key will fall into the same partition.

Prerequisites

- You have created a Kafka cluster.
- An enhanced datasource connection has been created for DLI to connect to Kafka clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Precautions

- When creating a Flink OpenSource SQL job, you need to set **Flink Version** to **1.12** on the **Running Parameters** tab of the job editing page, select **Save Job Log**, and set the OBS bucket for saving job logs.
- For details about how to use data types, see section [Format](#).
- The Upsert Kafka always works in the upsert fashion and requires to define the primary key in the DDL.
- By default, an Upsert Kafka sink ingests data with at-least-once guarantees into a Kafka topic if the query is executed with checkpointing enabled. This means that Flink may write duplicate records with the same key into the Kafka topic. Therefore, the Upsert Kafka connector achieves idempotent writes.

Syntax

```
create table kafkaSource(
  attr_name attr_type
  ('; attr_name attr_type)*
  (';PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'upsert-kafka',
  'topic' = "",
  'properties.bootstrap.servers' = "",
  'key.format' = "",
  'value.format' = ""
);
```

Parameters

Table 1-24 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	(none)	String	Connector to be used. Set this parameter to upsert-kafka .
topic	Yes	(none)	String	Kafka topic name.
properties.bootstrap.servers	Yes	(none)	String	Comma separated list of Kafka brokers.
key.format	Yes	(none)	String	Format used to deserialize and serialize the key part of Kafka messages. The key fields are specified by the PRIMARY KEY syntax. The following formats are supported: <ul style="list-style-type: none"> • csv • json • avro Refer to Format for more details and format parameters.

Parameter	Mandatory	Default Value	Data Type	Description
key.fields-prefix	No	(none)	String	<p>Defines a custom prefix for all fields of the key format to avoid name clashes with fields of the value format.</p> <p>By default, the prefix is empty. If a custom prefix is defined, both the table schema and key.fields will work with prefixed names. When constructing the data type of the key format, the prefix will be removed and the non-prefixed names will be used within the key format. Note that this option requires that value.fields-include must be set to EXCEPT_KEY.</p>
value.format	Yes	(none)	String	<p>Format used to deserialize and serialize the value part of Kafka messages. The following formats are supported:</p> <ul style="list-style-type: none"> • csv • json • avro <p>Refer to Format for more details and format parameters.</p>
value.fields-include	No	'ALL'	String	<p>Controls which fields should appear in the value part. Options:</p> <ul style="list-style-type: none"> • ALL: All fields in the schema, including the primary key field, are included in the value part. • EXCEPT_KEY: All the fields of the table schema are included, except the primary key field.
sink.parallelism	No	(none)	Integer	<p>Defines the parallelism of the Upsert Kafka sink operator. By default, the parallelism is determined by the framework using the same parallelism of the upstream chained operator.</p>

Parameter	Mandatory	Default Value	Data Type	Description
properties.*	No	(none)	String	<p>This option can set and pass arbitrary Kafka configurations.</p> <p>The suffix of this parameter must match the parameter defined in Kafka Configuration documentation. Flink will remove the properties. key prefix and pass the transformed key and value to the underlying KafkaClient.</p> <p>For example, you can disable automatic topic creation via 'properties.allow.auto.create.topics' = 'false'. But there are some configurations that do not support to set, because Flink will override them, for example, 'key.deserializer' and 'value.deserializer'.</p>

Example

In this example, Kafka source topic data is read from the Kafka source table and written to the Kafka sink topic through the Upsert Kafka result table.

1. Create an enhanced datasource connection in the VPC and subnet where Kafka locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Kafka address by referring to [Testing Address Connectivity](#). If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
3. Create a Flink OpenSource SQL job. Enter the following job script and submit the job.

When you create a job, set **Flink Version** to **1.12** on the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
```

```
"format" = "json"
);
CREATE TABLE UPSERTKAFKASINK (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'key.format' = 'json',
  'value.format' = 'json'
);
insert into UPSERTKAFKASINK
select * from orders;
```

4. Connect to the Kafka cluster and send the following test data to the Kafka source topic:

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

5. Connect to the Kafka cluster and read data from the Kafka sink topic. The result is as follows:

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

FAQ

None

1.3.2.11 FileSystem Result Table

Function

The FileSystem result (sink) table is used to export data to the HDFS or OBS file system. It is applicable to scenarios such as data dumping, big data analysis, data backup, and active, deep, or cold archiving.

Considering that the input stream can be unbounded, you can put the data in each bucket into **part** files of a limited size. Data can be written into a bucket

based on time. For example, you can write data into a bucket every hour. This bucket contains the records received within one hour, and

data in the bucket directory is split into multiple **part** files. Each sink bucket that receives data contains at least one **part** file for each subtask. Other **part** files are created based on the configured rolling policy. For Row Formats, the default rolling policy is based on the **part** file size. You need to specify the maximum timeout period for opening a file and the timeout period for the inactive state after closing a file. Bulk Formats are rolled each time a checkpoint is created. You can add other rolling conditions based on size or time.

NOTE

- To use FileSink in STREAMING mode, you need to enable the checkpoint function. **Part** files are generated only when the checkpoint is successful. If the checkpoint function is not enabled, the files remain in the in-progress or pending state, and downstream systems cannot securely read the file data.
- The number recorded by the sink end operator is the number of checkpoints, not the actual volume of the sent data. For the actual volume, see the number recorded by the streaming-writer or StreamingFileWriter operator.

Syntax

```
CREATE TABLE sink_table (  
  name string,  
  num INT,  
  p_day string,  
  p_hour string  
) partitioned by (p_day, p_hour) WITH (  
  'connector' = 'filesystem',  
  'path' = 'obs://**',  
  'format' = 'parquet',  
  'auto-compaction' = 'true'  
);
```

Usage

- **Rolling Policy**

The Rolling Policy defines when a given in-progress part file will be closed and moved to the pending and later to finished state. Part files in the "finished" state are the ones that are ready for viewing and are guaranteed to contain valid data that will not be reverted in case of failure.

In STREAMING mode, the Rolling Policy in combination with the checkpointing interval (pending files become finished on the next checkpoint) control how quickly part files become available for downstream readers and also the size and number of these parts. For details, see [Parameters](#).

- **Part File Lifecycle**

To use the output of the FileSink in downstream systems, we need to understand the naming and lifecycle of the output files produced.

Part files can be in one of three states:

- **In-progress:** The part file that is currently being written to is in-progress.
- **Pending:** Closed (due to the specified rolling policy) in-progress files that are waiting to be committed.
- **Finished:** On successful checkpoints (STREAMING) or at the end of input (BATCH) pending files transition to **Finished**

Only finished files are safe to read by downstream systems as those are guaranteed to not be modified later.

By default, the file naming strategy is as follows:

- **In-progress / Pending:** part-<uid>-<partFileIndex>.inprogress.uid
- **Finished:** part-<uid>-<partFileIndex>

uid is a random ID assigned to a subtask of the sink when the subtask is instantiated. This **uid** is not fault-tolerant so it is regenerated when the subtask recovers from a failure.

- **Compaction**

FileSink supports compaction of the pending files, which allows the application to have smaller checkpoint interval without generating a lot of small files.

Once enabled, the compaction happens between the files become pending and get committed. The pending files will be first committed to temporary files whose path starts with a dot (.). Then these files will be compacted according to the strategy by the compactor specified by the users, and the new compacted pending files will be generated. Then these pending files will be emitted to the committer to be committed to the formal files. After that, the source files will be removed.

- **Partitions**

Filesystem sink supports the partitioning function. Partitions are generated based on the selected fields by using the **partitioned by** syntax. The following is an example:

```

path
├── datetime=2022-06-25
│   ├── hour=10
│   │   ├── part-0.parquet
│   │   └── part-1.parquet
│   └── datetime=2022-06-26
│       ├── hour=16
│       │   └── part-0.parquet
│       └── hour=17
│           └── part-0.parquet
    
```

Similar to files, partitions also need to be submitted to notify downstream applications that files in the partitions can be securely read. Filesystem sink provides multiple configuration submission policies.

Parameters

Table 1-25 Parameter description

Parameter	Mandatory	Default Value	Type	Description
connector	Yes	None	String	The value is fixed at filesystem .
path	Yes	None	String	OBS path
format	Yes	None	String	File format Available values are: csv and parquet

Parameter	Mandatory	Default Value	Type	Description
sink.rolling-policy.file-size	No	128 MB	MemorySize	<p>Maximum size of a part file. If the size of a part file exceeds this value, a new file will be generated.</p> <p>NOTE The Rolling Policy defines when a given in-progress part file will be closed and moved to the pending and later to finished state. Part files in the "finished" state are the ones that are ready for viewing and are guaranteed to contain valid data that will not be reverted in case of failure. In STREAMING mode, the Rolling Policy in combination with the checkpointing interval (pending files become finished on the next checkpoint) control how quickly part files become available for downstream readers and also the size and number of these parts.</p>
sink.rolling-policy.rollover-interval	No	30 min	Duration	<p>Maximum duration that a part file can be opened. If a part file is opened longer than the maximum duration, a new file will be generated in rolling mode. The default value is 30 minutes so that there will not be a large number of small files. The check frequency is specified by sink.rolling-policy.check-interval.</p> <p>NOTE There must be a space between the number and the unit. The supported time units include d, h, min, s, and ms. For bulk files (parquet, orc, and avro), the checkpoint interval also controls the maximum open duration of a part file.</p>
sink.rolling-policy.check-interval	No	1 min	Duration	<p>Check interval of the time-based rolling policy</p> <p>This parameter controls the frequency of checking whether a file should be rolled based on sink.rolling-policy.rollover-interval.</p>
auto-compaction	No	false	Boolean	<p>Whether automatic compaction is enabled for the streaming sink. Data is first written to temporary files. After the checkpoint is complete, the temporary files generated by the checkpoint are compacted.</p>

Parameter	Mandatory	Default Value	Type	Description
compactio n.file-size	No	Size of sink.rolling- policy.file- size	MemorySize	Size of the files that will be compacted. The default value is the size of the files that will be rolled. NOTE <ul style="list-style-type: none"> • Only files in the same checkpoint are compacted. The final files must be more than or equal to the number of checkpoints. • If the compaction takes a long time, back pressure may occur and the checkpointing may be prolonged. • After this function is enabled, final files are generated during checkpoint and a new file is opened to receive the data generated at the next checkpoint.

Example 1

Use datagen to randomly generate data and write the data into the **fileName** directory in the OBS bucket **bucketName**. The file generation time is irrelevant to the checkpoint. When the file is opened more than 30 minutes or is bigger than 128 MB, a new file is generated.

```
create table orders(
  name string,
  num INT
) with (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.name.kind' = 'random',
  'fields.name.length' = '5'
);

CREATE TABLE sink_table (
  name string,
  num INT
) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://bucketName/fileName',
  'format' = 'csv',
  'sink.rolling-policy.file-size'='128m',
  'sink.rolling-policy.rollover-interval'='30 min'
);
INSERT into sink_table SELECT * from orders;
```

Example 2

Use datagen to randomly generate data and write the data into the **fileName** directory in the OBS bucket **bucketName**. The file generation time is relevant to the checkpoint. When the checkpoint interval is reached or the file size reaches 100 MB, a new file is generated.

```
create table orders(
  name string,
```



```
num INT
) with (
'connector' = 'datagen',
'rows-per-second' = '100',
'fields.name.kind' = 'random',
'fields.name.length' = '5'
);

CREATE TABLE sink_table (
name string,
num INT
) WITH (
'connector' = 'filesystem',
'path' = 'obs://bucketName/fileName',
'format' = 'csv',
'sink.rolling-policy.file-size'='128m',
'sink.rolling-policy.rollover-interval'='30 min',
'auto-compaction'='true',
'compaction.file-size'='100m'
);
INSERT into sink_table SELECT * from orders;
```

1.3.3 Creating Dimension Tables

1.3.3.1 GaussDB(DWS) Dimension Table

Function

Create a GaussDB(DWS) table to connect to source streams for wide table generation.

Prerequisites

- Ensure that you have created a GaussDB(DWS) cluster using your account.
- A DWS database table has been created.
- An enhanced datasource connection has been created for DLI to connect to DWS clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Precautions

When you create a Flink OpenSource SQL job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Syntax

```
create table dwsSource (
attr_name attr_type
(, attr_name attr_type)*
)
with (
```

```
'connector' = 'gaussdb',
'url' = "",
'table-name' = "",
'username' = "",
'password' = ""
);
```

Parameters

Table 1-26 Parameter description

Parameter	Mandatory	Default Value	Data Type	Description
connector	Yes	None	String	Connector type. Set this parameter to gaussdb .
url	Yes	None	String	JDBC connection address. If you use the gsjdbc4 driver, set the value in jdbc:postgresql://\${ip}:\${port}/\${dbName} format. If you use the gsjdbc200 driver, set the value in jdbc:gaussdb://\${ip}:\${port}/\${dbName} format.
table-name	Yes	None	String	Name of the table where the data will be read from the database
driver	No	None	String	JDBC connection driver. The default value is org.postgresql.Driver . <ul style="list-style-type: none"> If you use the gsjdbc4 driver for connection, set connector.driver to org.postgresql.Driver. If you use the gsjdbc200 driver for connection, set connector.driver to com.huawei.gauss200.jdbc.Driver.
username	No	None	String	Database authentication user name. This parameter must be configured in pair with password .
password	No	None	String	Database authentication password. This parameter must be configured in pair with username .

Parameter	Mandatory	Default Value	Data Types	Description
scan.partition.column	No	None	String	Name of the column used to partition the input This parameter must be set when scan.partition.lower-bound , scan.partition.upper-bound , and scan.partition.num are all configured, and should not be set when other three parameters are not.
scan.partition.lower-bound	No	None	Integer	Lower bound of values to be fetched for the first partition This parameter must be set when scan.partition.column , scan.partition.upper-bound , and scan.partition.num are all configured, and should not be set when other three parameters are not.
scan.partition.upper-bound	No	None	Integer	Upper bound of values to be fetched for the last partition This parameter must be set when scan.partition.column , scan.partition.lower-bound , and scan.partition.num are all configured, and should not be set when other three parameters are not.
scan.partition.num	No	None	Integer	Number of partitions to be created This parameter must be set when scan.partition.column , scan.partition.upper-bound , and scan.partition.upper-bound are all configured, and should not be set when other three parameters are not.
scan.fetch-size	No	0	Integer	Number of rows fetched from the database each time. The default value 0 indicates that the number of rows is not limited.
scan.auto-commit	No	true	Boolean	Automatic commit flag. It determines whether each statement is committed in a transaction automatically.

Parameter	Mandatory	Default Value	Data Types	Description
lookup.cache.max-rows	No	None	Integer	The max number of rows of lookup cache. Caches exceeding the TTL will be expired. Lookup cache is disabled by default.
lookup.cache.ttl	No	None	Duration	Maximum time to live (TTL) of for every rows in lookup cache. Caches exceeding the TTL will be expired. The format is {length value}{time unit label}, for example, 123ms , 321s . The supported time units include d, h, min, s, and ms (default unit). Lookup cache is disabled by default.
lookup.max-retries	No	3	Integer	Maximum retry times if lookup database failed.

Example

Read data from a Kafka source table, use a GaussDB(DWS) table as the dimension table. Write wide table information generated by the source and dimension tables to a Kafka result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where DWS and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set GaussDB(DWS) and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the DWS and Kafka address by referring to [Testing Address Connectivity](#). If the connection passes the test, it is bound to the queue.
3. Connect to the GaussDB(DWS) database instance, create a table as a dimension table, and name the table **area_info**. Example SQL statements are as follows:

```
create table public.area_info(
  area_id VARCHAR,
  area_province_name VARCHAR,
  area_city_name VARCHAR,
  area_county_name VARCHAR,
  area_street_name VARCHAR,
  region_name VARCHAR);
```

4. Connect to the database and run the following statement to insert test data into the dimension table **area_info**:

```
insert into area_info
(area_id, area_province_name, area_city_name, area_county_name, area_street_name, region_name)
values
('330102', 'a1', 'b1', 'c1', 'd1', 'e1'),
('330106', 'a1', 'b1', 'c2', 'd2', 'e1'),
('330108', 'a1', 'b1', 'c3', 'd3', 'e1'),
('330110', 'a1', 'b1', 'c4', 'd4', 'e1');
```

5. Create a Flink OpenSource SQL job Enter the following job script and submit the job. The job script uses Kafka as the data source and a GaussDB(DWS) table as the dimension table. Data is output to a Kafka result table.

When you create a job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Set the values of the parameters in bold in the following script as needed.**

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  proctime as Proctime()  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'KafkaSourceTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'dws-order',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
)  
);  
  
-- Create an address dimension table  
create table area_info (  
  area_id string,  
  area_province_name string,  
  area_city_name string,  
  area_county_name string,  
  area_street_name string,  
  region_name string  
) WITH (  
  'connector' = 'gaussdb',  
  'driver' = 'org.postgresql.Driver',  
  'url' = 'jdbc:gaussdb://DwsAddress:DwsPort/DwsDbName',  
  'table-name' = 'area_info',  
  'username' = 'DwsUserName',  
  'password' = 'DwsPassword',  
  'lookup.cache.max-rows' = '10000',  
  'lookup.cache.ttl' = '2h'  
)  
);  
  
-- Generate a wide table based on the address dimension table containing detailed order information.  
create table order_detail(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  area_province_name string,  
  area_city_name string,  
  area_county_name string,  
  area_street_name string,  
  region_name string  
) with (  
  'connector' = 'kafka',  
  'topic' = 'KafkaSinkTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'format' = 'json'  
)  
);
```

```
insert into order_detail
select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
area.area_street_name, area.region_name from orders
left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

6. Connect to the Kafka cluster and insert the following test data into the source topic in Kafka:

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

7. Connect to the Kafka cluster and read data from the sink topic of Kafka. The result is as follows:

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}
```

```
{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}
```

```
{"order_id":"202103251505050001","order_channel":"qqShop","order_time":"2021-03-25
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

FAQs

- Q: What should I do if Flink job logs contain the following error information?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

A: The datasource connection is not bound or the binding fails.

- To reconfigure datasource connections, refer to [Enhanced Datasource Connection](#). Rectify the fault by referring to [DLI Failed to Connect to GaussDB\(DWS\) Through an Enhanced Datasource Connection](#).

- Q: How can I configure a GaussDB(DWS) table that is in a schema?

A: In the following example configures the `area_info` table in the `dbuser2` schema.

```
-- Create an address dimension table
create table area_info (
area_id string,
area_province_name string,
area_city_name string,
area_county_name string,
area_street_name string,
region_name string
) WITH (
'connector' = 'gaussdb',
'driver' = 'org.postgresql.Driver',
```

```
'url' = 'jdbc:postgresql://DwsAddress:DwsPort/DwsDbname',  
'table-name' = 'dbuser2.area_info',  
'username' = 'DwsUserName',  
'password' = 'DwsPassword',  
'lookup.cache.max-rows' = '10000',  
'lookup.cache.ttl' = '2h'  
);
```

1.3.3.2 HBase Dimension Table

Function

Create a Hbase dimension table to connect to the source streams for wide table generation.

Prerequisites

- An enhanced datasource connection has been created for DLI to connect to HBase, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).
- If MRS HBase is used, IP addresses of all hosts in the MRS cluster have been added to host information of the enhanced datasource connection.

Precautions

- When you create a Flink OpenSource SQL job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.
- All the column families in HBase table must be declared as ROW type, the field name maps to the column family name, and the nested field names map to the column qualifier names. There is no need to declare all the families and qualifiers in the schema, users can declare what is used in the query. Except the ROW type fields, the single atomic type field (for example, STRING, BIGINT) will be recognized as HBase rowkey. The rowkey field can be an arbitrary name, but should be quoted using backticks if it is a reserved keyword.

Syntax

```
create table hbaseSource (  
  attr_name attr_type  
  (' attr_name attr_type)*  
)  
with (  
  'connector' = 'hbase-2.2',  
  'table-name' = "",  
  'zookeeper.quorum' = ""  
);
```

Parameters

Table 1-27 Parameter description

Parameter	Mandatory	Default Value	Type	Description
connector	Yes	None	String	Connector type. Set this parameter to hbase-2.2 .
table-name	Yes	None	String	Name of the HBase table
zookeeper.quorum	Yes	None	String	HBase Zookeeper quorum. The format is ZookeeperAddress:ZookeeperPort. The following describes how to obtain the ZooKeeper IP address and port number: <ul style="list-style-type: none"> On the MRS Manager console, choose Cluster > <i>Name of the desired cluster</i> > Service > ZooKeeper > Instance. On the displayed page, obtain the IP address of the ZooKeeper instance. On the MRS Manager console, choose Cluster > <i>Name of the desired cluster</i> > Service > ZooKeeper > Configuration, and click All Configurations. Search for the clientPort parameter, and obtain the ZooKeeper port number.
zookeeper.znode.parent	No	/hbase	String	Root directory in ZooKeeper for the HBase cluster.
lookup.async	No	false	Boolean	Whether async lookup is enabled.
lookup.cache.max-rows	No	-1	Long	The max number of rows of lookup cache. Caches exceeding the TTL will be expired. Lookup cache is disabled by default.

Parameter	Mandatory	Default Value	Type	Description
lookup.cache.ttl	No	-1	Long	Maximum time to live (TTL) of for every rows in lookup cache. Caches exceeding the TTL will be expired. The format is {length value}{time unit label}, for example, 123ms , 321s . The supported time units include d, h, min, s, and ms (default unit). Lookup cache is disabled by default.
lookup.max-retries	No	3	Integer	Maximum retry times if lookup database failed.

Data Type Mapping

HBase stores all data as byte arrays. The data needs to be serialized and deserialized during read and write operation.

When serializing and de-serializing, Flink HBase connector uses utility class **org.apache.hadoop.hbase.util.Bytes** provided by HBase (Hadoop) to convert Flink data types to and from byte arrays.

Flink HBase connector encodes null values to empty bytes, and decode empty bytes to null values for all data types except string type. For string type, the null literal is determined by null-string-literal option.

Table 1-28 Data type mapping

Flink SQL Type	HBase Conversion
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY / VARBINARY	Return byte[] as is.
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes

Flink SQL Type	HBase Conversion
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	Number of days since 1970-01-01 00:00:00 UTC. The value is an integer.
TIME	Number of milliseconds since 1970-01-01 00:00:00 UTC. The value is an integer.
TIMESTAMP	Number of milliseconds since 1970-01-01 00:00:00 UTC. The value is of the long type.
ARRAY	Not supported
MAP / MULTISSET	Not supported
ROW	Not supported

Example

In this example, data is read from a Kafka data source, an HBase table is used as a dimension table to generate a wide table, and the result is written to a Kafka result table. The procedure is as follows (the HBase versions in this example are 1.3.1 and 2.2.3):

1. Create an enhanced datasource connection in the VPC and subnet where HBase and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#). Add MRS host information for the enhanced datasource connection..
2. Set HBase and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the HBase and Kafka address by referring to [Testing Address Connectivity](#). If the connection passes the test, it is bound to the queue.
3. Create a HBase table and name it **area_info** using the HBase shell. The table has only one column family **detail**. The creation statement is as follows:

```
create 'area_info', {NAME => 'detail'}
```

4. Run the following statement in the HBase shell to insert dimension table data:

```
put 'area_info', '330106', 'detail:area_province_name', 'a1'  
put 'area_info', '330106', 'detail:area_city_name', 'b1'  
put 'area_info', '330106', 'detail:area_county_name', 'c2'  
put 'area_info', '330106', 'detail:area_street_name', 'd2'  
put 'area_info', '330106', 'detail:region_name', 'e1'  
  
put 'area_info', '330110', 'detail:area_province_name', 'a1'  
put 'area_info', '330110', 'detail:area_city_name', 'b1'  
put 'area_info', '330110', 'detail:area_county_name', 'c4'  
put 'area_info', '330110', 'detail:area_street_name', 'd4'  
put 'area_info', '330110', 'detail:region_name', 'e1'
```

5. Create a Flink OpenSource SQL job Enter the following job script and submit the job. The job script uses Kafka as the data source and an HBase table as the dimension table. Data is output to a Kafka result table.

When you create a job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Set the values of the parameters in bold in the following script as needed.**

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  proctime as Proctime()  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'KafkaSourceTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'GroupId',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
)  
);  
  
-- Create an address dimension table  
create table area_info (  
  area_id string,  
  detail row(  
    area_province_name string,  
    area_city_name string,  
    area_county_name string,  
    area_street_name string,  
    region_name string)  
) WITH (  
  'connector' = 'hbase-2.2',  
  'table-name' = 'area_info',  
  'zookeeper.quorum' = 'ZookeeperAddress:ZookeeperPort',  
  'lookup.async' = 'true',  
  'lookup.cache.max-rows' = '10000',  
  'lookup.cache.ttl' = '2h'  
)  
);  
  
-- Generate a wide table based on the address dimension table containing detailed order information.  
create table order_detail(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,
```

```
user_name string,
area_id string,
area_province_name string,
area_city_name string,
area_county_name string,
area_street_name string,
region_name string
) with (
'connector' = 'kafka',
'topic' = '<yourSinkTopic>',
'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
'format' = 'json'
);

insert into order_detail
select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
area.area_street_name, area.region_name from orders
left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

6. Connect to the Kafka cluster and insert the following test data into the source topic in Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

7. Connect to the Kafka cluster and read data from the sink topic of Kafka. The result data is as follows:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24
10:00:00", "pay_amount":100.0, "real_pay":100.0, "pay_time":"2021-03-24
10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106", "area_province_name":"a1", "area_ci
ty_name":"b1", "area_county_name":"c2", "area_street_name":"d2", "region_name":"e1"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24
16:06:06", "pay_amount":200.0, "real_pay":180.0, "pay_time":"2021-03-24
16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106", "area_province_name":"a1", "area_ci
ty_name":"b1", "area_county_name":"c2", "area_street_name":"d2", "region_name":"e1"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":60.0, "real_pay":60.0, "pay_time":"2021-03-25
12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110", "area_province_name":"a1", "area_cit
y_name":"b1", "area_county_name":"c4", "area_street_name":"d4", "region_name":"e1"}
```

FAQs

- Q: What should I do if Flink job logs contain the following error information?

```
org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from
server in 90069ms for connection id 0x0
```

A: The datasource connection is not bound or the binding fails. Configure the datasource connection by referring to [Enhanced Datasource Connection](#) or configure the security group of the Kafka cluster to allow access from the DLI queue.

1.3.3.3 JDBC Dimension Table

Create a JDBC dimension table to connect to the source stream.

Prerequisites

You have created a JDBC instance for your account.

Precautions

When you create a Flink OpenSource SQL job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Syntax

```
CREATE TABLE table_id (
  attr_name attr_type
  (,' attr_name attr_type)*
)
WITH (
  'connector' = 'jdbc',
  'url' = "",
  'table-name' = "",
  'driver' = "",
  'username' = "",
  'password' = ""
);
```

Parameters

Table 1-29 Parameter descriptions

Parameter	Mandatory	Description
connector	Yes	Data source type. The value is fixed to jdbc .
url	Yes	Database URL
table-name	Yes	Name of the table where the data will be read from the database
driver	No	Driver required for connecting to the database. If you do not set this parameter, the automatically extracted URL will be used.
username	No	Database authentication user name. This parameter must be configured in pair with password .
password	No	Database authentication password. This parameter must be configured in pair with username .
scan.partition.column	No	Name of the column used to partition the input This parameter must be set when scan.partition.lower-bound , scan.partition.upper-bound , and scan.partition.num are all configured, and should not be set when other three parameters are not.

Parameter	Mandatory	Description
scan.partition.lower-bound	No	Lower bound of values to be fetched for the first partition This parameter must be set when scan.partition.column , scan.partition.upper-bound , and scan.partition.num are all configured, and should not be set when other three parameters are not.
scan.partition.upper-bound	No	Upper bound of values to be fetched for the last partition This parameter must be set when scan.partition.column , scan.partition.lower-bound , and scan.partition.num are all configured, and should not be set when other three parameters are not.
scan.partition.num	No	Number of partitions to be created This parameter must be set when scan.partition.column , scan.partition.upper-bound , and scan.partition.lower-bound are all configured, and should not be set when other three parameters are not.
scan.fetch-size	No	Number of rows fetched from the database each time. The default value is 0 , indicating the hint is ignored.
lookup.cache.max-rows	No	Maximum number of cached rows in a dimension table. If the number of cached rows exceeds the value, old data will be deleted. The value -1 indicates that data cache disabled.
lookup.cache.ttl	No	Maximum time to live (TTL) of for every rows in lookup cache. Caches exceeding the TTL will be expired. The format is {length value}{time unit label}, for example, 123ms , 321s . The supported time units include d, h, min, s, and ms (default unit).
lookup.max-retries	No	Maximum number of attempts to obtain data from the dimension table. The default value is 3 .

Data Type Mapping

Table 1-30 Data type mapping

MySQL Type	PostgreSQL Type	Flink SQL Type
TINYINT	-	TINYINT
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

MySQL Type	PostgreSQL Type	Flink SQL Type
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

Example

Read data from a Kafka source table, use a JDBC table as the dimension table. Write table information generated by the source and dimension tables to a Kafka result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where MySQL and Kafka locate, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set MySQL and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the MySQL and Kafka address by referring to [Testing Address Connectivity](#). If the connection passes the test, it is bound to the queue.
3. Connect to the MySQL database instance, create a table in the flink database as a dimension table, and name the table **area_info**. Example SQL statements are as follows:

```
CREATE TABLE `flink`.`area_info` (
  `area_id` VARCHAR(32) NOT NULL,
  `area_province_name` VARCHAR(32) NOT NULL,
  `area_city_name` VARCHAR(32) NOT NULL,
  `area_county_name` VARCHAR(32) NOT NULL,
  `area_street_name` VARCHAR(32) NOT NULL,
  `region_name` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`area_id`)
) ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_general_ci;
```

4. Connect to the MySQL database and run the following statement to insert test data into the JDBC dimension table **area_info**:

```
insert into flink.area_info
(area_id, area_province_name, area_city_name, area_county_name, area_street_name, region_name)
values
('330102', 'a1', 'b1', 'c1', 'd1', 'e1'),
('330106', 'a1', 'b1', 'c2', 'd2', 'e1'),
('330108', 'a1', 'b1', 'c3', 'd3', 'e1'), ('330110', 'a1', 'b1', 'c4', 'd4', 'e1');
```

5. Create a Flink OpenSource SQL job Enter the following job script and submit the job. The job script uses Kafka as the data source and a JDBC table as the dimension table. Data is output to a Kafka result table.

When you create a job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Set the values of the parameters in bold in the following script as needed.**

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  proctime as Proctime()  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'KafkaSourceTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'jdbc-order',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
);  
  
-- Create an address dimension table  
create table area_info (  
  area_id string,  
  area_province_name string,  
  area_city_name string,  
  area_county_name string,  
  area_street_name string,  
  region_name string  
) WITH (  
  'connector' = 'jdbc',  
  'url' = 'jdbc:mysql://JDBCAddress:JDBCPort/flink',--flink is the MySQL database where the area_info  
table locates.  
  'table-name' = 'area_info',  
  'username' = 'JDBCUserName',  
  'password' = 'JDBCPassWord'  
);  
  
-- Generate a wide table based on the address dimension table containing detailed order information.  
create table order_detail(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  area_province_name string,  
  area_city_name string,  
  area_county_name string,  
  area_street_name string,  
  region_name string  
) with (  
  'connector' = 'kafka',  
  'topic' = 'KafkaSinkTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'format' = 'json'  
);  
  
insert into order_detail  
  select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,  
  orders.pay_time, orders.user_id, orders.user_name,  
  area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,  
  area.area_street_name, area.region_name from orders
```

```
left join area_info for system_time as of orders.proctime as area on orders.area_id =  
area.area_id;
```

6. Connect to the Kafka cluster and insert the following test data into the source topic in Kafka:

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",  
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",  
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25  
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",  
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",  
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",  
"user_name":"Cindy", "area_id":"330108"}
```

7. Connect to the Kafka cluster and read data from the sink topic of Kafka.

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24  
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24  
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_c  
ity_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}
```

```
{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25  
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25  
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_c  
ity_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}
```

```
{"order_id":"202103251505050001","order_channel":"qqShop","order_time":"2021-03-25  
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25  
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c  
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

FAQs

None

1.3.3.4 Redis Dimension Table

Function

Create a Redis table to connect to source streams for wide table generation.

Prerequisites

- An enhanced datasource connection with Redis has been established, so that you can configure security group rules as required.
 - For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
 - For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Precautions

- When you create a Flink OpenSource SQL job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.
- To obtain the key values, you can set the primary key in Flink. The primary key maps to the Redis key.

- If the primary key cannot be a composite primary key, and only can be one field.
- Constraints on **schema-syntax**:
 - If **schema-syntax** is **map** or **array**, there can be only one non-primary key and it must be of the same **map** or **array** type.
 - If **schema-syntax** is **fields-scores**, the number of non-primary keys must be an even number, and the second key of every two keys except the primary key must be of the **double** type. The **double** value is the score of the previous key. The following is an example:

```
CREATE TABLE redisSource (  
  redisKey string,  
  order_id string,  
  score1 double,  
  order_channel string,  
  score2 double,  
  order_time string,  
  score3 double,  
  pay_amount double,  
  score4 double,  
  real_pay double,  
  score5 double,  
  pay_time string,  
  score6 double,  
  user_id string,  
  score7 double,  
  user_name string,  
  score8 double,  
  area_id string,  
  score9 double,  
  primary key (redisKey) not enforced  
) WITH (  
  'connector' = 'redis',  
  'host' = 'RedisIP',  
  'password' = 'RedisPassword',  
  'data-type' = 'sorted-set',  
  'deploy-mode' = 'master-replica',  
  'schema-syntax' = 'fields-scores'  
);
```

- Restrictions on **data-type**:
 - When **data-type** is **set**, the types of non-primary keys defined in Flink must be the same.
 - If **data-type** is **sorted-set** and **schema-syntax** is **fields** or **array**, only **sorted set** values can be read from Redis, and the **score** value cannot be read.
 - If **data-type** is **string**, only one non-primary key field is allowed.
 - If **data-type** is **sorted-set** and **schema-syntax** is **map**, there can be only one non-primary key in addition to the primary key and the non-primary key must be of the **map** type. The **map** values of the non-primary key must be of the **double** type, indicating the score. The keys in the map are the values in the Redis set.
 - If **data-type** is **sorted-set** and **schema-syntax** is **array-scores**, only two non-primary keys are allowed and must be of the **array** type.

The first key indicates values in the Redis set. The second key is of the **array<double>** type, indicating index scores. The following is an example:

```
CREATE TABLE redisSink (  
  order_id string,  
  arrayField Array<String>,  
  arrayScore array<double>,  
);
```

```

primary key (order_id) not enforced
) WITH (
'connector' = 'redis',
'host' = 'RedisIP',
'password' = 'RedisPassword',
'data-type' = 'sorted-set',
"default-score" = '3',
'deploy-mode' = 'master-replica',
'schema-syntax' = 'array-scores'
);

```

Syntax

```

create table dwsSource (
attr_name attr_type
(, attr_name attr_type)*
(, watermark for rowtime_column_name as watermark-strategy_expression)
,PRIMARY KEY (attr_name, ...) NOT ENFORCED
)
with (
'connector' = 'redis',
'host' = "
);

```

Parameters

Table 1-31 Parameter description

Parameter	Mandatory	Default Value	Data Types	Description
connector	Yes	None	String	Connector type. Set this parameter to redis .
host	Yes	None	String	Redis connector address
port	No	6379	Integer	Redis connector port
password	No	None	String	Redis authentication password
namespace	No	None	String	Redis key namespace
delimiter	No	:	String	Delimiter between the Redis key and namespace
data-type	No	hash	String	Redis data type. Available values are as follows: <ul style="list-style-type: none"> hash list set sorted-set string For details about the constraints, see Constraints on data-type .

Parameter	Mandatory	Default Value	Data Types	Description
schema-syntax	No	fields	String	<p>Redis schema semantics. Available values are as follows:</p> <ul style="list-style-type: none"> • fields: applicable to all data types • fields-scores: applicable to sorted set data • array: applicable to list, set, and sorted set data • array-scores: applicable to sorted set data • map: applicable to hash and sorted set data <p>For details about the constraints, see Constraints on schema-syntax.</p>
deploy-mode	No	standalone	String	<p>Deployment mode of the Redis cluster. The value can be standalone, master-replica, or cluster. The default value is standalone.</p>
retry-count	Yes	5	Integer	<p>Size of each connection request queue. If the number of connection requests in a queue exceeds the queue size, command calling will cause RedisException. Setting requestQueueSize to a small value will cause exceptions to occur earlier during overload or disconnection. A larger value indicates more time required to reach the boundary, but more requests may be queued and more heap space may be used. The default value is 2147483647.</p>
connection-timeout-millis	No	10000	Integer	<p>Maximum timeout for connecting to the Redis cluster</p>
commands-timeout-millis	No	2000	Integer	<p>Maximum time for waiting for a completion response</p>
rebalancing-timeout-millis	No	15000	Integer	<p>Sleep time when the Redis cluster fails</p>

Parameter	Mandatory	Default Value	Data Types	Description
scan-keys-count	No	1000	Integer	Number of data records read in each scan
default-score	No	0	Double	Default score when data-type is sorted-set
deserialize-error-policy	No	fail-job	Enum	How to process a data parsing failure Available values are as follows: <ul style="list-style-type: none"> • fail-job: Fail the job • skip-row: Skip the current data. • null-field: Set the current data to null.
skip-null-values	No	true	Boolean	Whether null values will be skipped
lookup.async	No	false	Boolean	Whether asynchronous I/O will be used when this table is used as a dimension table

Example

Read data from a Kafka source table, use a Redis table as the dimension table. Write wide table information generated by the source and dimension tables to a Kafka result table. The procedure is as follows:

1. Create an enhanced datasource connection in the VPC and subnet where Redis and Kafka locates, and bind the connection to the required Flink elastic resource pool. For details, see [Enhanced Datasource Connections](#).
2. Set Redis and Kafka security groups and add inbound rules to allow access from the Flink queue. Test the connectivity using the Redis address by referring to [Testing Address Connectivity](#). If the connection passes the test, it is bound to the queue.
3. Run the following commands on the Redis client to send data to Redis:

```
HMSET 330102 area_province_name a1 area_province_name b1 area_county_name c1
area_street_name d1 region_name e1

HMSET 330106 area_province_name a1 area_province_name b1 area_county_name c2
area_street_name d2 region_name e1

HMSET 330108 area_province_name a1 area_province_name b1 area_county_name c3
area_street_name d3 region_name e1

HMSET 330110 area_province_name a1 area_province_name b1 area_county_name c4
area_street_name d4 region_name e1
```
4. Create a Flink OpenSource SQL job Enter the following job script and submit the job. The job script uses Kafka as the data source and a Redis table as the dimension table. Data is output to a Kafka result table.

When you create a job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs. **Set the values of the parameters in bold in the following script as needed.**

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  proctime as Proctime()  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'kafkaSourceTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'GroupId',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
);  
  
-- Create an address dimension table  
create table area_info (  
  area_id string,  
  area_province_name string,  
  area_city_name string,  
  area_county_name string,  
  area_street_name string,  
  region_name string,  
  primary key (area_id) not enforced -- Redis key  
) WITH (  
  'connector' = 'redis',  
  'host' = 'RedisIP',  
  'password' = 'RedisPassword',  
  'data-type' = 'hash',  
  'deploy-mode' = 'master-replica'  
);  
  
-- Generate a wide table based on the address dimension table containing detailed order information.  
create table order_detail(  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  area_province_name string,  
  area_city_name string,  
  area_county_name string,  
  area_street_name string,  
  region_name string  
) with (  
  'connector' = 'kafka',  
  'topic' = 'kafkaSinkTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'format' = 'json'  
);  
  
insert into order_detail  
  select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,  
  orders.pay_time, orders.user_id, orders.user_name,  
  area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,  
  area.area_street_name, area.region_name from orders  
  left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

5. Connect to the Kafka cluster and insert the following test data into the source topic in Kafka:

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",  
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",  
"user_name":"Alice", "area_id":"330106"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25  
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",  
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",  
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",  
"user_name":"Cindy", "area_id":"330108"}
```

6. Connect to the Kafka cluster and read data from the sink topic of Kafka. The result data is as follows:

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24  
16:06:06", "pay_amount":200.0, "real_pay":180.0, "pay_time":"2021-03-24  
16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106", "area_province_name":"a1", "area_ci  
ty_name":"b1", "area_county_name":"c2", "area_street_name":"d2", "region_name":"e1"}
```

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25  
12:02:02", "pay_amount":60.0, "real_pay":60.0, "pay_time":"2021-03-25  
12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110", "area_province_name":"a1", "area_cit  
y_name":"b1", "area_county_name":"c4", "area_street_name":"d4", "region_name":"e1"}
```

```
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25  
15:05:05", "pay_amount":500.0, "real_pay":400.0, "pay_time":"2021-03-25  
15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108", "area_province_name":"a1", "area_c  
ity_name":"b1", "area_county_name":"c3", "area_street_name":"d3", "region_name":"e1"}
```

FAQs

If Chinese characters are written to the Redis in the Windows environment, an exception will occur during data writing.

1.3.4 Format

1.3.4.1 Avro

Function

Apache Avro is supported for you to read and write Avro data based on an Avro schema with Flink. The Avro schema is derived from the table schema.

Supported Connectors

- Kafka
- Upsert Kafka

Parameters

Table 1-32 Parameter

Parameter	Mandatory	Default value	Type	Description
format	Yes	None	String	Format to be used. Set the value to avro .
avro.codec	No	None	String	Avro compression codec for the file system only The codec is disabled by default. Available values are deflate , snappy , bzip2 , and xz .

Data Type Mapping

Currently, the Avro schema is derived from the table schema and cannot be explicitly defined. The following table lists mappings between Flink to Avro types.

In addition to the following types, Flink supports reading/writing nullable types. Flink maps nullable types to Avro **union(something, null)**, where **something** is an Avro type converted from Flink type.

You can refer to [Apache Avro 1.11.0 Specification](#) for more information about Avro types.

Table 1-33 Data Type Mapping

Flink SQL Type	Avro Type	Avro Logical Type
CHAR / VARCHAR / STRING	string	-
BOOLEAN	boolean	-
BINARY / VARBINARY	bytes	-
DECIMAL	fixed	decimal
TINYINT	int	-
SMALLINT	int	-

Flink SQL Type	Avro Type	Avro Logical Type
INT	int	-
BIGINT	long	-
FLOAT	float	-
DOUBLE	double	-
DATE	int	date
TIME	int	time-millis
TIMESTAMP	long	timestamp-millis
ARRAY	array	-
MAP (keys must be of the string, char, or varchar type.)	map	-
MULTISET (elements must be of the string, char, or varchar type.)	map	-
ROW	record	-

Example

Read data from Kafka, deserialize the data to the Avro format, and outputs the data to print.

Step 1 Create a datasource connection for access to the VPC and subnet where Kafka locates and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka IP address. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Create a Flink OpenSource SQL job and select Flink 1.12. Copy the following statement and submit the job:

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' =
  '<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>,<yourKafkaAddress3>:<yourKafkaPort>',
```

```
'properties.group.id' = '<yourGroupId>',
'scan.startup.mode' = 'latest-offset',
"format" = "avro"
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from kafkaSource;
```

Step 3 Insert the following data to Kafka using Avro data serialization:

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

Step 4 Perform the following operations to view the output:

- Method 1: Locate the job and click **More > FlinkUI**. Choose **Task Managers > Stdout**.
- Method 2: If you allow DLI to save job logs in OBS, view the output in the **taskmanager.out** file.
+l(202103241000000001,webShop,2021-03-2410:00:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330106)
+l(202103241606060001,appShop,2021-03-2416:06:06,200.0,180.0,2021-03-2416:10:06,0001,Alice,330106)

----End

1.3.4.2 Canal

Function

Canal is a Changelog Data Capture (CDC) tool that can stream changes in real-time from MySQL into other systems. Canal provides a unified format schema for changelog and supports to serialize messages using JSON and protobuf (the default format for Canal).

Flink supports to interpret Canal JSON messages as INSERT, UPDATE, and DELETE messages into the Flink SQL system. This is useful in many cases to leverage this feature, such as:

- synchronizing incremental data from databases to other systems
- Auditing logs
- Real-time materialized view on databases
- Temporal join changing history of a database table, etc.

Flink also supports to encode the INSERT, UPDATE, and DELETE messages in Flink SQL as Canal JSON messages, and emit to storage like Kafka. However, currently Flink cannot combine UPDATE_BEFORE and UPDATE_AFTER into a single UPDATE message. Therefore, Flink encodes UPDATE_BEFORE and UPDATE_AFTER as DELETE and INSERT Canal messages.

Parameters

Table 1-34 Parameter description

Parameter	Mandatory	Default Value	Type	Description
format	Yes	None	String	Format to be used. In this example. Set this parameter to canal-json .
canal-json.ignore-parse-errors	No	false	Boolean	Whether fields and rows with parse errors will be skipped or failed. The default value is false , indicating that an error will be thrown. Fields are set to null in case of errors.
canal-json.timestamp-format.standard	No	'SQL'	String	Input and output timestamp formats. Currently supported values are SQL and ISO-8601 : <ul style="list-style-type: none"> • SQL will parse input timestamp in "yyyy-MM-dd HH:mm:ss.s{precision}" format, for example 2020-12-30 12:13:14.123 and output timestamp in the same format. • ISO-8601 will parse input timestamp in "yyyy-MM-ddTHH:mm:ss.s{precision}" format, for example 2020-12-30T12:13:14.123 and output timestamp in the same format.

Parameter	Mandatory	Default Value	Type	Description
canal-json.map-null-key.mode	No	'FALL'	String	Handling mode when serializing null keys for map data. Available values are as follows: <ul style="list-style-type: none"> ● FAIL will throw exception when encountering map value with null key. ● DROP will drop null key entries for map data. ● LITERAL replaces the empty key value in the map with a string constant. The string literal is defined by canal-json.map-null-key.literal option.
canal-json.map-null-key.literal	No	'null'	String	String literal to replace null key when canal-json.map-null-key.mode is LITERAL .
canal-json.database.include	No	None	String	An optional regular expression to only read the specific databases changelog rows by regular matching the database meta field in the Canal record.
canal-json.table.include	No	None	String	An optional regular expression to only read the specific tables changelog rows by regular matching the table meta field in the Canal record.

Supported Connectors

- Kafka

Example

Use Kafka to send data and output the data to print.

- Step 1** Create a datasource connection for the communication with the VPC and subnet where Kafka locates and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka IP address. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Create a Flink OpenSource SQL job and select Flink 1.12. Copy the following statement and submit the job:

```
create table kafkaSource(  
  id bigint,  
  name string,  
  description string,  
  weight DECIMAL(10, 2)  
) with (  
  'connector' = 'kafka',  
  'topic' = '<yourTopic>',  
  'properties.group.id' = '<yourGroupId>',  
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'canal-json'  
);  
create table printSink(  
  id bigint,  
  name string,  
  description string,  
  weight DECIMAL(10, 2)  
) with (  
  'connector' = 'print'  
);  
insert into printSink select * from kafkaSource;
```

Step 3 Insert the following data to the corresponding topic in Kafka:

```
{  
  "data": [  
    {  
      "id": "111",  
      "name": "scooter",  
      "description": "Big 2-wheel scooter",  
      "weight": "5.18"  
    }  
  ],  
  "database": "inventory",  
  "es": 1589373560000,  
  "id": 9,  
  "isDdl": false,  
  "mysqlType": {  
    "id": "INTEGER",  
    "name": "VARCHAR(255)",  
    "description": "VARCHAR(512)",  
    "weight": "FLOAT"  
  },  
  "old": [  
    {  
      "weight": "5.15"  
    }  
  ],  
  "pkNames": [  
    "id"  
  ],  
  "sql": "",  
  "sqlType": {  
    "id": 4,  
    "name": 12,  
    "description": 12,  
    "weight": 7  
  },  
  "table": "products",  
  "ts": 1589373560798,  
  "type": "UPDATE"  
}
```

Step 4 View the output through either of the following methods:

- Method 1: Locate the job and click **More > FlinkUI**. Choose **Task Managers > Stdout**.

- Method 2: If you allow DLI to save job logs in OBS, view the output in the **taskmanager.out** file.

```
-U(111,scooter,Big2-wheel scooter,5.15)
+U(111,scooter,Big2-wheel scooter,5.18)
```

----End

1.3.4.3 Confluent Avro

Function

The Avro Schema Registry (**avro-confluent**) format allows you to read records that were serialized by the **io.confluent.kafka.serializers.KafkaAvroSerializer** and to write records that can in turn be read by the **io.confluent.kafka.serializers.KafkaAvroDeserializer**.

When reading (deserializing) a record with this format the Avro writer schema is fetched from the configured Confluent Schema Registry based on the schema version ID encoded in the record while the reader schema is inferred from table schema.

When writing (serializing) a record with this format the Avro schema is inferred from the table schema and used to retrieve a schema ID to be encoded with the data. The lookup is performed with in the configured Confluent Schema Registry under the **subject**. The subject is specified by **avro-confluent.schema-registry.subject**.

Supported Connectors

- kafka
- upsert kafka

Parameters

Table 1-35 Parameter description

Parameter	Mandatory	Default Value	Type	Description
format	Yes	None	String	Format to be used. Set this parameter to avro-confluent .

Parameter	Mandatory	Default Value	Type	Description
avro-confluent.schema-registry.subject	No	None	String	The Confluent Schema Registry subject under which to register the schema used by this format during serialization. By default, kafka and upsert-kafka connectors use <topic_name>-value or <topic_name>-key as the default subject name if this format is used as the value or key format.
avro-confluent.schema-registry.url	Yes	None	String	URL of the Confluent Schema Registry to fetch/register schemas.

Example

1. Read JSON data from the source topic in Kafka and write the data in Confluent Avro format to the sink topic.

Step 1 Create a datasource connection for the communication with the VPC and subnet where Kafka and ECS locate and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka and ECS IP addresses. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Purchase an ECS cluster, download Confluent 5.5.2 (<https://packages.confluent.io/archive/5.5/>) and jdk1.8.0_232, and upload them to the ECS cluster. Run the following command to decompress the packages (assume that the decompression directories are **confluent-5.5.2** and **jdk1.8.0_232**):

```
tar zxvf confluent-5.5.2-2.11.tar.gz
tar zxvf jdk1.8.0_232.tar.gz
```

Step 3 Run the following commands to install jdk1.8.0_232 in the current ECS cluster. You can run the **pwd** command in the **jdk1.8.0_232** folder to view the value of **yourJdkPath**.

```
export JAVA_HOME=<yourJdkPath>
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
```

Step 4 Go to the **confluent-5.5.2/etc/schema-registry/** directory and modify the following configuration items in the **schema-registry.properties** file:

```
listeners=http://<yourEcsIp>:8081
kafkastore.bootstrap.servers=<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>
```


Step 5 Switch to the **confluent-5.5.2** directory and run the following command to start Confluent:

```
bin/schema-registry-start etc/schema-registry/schema-registry.properties
```

Step 6 Create a Flink opensource SQL job, select the Flink 1.12 version, and allow DLI to save job logs in OBS. Add the following statement to the job and submit it:

```
CREATE TABLE kafkaSource (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'properties.bootstrap.servers' =  
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',  
  'topic' = '<yourSourceTopic>',  
  'properties.group.id' = '<yourGroupld>',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
);  
CREATE TABLE kafkaSink (  
  order_id string,  
  order_channel string,  
  order_time string,  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string  
) WITH (  
  'connector' = 'kafka',  
  'properties.bootstrap.servers' =  
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',  
  'topic' = '<yourSinkTopic>',  
  'format' = 'avro-confluent',  
  'avro-confluent.schema-registry.url' = 'http://<yourEcsIp>:8081',  
  'avro-confluent.schema-registry.subject' = '<yourSubject>'  
);  
insert into kafkaSink select * from kafkaSource;
```

Step 7 Insert the following data into Kafka:

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",  
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",  
"user_name":"Alice", "area_id":"330106"}  
  
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",  
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",  
"user_name":"Alice", "area_id":"330106"}
```

Step 8 Read the data of the sink Kafka topic. You will find that the data has been written and the schema has been saved to the **_schema** topic of Kafka.

----End

1.3.4.4 CSV

Function

The CSV format allows you to read and write CSV data based on a CSV schema. Currently, the CSV schema is derived from table schema.

Supported Connectors

- Kafka
- Upsert Kafka

Parameters

Table 1-36

Parameter	Mandatory	Default value	Type	Description
format	Yes	None	String	Format to be used. Set the value to csv .
csv.field-delimiter	No	,	String	Field delimiter character, which must be a single character. You can use backslash to specify special characters, for example, <code>\t</code> represents the tab character. You can also use unicode to specify them in plain SQL, for example, <code>'csv.field-delimiter' = '\u0001'</code> represents the 0x01 character.
csv.disable-quote-character	No	false	Boolean	Disabled quote character for enclosing field values. If you set this parameter to true , csv.quote-character cannot be set.
csv.quote-character	No	"	String	Quote character for enclosing field values.
csv.allow-comments	No	false	Boolean	Ignore comment lines that start with <code>#</code> . If you set this parameter to true , make sure to also ignore parse errors to allow empty rows.
csv.ignore-parse-errors	No	false	Boolean	Whether fields and rows with parse errors will be skipped or failed. The default value is false , indicating that an error will be thrown. Fields are set to null in case of errors.

Parameter	Mandatory	Default value	Type	Description
csv.array-element-delimiter	No	;	String	Array element delimiter string for separating array and row element values.
csv.escape-character	No	None	String	Escape character for escaping values
csv.null-literal	No	None	String	Null literal string that is interpreted as a null value.

Example

Use Kafka to send data and output the data to print.

- Step 1** Create a datasource connection for the communication with the VPC and subnet where Kafka locates and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka IP address. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
- Step 2** Create a Flink OpenSource SQL job. Copy the following statement and submit the job:

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSourceTopic>',
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupld>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "csv"
);

CREATE TABLE kafkaSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
```

```
'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
"format" = "csv"
);
insert into kafkaSink select * from kafkaSource;
```

Step 3 Insert the following data into the source Kafka topic:

```
202103251505050001,qqShop,2021-03-25 15:05:05,500.00,400.00,2021-03-25 15:10:00,0003,Cindy,330108
202103241606060001,appShop,2021-03-24 16:06:06,200.00,180.00,2021-03-24 16:10:06,0001,Alice,330106
```

Step 4 Read data from the sink Kafka topic. The result is as follows:

```
202103251505050001,qqShop,"2021-03-25 15:05:05",500.0,400.0,"2021-03-25 15:10:00",0003,Cindy,330108
202103241606060001,appShop,"2021-03-24 16:06:06",200.0,180.0,"2021-03-24 16:10:06",0001,Alice,330106
```

----End

1.3.4.5 Debezium

Function

Debezium is a Changelog Data Capture (CDC) tool that can stream changes in real-time from other databases into Kafka. Debezium provides a unified format schema for changelog and supports to serialize messages using JSON.

Flink supports to interpret Debezium JSON and Avro messages as INSERT/UPDATE/DELETE messages into Flink SQL system. This is useful in many cases to leverage this feature, such as:

- synchronizing incremental data from databases to other systems
- Auditing logs
- Real-time materialized view on databases
- Temporal join changing history of a database table, etc.

Parameters

Table 1-37

Parameter	Mandatory	Default Value	Mandatory	Description
format	Yes	None	String	Format to be used. In this example. Set this parameter to debezium-json .
debezium-json.schema-include	No	false	Boolean	Whether the Debezium JSON messages contain the schema. When setting up Debezium Kafka Connect, enable the Kafka configuration value.converter.schemas.enable to include the schema in the message.

Parameter	Mandatory	Default Value	Mandatory	Description
debezium-json.ignore-parse-errors	No	false	Boolean	Whether fields and rows with parse errors will be skipped or failed. The default value is false , indicating that an error will be thrown. Fields are set to null in case of errors.
debezium-json.timestamp-format.standard	No	'SQL'	String	Input and output timestamp formats. Currently supported values are SQL and ISO-8601 . <ul style="list-style-type: none"> • SQL will parse input timestamp in "yyyy-MM-dd HH:mm:ss.s{precision}" format, for example 2020-12-30 12:13:14.123 and output timestamp in the same format. • ISO-8601 will parse input timestamp in "yyyy-MM-ddTHH:mm:ss.s{precision}" format, for example 2020-12-30T12:13:14.123 and output timestamp in the same format.
debezium-json.map-null-key.mode	No	'FAIL'	String	Handling mode when serializing null keys for map data. Available values are as follows: <ul style="list-style-type: none"> • FAIL will throw exception when encountering map value with null key. • DROP will drop null key entries for map data. • LITERAL replaces the empty key value in the map with a string constant. The string literal is defined by debezium-json.map-null-key.literal option.
debezium-json.map-null-key.literal	No	'null'	String	String literal to replace null key when debezium-json.map-null-key.mode is LITERAL .

Supported Connectors

- Kafka

Example

Use Kafka to send data and output the data to print.

Step 1 Create a datasource connection for the communication with the VPC and subnet where Kafka locates and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka IP address. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Create a Flink OpenSource SQL job. Copy the following statement and submit the job:

```
create table kafkaSource(  
  id BIGINT,  
  name STRING,  
  description STRING,  
  weight DECIMAL(10, 2)  
  ) with (  
    'connector' = 'kafka',  
    'topic' = '<yourTopic>',  
    'properties.group.id' = '<yourGroupId>',  
    'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',  
    'scan.startup.mode' = 'latest-offset',  
    'format' = 'debezium-json'  
  );  
create table printSink(  
  id BIGINT,  
  name STRING,  
  description STRING,  
  weight DECIMAL(10, 2)  
  ) with (  
    'connector' = 'print'  
  );  
insert into printSink select * from kafkaSource;
```

Step 3 Insert the following data to the corresponding topic in Kafka:

```
{  
  "before": {  
    "id": 111,  
    "name": "scooter",  
    "description": "Big 2-wheel scooter",  
    "weight": 5.18  
  },  
  "after": {  
    "id": 111,  
    "name": "scooter",  
    "description": "Big 2-wheel scooter",  
    "weight": 5.15  
  },  
  "source": {  
    "version": "0.9.5.Final",  
    "connector": "mysql",  
    "name": "fullfillment",  
    "server_id": 1,  
    "ts_sec": 1629607909,  
    "gtid": "mysql-bin.000001",  
    "pos": 2238,"row": 0,  
    "snapshot": false,  
    "thread": 7,  
    "db": "inventory",  
    "table": "test",  
    "query": null,  
    "op": "u",  
    "ts_ms": 1589362330904,  
  }  
}
```

```
"transaction": null
}
```

Step 4 View the output through either of the following methods:

- Method 1: Locate the job and click **More > FlinkUI**. Choose **Task Managers > Stdout**.
- Method 2: If you allow DLI to save job logs in OBS, view the output in the **taskmanager.out** file.

```
-U(111,scooter,Big2-wheel scooter,5.18)
+U(111,scooter,Big2-wheel scooter,5.15)
```

----End

1.3.4.6 JSON

Function

The JSON format allows you to read and write JSON data based on a JSON schema. Currently, the JSON schema is derived from table schema.

Supported Connectors

- Kafka
- Upsert Kafka
- Elasticsearch

Parameters

Table 1-38

Parameter	Mandatory	Default Value	Type	Description
format	Yes	None	String	Format to be used. Set this parameter to json .
json.fail-on-missing-field	No	false	Boolean	Whether to skip the field or row or throws an error when a field to be parsed is missing. The default value is false , indicating that no error will be thrown.
json.ignore-parse-errors	No	false	Boolean	Whether fields and rows with parse errors will be skipped or failed. The default value is false , indicating that an error will be thrown. Fields are set to null in case of errors.

Parameter	Mandatory	Default Value	Type	Description
json.timestamp-format.standard	No	'SQL'	String	<p>Input and output timestamp format for <code>TIMESTAMP</code> and <code>TIMESTAMP WITH LOCAL TIME ZONE</code>.</p> <p>Currently supported values are SQL and ISO-8601:</p> <ul style="list-style-type: none"> SQL will parse the input <code>TIMESTAMP</code> values in "yyyy-MM-dd HH:mm:ss.s{precision}" format, for example, 2020-12-30 12:13:14.123, parse <code>TIMESTAMP WITH LOCAL TIME ZONE</code> values in "yyyy-MM-dd HH:mm:ss.s{precision}'Z'" format, for example, 2020-12-30 12:13:14.123Z and output timestamp in the same format. ISO-8601 will parse the input <code>TIMESTAMP</code> values in "yyyy-MM-ddTHH:mm:ss.s{precision}" format, for example, 2020-12-30T12:13:14.123 parse <code>TIMESTAMP WITH LOCAL TIME ZONE</code> values in "yyyy-MM-ddTHH:mm:ss.s{precision}'Z'" format, for example, 2020-12-30T12:13:14.123Z and output timestamp in the same format.

Parameter	Mandatory	Default Value	Type	Description
json.map-null-key.mode	No	'FALL'	String	Handling mode when serializing null keys for map data. Available values are as follows: <ul style="list-style-type: none"> ● FAIL will throw exception when encountering map value with null key. ● DROP will drop null key entries for map data. ● LITERAL replaces the empty key value in the map with a string constant. The string literal is defined by json.map-null-key.literal option.
json.map-null-key.literal	No	'null'	String	String literal to replace null key when json.map-null-key.mode is LITERAL .

Example

In this example, data is read from a topic and written to another using a Kafka sink.

- Step 1** Create a datasource connection for the communication with the VPC and subnet where Kafka locates and bind the connection to the queue. Set an inbound rule for the security group to allow access of the queue and test the connectivity using the Kafka address. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.
- Step 2** Create a Flink OpenSource SQL job, select Flink 1.12, and allow DLI to save job logs in OBS. Use the following statement in the job and submit it:

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSourceTopic>',
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE kafkaSink (
  order_id string,
```

```
order_channel string,  
order_time string,  
pay_amount double,  
real_pay double,  
pay_time string,  
user_id string,  
user_name string,  
area_id string  
) WITH (  
  'connector' = 'kafka',  
  'topic' = '<yourSinkTopic>',  
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',  
  "format" = "json"  
);  
  
insert into kafkaSink select * from kafkaSource;
```

Step 3 Insert the following data into the source Kafka topic:

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24  
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24  
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}  
  
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24  
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24  
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

Step 4 Read data from the sink topic. The result is as follows:

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24  
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24  
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}  
  
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24  
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24  
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

----End

1.3.4.7 Maxwell

Function

Flink supports to interpret Maxwell JSON messages as INSERT/UPDATE/DELETE messages into Flink SQL system. This is useful in many cases to leverage this feature,

such as:

- Synchronizing incremental data from databases to other systems
- Auditing logs
- Real-time materialized views on databases
- Temporal join changing history of a database table and so on

Flink also supports to encode the INSERT/UPDATE/DELETE messages in Flink SQL as Maxwell JSON messages, and emit to external systems like Kafka. However, currently Flink cannot combine UPDATE_BEFORE and UPDATE_AFTER into a single UPDATE message. Therefore, Flink encodes UPDATE_BEFORE and UPDATE_AFTER as DELETE and INSERT Maxwell messages.

Parameters

Table 1-39

Parameter	Mandatory	Default Value	Type	Description
format	Yes	None	String	Format to be used. Set this parameter to maxwell-json .
maxwell-json.ignore-parse-errors	No	false	Boolean	Whether fields and rows with parse errors will be skipped or failed. Fields are set to null in case of errors.
maxwell-json.timestamp-format.standard	No	'SQL'	String	Input and output timestamp formats. Currently supported values are SQL and ISO-8601 : SQL will parse input timestamp in "yyyy-MM-dd HH:mm:ss.s{precision}" format, for example, 2020-12-30 12:13:14.123 and output timestamp in the same format. ISO-8601 will parse input timestamp in "yyyy-MM-ddTHH:mm:ss.s{precision}" format, for example 2020-12-30T12:13:14.123 and output timestamp in the same format.
maxwell-json.map-null-key.mode	No	'FAIL'	String	Handling mode when serializing null keys for map data. Currently supported values are 'FAIL', 'DROP' and 'LITERAL': FAIL will throw exception when encountering map with null key. DROP will drop null key entries for map data. LITERAL will replace null key with string literal. The string literal is defined by maxwell-json.map-null-key.literal option.
maxwell-json.map-null-key.literal	No	'null'	String	String literal to replace null key when maxwell-json.map-null-key.mode is LITERAL .

Supported Connectors

- Kafka

Example

Use Kafka to send data and output the data to print.

Step 1 Create a datasource connection for the communication with the VPC and subnet where Kafka locates and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka IP address. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Create a Flink OpenSource SQL job and select Flink 1.12. Copy the following statement and submit the job:

```
create table kafkaSource(  
  id bigint,  
  name string,  
  description string,  
  weight DECIMAL(10, 2)  
) with (  
  'connector' = 'kafka',  
  'topic' = '<yourTopic>',  
  'properties.group.id' = '<yourGroupId>',  
  'properties.bootstrap.servers' =  
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'maxwell-json'  
);  
create table printSink(  
  id bigint,  
  name string,  
  description string,  
  weight DECIMAL(10, 2)  
) with (  
  'connector' = 'print'  
);  
insert into printSink select * from kafkaSource;
```

Step 3 Insert the following data to the corresponding topic in Kafka:

```
{  
  "database": "test",  
  "table": "e",  
  "type": "insert",  
  "ts": 1477053217,  
  "xid": 23396,  
  "commit": true,  
  "position": "master.000006:800911",  
  "server_id": 23042,  
  "thread_id": 108,  
  "primary_key": [1, "2016-10-21 05:33:37.523000"],  
  "primary_key_columns": ["id", "c"],  
  "data": {  
    "id": 111,  
    "name": "scooter",  
    "description": "Big 2-wheel scooter",  
    "weight": 5.15  
  },  
  "old": {  
    "weight": 5.18  
  }  
}
```

Step 4 View the output through either of the following methods:

- Method 1: Locate the job and click **More > FlinkUI**. Choose **Task Managers > Stdout**.

- Method 2: If you allow DLI to save job logs in OBS, view the output in the **taskmanager.out** file.

```
+I(111,scooter,Big 2-wheel scooter,5.15)
```

----End

1.3.4.8 Raw

Function

The raw format allows you to read and write raw (byte based) values as a single column.

Note: This format encodes null values as **null** of the **byte[]** type. This may have limitation when used in **upsert-kafka**, because **upsert-kafka** treats null values as a tombstone message (DELETE on the key). Therefore, we recommend avoiding using **upsert-kafka** connector and the **raw** format as a **value.format** if the field can have a null value.

The raw format connector is built-in, no additional dependencies are required.

Parameters

Table 1-40

Parameter	Mandatory	Default Value	Type	Description
format	Yes	None	String	Format to be used. Set this parameter to raw .
raw.charset	No	UTF-8	String	Charset to encode the text string.
raw.endianness	No	big-endian	String	Endianness to encode the bytes of numeric value. Valid values are big-endian and little-endian . You can search for endianness for more details.

Supported Connectors

- Kafka
- UpsertKafka

Example

Use Kafka to send data and output the data to print.

Step 1 Create a datasource connection for the communication with the VPC and subnet where Kafka locates and bind the connection to the queue. Set a security group and inbound rule to allow access of the queue and test the connectivity of the queue using the Kafka IP address. For example, locate a general-purpose queue where the job runs and choose **More > Test Address Connectivity** in the **Operation** column. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Create a Flink OpenSource SQL job and select Flink 1.12. Copy the following statement and submit the job:

```
create table kafkaSource(  
  log string  
) with (  
  'connector' = 'kafka',  
  'topic' = '<yourTopic>',  
  'properties.group.id' = '<yourGroupId>',  
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'raw'  
);  
create table printSink(  
  log string  
) with (  
  'connector' = 'print'  
);  
insert into printSink select * from kafkaSource;
```

Step 3 Insert the following data to the corresponding topic in Kafka:

```
47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0" 200 5316 "https://domain.com/?  
p=1" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119  
Safari/537.36" "2.75"
```

Step 4 View the output through either of the following methods:

- Method 1: Locate the job and click **More > FlinkUI**. Choose **Task Managers > Stdout**.
- Method 2: If you allow DLI to save job logs in OBS, view the output in the **taskmanager.out** file.

```
+I(47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0"2005316"https://domain.com/?  
p=1"  
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/  
537.36" "2.75")
```

----End

1.4 DML Syntax

1.4.1 SELECT

SELECT

Syntax

```
SELECT [ ALL | DISTINCT ]  
{ * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ]
```

Description

SELECT is used to select data from a table.

ALL indicates that all results are returned.

DISTINCT indicates that the duplicated results are removed.

Precautions

- The to-be-queried table must exist. Otherwise, an error is reported.
- WHERE is used to specify the search condition, which can be the arithmetic operator, relational operator, or logical operator.
- GROUP BY is used to specify the grouping field, which can be one or more multiple fields.

Example

Select the order which contains more than 3 pieces of data.

```
insert into temp SELECT * FROM Orders WHERE units > 3;
```

Insert a group of constant data.

```
insert into temp select 'Lily', 'male', 'student', 17;
```

WHERE

Syntax

```
SELECT { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]
```

Description

This clause is used to filter the query results using the WHERE clause.

Precautions

- The to-be-queried table must exist.
- WHERE filters the records that do not meet the requirements.

Example

Search orders which contain more than 3 pieces and fewer than 10 pieces of data.

```
insert into temp SELECT * FROM Orders  
WHERE units > 3 and units < 10;
```

HAVING

Function

This clause is used to search for the query results that meet the search condition.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ]
```

Description

Generally, HAVING and GROUP BY are used together. You can use GROUP BY for grouping and then use HAVING for filtering. Arithmetic operations and aggregate functions are supported in the HAVING clause.

Precautions

If the filtering condition is subject to the results of GROUP BY, the HAVING clause, rather than the WHERE clause, must be used for search.

Example

Group the **student** table according to the **name** field and search for the records in which the maximum score is higher than 95 in the group.

```
insert into temp SELECT name, max(score) FROM student  
GROUP BY name  
HAVING max(score) >95;
```

Column-Based GROUP BY

Function

This clause is used to group a table based on columns.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]
```

Description

Column-based GROUP BY can be categorized into single-column GROUP BY and multi-column GROUP BY.

- Single-column GROUP BY indicates that the GROUP BY clause contains only one column.
- Multi-column GROUP BY indicates that the GROUP BY clause contains multiple columns. The table will be grouped according to all fields in the GROUP BY clause. The records whose fields are the same are grouped into one group.

Precautions

GroupBy generates update results in the stream processing table.

Example

Group the **student** table according to the score and name fields and return the grouping results.

```
insert into temp SELECT name,score, max(score) FROM student
GROUP BY name,score;
```

Expression-Based GROUP BY

Function

This clause is used to group streams according to expressions.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
```

Description

groupItem can have one or more fields. The fields can be called by string functions, but cannot be called by aggregate functions.

Precautions

None

Example

Use the substring function to obtain the character string from the name field, group the **student** table according to the obtained character string, and return each sub character string and the number of records.

```
insert into temp SELECT substring(name,6),count(name) FROM student
GROUP BY substring(name,6);
```

Grouping sets, Rollup, Cube

Function

- The GROUP BY GROUPING SETS generates a result set equivalent to that generated by multiple simple GROUP BY UNION ALL statements. Using GROUPING SETS is more efficient.
- The ROLLUP and CUBE generate multiple groups based on certain rules and then collect statistics by group.
- The result set generated by CUBE contains all the combinations of values in the selected columns.
- The result set generated by ROLLUP contains the combinations of a certain layer structure in the selected columns.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY groupingItem]
```

Description

Values of **groupingItem** can be **Grouping sets(columnName [, columnName]*), Rollup(columnName [, columnName]*), and Cube(columnName [, columnName]*)**.

Precautions

None

Example

Return the results generated based on **user** and **product**.

```
INSERT INTO temp SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product));
```

GROUP BY Using HAVING

Function

This clause filters a table after grouping it using the HAVING clause.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ]
```

Description

Generally, HAVING and GROUP BY are used together. You can use GROUP BY for grouping and the HAVING for filtering.

Precautions

- If the filtering condition is subject to the results of GROUP BY, the HAVING clause, rather than the WHERE clause, must be used for search. HAVING and GROUP BY are used together. Use GROUP BY for grouping and the HAVING for filtering.
- Fields used in HAVING, except for those used for aggregate functions, must exist in GROUP BY.
- The arithmetic operation and aggregate function are supported by the HAVING clause.

Example

Group the **transactions** by **num**, use the HAVING clause to search for the records in which the maximum value derived from multiplying **price** with **amount** is higher than 5000, and return the filtered results.

```
insert into temp SELECT num, max(price*amount) FROM transactions
WHERE time > '2016-06-01'
GROUP BY num
HAVING max(price*amount)>5000;
```

1.4.2 Set Operations

Union/Union ALL/Intersect/Except

Syntax

```
query UNION [ ALL ] | Intersect | Except query
```

Description

- UNION is used to return the union set of multiple query results.
- INTERSECT is used to return the intersection of multiple query results.
- EXCEPT is used to return the difference set of multiple query results.

Precautions

- Set operations join tables from head to tail under certain conditions. The quantity of columns returned by each SELECT statement must be the same. Column types must be the same. Column names can be different.
- By default, UNION takes only distinct records while UNION ALL does not remove duplicates from the result.

Example

Output distinct records found in either Orders1 and Orders2 tables.

```
insert into temp SELECT * FROM Orders1  
UNION SELECT * FROM Orders2;
```

IN

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
WHERE column_name IN (value (, value)* ) | query
```

Description

The IN operator allows multiple values to be specified in the WHERE clause. It returns true if the expression exists in the given table subquery.

Precautions

The subquery table must consist of a single column, and the data type of the column must be the same as that of the expression.

Example

Return **user** and **amount** information of the products in **NewProducts** of the **Orders** table.

```
insert into temp SELECT user, amount  
FROM Orders  
WHERE product IN (  
    SELECT product FROM NewProducts  
);
```

1.4.3 Window

GROUP WINDOW

Description

Group Window is defined in GROUP BY. One record is generated from each group. Group Window involves the following functions:

- Array functions

Table 1-41 Array functions

Grouping Window Function	Description
TUMBLE(time_attr, interval)	Defines a tumbling time window. A tumbling time window assigns rows to non-overlapping, continuous windows with a fixed duration (interval). For example, a tumbling window of 5 minutes groups rows in 5 minutes intervals. Tumbling windows can be defined on event-time (stream + batch) or processing-time (stream).
HOP(time_attr, interval, interval)	Defines a hopping time window (called sliding window in the Table API). A hopping time window has a fixed duration (second interval parameter) and hops by a specified hop interval (first interval parameter). If the hop interval is smaller than the window size, hopping windows are overlapping. Thus, rows can be assigned to multiple windows. For example, a hopping window of 15 minutes size and 5 minute hop interval assigns each row to 3 different windows of 15 minute size, which are evaluated in an interval of 5 minutes. Hopping windows can be defined on event-time (stream + batch) or processing-time (stream).
SESSION(time_attr, interval)	Defines a session time window. Session time windows do not have a fixed duration but their bounds are defined by a time interval of inactivity, that is, a session window is closed if no event appears for a defined gap period. For example a session window with a 30 minute gap starts when a row is observed after 30 minutes inactivity (otherwise the row would be added to an existing window) and is closed if no row is added within 30 minutes. Session windows can work on event-time (stream + batch) or processing-time (stream).

Notes:

 **CAUTION**

In streaming mode, the **time_attr** argument of the group window function must refer to a valid time attribute that specifies the processing time or event time of rows.

- **event-time**: The type is timestamp(3).
- **processing-time**: No need to specify the type.

In batch mode, the **time_attr** argument of the group window function must be an attribute of type timestamp.

- Window auxiliary functions
The start and end timestamps of group windows as well as time attributes can be selected with the following auxiliary functions.

Table 1-42 Window auxiliary functions

Auxiliary Function	Description
TUMBLE_START(time_attr, interval) HOP_START(time_attr, interval, interval) SESSION_START(time_attr, interval)	Returns the timestamp of the inclusive lower bound of the corresponding tumbling, hopping, or session window.
TUMBLE_END(time_attr, interval) HOP_END(time_attr, interval, interval) SESSION_END(time_attr, interval)	Returns the timestamp of the exclusive upper bound of the corresponding tumbling, hopping, or session window. Note: The exclusive upper bound timestamp cannot be used as a rowtime attribute in subsequent time-based operations, such as interval joins and group window or over window aggregations.
TUMBLE_ROWTIME(time_attr, interval) HOP_ROWTIME(time_attr, interval, interval) SESSION_ROWTIME(time_attr, interval)	Returns the timestamp of the inclusive upper bound of the corresponding tumbling, hopping, or session window. The resulting attribute is a rowtime attribute that can be used in subsequent time-based operations such as interval joins and group window or over window aggregations.

Auxiliary Function	Description
TUMBLE_PROCTIME(time_attr, interval) HOP_PROCTIME(time_attr, interval, interval) SESSION_PROCTIME(time_attr, interval)	Returns a proctime attribute that can be used in subsequent time-based operations such as interval joins and group window or over window aggregations.

Note: Auxiliary functions must be called with exactly same arguments as the group window function in the GROUP BY clause.

Example

```
// Calculate the SUM every day (event time).
insert into temp SELECT name,
  TUMBLE_START(ts, INTERVAL '1' DAY) as wStart,
  SUM(amount)
FROM Orders
GROUP BY TUMBLE(ts, INTERVAL '1' DAY), name;

// Calculate the SUM every day (processing time).
insert into temp SELECT name,
  SUM(amount)
FROM Orders
GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;

// Calculate the SUM over the recent 24 hours every hour (event time).
insert into temp SELECT product,
  SUM(amount)
FROM Orders
GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

// Calculate the SUM of each session and an inactive interval every 12 hours (event time).
insert into temp SELECT name,
  SESSION_START(ts, INTERVAL '12' HOUR) AS sStart,
  SESSION_END(ts, INTERVAL '12' HOUR) AS sEnd,
  SUM(amount)
FROM Orders
GROUP BY SESSION(ts, INTERVAL '12' HOUR), name;
```

TUMBLE WINDOW Extension

Function

The extension functions of the DLI tumbling window are as follows:

- A tumbling window is triggered periodically to reduce latency.
Before the tumbling window ends, the window can be periodically triggered based on the configured frequency. The compute result from the start to the current time is output, which does not affect the final output. The latest result can be viewed in each period before the window ends.
- Data accuracy is improved.
You can set a latency for the end of the window. The output of the window is updated according to the configured latency each time a piece of late data reaches.

Precautions

- If you use the INSERT statement to write results to a sink, it must support the upsert mode. Ensure that the result table supports upsert operations and the primary key is defined.
- Latency settings only take effect for event time and not for proctime.
- Auxiliary functions must be called with the same parameters as the grouping window functions in the GROUP BY clause.
- If event time is used, watermark must be used. The code is as follows (**order_time** is identified as the event time column and watermark is set to 3 seconds):

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  watermark for order_time as order_time - INTERVAL '3' SECOND  
) WITH (  
  'connector' = 'kafka',  
  'topic' = '<yourTopic>',  
  'properties.bootstrap.servers' = '<yourKafka>:<port>',  
  'properties.group.id' = '<yourGroupId>',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
);
```

- If the proctime is used, you need to use the computed column. The code is as follows (**proc** is the processing time column):

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  proc as proctime()  
) WITH (  
  'connector' = 'kafka',  
  'topic' = '<yourTopic>',  
  'properties.bootstrap.servers' = '<yourKafka>:<port>',  
  'properties.group.id' = '<yourGroupId>',  
  'scan.startup.mode' = 'latest-offset',  
  'format' = 'json'  
);
```

Syntax

```
TUMBLE(time_attr, window_interval, period_interval, lateness_interval)
```

Example

The current time attribute column is **testtime**, the window interval is 10 seconds, and the latency is 10 seconds.

```
TUMBLE(testtime, INTERVAL '10' SECOND, INTERVAL '10' SECOND, INTERVAL '10' SECOND)
```

Description

Table 1-43 Parameters

Parameter	Description	Format
time_attribute	Event time or processing time attribute column <ul style="list-style-type: none"> • event-time: The type is timestamp(3). • processing-time: No need to specify the type. 	-
window_interval	Duration of the window	<ul style="list-style-type: none"> • Format 1: INTERVAL '10' SECOND The window interval is 10 seconds. You can change the value as needed. • Format 2: INTERVAL '10' MINUTE The window interval is 10 minutes. You can change the value as needed. • Format 3: INTERVAL '10' DAY The window interval is 10 days. You can change the value as needed.
period_interval	Frequency of periodic triggering within the window range. That is, before the window ends, the output result is updated at an interval specified by period_interval from the time when the window starts. If this parameter is not set, the periodic triggering policy is not used by default.	
lateness_interval	Time to postpone the end of the window. The system continues to collect the data that reaches the window within lateness_interval after the window ends. The output is updated for each data that reaches the window within lateness_interval . NOTE If the time window is for processing time, lateness_interval does not take effect.	

 **NOTE**

Values of **period_interval** and **lateness_interval** cannot be negative numbers.

- If **period_interval** is set to **0**, periodic triggering is disabled for the window.
- If **lateness_interval** is set to **0**, the latency after the window ends is disabled.
- If neither of the two parameters is set, both periodic triggering and latency are disabled and only the regular tumbling window functions are available .
- If only the latency function needs to be used, set period_interval **INTERVAL '0' SECOND**.

Auxiliary Functions

Table 1-44 Auxiliary function

Auxiliary Function	Description
TUMBLE_START(time_attr, window_interval, period_interval, lateness_interval)	Returns the timestamp of the inclusive lower bound of the corresponding tumbling window.
TUMBLE_END(time_attr, window_interval, period_interval, lateness_interval)	Returns the timestamp of the exclusive upper bound of the corresponding tumbling window.

Example

1. The Kafka is used as the data source table containing the order information, and the JDBC is used as the data result table for statistics on the number of orders settled by a user within 30 seconds. The order ID and window opening time are used as primary keys to collect result statistics in real time to JDBC.

Step 1 Create a datasource connection for the communication with the VPC and subnet where MySQL and Kafka locate and bind the connection to the queue. Set an inbound rule for the security group to allow access of the queue, and test the connectivity of the queue using the MySQL and Kafka addresses. If the connection is successful, the datasource is bound to the queue. Otherwise, the binding fails.

Step 2 Run the following statement to create the **order_count** table in the MySQL Flink database:

```
CREATE TABLE `flink`.`order_count` (  
  `user_id` VARCHAR(32) NOT NULL,  
  `window_start` TIMESTAMP NOT NULL,  
  `window_end` TIMESTAMP NULL,  
  `total_num` BIGINT UNSIGNED NULL,  
  PRIMARY KEY (`user_id`, `window_start`)  
) ENGINE = InnoDB  
  DEFAULT CHARACTER SET = utf8mb4  
  COLLATE = utf8mb4_general_ci;
```

Step 3 Create a Flink OpenSource SQL job and submit the job. In this example, the window size is 30 seconds, the triggering period is 10 seconds, and the latency is 5 seconds. That is, if the result is updated before the window ends, the intermediate result will be output every 10 seconds. After the watermark is reached and the window ends, the data whose event time is within 5 seconds of the watermark will still be processed and counted in the current window. If the event time exceeds 5 seconds of the watermark, the data will be discarded.

```
CREATE TABLE orders (  
  order_id string,  
  order_channel string,  
  order_time timestamp(3),  
  pay_amount double,  
  real_pay double,  
  pay_time string,  
  user_id string,  
  user_name string,  
  area_id string,  
  watermark for order_time as order_time - INTERVAL '3' SECOND  
) WITH (  
  'connector' = 'kafka',  
  'topic' = '<yourTopic>',  
  'properties.bootstrap.servers' = '<yourKafka>:<port>',
```

```
'properties.group.id' = '<yourGroupId>',
'scan.startup.mode' = 'latest-offset',
'format' = 'json'
);

CREATE TABLE jdbcSink (
  user_id string,
  window_start timestamp(3),
  window_end timestamp(3),
  total_num BIGINT,
  primary key (user_id, window_start) not enforced
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://<yourMySQL>:3306/flink',
  'table-name' = 'order_count',
  'username' = '<yourUserName>',
  'password' = '<yourPassword>',
  'sink.buffer-flush.max-rows' = '1'
);

insert into jdbcSink select
  order_id,
  TUMBLE_START(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5' SECOND),
  TUMBLE_END(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5' SECOND),
  COUNT(*) from orders
  GROUP BY user_id, TUMBLE(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5'
SECOND);
```

Step 4 Insert data to Kafka. Assume that orders are settled at different time and the order data at 10:00:13 arrives late.

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000002", "order_channel":"webShop", "order_time":"2021-03-24 10:00:20",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000003", "order_channel":"webShop", "order_time":"2021-03-24 10:00:33",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000004", "order_channel":"webShop", "order_time":"2021-03-24 10:00:13",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

Step 5 Run the following statement in the MySQL database to view the output result. The final result is displayed as follows because the periodic output result cannot be collected:

```
select * from order_count
user_id  window_start  window_end  total_num
0001    2021-03-24 10:00:00 2021-03-24 10:00:30 3
0001    2021-03-24 10:00:30 2021-03-24 10:01:00 1
```

----End

OVER WINDOW

The difference between Over Window and Group Window is that one record is generated from one row in Over Window.

Syntax

```
SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
```

```
ROWS
BETWEEN (UNBOUNDED|rowCount) PRECEDING AND CURRENT ROW FROM TABLENAME

SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  RANGE
  BETWEEN (UNBOUNDED|timeInterval) PRECEDING AND CURRENT ROW FROM TABLENAME
```

Description

Table 1-45 Parameter description

Parameter	Description
PARTITION BY	Indicates the primary key of the specified group. Each group separately performs calculation.
ORDER BY	Indicates the processing time or event time as the timestamp for data.
ROWS	Indicates the count window.
RANGE	Indicates the time window.

Precautions

- All aggregates must be defined in the same window, that is, in the same partition, sort, and range.
- Currently, only windows from PRECEDING (unbounded or bounded) to CURRENT ROW are supported. The range described by FOLLOWING is not supported.
- ORDER BY must be specified for a single time attribute.

Example

```
// Calculate the count and total number from syntax rules enabled to now (in proctime).
insert into temp SELECT name,
  count(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt1,
  sum(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt2
FROM Orders;

// Calculate the count and total number of the recent four records (in proctime).
insert into temp SELECT name,
  count(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) as cnt1,
  sum(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) as cnt2
FROM Orders;

// Calculate the count and total number last 60s (in eventtime). Process the events based on event time,
which is the timeattr field in Orders.
insert into temp SELECT name,
  count(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND PRECEDING AND CURRENT ROW) as cnt1,
  sum(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND PRECEDING AND CURRENT ROW) as cnt2
FROM Orders;
```

1.4.4 JOIN

Equi-join

Syntax

```
FROM tableExpression INNER | LEFT | RIGHT | FULL JOIN tableExpression  
ON value11 = value21 [ AND value12 = value22]
```

Precautions

- Currently, only equi-joins are supported, for example, joins that have at least one conjunctive condition with an equality predicate. Arbitrary cross or theta joins are not supported.
- Tables are joined in the order in which they are specified in the FROM clause. Make sure to specify tables in an order that does not yield a cross join (Cartesian product), which are not supported and would cause a query to fail.
- For streaming queries the required state to compute the query result might grow infinitely depending on the type of aggregation and the number of distinct grouping keys. Provide a query configuration with valid retention interval to prevent excessive state size.

Example

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

Time-windowed Join

Function

Each piece of data in a stream is joined with data in different time zones in another stream.

Syntax

```
from t1 JOIN t2 ON t1.key = t2.key AND TIMEBOUND_EXPRESSION
```

Description

TIMEBOUND_EXPRESSION can be in either of the following formats:

- L.time between LowerBound(R.time) and UpperBound(R.time)
- R.time between LowerBound(L.time) and UpperBound(L.time)
- Comparison expression with the time attributes (L.time/R.time)

Precautions

A time window join requires at least one equi join predicate and a join condition that limits the time of both streams.

For example, use two range predicates (<, <=, >=, or >), a BETWEEN predicate, or an equal predicate that compares the same type of time attributes (such as processing time and event time) in two input tables.

For example, the following predicate is a valid window join condition:

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

Example

Join all orders shipped within 4 hours with their associated shipments.

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

Expanding arrays into a relation

Precautions

This clause is used to return a new row for each element in the given array. Unnesting WITH ORDINALITY is not yet supported.

Example

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

User-Defined Table Functions

Function

This clause is used to join a table with the results of a table function. Each row of the left (outer) table is joined with all rows produced by the corresponding call of the table function.

Precautions

A left outer join against a lateral table requires a TRUE literal in the ON clause.

Example

The row of the left (outer) table is dropped, if its table function call returns an empty result.

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

If a table function call returns an empty result, the corresponding outer row is preserved, and the result padded with null values.

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

Join Temporal Table Function

Function

Precautions

Currently only inner join and left outer join with temporal tables are supported.

Example

Assuming Rates is a temporal table function, the join can be expressed in SQL as follows:

```
SELECT
  o_amount, r_rate
FROM
  Orders,
  LATERAL TABLE (Rates(o_proctime))
WHERE
  r_currency = o_currency;
```

Join Temporal Tables

Function

This clause is used to join the Temporal table.

Syntax

```
SELECT column-names
FROM table1 [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.proctime [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

Description

- **table1.proctime** indicates the processing time attribute (computed column) of **table1**.
- **FOR SYSTEM_TIME AS OF table1.proctime** indicates that when the records in the left table are joined with the dimension table on the right, only the snapshot data is used for matching the current processing time dimension table.

Precautions

Only inner and left joins are supported for temporal tables with processing time attributes.

Example

LatestRates is a dimension table (such as HBase table) that is materialized with the latest rate.

```
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;
```

1.4.5 OrderBy & Limit

OrderBy

Function

This clause is used to sort data in ascending order on a time attribute.

Precautions

Currently, only sorting by time attribute is supported.

Example

Sort data in ascending order on the time attribute.

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

Limit

Function

This clause is used to constrain the number of rows returned.

Precautions

This clause is used in conjunction with ORDER BY to ensure that the results are deterministic.

Example

```
SELECT *  
FROM Orders  
ORDER BY orderTime  
LIMIT 3;
```

1.4.6 Top-N

Function

Top-N queries ask for the N smallest or largest values ordered by columns. Both smallest and largest values sets are considered Top-N queries. Top-N queries are useful in cases where the need is to display only the N bottom-most or the N top-most records from batch/streaming table on a condition.

Syntax

```
SELECT [column_list]  
FROM (  
  SELECT [column_list],  
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]  
    ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum  
  FROM table_name)  
WHERE rownum <= N [AND conditions]
```

Description

- ROW_NUMBER(): Allocate a unique and consecutive number to each line starting from the first line in the current partition. Currently, we only support ROW_NUMBER as the over window function. In the future, we will support RANK() and DENSE_RANK().
- PARTITION BY col1[, col2...]: Specifies the partition columns. Each partition will have a Top-N result.

- **ORDER BY col1 [asc|desc][, col2 [asc|desc]...]:** Specifies the ordering columns. The ordering directions can be different on different columns.
- **WHERE rownum <= N:** The rownum <= N is required for Flink to recognize this query is a Top-N query. The N represents the N smallest or largest records will be retained.
- **[AND conditions]:** It is free to add other conditions in the where clause, but the other conditions can only be combined with rownum <= N using AND conjunction.

Precautions

- The TopN query is Result Updating.
- Flink SQL will sort the input data stream according to the order key,
- so if the top N records have been changed, the changed ones will be sent as retraction/update records to downstream.
- If the top N records need to be stored in external storage, the result table should have the same unique key with the Top-N query.

Example

This is an example to get the top five products per category that have the maximum sales in realtime.

```
SELECT *
FROM (
  SELECT *,
  ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) as row_num
  FROM ShopSales)
WHERE row_num <= 5;
```

1.4.7 Deduplication

Function

Deduplication removes rows that duplicate over a set of columns, keeping only the first one or the last one.

Syntax

```
SELECT [column_list]
FROM (
  SELECT [column_list],
  ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
  ORDER BY time_attr [asc|desc]) AS rownum
  FROM table_name)
WHERE rownum = 1
```

Description

- **ROW_NUMBER():** Assigns a unique, sequential number to each row, starting with one.
- **PARTITION BY col1[, col2...]:** Specifies the partition columns, i.e. the deduplicate key.

- ORDER BY time_attr [asc|desc]: Specifies the ordering column, it must be a time attribute. Currently Flink supports proctime only. Ordering by ASC means keeping the first row, ordering by DESC means keeping the last row.
- WHERE rownum = 1: The rownum = 1 is required for Flink to recognize this query is deduplication.

Precautions

None

Example

The following examples show how to remove duplicate rows on **order_id**. The proctime is an event time attribute.

```
SELECT order_id, user, product, number
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY proctime ASC) as row_num
  FROM Orders)
WHERE row_num = 1;
```

1.5 Functions

1.5.1 User-Defined Functions (UDFs)

Overview

DLI supports the following three types of user-defined functions (UDFs):

- Regular UDF: takes in one or more input parameters and returns a single result.
- User-defined table-generating function (UDTF): takes in one or more input parameters and returns multiple rows or columns.
- User-defined aggregate function (UDAF): aggregates multiple records into one value.

NOTE

- UDFs can only be used in dedicated queues.
- **Currently, Python is not supported for programming UDFs, UDTFs, and UDAFs.**

POM Dependency

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-table-common</artifactId>
  <version>1.10.0</version>
  <scope>provided</scope>
</dependency>
```

Using UDFs

1. Encapsulate the implemented UDFs into a JAR package and upload the package to OBS.

2. In the navigation pane of the DLI management console, choose **Data Management > Package Management**. On the displayed page, click **Create** and use the JAR package uploaded to OBS to create a package.
3. In the left navigation, choose **Job Management** and click **Flink Jobs**. Locate the row where the target resides and click **Edit** in the **Operation** column to switch to the page where you can edit the job.
4. Click the **Running Parameters** tab of your job, select the UDF JAR and click **Save**.
5. Add the following statement to the SQL statements to use the functions:

```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
```

UDF

The regular UDF must inherit the `ScalarFunction` function and implement the `eval` method. The open and close functions are optional.

Example code

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class UdfScalarFunction extends ScalarFunction {
    private int factor = 12;
    public UdfScalarFunction() {
        this.factor = 12;
    }
    /**
     * (optional) Initialization
     * @param context
     */
    @Override
    public void open(FunctionContext context) {}
    /**
     * Custom logic
     * @param s
     * @return
     */
    public int eval(String s) {
        return s.hashCode() * factor;
    }
    /**
     * Optional
     */
    @Override
    public void close() {}
}
```

Example

```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

The UDTF must inherit the `TableFunction` function and implement the `eval` method. The open and close functions are optional. If the UDTF needs to return multiple columns, you only need to declare the returned value as **Tuple** or **Row**. If **Row** is used, you need to overload the `getResultType` method to declare the returned field type.

Example code

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
```

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class UdfTableFunction extends TableFunction<Row> {
    private Logger log = LoggerFactory.getLogger(TableFunction.class);
    /**
     * (optional) Initialization
     * @param context
     */
    @Override
    public void open(FunctionContext context) {}
    public void eval(String str, String split) {
        for (String s : str.split(split)) {
            Row row = new Row(2);
            row.setField(0, s);
            row.setField(1, s.length());
            collect(row);
        }
    }
    /**
     * Declare the type returned by the function
     * @return
     */
    @Override
    public TypeInformation<Row> getResultType() {
        return Types.ROW(Types.STRING, Types.INT);
    }
    /**
     * Optional
     */
    @Override
    public void close() {}
}
```

Example

The UDTF supports CROSS JOIN and LEFT JOIN. When the UDTF is used, the **LATERAL** and **TABLE** keywords must be included.

- **CROSS JOIN**: does not output the data of a row in the left table if the UDTF does not output the result for the data of the row.
- **LEFT JOIN**: outputs the data of a row in the left table even if the UDTF does not output the result for the data of the row, but pads null with UDTF-related fields.

```
CREATE FUNCTION udtf_test AS 'com.huaweicompany.udf.TableFunction';
// CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);
// LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

UDAF

The UDAF must inherit the AggregateFunction function. You need to create an accumulator for storing the computing result, for example, **WeightedAvgAccum** in the following example code.

Example code

```
public class WeightedAvgAccum {
    public long sum = 0;
    public int count = 0;
}
```

```
import org.apache.flink.table.functions.AggregateFunction;
import java.util.Iterator;
/**
 * The first type variable is the type returned by the aggregation function, and the second type variable is of
 * the Accumulator type.
 * Weighted Average user-defined aggregate function.
 */
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {
    // Initialize the accumulator.
    @Override
    public WeightedAvgAccum createAccumulator() {
        return new WeightedAvgAccum();
    }
    // Return the intermediate computing value stored in the accumulator.
    @Override
    public Long getValue(WeightedAvgAccum acc) {
        if (acc.count == 0) {
            return null;
        } else {
            return acc.sum / acc.count;
        }
    }
    // Update the intermediate computing value according to the input.
    public void accumulate(WeightedAvgAccum acc, long iValue) {
        acc.sum += iValue;
        acc.count += 1;
    }
    // Perform the retraction operation, which is opposite to the accumulate operation.
    public void retract(WeightedAvgAccum acc, long iValue) {
        acc.sum -= iValue;
        acc.count -= 1;
    }
    // Combine multiple accumulator values.
    public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {
        Iterator<WeightedAvgAccum> iter = it.iterator();
        while (iter.hasNext()) {
            WeightedAvgAccum a = iter.next();
            acc.count += a.count;
            acc.sum += a.sum;
        }
    }
    // Reset the intermediate computing value.
    public void resetAccumulator(WeightedAvgAccum acc) {
        acc.count = 0;
        acc.sum = 0L;
    }
}
```

Example

```
CREATE FUNCTION udaf_test AS 'com.huaweicompany.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

1.5.2 Built-In Functions

1.5.2.1 Mathematical Operation Functions

Relational Operators

All data types can be compared by using relational operators and the result is returned as a BOOLEAN value.

Relationship operators are binary operators. Two compared data types must be of the same type or they must support implicit conversion.

Table 1-46 lists all relational operators supported by Flink SQL.

Table 1-46 Relational Operators

Operator	Returned Data Type	Description
A = B	BOOLEAN	If A is equal to B, then TRUE is returned. Otherwise, FALSE is returned. This operator is used for value assignment.
A <> B	BOOLEAN	If A is not equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned. This operator follows the standard SQL syntax.
A < B	BOOLEAN	If A is less than B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A <= B	BOOLEAN	If A is less than or equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A > B	BOOLEAN	If A is greater than B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A >= B	BOOLEAN	If A is greater than or equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A IS NULL	BOOLEAN	If A is NULL , then TRUE is returned. Otherwise, FALSE is returned.
A IS NOT NULL	BOOLEAN	If A is not NULL , then TRUE is returned. Otherwise, FALSE is returned.
A IS DISTINCT FROM B	BOOLEAN	If A is not equal to B, TRUE is returned. NULL indicates A equals B.
A IS NOT DISTINCT FROM B	BOOLEAN	If A is equal to B, TRUE is returned. NULL indicates A equals B.

Operator	Returned Data Type	Description
A BETWEEN [ASYMMETRIC SYMMETRIC] B AND C	BOOLEAN	<p>If A is greater than or equal to B but less than or equal to C, TRUE is returned.</p> <ul style="list-style-type: none"> ASYMMETRIC: indicates that B and C are location-related. For example, "A BETWEEN ASYMMETRIC B AND C" is equivalent to "A BETWEEN B AND C". SYMMETRIC: indicates that B and C are not location-related. For example, "A BETWEEN SYMMETRIC B AND C" is equivalent to "A BETWEEN B AND C OR (A BETWEEN C AND B)".
A NOT BETWEEN B [ASYMMETRIC SYMMETRIC] AND C	BOOLEAN	<p>If A is less than B or greater than C, TRUE is returned.</p> <ul style="list-style-type: none"> ASYMMETRIC: indicates that B and C are location-related. For example, "A NOT BETWEEN ASYMMETRIC B AND C" is equivalent to "A NOT BETWEEN B AND C". SYMMETRIC: indicates that B and C are not location-related. For example, "A NOT BETWEEN SYMMETRIC B AND C" is equivalent to "(A NOT BETWEEN B AND C) OR (A NOT BETWEEN C AND B)".
A LIKE B [ESCAPE C]	BOOLEAN	If A matches pattern B, TRUE is returned. The escape character C can be defined as required.
A NOT LIKE B [ESCAPE C]	BOOLEAN	If A does not match pattern B, TRUE is returned. The escape character C can be defined as required.
A SIMILAR TO B [ESCAPE C]	BOOLEAN	If A matches regular expression B, TRUE is returned. The escape character C can be defined as required.
A NOT SIMILAR TO B [ESCAPE C]	BOOLEAN	If A does not match regular expression B, TRUE is returned. The escape character C can be defined as required.
value IN (value [, value]*)	BOOLEAN	If the value is equal to any value in the list, TRUE is returned.
value NOT IN (value [, value]*)	BOOLEAN	If the value is not equal to any value in the list, TRUE is returned.
EXISTS (sub-query)	BOOLEAN	If sub-query returns at least one row, TRUE is returned.

Operator	Returned Data Type	Description
value IN (sub-query)	BOOLEAN	If value is equal to a row returned by subquery, TRUE is returned.
value NOT IN (sub-query)	BOOLEAN	If value is not equal to a row returned by subquery, TRUE is returned.

Precautions

- Values of the double, real, and float types may be different in precision. The equal sign (=) is not recommended for comparing two values of the double type. You are advised to obtain the absolute value by subtracting these two values of the double type and determine whether they are the same based on the absolute value. If the absolute value is small enough, the two values of the double data type are regarded equal. For example:
`abs(0.9999999999 - 1.0000000000) < 0.000000001` //The precision decimal places of 0.9999999999 and 1.0000000000 are 10, while the precision decimal place of 0.000000001 is 9. Therefore, 0.9999999999 can be regarded equal to 1.0000000000.
- Comparison between data of the numeric type and character strings is allowed. During comparison using relational operators, including >, <, ≤, and ≥, data of the string type is converted to numeric type by default. No characters other than numeric characters are allowed.
- Character strings can be compared using relational operators.

Logical Operators

Common logical operators are AND, OR, and NOT. Their priority order is NOT > AND > OR.

Table 1-47 lists the calculation rules. A and B indicate logical expressions.

Table 1-47 Logical Operators

Operator	Returned Data Type	Description
A OR B	BOOLEAN	If A or B is TRUE, TRUE is returned. Three-valued logic is supported.
A AND B	BOOLEAN	If both A and B are TRUE, TRUE is returned. Three-valued logic is supported.
NOT A	BOOLEAN	If A is not TRUE, TRUE is returned. If A is UNKNOWN, UNKNOWN is returned.
A IS FALSE	BOOLEAN	If A is TRUE, TRUE is returned. If A is UNKNOWN, FALSE is returned.
A IS NOT FALSE	BOOLEAN	If A is not FALSE, TRUE is returned. If A is UNKNOWN, TRUE is returned.

Operator	Returned Data Type	Description
A IS TRUE	BOOLEAN	If A is TRUE, TRUE is returned. If A is UNKNOWN, FALSE is returned.
A IS NOT TRUE	BOOLEAN	If A is not TRUE, TRUE is returned. If A is UNKNOWN, TRUE is returned.
A IS UNKNOWN	BOOLEAN	If A is UNKNOWN, TRUE is returned.
A IS NOT UNKNOWN	BOOLEAN	If A is not UNKNOWN, TRUE is returned.

Precautions

Only data of the Boolean type can be used for calculation using logical operators. Implicit type conversion is not supported.

Arithmetic Operators

Arithmetic operators include binary operators and unary operators, for all of which, the returned results are of the numeric type. [Table 1-48](#) lists arithmetic operators supported by Flink SQL.

Table 1-48 Arithmetic Operators

Operator	Returned Data Type	Description
+ numeric	All numeric types	Returns numbers.
- numeric	All numeric types	Returns negative numbers.
A + B	All numeric types	A plus B. The result type is associated with the operation data type. For example, if floating-point number is added to an integer, the result will be a floating-point number.

Operator	Returned Data Type	Description
A - B	All numeric types	A minus B. The result type is associated with the operation data type.
A * B	All numeric types	Multiply A and B. The result type is associated with the operation data type.
A / B	All numeric types	Divide A by B. The result is a number of the double type (double-precision number).
POWER(A, B)	All numeric types	Returns the value of A raised to the power B.
ABS(numeric)	All numeric types	Returns the absolute value of a specified value.
MOD(A, B)	All numeric types	Returns the remainder (modulus) of A divided by B. A negative value is returned only when A is a negative value.
SQRT(A)	All numeric types	Returns the square root of A.
LN(A)	All numeric types	Returns the nature logarithm of A (base e).
LOG10(A)	All numeric types	Returns the base 10 logarithms of A.
LOG2(A)	All numeric types	Returns the base 2 logarithm of A.

Operator	Returned Data Type	Description
LOG(B) LOG(A, B)	All numeric types	When called with one argument, returns the natural logarithm of B. When called with two arguments, this function returns the logarithm of B to the base A. B must be greater than 0 and A must be greater than 1.
EXP(A)	All numeric types	Return the value of e raised to the power of a .
CEIL(A) CEILING(A)	All numeric types	Return the smallest integer that is greater than or equal to a . For example: <code>ceil(21.2) = 22</code> .
FLOOR(A)	All numeric types	Return the largest integer that is less than or equal to a . For example: <code>floor(21.2) = 21</code> .
SIN(A)	All numeric types	Returns the sine value of A.
COS(A)	All numeric types	Returns the cosine value of A.
TAN(A)	All numeric types	Returns the tangent value of A.
COT(A)	All numeric types	Returns the cotangent value of A.
ASIN(A)	All numeric types	Returns the arc sine value of A.

Operator	Returned Data Type	Description
ACOS(A)	All numeric types	Returns the arc cosine value of A.
ATAN(A)	All numeric types	Returns the arc tangent value of A.
ATAN2(A, B)	All numeric types	Returns the arc tangent of a coordinate (A, B).
COSH(A)	All numeric types	Returns the hyperbolic cosine of A. Return value type is DOUBLE.
DEGREES(A)	All numeric types	Convert the value of a from radians to degrees.
RADIANS(A)	All numeric types	Convert the value of a from degrees to radians.
SIGN(A)	All numeric types	Returns the sign of A. 1 is returned if A is positive. -1 is returned if A is negative. Otherwise, 0 is returned.
ROUND(A, d)	All numeric types	Returns a number rounded to d decimal places for A. For example: round(21.263,2) = 21.26.
PI	All numeric types	Returns the value of pi .
E()	All numeric types	Returns the value of e .

Operator	Returned Data Type	Description
RAND()	All numeric types	Returns a pseudorandom double value in the range [0.0, 1.0)
RAND(A)	All numeric types	Returns a pseudorandom double value in the range [0.0, 1.0) with an initial seed A. Two RAND functions will return identical sequences of numbers if they have the same initial seed.
RAND_INTEGER(A)	All numeric types	Returns a pseudorandom double value in the range [0.0, A)
RAND_INTEGER(A, B)	All numeric types	Returns a pseudorandom double value in the range [0.0, B) with an initial seed A.
UUID()	All numeric types	Returns a UUID string.
BIN(A)	All numeric types	Returns a string representation of integer A in binary format. Returns NULL if A is NULL.
HEX(A) HEX(B)	All numeric types	Returns a string representation of an integer A value or a string B in hex format. Returns NULL if the argument is NULL.
TRUNCATE(A, d)	All numeric types	Returns a number of truncated to d decimal places. Returns NULL if A or d is NULL. Example: truncate (42.345, 2) = 42.340 truncate(42.345) = 42.000
PI()	All numeric types	Returns the value of pi .

Precautions

Data of the string type is not allowed in arithmetic operations.

1.5.2.2 String Functions

Table 1-49 String Functions

Function	Return Type	Description
string1 string2	STRING	Returns the concatenation of string1 and string2.
CHAR_LENGTH(string) CHARACTER_LENGTH(string)	INT	Returns the number of characters in the string.
UPPER(string)	STRING	Returns the string in uppercase.
LOWER(string)	STRING	Returns the string in lowercase.
POSITION(string1 IN string2)	INT	Returns the position (start 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)	STRING	Returns a string that removes leading and/or trailing characters string2 from string1.
LTRIM(string)	STRING	Returns a string that removes the left whitespaces from the specified string. For example, LTRIM(' This is a test String.') returns "This is a test String." .
RTRIM(string)	STRING	Returns a string that removes the right whitespaces from the specified string. For example, RTRIM('This is a test String. ') returns "This is a test String." .
REPEAT(string, integer)	STRING	Returns a string that repeats the base string integer times. For example, REPEAT('This is a test String.', 2) returns "This is a test String.This is a test String." .
REGEXP_REPLACE(string1, string2, string3)	STRING	Returns a string from string1 with all the substrings that match a regular expression string2 consecutively being replaced with string3. For example, REGEXP_REPLACE('foobar', 'oo ar', '') returns "fb" . REGEXP_REPLACE('ab\ab', '\\', 'e') returns "abeab" .

Function	Return Type	Description
OVERLAY(string1 PLACING string2 FROM integer1 [FOR integer2])	STRING	Returns a string that replaces integer2 characters of STRING1 with STRING2 from position integer1. The default value of integer2 is the length of string2. For example, OVERLAY('This is an old string' PLACING ' new' FROM 10 FOR 5) returns "This is a new string" .
SUBSTRING(string FROM integer1 [FOR integer2])	STRING	Returns a substring of the specified string starting from position integer1 with length integer2 (to the end by default). If integer2 is not configured, the substring from integer1 to the end is returned by default.
REPLACE(string1, string2, string3)	STRING	Returns a new string which replaces all the occurrences of string2 with string3 (non-overlapping) from string1. For example, REPLACE('hello world', 'world', 'flink') returns "hello flink" ; REPLACE('ababab', 'abab', 'z') returns "zab" . REPLACE('ab\\ab', '\\', 'e') returns "abeab" .
REGEXP_EXTRACT(string1, string2[, integer])	STRING	Returns a string from string1 which extracted with a specified regular expression string2 and a regex match group index integer. Returns NULL, if the parameter is NULL or the regular expression is invalid. For example, REGEXP_EXTRACT('foothebar', 'foo(?:)(bar)', 2) returns "bar" .
INITCAP(string)	STRING	Returns a new form of STRING with the first character of each word converted to uppercase and the rest characters to lowercase.
CONCAT(string1, string2,...)	STRING	Returns a string that concatenates string1, string2, For example, CONCAT('AA', 'BB', 'CC') returns "AABBCC" .
CONCAT_WS(string1, string2, string3,...)	STRING	Returns a string that concatenates string2, string3, ... with a separator string1. The separator is added between the strings to be concatenated. Returns NULL if string1 is NULL. If other arguments are NULL, this function automatically skips NULL arguments. For example, CONCAT_WS('~', 'AA', NULL, 'BB', 'CC') returns "AA~BB~CC" .

Function	Return Type	Description
LPAD(string1, integer, string2)	STRING	Returns a new string from string1 left-padded with string2 to a length of integer characters. If any argument is NULL, NULL is returned. If integer is negative, NULL is returned. If the length of string1 is shorter than integer, returns string1 shortened to integer characters. For example, LPAD(Symbol,4,Symbol) returns "Symbol hi" . LPAD('hi',1,'?') returns "h" .
RPAD(string1, integer, string2)	STRING	Returns a new string from string1 right-padded with string2 to a length of integer characters. If any argument is NULL, NULL is returned. If integer is negative, NULL is returned. If the length of string1 is shorter than integer, returns string1 shortened to integer characters. For example, RPAD('hi',4,'?') returns "hi???" . RPAD('hi',1,'?') returns "h" .
FROM_BASE64(string)	STRING	Returns the base64-decoded result from string. Returns NULL if string is NULL. For example, FROM_BASE64('aGVsbG8gd29ybGQ=') returns "hello world" .
TO_BASE64(string)	STRING	Returns the base64-encoded result from string; if string is NULL. Returns NULL if string is NULL. For example, TO_BASE64(hello world) returns "aGVsbG8gd29ybGQ=" .
ASCII(string)	INT	Returns the numeric value of the first character of string. Returns NULL if string is NULL. For example, ascii('abc') returns 97 . ascii(CAST(NULL AS VARCHAR)) returns NULL .

Function	Return Type	Description
CHR(integer)	STRING	Returns the ASCII character having the binary equivalent to integer. If integer is larger than 255, we will get the modulus of integer divided by 255 first, and returns CHR of the modulus. Returns NULL if integer is NULL. chr(97) returns a . chr(353) Return a .
DECODE(binary, string)	STRING	Decodes the first argument into a String using the provided character set (one of 'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16'). If either argument is NULL, the result will also be NULL.
ENCODE(string1, string2)	STRING	Encodes the string1 into a BINARY using the provided string2 character set (one of 'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16'). If either argument is NULL, the result will also be NULL.
INSTR(string1, string2)	INT	Returns the position of the first occurrence of string2 in string1. Returns NULL if any argument is NULL.
LEFT(string, integer)	STRING	Returns the leftmost integer characters from the string. Returns EMPTY String if integer is negative. Returns NULL if any argument is NULL.
RIGHT(string, integer)	STRING	Returns the rightmost integer characters from the string. Returns EMPTY String if integer is negative. Returns NULL if any argument is NULL.
LOCATE(string1, string2[, integer])	INT	Returns the position of the first occurrence of string1 in string2 after position integer. Returns 0 if not found. The value of integer defaults to 0 . Returns NULL if any argument is NULL.

Function	Return Type	Description
PARSE_URL(string1, string2[, string3])	STRING	<p>Returns the specified part from the URL.</p> <p>Valid values for string2 include 'HOST', 'PATH', 'QUERY', 'REF', 'PROTOCOL', 'AUTHORITY', 'FILE', and 'USERINFO'.</p> <p>Returns NULL if any argument is NULL.</p> <p>If string2 is QUERY, the key in QUERY can be specified as string3.</p> <p>Example:</p> <p>The <code>parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST')</code> returns 'facebook.com'.</p> <p><code>parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1')</code> returns 'v1'.</p>
REGEXP(string1, string2)	BOOLEAN	<p>Performs a regular expression search on the specified string and returns a BOOLEAN value indicating whether the specified match pattern is found. If it is found, TRUE is returned. string1 indicates the specified string, and string2 indicates the regular expression.</p> <p>Returns NULL if any argument is NULL.</p>
REVERSE(string)	STRING	<p>Returns the reversed string.</p> <p>Returns NULL if any argument is NULL.</p> <p>NOTE Note that backquotes must be added to this function, for example, `REVERSE`.</p>
SPLIT_INDEX(string1, string2, integer1)	STRING	<p>Splits string1 by the delimiter string2, returns the integerth (zero-based) string of the split strings. Returns NULL if integer is negative.</p> <p>Returns NULL if integer is negative.</p> <p>Returns NULL if any argument is NULL.</p>
STR_TO_MAP(string1[, string2, string3])	MAP	<p>Returns a map after splitting the string1 into key/value pairs using delimiters.</p> <p>The default value of string2 is ','.</p> <p>The default value of string3 is '='.</p>
SUBSTR(string[, integer1[, integer2]])	STRING	<p>Returns a substring of string starting from position integer1 with length integer2.</p> <p>If integer2 is not specified, the string is truncated to the end.</p>

Function	Return Type	Description
JSON_VAL(STRING json_string, STRING json_path)	STRING	<p>Returns the value of the specified json_path from the json_string. For details about how to use the functions, see JSON_VAL Function.</p> <p>NOTE The following rules are listed in descending order of priority.</p> <ol style="list-style-type: none"> 1. The two arguments json_string and json_path cannot be NULL. 2. The value of json_string must be a valid JSON string. Otherwise, the function returns NULL. 3. If json_string is an empty string, the function returns an empty string. 4. If json_path is an empty string or the path does not exist, the function returns NULL.

JSON_VAL Function

- Syntax

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

Table 1-50 Parameters

Parameter	Data Types	Description
json_string	STRING	JSON object to be parsed
json_path	STRING	Path expression for parsing the JSON string For the supported expressions, see Table 1-51 .

Table 1-51 Expressions supported

Expression	Description
\$	Root node in the path
[]	Access array elements
*	Array wildcard
.	Access child elements

- Example
 - a. Test input data.
Test the data source kafka. The message content is as follows:

```
{name:James,age:24,sex:male,grade:{math:95,science:[80,85],english:100}}
{name:James,age:24,sex:male,grade:{math:95,science:[80,85],english:100}}
```

b. Use JSON_VAL in SQL statements.

```
CREATE TABLE kafkaSource (
  `message` string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSourceTopic>',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "csv",
  "csv.field-delimiter" = "\u0001",
  "csv.quote-character" = ""
);

CREATE TABLE kafkaSink(
  message1 STRING,
  message2 STRING,
  message3 STRING,
  message4 STRING,
  message5 STRING,
  message6 STRING
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  "format" = "json"
);

insert into kafkaSink select
JSON_VAL(message,""),
JSON_VAL(message,"$.name"),
JSON_VAL(message,"$.grade.science"),
JSON_VAL(message,"$.grade.science[*]"),
JSON_VAL(message,"$.grade.science[1]"),JSON_VAL(message,"$.grade.dddd")
from kafkaSource;
```

c. Check the output result of the Kafka topic in the sink.

```
{"message1":null,"message2":"swq","message3":"[80,85]","message4":"[80,85]","message5":"85"
,"message6":null}
{"message1":null,"message2":null,"message3":null,"message4":null,"message5":null,"message6":
null}
```

1.5.2.3 Temporal Functions

[Table 1-52](#) lists the time functions supported by Flink OpenSource SQL.

Description

Table 1-52 Temporal Functions

Function	Return Type	Description
DATE string	DATE	Parse the date string (yyyy-MM-dd) to a SQL date.
TIME string	TIME	Parse the time string (HH:mm:ss[.fff]) to a SQL time.

Function	Return Type	Description
TIMESTAMP string	TIMESTAMP	Convert the time string into a timestamp. The time string format is yyyy-MM-dd HH:mm:ss[.fff] .
INTERVAL string range	INTERVAL	interval indicates the interval. There are two forms: <ul style="list-style-type: none"> • yyyy-MM for SQL intervals of months. An interval range might be YEAR or YEAR TO MONTH for intervals of months. • dd hh:mm:ss.fff for SQL intervals of milliseconds. An interval range might be DAY, MINUTE, DAY TO HOUR, or DAY TO SECOND. <p>Example:</p> <p>INTERVAL '10 00:00:00.004' DAY TO second indicates that the interval is 10 days and 4 milliseconds.</p> <p>INTERVAL '10' DAY: indicates that the interval is 10 days.</p> <p>INTERVAL '2-10' YEAR TO MONTH indicates that the interval is two years and ten months.</p>
CURRENT_DATE	DATE	Return the SQL date of UTC time zone.
CURRENT_TIME	TIME	Return the SQL time of UTC time zone.
CURRENT_TIMESTAMP	TIMESTAMP	Return the SQL timestamp of UTC time zone.
LOCALTIME	TIME	Return the SQL time of the current time zone.
LOCALTIMESTAMP	TIMESTAMP	Return the SQL timestamp of the current time zone.
EXTRACT(timeintervalunit FROM temporal)	BIGINT	Extract part of the time point or interval. Return the part in the int type. For example, extract the date 2006-06-05 and return 5. EXTRACT(DAY FROM DATE '2006-06-05') returns 5.
YEAR(date)	BIGINT	Return the year from SQL date. For example, YEAR(DATE'1994-09-27') returns 1994 .

Function	Return Type	Description
QUARTER(date)	BIGINT	Return the quarter of a year (an integer between 1 and 4) from SQL date.
MONTH(date)	BIGINT	Return the month of a year (an integer between 1 and 12) from SQL date. For example, MONTH(DATE '1994-09-27') returns 9 .
WEEK(date)	BIGINT	Return the week of a year (an integer between 1 and 53) from SQL date. For example, WEEK(DATE'1994-09-27') returns 39 .
DAYOFYEAR(date)	BIGINT	Returns the day of a year (an integer between 1 and 366) from SQL date. For example, DAYOFYEAR(DATE '1994-09-27') is 270 .
DAYOFMONTH(date)	BIGINT	Return the day of a month (an integer between 1 and 31) from SQL date. For example, DAYOFMONTH(DATE'1994-09-27') returns 27 .
DAYOFWEEK(date)	BIGINT	Return the day of a week (an integer between 1 and 7) from SQL date. Sunday is set to 1 . For example, DAYOFWEEK(DATE'1994-09-27') returns 3 .
HOUR(timestamp)	BIGINT	Returns the hour of a day (an integer between 0 and 23) from SQL timestamp. For example, HOUR(TIMESTAMP '1994-09-27 13:14:15') returns 13 .
MINUTE(timestamp)	BIGINT	Returns the minute of an hour (an integer between 0 and 59) from SQL timestamp. For example, MINUTE(TIMESTAMP '1994-09-27 13:14:15') returns 14 .
SECOND(timestamp)	BIGINT	Returns the second of a minute (an integer between 0 and 59) from SQL timestamp. For example, SECOND(TIMESTAMP '1994-09-27 13:14:15') returns 15 .
FLOOR(timepoint TO timeintervalunit)	TIME	Round a time point down to the given unit. For example, 12:44:00 is returned from FLOOR(TIME '12:44:31' TO MINUTE) .

Function	Return Type	Description
CEIL(timepoint TO timeintervalunit)	TIME	Round a time point up to the given unit. For example, CEIL(TIME '12:44:31' TO MINUTE) returns 12:45:00 .
(timepoint1, temporal1) OVERLAPS (timepoint2, temporal2)	BOOLEAN	Return TRUE if two time intervals defined by (timepoint1, temporal1) and (timepoint2, temporal2) overlap. Example: (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) returns TRUE . (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) returns FALSE .
DATE_FORMAT(timestamp, string)	STRING	Convert timestamp to a value of string in the format specified by the date format string.
TIMESTAMPADD(timeintervalunit, interval, timepoint)	TIMESTAMP/ DATE/ TIME	Return the date and time added to timepoint based on the result of interval and timeintervalunit . For example, TIMESTAMPADD(WEEK, 1, DATE '2003-01-02') returns 2003-01-09 .
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	INT	Return the (signed) number of timepointunit between timepoint1 and timepoint2 . The unit for the interval is given by the first argument, which should be one of the following values: SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR. For example, TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00') returns 1 .
CONVERT_TZ(string1, string2, string3)	TIMESTAMP	Convert a datetime string1 from time zone string2 to time zone string3 . For example, CONVERT_TZ('1970-01-01 00:00:00', 'UTC', 'America/Los_Angeles') returns '1969-12-31 16:00:00' .

Function	Return Type	Description
FROM_UNIXTIME(numeric[, string])	STRING	Return a string representation of the numeric argument (in seconds) in the current time zone. The default string format is YYYY-MM-DD hh:mm:ss. For example, FROM_UNIXTIME(44) returns 1970-01-01 09:00:44 .
UNIX_TIMESTAMP()	BIGINT	Get current Unix timestamp in seconds.
UNIX_TIMESTAMP(string1[, string2])	BIGINT	Convert date time string string1 in format string2 to Unix timestamp (in seconds), using the specified timezone in table config. The default format of string2 is yyyy-MM-dd HH:mm:ss.
TO_DATE(string1[, string2])	DATE	Convert a date string string1 with format string2 to a date. The default format of string2 is yyyy-MM-dd.
TO_TIMESTAMP(string1[, string2])	TIMESTAMP	Converts date time string string1 with format string2 under the 'UTC+0' time zone to a timestamp. The default format of string2 is yyyy-MM-dd HH:mm:ss.

DATE

- **Function**
Returns a SQL date parsed from string in form of **yyyy-MM-dd**.

- **Description**
DATE DATE string

- **Input parameters**

Parameter	Data Types	Parameters
string	STRING	String in the SQL date format. Note that the string must be in the yyyy-MM-dd format. Otherwise, an error will be reported.

- **Example**

- Test statement

```
SELECT
  DATE "2021-08-19" AS `result`
FROM
  testtable;
```

- Test Result

result
2021-08-19

TIME

- **Function**
Returns a SQL time parsed from string in form of **HH:mm:ss[.fff]**.

- **Description**
TIME TIME string

- **Input parameters**

Parameter	Data Types	Parameters
string	STRING	Time Note that the string must be in the format of HH:mm:ss[.fff] . Otherwise, an error will be reported.

- **Example**

- Test statement

```
SELECT
  TIME "10:11:12" AS `result`,
  TIME "10:11:12.032" AS `result2`
FROM
  testtable;
```

- Test result

result	result2
10:11:12	10:11:12.032

TIMESTAMP

- **Function**
Converts the time string into timestamp. The time string format is **yyyy-MM-dd HH:mm:ss[.fff]**. The return value is of the **TIMESTAMP(3)** type.

- **Description**
TIMESTAMP(3) TIMESTAMP string

- **Input parameters**

Parameter	Data Types	Parameters
string	STRING	Time Note that the string must be in the format of yyyy-MM-dd HH:mm:ss[.fff] . Otherwise, an error will be reported.

- **Example**

- Test statement

```
SELECT
  TIMESTAMP "1997-04-25 13:14:15" AS `result`,
  TIMESTAMP "1997-04-25 13:14:15.032" AS `result2`
FROM
  testtable;
```

- Test result

result	result2
1997-04-25 13:14:15	1997-04-25 13:14:15.032

INTERVAL

- **Function**

Parses an interval string.

- **Description**

INTERVAL INTERVAL string range

- **Input parameters**

Parameter	Data Types	Parameters
string	STRING	Timestamp string used together with the range parameter. The string is in either of the following two formats: <ul style="list-style-type: none"> • yyyy-MM for SQL intervals of months. An interval range might be YEAR or YEAR TO MONTH for intervals of months. • dd hh:mm:ss.fff for SQL intervals of milliseconds. An interval range might be DAY, MINUTE, DAY TO HOUR, or DAY TO SECOND.
range	INTERVAL	Interval range. This parameter is used together with the string parameter. Available values are as follows: YEAR , YEAR To Month , DAY , MINUTE , DAY TO HOUR and DAY TO SECOND .

- **Example**

Test statement

```
-- indicates that the interval is 10 days and 4 milliseconds.
INTERVAL '10 00:00:00.004' DAY TO second
-- The interval is 10 days.
INTERVAL '10'
-- The interval is 2 years and 10 months.
INTERVAL '2-10' YEAR TO MONTH
```

CURRENT_DATE

- **Function**

Returns the current SQL time (**yyyy-MM-dd**) in the local time zone. The return value is of the **DATE** type.

- **Description**

DATE CURRENT_DATE

- **Input parameters**

None

- **Example**

- Test statement


```
SELECT
  CURRENT_DATE AS `result`
FROM
  testtable;
```

- Test result

result
2021-10-28

CURRENT_TIME

- **Function**

Returns the current SQL time (**HH:mm:sss.fff**) in the local time zone. The return value is of the **TIME** type.

- **Description**

TIME CURRENT_TIME

- **Input parameters**

None

- **Example**

- Test statement


```
SELECT
  CURRENT_TIME AS `result`
FROM
  testtable;
```

- Test Result

result
08:29:19.289

CURRENT_TIMESTAMP

- **Function**

Returns the current SQL timestamp in the local time zone. The return value is of the **TIMESTAMP(3)** type.

- **Description**

TIMESTAMP(3) CURRENT_TIMESTAMP

- **Input parameters**

None

- **Example**

- Test statement

```
SELECT
  CURRENT_TIMESTAMP AS `result`
FROM
  testtable;
```

- Test Result

result
2021-10-28 08:33:51.606

LOCALTIME

- **Function**

Returns the current SQL time in the local time zone. The return value is of the **TIME** type.

- **Description**

TIME LOCALTIME

- **Input parameters**

None

- **Example**

- Test statement

```
SELECT
  LOCALTIME AS `result`
FROM
  testtable;
```

- Test Result

result
16:39:37.706

LOCALTIMESTAMP

- **Function**

Returns the current SQL timestamp in the local time zone. The return value is of the **TIMESTAMP(3)** type.

- **Description**

TIMESTAMP(3) LOCALTIMESTAMP

- **Input parameters**

None

- **Example**

- Test statement

```
SELECT
  LOCALTIMESTAMP AS `result`
FROM
  testtable;
```

- Test Result

result
2021-10-28 16:43:17.625

EXTRACT

- **Function**

Returns a value extracted from the **timeintervalunit** part of temporal. The return value is of the **BIGINT** type.

- **Description**

BIGINT **EXTRACT**(timeinteravlunit **FROM** temporal)

- **Input parameters**

Parameter	Data Types	Parameters
timeinteravlunit	TIMEUNIT	Time unit to be extracted from a time point or interval. The value can be YEAR, QUARTER, MONTH, WEEK, DAY, DOY, HOUR, MINUTE, SECOND .
temporal	DATE/TIME/TIMESTAMP/INTERVAL	Time point or interval

 **CAUTION**

Do not specify a time unit that is not of any time points or intervals. Otherwise, the job fails to be submitted.

For example, an error message is displayed when the following statement is executed because **YEAR** cannot be extracted from **TIME**.

```
SELECT
  EXTRACT(YEAR FROM TIME '12:44:31') AS `result`
FROM
  testtable;
```

- **Example**

- Test statement

```
SELECT
  EXTRACT(YEAR FROM DATE '1997-04-25') AS `result`,
  EXTRACT(MINUTE FROM TIME '12:44:31') AS `result2`,
  EXTRACT(SECOND FROM TIMESTAMP '1997-04-25 13:14:15') AS `result3`;
```

```
EXTRACT(YEAR FROM INTERVAL '2-10' YEAR TO MONTH) AS `result4`,
FROM
testtable;
```

- Test result

result	result2	result3	result4
1997	44	15	2

YEAR

- **Function**

Returns the year from a SQL date date. The return value is of the **BIGINT** type.

- **Description**

BIGINT YEAR(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
YEAR(DATE '1997-04-25' ) AS `result`
FROM
testtable;
```

- Test result

result
1997

QUARTER

- **Function**

Returns the quarter of a year (an integer between 1 and 4) from a SQL date date. The return value is of the **BIGINT** type.

- **Description**

BIGINT QUARTER(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
QUARTER(DATE '1997-04-25' ) AS `result`
```

```
FROM
  testtable;
```

- Test result

result
2

MONTH

- **Function**

Returns the month of a year (an integer between 1 and 12) from a SQL date date. The return value is of the **BIGINT** type.

- **Description**

BIGINT MONTH(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  MONTH(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

result
4

WEEK

- **Function**

Returns the week of a year from a SQL date date. The return value is of the **BIGINT** type.

- **Description**

BIGINT WEEK(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  WEEK(DATE '1997-04-25' ) AS `result`
```

```
FROM
  testtable;
```

- Test result

result
17

DAYOFYEAR

- **Function**

Returns the day of a year (an integer between 1 and 366) from SQL date date. The return value is of the **BIGINT** type.

- **Description**

BIGINT DAYOFYEAR(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  DAYOFYEAR(DATE '1997-04-25') AS `result`
FROM
  testtable;
```

- Test Result

result
115

DAYOFMONTH

- **Function**

Returns the day of a month (an integer between 1 and 31) from a SQL date date. The return value is of the **BIGINT** type.

- **Description**

BIGINT DAYOFMONTH(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  DAYOFMONTH(DATE '1997-04-25') AS `result`
```

```
FROM
  testtable;
```

– Test Result

result
25

DAYOFWEEK

- **Function**

Returns the day of a week (an integer between 1 and 7) from a SQL date date. The return value is of the **BIGINT** type.

 **NOTE**

Note that the start day of a week is Sunday.

- **Description**

BIGINT DAYOFWEEK(date)

- **Input parameters**

Parameter	Data Types	Parameters
date	DATE	SQL date

- **Example**

– Test statement

```
SELECT
  DAYOFWEEK(DATE '1997-04-25') AS `result`
FROM
  testtable;
```

– Test Result

result
6

HOUR

- **Function**

Returns the hour of a day (an integer between 0 and 23) from SQL timestamp timestamp. The return value is of the **BIGINT** type.

- **Description**

BIGINT HOUR(timestamp)

- **Input parameters**

Parameter	Data Types	Parameters
timestamp	TIMESTAMP	SQL timestamp

- **Example**

- Test statement

```
SELECT
  HOUR(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
  testtable;
```

- Test Result

result
10

MINUTE

- **Function**

Returns the minute of an hour (an integer between 0 and 59) from a SQL timestamp. The return value is of the **BIGINT** type.

- **Description**

BIGINT **MINUTE**(timestamp)

- **Input parameters**

Parameter	Data Types	Parameters
timestamp	TIMESTAMP	SQL timestamp

- **Example**

- Test statement

```
SELECT
  MINUTE(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
  testtable;
```

- Test Result

result
11

SECOND

- **Function**

Returns the second of an hour (an integer between 0 and 59) from a SQL timestamp. The return value is of the **BIGINT** type.

- **Description**

BIGINT **SECOND**(timestamp)

- **Input parameters**

Parameter	Data Types	Parameters
timestamp	TIMESTAMP	SQL timestamp

- **Example**

- Test statement

```
SELECT
  SECOND(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
  testtable;
```

- Test result

result
12

FLOOR

- **Function**

Returns a value that rounds **timepoint** down to the time unit **timeintervalunit**.

- **Description**

TIME/TIMESTAMP(3) **FLOOR**(timepoint TO timeintervalunit)

- **Input parameters**

Parameter	Data Types	Parameters
timepoint	TIMESTAMP /TIME	SQL time or SQL timestamp
timeintervalunit	TIMEUNIT	Time unit. The value can be YEAR, QUARTER, MONTH, WEEK, DAY, DOY, HOUR, MINUTE, or SECOND.

- **Example**

- Test statement

```
SELECT
  FLOOR(TIME '13:14:15' TO MINUTE) AS `result`
  FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
  FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`
FROM testtable;
```

- Test result

message	message2	message3
13:14	13:14	1997-04-25T13:14

CEIL

- **Function**

Returns a value that rounds **timepoint** up to the time unit **timeintervalunit**.

- **Description**

TIME/TIMESTAMP(3) **CEIL**(timepoint TO timeintervalunit)

- **Input parameters**

Parameter	Data Types	Parameters
timepoint	TIMESTAMP /TIME	SQL time or SQL timestamp
timeintervalunit	TIMEUNIT	Time unit. The value can be YEAR, QUARTER, MONTH, WEEK, DAY, DOY, HOUR, MINUTE, or SECOND.

- **Example**

- Test statement

```
SELECT
  CEIL(TIME '13:14:15' TO MINUTE) AS `result`
  CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
  CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`
FROM testtable;
```

- Test Result

result	result2	result3
13:15	13:15	1997-04-25T13:15

OVERLAPS

- **Function**

Returns **TRUE** if two time intervals overlap; returns **FALSE** otherwise.

- **Description**

BOOLEAN (timepoint1, temporal1) **OVERLAPS** (timepoint2, temporal2)

- **Input parameters**

Parameter	Data Types	Parameters
timepoint1/ timepoint2	DATE/TIME/ TIMESTAMP	Time point
temporal1/ temporal2	DATE/TIME/ TIMESTAMP/ INTERVAL	Time point or interval

 **NOTE**

- **(timepoint, temporal)** is a closed interval.
- The temporal can be of the **DATE, TIME, TIMESTAMP, or INTERVAL** type.
 - When the temporal is **DATE, TIME, or TIMESTAMP, (timepoint, temporal)** indicates an interval between **timepoint** and **temporal**. The temporal can be earlier than the value of **timepoint**, for example, (**DATE '1997-04-25', DATE '1997-04-23'**).
 - When the temporal is **INTERVAL, (timepoint, temporal)** indicates an interval between **timepoint** and **timepoint + temporal**.
- Ensure that **(timepoint1, temporal1)** and **(timepoint2, temporal2)** are intervals of the same data type.

- **Example**

- Test statement

```
SELECT
  (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS `result`,
  (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS `result2`,
  (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:31:00', INTERVAL '2' HOUR) AS `result3`,
  (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:00:00', INTERVAL '3' HOUR) AS `result4`,
  (TIMESTAMP '1997-04-25 12:00:00', TIMESTAMP '1997-04-25 12:20:00') OVERLAPS
  (TIMESTAMP '1997-04-25 13:00:00', INTERVAL '2' HOUR) AS `result5`,
  (DATE '1997-04-23', INTERVAL '2' DAY) OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
  AS `result6`,
  (DATE '1997-04-25', DATE '1997-04-23') OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
  AS `result7`
FROM
  testtable;
```

- Test Result

res ult	res ult 2	res ult 3	res ult 4	resu lt5	resu lt6	result7
tru e	tru e	fals e	tru e	fals e	true	true

DATE_FORMAT

- **Function**

Converts a timestamp to a value of string in the format specified by the date format string.

- **Description**

STRING **DATE_FORMAT**(timestamp, dateformat)

- **Input parameters**

Parameter	Data Types	Parameters
timestamp	TIMESTAMP/ STRING	Time point
dateformat	STRING	String in the date format

- **Example**

- Test statement

```
SELECT
  DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd HH:mm:ss') AS `result`,
  DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result2`,
  DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yy/MM/dd HH:mm') AS `result3`,
  DATE_FORMAT('1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result4`
FROM testtable;
```

- Test Result

result	result2	result3	result4
1997-04-25 10:11:12	1997-04-25	97/04/25 10:11	1997-04-25

TIMESTAMPADD

- **Function**

Returns the date and time by combining **interval** and **timeintervalunit** and adding the combination to **timepoint**.

 **NOTE**

The return value of **TIMESTAMPADD** is the value of **timepoint**. An exception is that if the input **timepoint** is of the **TIMESTAMP** type, the return value can be inserted into a table field of the **DATE** type.

- **Description**

TIMESTAMP(3)/DATE/TIME **TIMESTAMPADD**(timeintervalunit, interval, timepoint)

- **Input parameters**

Parameter	Data Types	Parameters
timeintervalunit	TIMEUNIT	Time unit.
interval	INT	Interval
timepoint	TIMESTAMP/ DATE/TIME	Time point

- **Example**

- Test statement

```
SELECT
  TIMESTAMPADD(WEEK, 1, DATE '1997-04-25') AS `result`,
  TIMESTAMPADD(QUARTER, 1, TIMESTAMP '1997-04-25 10:11:12') AS `result2`,
  TIMESTAMPADD(SECOND, 2, TIME '10:11:12') AS `result3`
FROM testtable;
```

- Test Result

result	result2	result3
1997-05-02	<ul style="list-style-type: none"> • If this field is inserted into a table field of the TIMESTAMP type, 1997-07-25T10:11:12 is returned. • If this field is inserted into a table field of the TIMESTAMP type, 1997-07-25 is returned. 	10:11:14

TIMESTAMPDIFF

- **Function**

Returns the (signed) number of **timepointunit** between **timepoint1** and **timepoint2**. The unit for the interval is given by the first argument.

- **Description**

INT **TIMESTAMPDIFF**(timepointunit, timepoint1, timepoint2)

- **Input parameters**

Parameter	Data Types	Parameters
timepointunit	TIMEUNIT	Time unit. The value can be SECOND, MINUTE, HOUR, DAY, MONTH or YEAR .
timepoint1/ timepoint2	TIMESTAMP/ DATE	Time point

- **Example**

- Test statement

```
SELECT
    TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-25 10:00:00', TIMESTAMP '1997-04-28 10:00:00')
    AS `result`,
    TIMESTAMPDIFF(DAY, DATE '1997-04-25', DATE '1997-04-28') AS `result2`,
    TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-27 10:00:20', TIMESTAMP '1997-04-25 10:00:00')
    AS `result3`
FROM testtable;
```

- Test result

result	result2	result3
3	3	-2

CONVERT_TZ

- **Function**

Converts a datetime **string1** (with default ISO timestamp format '**yyyy-MM-dd HH:mm:ss**') from time zone **string2** to time zone **string3**.

- **Description**

STRING **CONVERT_TZ**(string1, string2, string3)

- **Input parameters**

Parameter	Data Types	Parameters
string1	STRING	SQL timestamp. If the value does not meet the format requirements, NULL is returned.

Parameter	Data Types	Parameters
string2	STRING	Time zone before conversion. The format of time zone should be either an abbreviation such as PST , a full name such as America/Los_Angeles , or a custom ID such as GMT-08:00 .
string3	STRING	Time zone after conversion. The format of time zone should be either an abbreviation such as PST , a full name such as America/Los_Angeles , or a custom ID such as GMT-08:00 .

- **Example**

- Test statement

```
SELECT
  CONVERT_TZ(1970-01-01 00:00:00, UTC, America/Los_Angeles) AS `result`,
  CONVERT_TZ(1997-04-25 10:00:00, UTC, GMT-08:00) AS `result2`
FROM testtable;
```

- Test Result

result	result2
1969-12-31 16:00:00	1997-04-25 02:00:00

FROM_UNIXTIME

- **Function**

Returns a representation of the **numeric** argument as a value in string format.

- **Description**

STRING FROM_UNIXTIME(numeric[, string])

- **Input parameters**

Parameter	Data Types	Parameters
numeric	BIGINT	An internal timestamp representing the number of seconds since 1970-01-01 00:00:00 UTC. The value can be generated by the UNIX_TIMESTAMP() function.
string	STRING	Time. If this parameter is not specified, the default time format is yyyy-MM-dd HH:mm:ss format.

- **Example**

- Test statement

```
SELECT
  FROM_UNIXTIME(44) AS `result`,
  FROM_UNIXTIME(44, 'yyyy:MM:dd') AS `result2`
FROM testtable;
```

- Test Result

result	result2
1970-01-01 08:00:44	1970:01:01

UNIX_TIMESTAMP

- **Function**

Gets current Unix timestamp in seconds. The return value is of the **BIGINT** type.

- **Description**

BIGINT UNIX_TIMESTAMP()

- **Input parameters**

None

- **Example**

- Test statement

```
SELECT
  UNIX_TIMESTAMP() AS `result`
FROM
  table;
```

- Test result

result
1635401982

UNIX_TIMESTAMP(string1[, string2])

- **Function**

Converts date time **string1** in format **string2** to Unix timestamp (in seconds). The return value is of the **BIGINT** type.

- **Description**

BIGINT UNIX_TIMESTAMP(string1[, string2])

- **Input parameters**

Parameter	Data Types	Parameters
string1	STRING	SQL timestamp string. An error is reported if the value does not comply with the string2 format.
string2	STRING	Time. If this parameter is not specified, the default time format is yyyy-MM-dd HH:mm:ss .

- **Example**

- Test statement

```
SELECT
  UNIX_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`;
```



```
UNIX_TIMESTAMP('1997-04-25 00:00:10', 'yyyy-MM-dd HH:mm:ss') AS `result2`,
UNIX_TIMESTAMP('1997-04-25 00:00:00') AS `result3`
FROM
testtable;
```

- Test result

result	result2	result3
861897600	861897610	861897600

TO_DATE

- **Function**

Converts a date **string1** with format **string2** to a date.

- **Description**

```
DATE TO_DATE(string1[, string2])
```

- **Input parameters**

Parameter	Data Types	Parameters
string1	STRING	SQL timestamp string. If the value is not in the required format, an error is reported.
string2	STRING	Format. If this parameter is not specified, the default time format is yyyy-MM-dd .

- **Example**

- Test statement

```
SELECT
TO_DATE('1997-04-25') AS `result`,
TO_DATE('1997:04:25', 'yyyy-MM-dd') AS `result2`,
TO_DATE('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
testtable;
```

- Test result

result	result2	result3
1997-04-25	1997-04-25	1997-04-25

TO_TIMESTAMP

- **Function**

Converts date time **string1** with format **string2** to a timestamp.

- **Description**

```
TIMESTAMP TO_TIMESTAMP(string1[, string2])
```

- **Input parameters**

Parameter	Data Types	Parameters
string1	STRING	SQL timestamp string. If the value is not in the required format, NULL is returned.
string2	STRING	Date format. If this parameter is not specified, the default format is yyyy-MM-dd HH:mm:ss .

• **Example**

- Test statement

```
SELECT
  TO_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
  TO_TIMESTAMP('1997-04-25 00:00:00') AS `result2`,
  TO_TIMESTAMP('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
  testtable;
```

- Test result

result	result2	result3
1997-04-25 00:00	1997-04-25 00:00	1997-04-25 00:00

1.5.2.4 Conditional Functions

Description

Table 1-53 Conditional Functions

Conditional Functions	Description
CASE value WHEN value1_1 [, value1_2]* THEN result1 [WHEN value2_1 [, value2_2]* THEN result2]* [ELSE resultZ] END	Returns resultX when the value is contained in (valueX_1, valueX_2, ...). Only the first matched value is returned. When no value matches, returns result_z if it is provided and returns NULL otherwise.
CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2]* [ELSE resultZ] END	Returns resultX when the first conditionX is met. Only the first matched value is returned. When no condition is met, returns result_z if it is provided and returns NULL otherwise.

Conditional Functions	Description
NULLIF(value1, value2)	Returns NULL if value1 is equal to value2; returns value1 otherwise. For example, NullIF (5, 5) returns NULL . NULLIF(5, 0) returns 5 .
COALESCE(value1, value2 [, value3]*)	Returns the first value (from left to right) that is not NULL from value1, value2, For example, COALESCE(NULL, 5) returns 5 .
IF(condition, true_value, false_value)	Returns the true_value if condition is met, otherwise false_value . For example, IF(5 > 3, 5, 3) returns 5 .
IS_ALPHA(string)	Returns TRUE if all characters in the string are letters, otherwise FALSE .
IS_DECIMAL(string)	Returns TRUE if string can be parsed to a valid numeric, otherwise FALSE .
IS_DIGIT(string)	Returns TRUE if all characters in string are digits, otherwise FALSE . Otherwise, FALSE is returned.

1.5.2.5 Type Conversion Functions

Syntax

```
CAST(value AS type)
```

Description

This function is used to forcibly convert types.

Precautions

If the input is **NULL**, **NULL** is returned.

Example 1: Convert the amount value to an integer.

The following example converts the **amount** value to an integer.

```
insert into temp select cast(amount as INT) from source_stream;
```

Table 1-54 Examples of type conversion functions

Example	Description	Example
cast(v1 as string)	Converts v1 to a string. The value of v1 can be of the numeric type or of the timestamp, date, or time type.	<p>Table T1:</p> <pre> content (INT) ----- 5 </pre> <p>Statement:</p> <pre>SELECT cast(content as varchar) FROM T1;</pre> <p>Result:</p> <pre>"5"</pre>
cast (v1 as int)	Converts v1 to the int type. The value of v1 can be a number or a character.	<p>Table T1:</p> <pre> content (STRING) ----- "5" </pre> <p>Statement:</p> <pre>SELECT cast(content as int) FROM T1;</pre> <p>Result:</p> <pre>5</pre>
cast(v1 as timestamp)	Converts v1 to the timestamp type. The value of v1 can be of the string , date , or time type.	<p>Table T1:</p> <pre> content (STRING) ----- "2018-01-01 00:00:01" </pre> <p>Statement:</p> <pre>SELECT cast(content as timestamp) FROM T1;</pre> <p>Result:</p> <pre>1514736001000</pre>
cast(v1 as date)	Converts v1 to the date type. The value of v1 can be of the string or timestamp type.	<p>Table T1:</p> <pre> content (TIMESTAMP) ----- 1514736001000 </pre> <p>Statement:</p> <pre>SELECT cast(content as date) FROM T1;</pre> <p>Result:</p> <pre>"2018-01-01"</pre>

 **NOTE**

Flink jobs do not support the conversion of **bigint** to **timestamp** using CAST. You can convert it using **to_timestamp**.

Example 2:

1. Create a Flink opensource SQL job by referring to [Kafka Source Table](#) and [Print Result Table](#), enter the following job running script, and submit the job.

Note: When creating a job, set Flink Version to 1.12 in the Running Parameters area on the job editing page, select Save Job Log, and set the OBS bucket for saving job logs to facilitate subsequent job log viewing. Change the values of the parameters in bold in the following script according to the actual situation.

```
CREATE TABLE kafkaSource (  
  cast_int_to_string int,  
  cast_String_to_int string,  
  case_string_to_timestamp string,  
  case_timestamp_to_date timestamp  
) WITH (  
  'connector' = 'kafka',  
  'topic' = 'KafkaTopic',  
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',  
  'properties.group.id' = 'GroupId',  
  'scan.startup.mode' = 'latest-offset',  
  "format" = "json"  
);  
  
CREATE TABLE printSink (  
  cast_int_to_string string,  
  cast_String_to_int int,  
  case_string_to_timestamp timestamp,  
  case_timestamp_to_date date  
) WITH (  
  'connector' = 'print'  
);  
  
insert into printSink select  
  cast(cast_int_to_string as string),  
  cast(cast_String_to_int as int),  
  cast(case_string_to_timestamp as timestamp),  
  cast(case_timestamp_to_date as date)  
from kafkaSource;
```

2. Connect to the Kafka cluster and send the following test data to the Kafka topic:

```
{"cast_int_to_string": "1", "cast_String_to_int": "1", "case_string_to_timestamp": "2022-04-02 15:00:00",  
"case_timestamp_to_date": "2022-04-02 15:00:00"}
```

3. View output.
 - Method 1:
 - i. Log in to the DLI management console and choose Job Management > Flink Streaming Jobs.
 - ii. Locate the row that contains the target Flink job, and choose More & > FlinkUI in the Operation column.
 - iii. On the Flink UI, choose Task Managers, click the task name, and select Stdout to view the job run logs.
 - Method 2: If you select Save Job Log for Running Parameters before submitting the job, perform the following operations:
 - i. Log in to the DLI management console and choose Job Management > Flink Streaming Jobs.
 - ii. Click the name of the corresponding Flink job, choose Run Log, click OBS Bucket, and locate the folder of the corresponding log based on the job running date.

- iii. Go to the folder of the corresponding date, find the folder whose name contains taskmanager, download the taskmanager.out file, and view the result log.

The query result is as follows:

```
+I(1,1,2022-04-02T15:00,2022-04-02)
```

1.5.2.6 Collection Functions

Description

Table 1-55 Collection functions

Collection Functions	Description
CARDINALITY(array)	Returns the number of elements in array.
array '[' integer ']	Returns the element at position INT 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 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.

1.5.2.7 Value Construction Functions

Description

Table 1-56 Value construction functions

Value Construction Functions	Description
ROW(value1, [, value2]*) (value1, [, value2]*)	Returns a row created from a list of values (value1, value2,...).
ARRAY '[' value1 [, value2]* ']	Returns an array created from a list of values (value1, value2, ...).
MAP '[' key1, value1 [, key2, value2]* ']	Returns a map created from a list of key-value pairs ((value1, value2), (value3, value4), ...). The key-value pair is (key1, value1), (key2, value2).

1.5.2.8 Value Access Functions

Description

Table 1-57 Value access functions

Function	Description
tableName.compositeType.field	Returns the value of a field from a Flink composite type (e.g., Tuple, POJO) by name.
tableName.compositeType.*	Returns a flat representation of a Flink composite type (e.g., Tuple, POJO) that converts each of its direct subtype into a separate field.

1.5.2.9 Hash Functions

Description

Table 1-58 Hash functions

Hash Functions	Description
MD5(string)	Returns the MD5 hash of string as a string of 32 hexadecimal digits. Returns NULL if string is NULL .
SHA1(string)	Returns the SHA-1 hash of string as a string of 40 hexadecimal digits. Returns NULL if string is NULL .
SHA224(string)	Returns the SHA-224 hash of string as a string of 56 hexadecimal digits. Returns NULL if string is NULL .
SHA256(string)	Returns the SHA-256 hash of string as a string of 64 hexadecimal digits. Returns NULL if string is NULL .
SHA384(string)	Returns the SHA-384 hash of string as a string of 96 hexadecimal digits. Returns NULL if string is NULL .
SHA512(string)	Returns the SHA-512 hash of string as a string of 128 hexadecimal digits. Returns NULL if string is NULL .

Hash Functions	Description
SHA2(string, hashLength)	Returns the hash using the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, or SHA-512). The first argument string is the string to be hashed and the second argument hashLength is the bit length of the result (224, 256, 384, or 512). If either argument is NULL, the result will also be NULL.

1.5.2.10 Aggregate Functions

An aggregate function performs a calculation operation on a set of input values and returns a value. For example, the COUNT function counts the number of rows retrieved by an SQL statement. [Table 1-59](#) lists aggregate functions.

Table 1-59 Aggregate functions

Function	Return Type	Description
COUNT([ALL] expression DISTINCT expression1 [, expression2]*)	BIGINT	Returns the number of input rows for which the expression is not NULL. Use DISTINCT for one unique instance of each value.
COUNT(*) COUNT(1)	BIGINT	Returns the number of input rows.
AVG([ALL DISTINCT] expression)	DOUBLE	Returns the average (arithmetic mean) of expression across all input rows. Use DISTINCT for one unique instance of each value.
SUM([ALL DISTINCT] expression)	DOUBLE	Returns the sum of expression across all input rows. Use DISTINCT for one unique instance of each value.
MAX([ALL DISTINCT] expression)	DOUBLE	Returns the maximum value of expression across all input rows.
MIN([ALL DISTINCT] expression)	DOUBLE	Returns the minimum value of expression across all input rows.
STDDEV_POP([ALL DISTINCT] expression)	DOUBLE	Returns the population standard deviation of expression across all input rows.
STDDEV_SAMP([ALL DISTINCT] expression)	DOUBLE	Returns the sample standard deviation of expression across all input rows.

Function	Return Type	Description
VAR_POP([ALL DISTINCT] expression)	DOUBLE	Returns the population variance (square of the population standard deviation) of expression across all input rows.
VAR_SAMP([ALL DISTINCT] expression)	DOUBLE	Returns the sample variance (square of the sample standard deviation) of expression across all input rows.
COLLECT([ALL DISTINCT] expression)	MULTISET	Returns a multiset of expression across all input rows.
VARIANCE([ALL DISTINCT] expression)	DOUBLE	Returns the sample variance (square of the sample standard deviation) of expression across all input rows.
FIRST_VALUE(expression)	Actual type	Returns the first value in an ordered set of values.
LAST_VALUE(expression)	Actual type	Returns the last value in an ordered set of values.

1.5.2.11 Table-Valued Functions

1.5.2.11.1 string_split

The **string_split** function splits a target string into substrings based on the specified separator and returns a substring list.

Description

```
string_split(target, separator)
```

Table 1-60 string_split parameters

Parameter	Data Types	Description
target	STRING	<p>Target string to be processed</p> <p>NOTE</p> <ul style="list-style-type: none"> • If target is NULL, an empty line is returned. • If target contains two or more consecutive separators, an empty substring is returned. • If target does not contain a specified separator, the original string passed to target is returned.

Parameter	Data Types	Description
separator	VARCHAR	Separator. Currently, only single-character separators are supported.

Example

1. Create a Flink OpenSource SQL job by referring to [Kafka Source Table](#) and [Print Result Table](#), enter the following job running script, and submit the job.

When you create a job, set **Flink Version** to **1.12** in the **Running Parameters** tab. Select **Save Job Log**, and specify the OBS bucket for saving job logs.

Change the values of the parameters in bold as needed in the following script.

```
CREATE TABLE kafkaSource (
  target STRING,
  separator VARCHAR
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE printSink (
  target STRING,
  item STRING
) WITH (
  'connector' = 'print'
);

insert into printSink
select target,
item from
kafkaSource,
lateral table(string_split(target, separator)) as T(item);
```

2. Connect to the Kafka cluster and send the following test data to the Kafka topic:

```
{"target":"test-flink","separator":"-"}
{"target":"flink","separator":"-"}
{"target":"one-two-ww-three","separator":"-"}

```

The data is as follows:

Table 1-61 Test table data

target (STRING)	separator (VARCHAR)
test-flink	-
flink	-
one-two-ww-three	-

3. View output.

- Method 1:
 - i. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - ii. Locate the row that contains the target Flink job, and choose **More > FlinkUI** in the **Operation** column.
 - iii. On the Flink UI, choose **Task Managers**, click the task name, and select **Stdout** to view job logs.
- Method 2: If you select **Save Job Log** on the **Running Parameters** tab before submitting the job, perform the following operations:
 - i. Log in to the DLI console. In the navigation pane, choose **Job Management > Flink Jobs**.
 - ii. Click the name of the corresponding Flink job, choose **Run Log**, click **OBS Bucket**, and locate the folder of the log you want to view according to the date.
 - iii. Go to the folder of the date, find the folder whose name contains **taskmanager**, download the **taskmanager.out** file, and view result logs.

The query result is as follows:

```
+l(test-flink,test)
+l(test-flink,flink)
+l(flink,flink)
+l(one-two-ww-three,one)
+l(one-two-ww-three,two)
+l(one-two-ww-three,ww)
+l(one-two-ww-three,three)
```

The output data is as follows:

Table 1-62 Result table data

target (STRING)	item (STRING)
test-flink	test
test-flink	flink
flink	flink
one-two-ww-three	one
one-two-ww-three	two
one-two-ww-three	ww
one-two-ww-three	three

2 Flink Opensource SQL 1.10 Syntax Reference

2.1 Constraints and Definitions

2.1.1 Supported Data Types

STRING, BOOLEAN, BYTES, DECIMAL, TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, DATE, TIME, TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL, ARRAY, MULTISSET, MAP, ROW

2.1.2 Syntax Definition

2.1.2.1 Data Definition Language (DDL)

2.1.2.1.1 CREATE TABLE

Syntax

```
CREATE TABLE table_name
(
  { <column_definition> | <computed_column_definition> }[, ...n]
  [ <watermark_definition> ]
  [ <table_constraint> ][, ...n]
)
[COMMENT table_comment]
[PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
WITH (key1=val1, key2=val2, ...)
```

<column_definition>:
column_name column_type [<column_constraint>] [COMMENT column_comment]

<column_constraint>:
[CONSTRAINT constraint_name] PRIMARY KEY NOT ENFORCED

<table_constraint>:
[CONSTRAINT constraint_name] PRIMARY KEY (column_name, ...) NOT ENFORCED

<computed_column_definition>:

```
column_name AS computed_column_expression [COMMENT column_comment]
<watermark_definition>:
  WATERMARK FOR rowtime_column_name AS watermark_strategy_expression
<source_table>:
  [catalog_name.][db_name.]table_name
```

Function

This clause is used to create a table with a specified name.

Description

COMPUTED COLUMN

A computed column is a virtual column generated using **column_name AS computed_column_expression**. A computed column evaluates an expression that can reference other columns declared in the same table. The column itself is not physically stored within the table. A computed column could be defined using **cost AS price * quantity**. This expression can contain any combination of physical columns, constants, functions, or variables, but cannot contain any subquery.

In Flink, a computed column is used to define the time attribute in **CREATE TABLE** statements. A processing time attribute can be defined easily via **proc AS PROCTIME()** using the system's **PROCTIME()** function. The event time column may be obtained from an existing field. In this case, you can use the computed column to obtain event time. For example, if the original field is not of the **TIMESTAMP(3)** type or is nested in a JSON string, you can use computed columns.

Notes:

- An expression that define a computed column in a source table is calculated after data is read from the data source. The column can be used in the **SELECT** statement.
- A computed column cannot be the target of an **INSERT** statement. In an **INSERT** statement, the schema of the **SELECT** statement must be the same as that of the target table that does not have a computed column.

WATERMARK

The **WATERMARK** clause defines the event time attribute of a table and takes the form **WATERMARK FOR rowtime_column_name AS watermark_strategy_expression**.

rowtime_column_name defines an existing column that is marked as the event time attribute of the table. The column must be of the **TIMESTAMP(3)** type and must be the top-level column in the schema. It can also be a computed column.

watermark_strategy_expression defines the watermark generation strategy. It allows arbitrary non-query expression, including computed columns, to calculate the watermark. The expression return type must be **TIMESTAMP(3)**, which represents the timestamp since the Epoch. The returned watermark will be emitted only if it is non-null and its value is larger than the previously emitted local watermark (to preserve the contract of ascending watermarks). The watermark generation expression is evaluated by the framework for every record.

The framework will periodically emit the largest generated watermark. If the current watermark is still identical to the previous one, or is null, or the value of the returned watermark is smaller than that of the last emitted one, then no new watermark will be emitted. Watermark is emitted in an interval defined by **pipeline.auto-watermark-interval** configuration. If watermark interval is 0 ms, the generated watermarks will be emitted per-record if it is not null and greater than the last emitted one.

When using event time semantics, tables must contain an event time attribute and watermarking strategy.

Flink provides several commonly used watermark strategies.

- Strictly ascending timestamps: **WATERMARK FOR rowtime_column AS rowtime_column.**
Emits a watermark of the maximum observed timestamp so far. Rows that have a timestamp bigger to the max timestamp are not late.
- Ascending timestamps: **WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SECOND.**
Emits a watermark of the maximum observed timestamp so far minus 1. Rows that have a timestamp bigger or equal to the max timestamp are not late.
- Bounded out of order timestamps: **WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'string' timeUnit.**
Emits watermarks, which are the maximum observed timestamp minus the specified delay, for example, **WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '5' SECOND** is a 5 seconds delayed watermark strategy.

```
CREATE TABLE Orders (  
  user BIGINT,  
  product STRING,  
  order_time TIMESTAMP(3),  
  WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND  
) WITH (...);
```

PRIMARY KEY

Primary key constraint is a hint for Flink to leverage for optimizations. It tells that a column or a set of columns of a table or a view are unique and they do not contain null. Neither of columns in a primary can be nullable. The primary key therefore uniquely identifies a row in a table.

Primary key constraint can be either declared along with a column definition (a column constraint) or as a single line (a table constraint). For both cases, it should only be declared as a singleton. If you define multiple primary key constraints at the same time, an exception would be thrown.

Validity Check

SQL standard specifies that a constraint can either be **ENFORCED** or **NOT ENFORCED**. This controls if the constraint checks are performed on the incoming/outgoing data. Flink does not own the data therefore the only mode we want to support is the **NOT ENFORCED** mode. It is up to the user to ensure that the query enforces key integrity.

Flink will assume correctness of the primary key by assuming that the columns nullability is aligned with the columns in primary key. Connectors should ensure those are aligned.

Notes: In a **CREATE TABLE** statement, creating a primary key constraint will alter the columns nullability, that means, a column with primary key constraint is not nullable.

PARTITIONED BY

Partition the created table by the specified columns. A directory is created for each partition if this table is used as a filesystem sink.

WITH OPTIONS

Table properties used to create a table source/sink. The properties are usually used to find and create the underlying connector.

The key and value of expression `key1=val1` should both be string literal.

Notes: The table registered with **CREATE TABLE** statement can be used as both table source and table sink. We cannot decide if it is used as a source or sink until it is referenced in the DMLs.

2.1.2.1.2 CREATE VIEW

Syntax

```
CREATE VIEW [IF NOT EXISTS] view_name  
  [{columnName [, columnName ]* }] [COMMENT view_comment]  
AS query_expression
```

Function

Create a view with multiple layers nested in it to simplify the development process.

Description

IF NOT EXISTS

If the view already exists, nothing happens.

Example

Create a view named **viewName**.

```
create view viewName as select * from dataSource
```

2.1.2.1.3 CREATE FUNCTION

Syntax

```
CREATE FUNCTION  
  [IF NOT EXISTS] function_name  
AS identifier [LANGUAGE JAVA|SCALA]
```

Function

Create a user-defined function.

Description

IF NOT EXISTS

If the function already exists, nothing happens.

LANGUAGE JAVA|SCALA

Language tag is used to instruct Flink runtime how to execute the function. Currently only **JAVA** and **SCALA** are supported, the default language for a function is **JAVA**.

Example

Create a function named **STRINGBACK**.

```
create function STRINGBACK as 'com.dli.StringBack'
```

2.1.2.2 Data Manipulation Language (DML)

Statements

Syntax

```
INSERT INTO table_name [PARTITION part_spec] query

part_spec: (part_col_name1=val1 [, part_col_name2=val2, ...])

query:
  values
  | {
    | select
    | selectWithoutFrom
    | query UNION [ ALL ] query
    | query EXCEPT query
    | query INTERSECT query
    }
  [ ORDER BY orderItem [, orderItem ]* ]
  [ LIMIT { count | ALL } ]
  [ OFFSET start { ROW | ROWS } ]
  [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]

orderItem:
  expression [ ASC | DESC ]

select:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
  [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]

selectWithoutFrom:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }

projectItem:
```



```

expression [ [ AS ] columnAlias ]
| tableAlias . *

tableExpression:
tableReference [ , tableReference ]*
| tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression [ joinCondition ]

joinCondition:
ON booleanExpression
| USING '(' column [ , column ]* ')'

tableReference:
tablePrimary
[ matchRecognize ]
[ [ AS ] alias [ '(' columnAlias [ , columnAlias ]* ')' ] ]

tablePrimary:
[ TABLE ] [ [ catalogName . ] schemaName . ] tableName
| LATERAL TABLE '(' functionName '(' expression [ , expression ]* ')' ')'
| UNNEST '(' expression ')'

values:
VALUES expression [ , expression ]*

groupItem:
expression
| '(' ')'
| '(' expression [ , expression ]* ')'
| CUBE '(' expression [ , expression ]* ')'
| ROLLUP '(' expression [ , expression ]* ')'
| GROUPING SETS '(' groupItem [ , groupItem ]* ')'

windowRef:
windowName
| windowSpec

windowSpec:
[ windowName ]
 '('
 [ ORDER BY orderItem [ , orderItem ]* ]
 [ PARTITION BY expression [ , expression ]* ]
 [
 RANGE numericOrIntervalExpression {PRECEDING}
 | ROWS numericExpression {PRECEDING}
 ]
 ')'

matchRecognize:
MATCH_RECOGNIZE '('
 [ PARTITION BY expression [ , expression ]* ]
 [ ORDER BY orderItem [ , orderItem ]* ]
 [ MEASURES measureColumn [ , measureColumn ]* ]
 [ ONE ROW PER MATCH ]
 [ AFTER MATCH
 ( SKIP TO NEXT ROW
 | SKIP PAST LAST ROW
 | SKIP TO FIRST variable
 | SKIP TO LAST variable
 | SKIP TO variable )
 ]
 PATTERN '(' pattern ')'
 [ WITHIN intervalLiteral ]
 DEFINE variable AS condition [ , variable AS condition ]*
 ')'

measureColumn:
expression AS alias

pattern:

```

```

patternTerm [ '|' patternTerm ]*
patternTerm:
  patternFactor [ patternFactor ]*
patternFactor:
  variable [ patternQuantifier ]
patternQuantifier:
  '*'
  | '*?'
  | '+'
  | '+?'
  | '?'
  | '??'
  | '{ [ minRepeat ], [ maxRepeat ] }' ['?']
  | 'repeat'

```

Precautions

Flink SQL uses a lexical policy for identifier (table, attribute, function names) similar to Java:

- The case of identifiers is preserved whether they are quoted.
- Identifiers are matched case-sensitively.
- Unlike Java, back-ticks allow identifiers to contain non-alphanumeric characters (for example **SELECT a AS `my field` FROM t**).

String literals must be enclosed in single quotes (for example, **SELECT'Hello World'**). Two single quotation marks are used for escaping (for example, **SELECT'It's me.'**). Unicode characters are supported in string literals. If explicit Unicode points are required, use the following syntax:

- Use the backslash (\) as escaping character (default): **SELECT U&'\263A'**
- Use a custom escaping character: **SELECT U&'#263A' UESCAPE '#'**

2.2 Flink OpenSource SQL 1.10 Syntax

This section describes the Flink OpenSource SQL syntax supported by DLI. For details about the parameters and examples, see the syntax description.

Creating Tables

Table 2-1 Syntax for creating tables

Classification	Function
Creating a Source Table	Kafka Source Table
	DIS Source Table
	JDBC Source Table
	GaussDB(DWS) Source Table
	Redis Source Table
	HBase Source Table

Classification	Function
	userDefined Source Table
Creating a Result Table	ClickHouse Result Table
	Kafka Result Table
	Upsert Kafka Result Table
	DIS Result Table
	JDBC Result Table
	GaussDB(DWS) Result Table
	Redis Result Table
	SMN Result Table
	HBase Result Table
	Elasticsearch Result Table
User-defined Result Table	
Creating a Dimension Table	JDBC Dimension Table
	GaussDB(DWS) Dimension Table
	HBase Dimension Table

2.3 Data Definition Language (DDL)

2.3.1 Creating a Source Table

2.3.1.1 Kafka Source Table

Function

Create a source stream to obtain data from Kafka as input data for jobs.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

Kafka is an offline cluster. You have built an enhanced datasource connection to connect Flink jobs to Kafka. You have set security group rules as required.

Precautions

SASL_SSL cannot be enabled for the interconnected Kafka cluster.

Syntax

```
create table kafkaSource(
  attr_name attr_type
  (' attr_name attr_type)*
  (' PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
)
with (
  'connector.type' = 'kafka',
  'connector.version' = "",
  'connector.topic' = "",
  'connector.properties.bootstrap.servers' = "",
  'connector.properties.group.id' = "",
  'connector.startup-mode' = "",
  'format.type' = ""
);
```

Parameters

Table 2-2 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to kafka .
connector.version	Yes	Kafka version. The value can be '0.10' or '0.11', which corresponds to Kafka 2.11 to 2.4.0 and other historical versions, respectively.
format.type	Yes	Data deserialization format. The value can be csv , json , or avro .
format.field-delimiter	No	Attribute delimiter. You can customize the attribute delimiter only when the encoding format is CSV. The default delimiter is a comma (,).
connector.topic	Yes	Kafka topic name. Either this parameter or connector.topic-pattern is used.
connector.topic-pattern	No	Regular expression for matching the Kafka topic name. Either this parameter or connector.topic is used. Example: 'topic.*' '(topic-c topic-d)' '(topic-a topic-b topic-\\d*)' '(topic-a topic-b topic-[0-9]*)'

Parameter	Mandatory	Description
connector.properties.bootstrap.servers	Yes	Kafka broker addresses. Use commas (,) to separated them.
connector.properties.group.id	No	Consumer group name
connector.startup-mode	No	Consumer startup mode. The value can be earliest-offset , latest-offset , group-offsets , specific-offsets or timestamp . The default value is group-offsets .
connector.specific-offsets	No	Consumption offset. This parameter is mandatory when startup-mode is specific-offsets . The value is in the 'partition:0,offset:42;partition:1,offset:300' format.
connector.startup-timestamp-millis	No	Consumption start timestamp. This parameter is mandatory when startup-mode is timestamp .
connector.properties.*	No	Native Kafka property

Example

- Create table **kafkaSource** and read data encoded in CSV format from Kafka.

```
create table kafkaSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.properties.group.id' = 'test-group',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'csv'
);
```

- Create table **kafkaSource** and read data in non-nested JSON strings from Kafka.

Assume that the non-nested JSON strings are as follows:

```
{"car_id": 312, "car_owner": "wang", "car_brand": "tang"}
{"car_id": 313, "car_owner": "li", "car_brand": "lin"}
{"car_id": 314, "car_owner": "zhao", "car_brand": "han"}
```

You can create the table as follows:

```
create table kafkaSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING
)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
```

```
'connector.topic' = 'test-topic',  
'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',  
'connector.properties.group.id' = 'test-group',  
'connector.startup-mode' = 'latest-offset',  
'format.type' = 'json'  
);
```

- Create table **kafkaSource** and read the nested JSON data from Kafka.

Assume that the JSON data is as follows:

```
{  
  "id": "1",  
  "type": "online",  
  "data": {  
    "patient_id": 1234,  
    "name": "bob1234",  
    "age": "Bob",  
    "gmt_create": "Bob",  
    "gmt_modify": "Bob"  
  }  
}
```

You can create the table as follows:

```
CREATE table kafkaSource(  
  id STRING,  
  type STRING,  
  data ROW(  
    patient_id STRING,  
    name STRING,  
    age STRING,  
    gmt_create STRING,  
    gmt_modify STRING)  
)  
with (  
  'connector.type' = 'kafka',  
  'connector.version' = '0.11',  
  'connector.topic' = 'test-topic',  
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',  
  'connector.properties.group.id' = 'test-group',  
  'connector.startup-mode' = 'latest-offset',  
  'format.type' = 'json'  
);
```

2.3.1.2 DIS Source Table

Function

Create a source stream to read data from DIS. DIS accesses user data and Flink job reads data from the DIS stream as input data for jobs. Flink jobs can quickly remove data from producers using DIS source sources for continuous processing. Flink jobs are applicable to scenarios where data outside the cloud service is imported to the cloud service for filtering, real-time analysis, monitoring reports, and dumping.

DIS addresses the challenge of transmitting data outside cloud services to cloud services. DIS builds data intake streams for custom applications capable of processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, Internet of Things (IoT) data, social media feeds, website clickstreams, and location-tracking events. For more information about DIS, see the .

Syntax

```
create table disSource (  
  attr_name attr_type
```

```
(, attr_name attr_type)*
(, PRIMARY KEY (attr_name, ...) NOT ENFORCED)
(, watermark for rowtime_column_name as watermark_strategy_expression)
)
with (
'connector.type' = 'dis',
'connector.region' = "",
'connector.channel' = "",
'format-type' = ""
);
```

Parameters

Table 2-3 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Data source type. Set this parameter to dis .
connector.region	Yes	Region where DIS for storing the data locates.
connector.ak	No	Access key ID. This parameter must be set in pair with sk .
connector.sk	No	Secret access key. This parameter must be set in pair with ak .
connector.channel	Yes	Name of the DIS stream where data is located.
connector.partition-count	No	Number of partitions where data will be read. Data in partition 0 to partition-count will be read. Neither this parameter or partition-range can be configured. If neither of the two parameters is set, all partition data will be read by default.
connector.partition-range	No	Range of partitions where data will be read. Neither this parameter or partition-count can be configured. If neither of the two parameters is set, all partition data will be read by default. For example, if you set partition-range to [0:2] , data in partitions 1, 2, and 3 will be read. The range must be within the DIS stream.
connector.offset	No	Start position from which data will be read. Either this parameter or start-time can be configured.

Parameter	Mandatory	Description
connector.start-time	No	Time from which DLI reads data If this parameter is specified, DLI reads data read from the specified time. The format is yyyy-MM-dd HH:mm:ss . If neither start-time nor offset is specified, the latest data is read.
connector.enable-checkpoint	No	Whether to enable the checkpoint function. The value can be true (enabled) or false (disabled). The default value is false . Do not set this parameter when offset or start-time is set. If this parameter is set to true , checkpoint-app-name must be configured.
connector.checkpoint-app-name	No	ID of a DIS consumer. If a DIS stream is consumed by different jobs, you need to configure the consumer ID for each job to avoid checkpoint confusion. Do not set this parameter when offset or start-time is set. If checkpoint-app-name is set to true , this parameter is mandatory.
connector.checkpoint-interval	No	Interval of checkpoint operations on the DIS source operator. The default value is 60s . Available value units: d, day/h, hour/min, minute/s, sec, second Do not set this parameter when offset or start-time is configured.
format.type	Yes	Data coding format. The value can be csv or json .
format.field-delimiter	No	Attribute delimiter. You can customize the attribute delimiter only when the encoding format is CSV. The default delimiter is a comma (,).

Precautions

None

Example

```
create table disCsvSource (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT)
with (
```



```
'connector.type' = 'dis',  
'connector.region' = '',  
'connector.channel' = 'disInput',  
'format.type' = 'csv'  
);
```

2.3.1.3 JDBC Source Table

Function

The JDBC connector is a Flink's built-in connector to read data from a database.

Prerequisites

- An enhanced datasource connection with the database has been established, so that you can configure security group rules as required.

Syntax

```
create table jdbcSource (  
  attr_name attr_type  
  (,' attr_name attr_type)*  
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
  (,' watermark for rowtime_column_name as watermark-strategy_expression)  
)  
with (  
  'connector.type' = 'jdbc',  
  'connector.url' = '',  
  'connector.table' = '',  
  'connector.username' = '',  
  'connector.password' = ''  
);
```

Parameters

Table 2-4 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Data source type. Set this parameter to jdbc .
connector.url	Yes	Database URL
connector.table	Yes	Name of the table where the data to be read from the database is located
connector.driver	No	Driver required for connecting to the database. If you do not set this parameter, the automatically extracted URL will be used.
connector.username	No	Database authentication username. This parameter must be configured in pair with connector.password .

Parameter	Mandatory	Description
connector.password	No	Database authentication password. This parameter must be configured in pair with connector.username .
connector.read.partition.column	No	Name of the column used to partition the input This parameter is mandatory if connector.read.partition.lower-bound , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.lower-bound	No	Lower bound of values to be fetched for the first partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.upper-bound	No	Upper bound of values to be fetched for the last partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.lower-bound , and connector.read.partition.num are configured.
connector.read.partition.num	No	Number of partitions to be created This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.upper-bound are configured.
connector.read.fetch-size	No	Number of rows fetched from the database each time The default value is 0 , indicating the hint is ignored.

Precautions

None

Example

```
create table jdbcSource (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
```

```
total_miles INT)
with (
  'connector.type' = 'jdbc',
  'connector.url' = 'jdbc:mysql://xx.xx.xx.xx:3306/xx',
  'connector.table' = 'jdbc_table_name',
  'connector.driver' = 'com.mysql.jdbc.Driver',
  'connector.username' = 'xxx',
  'connector.password' = 'xxxxxx'
);
```

2.3.1.4 GaussDB(DWS) Source Table

Function

DLI reads data of Flink jobs from GaussDB(DWS). GaussDB(DWS) database kernel is compliant with PostgreSQL. The PostgreSQL database can store data of more complex types and delivers space information services, multi-version concurrent control (MVCC), and high concurrency. It applies to location applications, financial insurance, and e-commerce.

GaussDB(DWS) is an online data processing database based on the cloud infrastructure and platform and helps you mine and analyze massive sets of data.

Prerequisites

- Ensure that you have created a GaussDB(DWS) cluster using your account. For details about how to create a GaussDB(DWS) cluster, see "Creating a Cluster" in *Data Warehouse Service Management Guide*.
- A GaussDB(DWS) database table has been created.
- An enhanced datasource connection has been created for DLI to connect to GaussDB(DWS) clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.

Syntax

```
create table dwsSource (
  attr_name attr_type
  (,' attr_name attr_type)*
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (,' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'gaussdb',
  'connector.url' = "",
  'connector.table' = "",
  'connector.username' = "",
  'connector.password' = ""
);
```

Parameters

Table 2-5 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to gaussdb .
connector.url	Yes	JDBC connection address. The format is <code>jdbc:postgresql://\${ip}:\${port}/\${dbName}</code> . If the database version is later than 8.1.0, the value format is <code>jdbc:gaussdb://\${ip}:\${port}/\${dbName}</code> .
connector.table	Yes	Name of the table to be operated. If the GaussDB(DWS) table is in a schema, the format is <code>schema`.`Table name</code> . For details, see the Example .
connector.driver	No	JDBC connection driver. The default value is org.postgresql.Driver . If the database version is later than 8.1.0, the value is com.huawei.gauss200.jdbc.Driver .
connector.username	No	Database authentication user name. This parameter must be configured in pair with connector.password .
connector.password	No	Database authentication password. This parameter must be configured in pair with connector.username .
connector.read.partition.column	No	Name of the column used to partition the input This parameter is mandatory if connector.read.partition.lower-bound , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.lower-bound	No	Lower bound of values to be fetched for the first partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.upper-bound	No	Upper bound of values to be fetched for the last partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.lower-bound , and connector.read.partition.num are configured.

Parameter	Mandatory	Description
connector.read.partition.num	No	Number of partitions to be created This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.upper-bound are configured.
connector.read.fetch-size	No	Number of rows fetched from the database each time The default value is 0 , indicating the hint is ignored.

Example

- If you use the `gsjdbc4` driver for connection, set **connector.driver** to **org.postgresql.Driver**. You can omit this parameter because the `gsjdbc4` driver is the default one.

Create table **dwsSource** with data fetched from the **car_info** table that is not in a schema:

```
create table dwsSource(  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_speed INT  
) with (  
  'connector.type' = 'gaussdb',  
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',  
  'connector.table' = 'car_info',  
  'connector.username' = 'xx',  
  'connector.password' = 'xx'  
);
```

Create table **dwsSource** with data fetched from GaussDB(DWS) table **test** that is in a schema named **test_schema**:

```
create table dwsSource(  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_speed INT  
) with (  
  'connector.type' = 'gaussdb',  
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',  
  'connector.table' = 'test_schema"."test',  
  'connector.username' = 'xx',  
  'connector.password' = 'xx'  
);
```

- If you use the `gsjdbc200` driver for connection, set **connector.driver** to **com.huawei.gauss200.jdbc.Driver**.

Create table **dwsSource** with data fetched from GaussDB(DWS) table **test** that is in a schema named **ads_game_sdk_base**:

```
create table dwsSource(  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_speed INT  
) with (  
  'connector.type' = 'gaussdb',  
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',  
  'connector.table' = 'ads_game_sdk_base.test',  
  'connector.username' = 'xx',  
  'connector.password' = 'xx'  
);
```

```
'connector.type' = 'gaussdb',
'connector.table' = 'ads_game_sdk_base\'.\"test',
'connector.driver' = 'com.huawei.gauss200.jdbc.Driver',
'connector.url' = 'jdbc:gaussdb://xx.xx.xx.xx:8000/xx',
'connector.username' = 'xx',
'connector.password' = 'xx'
);
```

2.3.1.5 Redis Source Table

Function

Create a source stream to obtain data from Redis as input for jobs.

Prerequisites

An enhanced datasource connection with Redis has been established, so that you can configure security group rules as required.

Syntax

```
create table dwsSource (
  attr_name attr_type
  (,' attr_name attr_type)*
  (,' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'redis',
  'connector.host' = "",
  'connector.port' = ""
);
```

Parameters

Table 2-6 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to redis .
connector.host	Yes	Redis connector address
connector.port	Yes	Redis connector port
connector.password	No	Redis authentication password
connector.deploy-mode	No	Redis deployment mode. The value can be standalone or cluster . The default value is standalone .

Parameter	Mandatory	Description
connector.table-name	No	Name of the table stored in the Redis. This parameter is mandatory in the Redis Hashmap storage pattern. In this pattern, data is stored to Redis in hashmaps. The hash key is `\${table-name}:\${ext-key} , and the field name is the column name. NOTE Table storage pattern: connector.table-name and connector.key-column are used as Redis keys. For the Redis hash type, each key corresponds to a hashmap. A hash key is a field name of the source table, and a hash value is a field value of the source table.
connector.use-internal-schema	No	Whether to use the existing schema in the Redis. This parameter is optional in the Redis Hashmap storage pattern. The default value is false .
connector.key-column	No	This parameter is optional in table storage pattern. The value is used as the value of ext-key in the Redis. If this parameter is not set, the value of ext-key is the generated UUID.

Example

Reads data from Redis.

```
create table redisSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'redis',
  'connector.host' = 'xx.xx.xx.xx',
  'connector.port' = '6379',
  'connector.password' = 'xx',
  'connector.table-name' = 'car_info'
);
```

2.3.1.6 HBase Source Table

Function

Create a source stream to obtain data from HBase as input for jobs. HBase is a column-oriented distributed cloud storage system that features enhanced reliability, excellent performance, and elastic scalability. It applies to the storage of massive amounts of data and distributed computing. You can use HBase to build a storage system capable of storing TB- or even PB-level data. With HBase, you can filter and analyze data with ease and get responses in milliseconds, rapidly mining data value. DLI can read data from HBase for filtering, analysis, and data dumping.

Prerequisites

- An enhanced datasource connection has been created for DLI to connect to HBase, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
- **If MRS HBase is used, IP addresses of all hosts in the MRS cluster have been added to host information of the enhanced datasource connection.**

Syntax

```
create table hbaseSource (
  attr_name attr_type
  (,' attr_name attr_type)*
  (,' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = "",
  'connector.zookeeper.quorum' = ""
);
```

Parameters

Table 2-7 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to hbase .
connector.version	Yes	The value must be 1.4.3 .
connector.table-name	Yes	HBase table name
connector.zookeeper.quorum	Yes	ZooKeeper address
connector.zookeeper.znode.parent	No	Root directory for ZooKeeper. The default value is / hbase .

Parameter	Man dator y	Description
connector.rowkey y	No	Content of a compound rowkey to be assigned. The content is assigned to a new field based on the configuration. Example: rowkey1:3,rowkey2:3,... The value 3 indicates the first three bytes of the field. The number cannot be greater than the byte size of the field and cannot be less than 1. rowkey1:3,rowkey2:3 indicates that the first three bytes of the compound rowkey are assigned to rowkey1 , and the last three bytes are assigned to rowkey2 .

Example

```
create table hbaseSource(
  rowkey1 string,
  rowkey2 string,
  info Row<owner string>,
  car ROW<miles string, speed string>
) with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = 'carinfo',
  'connector.rowkey' = 'rowkey1:1,rowkey2:3',
  'connector.zookeeper.quorum' = 'xxx:2181'
);
```

2.3.1.7 userDefined Source Table

Function

You can call APIs to obtain data from the cloud ecosystem or an open source ecosystem and use the obtained data as input of Flink jobs.

Prerequisites

The customized source class needs to inherit the **RichParallelSourceFunction** class and specify the data type as Row.

For example, run **public class MySource extends RichParallelSourceFunction<Row>{}** to declare custom class **MySource**. You need to implement the **open**, **run**, **close**, and **cancel** functions. Encapsulate the class into a JAR file and upload the file through the UDF JAR on the SQL editing page.

Content of the dependent pom configuration file is as follows:

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_2.11</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
```

```

</dependency>

<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-core</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>

```

Syntax

```

create table userDefinedSource (
  attr_name attr_type
  (' attr_name attr_type)*
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = ''
);

```

Parameters

Table 2-8 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Source type. The value can only be user-defined , indicating a custom source.
connector.class-name	Yes	Fully qualified class name of the source class
connector.class-parameter	No	Parameter of the constructor of the source class. Only one parameter of the string type is supported.

Precautions

connector.class-name must be a fully qualified class name.

Example

```

create table userDefinedSource (
  attr1 int,
  attr2 int
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'xx.xx.MySource'
);

```

2.3.2 Creating a Result Table

2.3.2.1 ClickHouse Result Table

Function

DLI exports Flink job data to ClickHouse result tables.

ClickHouse is a column-based database oriented to online analysis and processing. It supports SQL query and provides good query performance. The aggregation analysis and query performance based on large and wide tables is excellent, which is one order of magnitude faster than other analytical databases.

Prerequisites

- Ensure your jobs run on an exclusive queue (non-shared queue) of DLI.
- You have established an enhanced datasource connection to ClickHouse and set the port in the security group rule of the ClickHouse cluster as needed.

For details about how to set up an enhanced datasource connection. For details, see "Enhanced Datasource Connection" in the *Data Lake Insight User Guide*.

Precautions

- When you create a ClickHouse cluster for MRS, set the cluster version to MRS 3.1.0 and do not enable Kerberos authentication.
- Do not define a primary key in Flink SQL statements. Do not use any syntax that generates primary keys, such as **insert into clickhouseSink select id, cout(*) from sourceName group by id**.
- Flink supports the following data types: string, tinyint, smallint, int, long, float, double, date, timestamp, decimal, and Array.

The array supports only the int, bigint, string, float, and double data types.

Syntax

```
create table clickhouseSink (  
  attr_name attr_type  
  ('; attr_name attr_type)*  
)  
with (  
  'connector.type' = 'clickhouse',  
  'connector.url' = "",  
  'connector.table' = ""  
);
```

Parameters

Table 2-9 Parameter description

Parameter	Man dato ry	Description
connector.type	Yes	Result table type. Set this parameter to clickhouse .

Parameter	Mandatory	Description
connector.url	Yes	<p>ClickHouse URL.</p> <p>Parameter format: jdbc:clickhouse:// <i>ClickHouseBalancer instance IP address:HTTP port number for ClickHouseBalancer instances/Database name</i></p> <ul style="list-style-type: none"> • IP address of a ClickHouseBalancer instance: Log in to the MRS management console, click a cluster name, and choose Components > ClickHouse > Instance to obtain the service IP address of the ClickHouseBalancer instance. • HTTP port of a ClickHouseBalancer instance: Log in to the MRS management console, click the target cluster name. On the displayed page, choose Components > ClickHouse. In the Service Configuration tab, choose ClickHouseBalancer from the All Roles dropdown list and search for lb_http_port to configure the parameter. The default value is 21425. • The database name is the name of the database created for the ClickHouse cluster.
connector.table	Yes	Name of the ClickHouse table to be created
connector.driver	No	<p>Driver required for connecting to the database</p> <ul style="list-style-type: none"> • If this parameter is not specified during table creation, the driver automatically extracts the value from the ClickHouse URL. • If this parameter is specified during table creation, the value must be ru.yandex.clickhouse.ClickHouseDriver.
connector.username	No	Account for connecting the ClickHouse database
connector.password	No	Password for accessing the ClickHouse database
connector.write.flush.max-rows	No	Maximum number of rows to be updated when data is written. The default value is 5000 .
connector.write.flush.interval	No	Interval for data update. The unit can be ms, milli, millisecond/s, sec, second/min or minute.
connector.write.max-retries	No	Maximum number of attempts to write data if failed. The default value is 3 .

Example

Read data from a DIS table and insert the data into the **test** table of ClickHouse database **flinktest**.

1. Create a DIS source table **disSource**.

```
create table disSource(  
  attr0 string,  
  attr1 TINYINT,  
  attr2 smallint,  
  attr3 int,  
  attr4 bigint,  
  attr5 float,  
  attr6 double,  
  attr7 String,  
  attr8 string,  
  attr9 timestamp(3),  
  attr10 timestamp(3),  
  attr11 date,  
  attr12 decimal(38, 18),  
  attr13 decimal(38, 18)  
) with (  
  "connector.type" = "dis",  
  "connector.region" = "cn-xxxx-x",  
  "connector.channel" = "xxxx",  
  "format.type" = 'csv'  
);
```

2. Create ClickHouse result table **clickhouse** and insert the data from the **disSource** table to the result table.

```
create table clickhouse(  
  attr0 string,  
  attr1 TINYINT,  
  attr2 smallint,  
  attr3 int,  
  attr4 bigint,  
  attr5 float,  
  attr6 double,  
  attr7 String,  
  attr8 string,  
  attr9 timestamp(3),  
  attr10 timestamp(3),  
  attr11 date,  
  attr12 decimal(38, 18),  
  attr13 decimal(38, 18),  
  attr14 array < int >,  
  attr15 array < bigint >,  
  attr16 array < float >,  
  attr17 array < double >,  
  attr18 array < varchar >,  
  attr19 array < String >  
) with (  
  'connector.type' = 'clickhouse',  
  'connector.url' = 'jdbc:clickhouse://xx.xx.xx.xx:xx/flinktest',  
  'connector.table' = 'test'  
);  
  
insert into  
  clickhouse  
select  
  attr0,  
  attr1,  
  attr2,  
  attr3,  
  attr4,  
  attr5,  
  attr6,  
  attr7,  
  attr8,
```

```

attr9,
attr10,
attr11,
attr12,
attr13,
array [attr3, attr3+1],
array [cast(attr4 as bigint), cast(attr4+1 as bigint)],
array [cast(attr12 as float), cast(attr12+1 as float)],
array [cast(attr13 as double), cast(attr13+1 as double)],
array ['TEST1', 'TEST2'],
array [attr7, attr7]
from
disSource;

```

2.3.2.2 Kafka Result Table

Function

DLI exports the output data of the Flink job to Kafka.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

Kafka is an offline cluster. You have built an enhanced datasource connection to connect Flink jobs to Kafka. You have set security group rules as required.

Precautions

SASL_SSL cannot be enabled for the interconnected Kafka cluster.

Syntax

```

create table kafkaSource(
  attr_name attr_type
  (,' attr_name attr_type)*
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'kafka',
  'connector.version' = "",
  'connector.topic' = "",
  'connector.properties.bootstrap.servers' = "",
  'format.type' = ""
);

```

Parameters

Table 2-10 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to kafka .

Parameter	Mandatory	Description
connector.version	No	Kafka version. The value can be '0.10' or '0.11', which corresponds to Kafka 2.11 to 2.4.0 and other historical versions, respectively.
format.type	Yes	Data serialization format. The value can be csv , json , or avro .
format.field-delimiter	No	Attribute delimiter. You can customize the attribute delimiter only when the encoding format is CSV. The default delimiter is a comma (,).
connector.topic	Yes	Kafka topic name.
connector.properties.bootstrap.servers	Yes	Kafka broker addresses. Use commas (,) to separated them.
connector.sink-partitioner	No	Partitioner type. The value can be fixed , round-robin , or custom .
connector.sink-partitioner-class	No	Custom partitioner. This parameter is mandatory when sink-partitioner is custom , for example, org.mycompany.MyPartitioner .
update-mode	No	Data update mode. Three write modes are supported: append , retract , and upsert .
connector.properties.*	No	Native properties of Kafka

Example

Output the data in **kafkaSink** to Kafka.

```
create table kafkaSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.10',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.sink-partitioner' = 'round-robin',
  'format.type' = 'csv'
);
```

2.3.2.3 Upsert Kafka Result Table

Function

DLI exports the output data of the Flink job to Kafka in upsert mode.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and

provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

Kafka is an offline cluster. You have built an enhanced datasource connection to connect Flink jobs to Kafka. You have set security group rules as required.

Precautions

SASL_SSL cannot be enabled for the interconnected Kafka cluster.

Syntax

```
create table kafkaSource(
  attr_name attr_type
  (,' attr_name attr_type)*
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'upsert-kafka',
  'connector.version' = "",
  'connector.topic' = "",
  'connector.properties.bootstrap.servers' = "",
  'format.type' = ""
);
```

Parameters

Table 2-11 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to upsert-kafka .
connector.version	No	Kafka version. The value can only be 0.11 .
format.type	Yes	Data serialization format. The value can be csv , json , or avro .
connector.topic	Yes	Kafka topic name
connector.properties.bootstrap.servers	Yes	Kafka broker addresses. Use commas (,) to separated them.
connector.sink-partitioner	No	Partitioner type. The value can be fixed , round-robin , or custom .

Parameter	Mandatory	Description
connector.sink-partitioner-class	No	Custom partitioner. This parameter is mandatory when sink-partitioner is custom , for example, org.mycompany.MyPartitioner .
connector.sink.ignore-retraction	No	Whether to ignore the retraction message. The default value is false , indicating that the retraction message is written to Kafka as null .
update-mode	No	Data update mode. Three write modes are supported: append , retract , and upsert .
connector.properties.*	No	Native properties of Kafka

Example

```
create table upsertKafkaSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT,
  primary key (car_id) not enforced
)
with (
  'connector.type' = 'upsert-kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'format.type' = 'csv'
);
```

2.3.2.4 DIS Result Table

Function

DLI writes the Flink job output data into DIS. The data is filtered and imported to the DIS stream for future processing.

DIS addresses the challenge of transmitting data outside cloud services to cloud services. DIS builds data intake streams for custom applications capable of processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, Internet of Things (IoT) data, social media feeds, website clickstreams, and location-tracking events. For more information about DIS, see the *Data Ingestion Service User Guide*.

Syntax

```
create table disSink (
  attr_name attr_type
  (' attr_name attr_type)*
```

```
(,PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
'connector.type' = 'dis',
'connector.region' = "",
'connector.channel' = "",
'format.type' = ""
);
```

Parameters

Table 2-12 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Data source type. Set this parameter to dis .
connector.region	Yes	Region where DIS for storing the data locates.
connector.ak	No	Access key ID. This parameter must be set in pair with sk .
connector.sk	No	Secret access key. This parameter must be set in pair with ak .
connector.channel	Yes	Name of the DIS stream where data is located.
format.type	Yes	Data coding format. The value can be csv or json .
format.field-delimiter	No	Attribute delimiter. You can customize the attribute delimiter only when the encoding format is CSV. The default delimiter is a comma (,).
connector.partition-key	No	Group primary key. Multiple primary keys are separated by commas (,). If this parameter is not specified, data is randomly written to DIS partitions.

Precautions

None

Example

Output the data in the **disSink** stream to DIS.

```
create table disSink(
car_id STRING,
car_owner STRING,
car_brand STRING,
car_speed INT
)
with (
'connector.type' = 'dis',
'connector.region' = "",
```

```
'connector.channel' = 'disOutput',
'connector.partition-key' = 'car_id,car_owner',
'format.type' = 'csv'
);
```

2.3.2.5 JDBC Result Table

Function

DLI exports the output data of the Flink job to RDS.

Prerequisites

- An enhanced datasource connection with the database has been established, so that you can configure security group rules as required.

Syntax

```
create table jdbcSink (
  attr_name attr_type
  (,' attr_name attr_type)*
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'jdbc',
  'connector.url' = "",
  'connector.table' = "",
  'connector.driver' = "",
  'connector.username' = "",
  'connector.password' = ""
);
```

Parameters

Table 2-13 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Data source type. Set this parameter to jdbc .
connector.url	Yes	Database URL
connector.table	Yes	Name of the table where the data to be read from the database is located
connector.driver	No	Driver required for connecting to the database If you do not set this parameter, the automatically extracted URL will be used.
connector.username	No	Username for accessing the database
connector.password	No	Password for accessing the database

Parameter	Mandatory	Description
connector.write.flush.max-rows	No	Maximum number of rows to be updated when data is written. The default value is 5000 .
connector.write.flush.interval	No	Interval for data update. The unit can be ms, milli, millisecond/s, sec, second/min or minute. If this parameter is not set, the value is not updated based on the interval by default.
connector.write.max-retries	No	Maximum number of attempts to write data if failed. The default value is 3 .
connector.write.exclude-update-columns	No	Columns excluded for data update. The default value is empty, indicating that when data with the same primary key is updated, the update of the specified field is ignored. The primary key column is ignored by default.

Precautions

None

Example

Output data from stream **jdbcSink** to the MySQL database.

```
create table jdbcSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
)
with (
  'connector.type' = 'jdbc',
  'connector.url' = 'jdbc:mysql://xx.xx.xx.xx:3306/xx',
  'connector.table' = 'jdbc_table_name',
  'connector.driver' = 'com.mysql.jdbc.Driver',
  'connector.username' = 'xxx',
  'connector.password' = 'xxxxxx'
);
```

2.3.2.6 GaussDB(DWS) Result Table

Function

DLI outputs the Flink job output data to GaussDB(DWS). GaussDB(DWS) database kernel is compliant with PostgreSQL. The PostgreSQL database can store data of more complex types and delivers space information services, multi-version concurrent control (MVCC), and high concurrency. It applies to location applications, financial insurance, and e-commerce.

GaussDB(DWS) is an online data processing database based on the cloud infrastructure and platform and helps you mine and analyze massive sets of data.

Prerequisites

- Ensure that you have created a GaussDB(DWS) cluster using your account. For details about how to create a GaussDB(DWS) cluster, see "Creating a Cluster" in *Data Warehouse Service Management Guide*.
- A GaussDB(DWS) database table has been created.
- An enhanced datasource connection has been created for DLI to connect to GaussDB(DWS) clusters, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.

Syntax

```
create table dwsSink (
  attr_name attr_type
  ('; attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'gaussdb',
  'connector.url' = "",
  'connector.table' = "",
  'connector.driver' = "",
  'connector.username' = "",
  'connector.password' = ""
);
```

Parameters

Table 2-14 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to gaussdb .
connector.url	Yes	JDBC connection address. The format is jdbc:postgresql://{ip}:{port}/{dbName}.
connector.table	Yes	Name of the table to be operated. If the GaussDB(DWS) table is in a schema, the format is schema"."Table name . For details, see the Example .
connector.driver	No	JDBC connection driver. The default value is org.postgresql.Driver .
connector.username	No	Database authentication user name. This parameter must be configured in pair with connector.password .
connector.password	No	Database authentication password. This parameter must be configured in pair with connector.username .

Parameter	Mandatory	Description
connector.write.mode	No	Data write mode. The value can be copy , insert , or upsert . The default value is upsert . This parameter must be configured depending on primary key . <ul style="list-style-type: none"> If primary key is not configured, data can be appended in copy and insert modes. If primary key is configured, all the three modes are available. Note: GaussDB(DWS) does not support the update of distribution columns. The primary keys of columns to be updated must cover all distribution columns defined in the GaussDB(DWS) table.
connector.write.flush.max-rows	No	Maximum rows allowed for data flush. If the data size exceeds the value, data flush is triggered. The default value is 5000 .
connector.write.flush.interval	No	Data flush period. Data flush is triggered periodically. The format is {length value}{time unit label}, for example, 123ms , 321s . The supported time units include d, h, min, s, and ms (default unit). If this parameter is not set, the value is not updated based on the interval by default.
connector.write.max-retries	No	Maximum number of attempts to write data. The default value is 3 .
connector.write.merge.filter-key	No	Column to be merged. This parameter takes effects only when PRIMARY KEY is configured and connector.write.mode is set to copy .
connector.write.escape-string-value	No	Whether to escape values of the string type. The default value is false .

Precautions

None

Example

- If you use the gsjdbc4 driver for connection, set **connector.driver** to **org.postgresql.Driver**. You can omit this parameter because the gsjdbc4 driver is the default one.
 - Write data to GaussDB(DWS) in **upsert** mode.

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
```

```
) with (  
  'connector.type' = 'gaussdb',  
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',  
  'connector.table' = 'car_info',  
  'connector.username' = 'xx',  
  'connector.password' = 'xx',  
  'connector.write.mode' = 'upsert',  
  'connector.write.flush.interval' = '30s'  
);
```

Create table **dwsSource** with data fetched from GaussDB(DWS) table **test** that is in a schema named **ads_game_sdk_base**:

```
CREATE TABLE ads_rpt_game_sdk_realtime_ada_reg_user_pay_mm (  
  ddate DATE,  
  dmin TIMESTAMP(3),  
  game_appkey VARCHAR,  
  channel_id VARCHAR,  
  pay_user_num_1m bigint,  
  pay_amt_1m bigint,  
  PRIMARY KEY (ddate, dmin, game_appkey, channel_id) NOT ENFORCED  
) WITH (  
  'connector.type' = 'gaussdb',  
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/dws_bigdata_db',  
  'connector.table' = 'ads_game_sdk_base"."test',  
  'connector.username' = 'xxxx',  
  'connector.password' = 'xxxx',  
  'connector.write.mode' = 'upsert',  
  'connector.write.flush.interval' = '30s'  
);
```

- If you use the gsjdbc200 driver for connection, set **connector.driver** to **com.huawei.gauss200.jdbc.Driver**.

Create table **dwsSource** with data fetched from GaussDB(DWS) table **test** that is in a schema named **ads_game_sdk_base**:

```
create table dwsSink(  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_speed INT  
) with (  
  'connector.type' = 'gaussdb',  
  'connector.table' = 'ads_game_sdk_base"."test',  
  'connector.driver' = 'com.huawei.gauss200.jdbc.Driver',  
  'connector.url' = 'jdbc:gaussdb://xx.xx.xx.xx:8000/xx',  
  'connector.username' = 'xx',  
  'connector.password' = 'xx',  
  'connector.write.mode' = 'upsert',  
  'connector.write.flush.interval' = '30s'  
);
```

2.3.2.7 Redis Result Table

Function

DLI exports the output data of the Flink job to Redis. Redis is a storage system that supports multiple types of data structures such as key-value. It can be used in scenarios such as caching, event pub/sub, and high-speed queuing. Redis supports direct read/write of strings, hashes, lists, queues, and sets. Redis works with in-memory dataset and provides persistence. For more information about Redis, visit <https://redis.io/>.

Prerequisites

An enhanced datasource connection with Redis has been established, so that you can configure security group rules as required.

Syntax

```
create table dwsSink (
  attr_name attr_type
  (' attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'redis',
  'connector.host' = "",
  'connector.port' = "",
  'connector.password' = "",
  'connector.table-name' = "",
  'connector.key-column' = ""
);
```

Parameters

Table 2-15 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to redis .
connector.host	Yes	Redis connector address
connector.port	Yes	Redis connector port
connector.password	No	Redis authentication password
connector.deploy-mode	No	Redis deployment mode. The value can be standalone or cluster . The default value is standalone .
connector.table-name	No	Name of the table stored in the Redis. This parameter is mandatory in the Redis Hashmap storage pattern. In this pattern, data is stored to Redis in hashmaps. The hash key is \$(table-name):\$(ext-key) , and the field name is the column name. NOTE Table storage pattern: connector.table-name and connector.key-column are used as Redis keys. For the Redis hash type, each key corresponds to a hashmap. A hash key is a field name of the source table, and a hash value is a field value of the source table.

Parameter	Mandatory	Description
connector.key-column	No	This parameter is optional in table storage pattern. The value is used as the value of ext-key in the Redis. If this parameter is not set, the value of ext-key is the generated UUID.
connector.write-schema	No	Whether to write the current schema to the Redis. This parameter is available in table storage pattern. The default value is false .
connector.data-type	No	Data types for storage. This parameter is mandatory for a custom storage pattern. Supported values include string, list, hash, and set. In a string, list or set, the number of schema fields must be 2, and the number of hash fields must be 3.
connector.ignore-retraction	No	Whether to ignore the retraction message. The default value is false .

Precautions

Either **connector.table-name** or **connector.data-type** must be set.

Example

- Configure the table storage pattern when you configure **connector.table-name**.

In table storage pattern, data is stored in hash mode, which is different from the basic hash pattern in which the three fields of a table are used as the **key**, **hash_key**, and **hash_value**. The key in table pattern can be specified by **connector.table-name** and **connector.key-column** parameters, all field names in the table are used as **hash_key**, and the field values are written to the hash table as **hash_value**.

```
create table redisSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'redis',
  'connector.host' = 'xx.xx.xx.xx',
  'connector.port' = '6379',
  'connector.password' = 'xx',
  'connector.table-name'='car_info',
  'connector.key-column'='car_id'
);

insert into redisSink
(car_id,car_owner,car_brand,car_speed)
VALUES
("A1234","OwnA","A1234",30);
```

- The following example shows how to create a table when **connector.data-type** is set to **string**, **list**, **hash**, or **set**, respectively.

- String type

The table contains two columns: key and value.

```
create table redisSink(  
  attr1 STRING,  
  attr2 STRING  
) with (  
  'connector.type' = 'redis',  
  'connector.host' = 'xx.xx.xx.xx',  
  'connector.port' = '6379',  
  'connector.password' = 'xx',  
  'connector.data-type' = 'string'  
);  
  
insert into redisSink  
  (attr1,attr2)  
VALUES  
  ("car_id","A1234");
```

- List type

The table contains two columns: key and value.

```
create table redisSink(  
  attr1 STRING,  
  attr2 STRING  
) with (  
  'connector.type' = 'redis',  
  'connector.host' = 'xx.xx.xx.xx',  
  'connector.port' = '6379',  
  'connector.password' = 'xx',  
  'connector.data-type' = 'list'  
);  
  
insert into redisSink  
  (attr1,attr2)  
VALUES  
  ("car_id","A1234");
```

- Set type

The table contains two columns: key and value.

```
create table redisSink(  
  attr1 STRING,  
  attr2 STRING  
) with (  
  'connector.type' = 'redis',  
  'connector.host' = 'xx.xx.xx.xx',  
  'connector.port' = '6379',  
  'connector.password' = 'xx',  
  'connector.data-type' = 'set'  
);  
  
insert into redisSink  
  (attr1,attr2)  
VALUES  
  ("car_id","A1234");
```

- Hash type

The table contains three columns: key, hash_key, and hash_value.

```
create table redisSink(  
  attr1 STRING,  
  attr2 STRING,  
  attr3 STRING  
) with (  
  'connector.type' = 'redis',  
  'connector.host' = 'xx.xx.xx.xx',  
  'connector.port' = '6379',  
  'connector.password' = 'xx',
```

```
'connector.data-type' = 'hash'
);

insert into redisSink
(attr1,attr2,attr3)
VALUES
("car_info","car_id","A1234");
```

2.3.2.8 SMN Result Table

Function

DLI exports Flink job output data to SMN.

SMN provides reliable and flexible large-scale message notification services to DLI. It significantly simplifies system coupling and pushes messages to subscription endpoints based on requirements. SMN can be connected to other cloud services or integrated with any application that uses or generates message notifications to push messages over multiple protocols.

Syntax

```
create table smnSink (
  attr_name attr_type
  (' attr_name attr_type)*
  ('PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'smn',
  'connector.region' = "",
  'connector.topic-urn' = "",
  'connector.message-subject' = "",
  'connector.message-column' = ""
);
```

Parameters

Table 2-16 Parameter description

Parameter	Man dato ry	Description
connector.type	Yes	Sink data type. Set this parameter to smn , which means that data is stored to SMN.
connector.region	Yes	Region where SMN belongs
connector.topic-urn	No	URN of an SMN topic, which is used for the static topic URN configuration. The SMN topic serves as the destination for short message notification and needs to be created in SMN. Either of topic_urn and urn_column must be configured. If both of them are configured, the topic_urn setting takes precedence.

Parameter	Mandatory	Description
connector.urn-column	No	Field name of the topic URN content, which is used for the dynamic topic URN configuration. One of topic_urn and urn_column must be configured. If both of them are configured, the topic_urn setting takes precedence.
connector.message-subject	Yes	Message subject sent by SMN. This parameter can be customized.
connector.message-column	Yes	Column name in the current table. Data in this column is the message content and is customized. Currently, only text messages are supported.

Precautions

None

Example

Write the data to the target of SMN topic. The topic of the message sent by SMN is **test**, and the message content is the data in the **attr1** column.

```
create table smnSink (
  attr1 STRING,
  attr2 STRING
)
with (
  'connector.type' = 'smn',
  'connector.region' = '',
  'connector.topic-urn' = 'xxxxxx',
  'connector.message-subject' = 'test',
  'connector.message-column' = 'attr1'
);
```

2.3.2.9 HBase Result Table

Function

DLI outputs the job data to HBase. HBase is a column-oriented distributed cloud storage system that features enhanced reliability, excellent performance, and elastic scalability. It applies to the storage of massive amounts of data and distributed computing. You can use HBase to build a storage system capable of storing TB- or even PB-level data. With HBase, you can filter and analyze data with ease and get responses in milliseconds, rapidly mining data value. Structured and semi-structured key-value data can be stored, including messages, reports, recommendation data, risk control data, logs, and orders. With DLI, you can write massive volumes of data to HBase at a high speed and with low latency.

Prerequisites

An enhanced datasource connection has been created for DLI to connect to HBase, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.

- **If MRS HBase is used, IP addresses of all hosts in the MRS cluster have been added to host information of the enhanced datasource connection.**

Syntax

```
create table hbaseSink (
  attr_name attr_type
  (' attr_name attr_type)*
)
with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = '',
  'connector.zookeeper.quorum' = ''
);
```

Parameters

Table 2-17 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to hbase .
connector.version	Yes	The value must be 1.4.3 .
connector.table-name	Yes	HBase table name
connector.zookeeper.quorum	Yes	ZooKeeper address
connector.zookeeper.znode.parent	No	Root directory for ZooKeeper. The default value is /hbase .
connector.write.buffer-flush.max-size	No	Maximum buffer size for each data write. The default value is 2 MB. The unit is MB.
connector.write.buffer-flush.max-rows	No	Maximum number of data records that can be updated each time

Parameter	Mandatory	Description
connector.write.buffer-flush.interval	No	Update time. The default value is 0s . Example value: 2s .
connector.rowkey	No	Content of a compound rowkey to be assigned. The content is assigned to a new field based on the configuration. Example: rowkey1:3,rowkey2:3, ... The value 3 indicates the first three bytes of the field. The number cannot be greater than the byte size of the field and cannot be less than 1.

Example

```
create table hbaseSink(
  rowkey string,
  name string,
  i Row<gender string, age int>,
  j Row<address string>
) with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = 'sink',
  'connector.rowkey' = 'rowkey:1,name:3',
  'connector.write.buffer-flush.max-rows' = '5',
  'connector.zookeeper.quorum' = 'xxx:2181'
);
```

2.3.2.10 Elasticsearch Result Table

Function

DLI exports Flink job output data to Elasticsearch of Cloud Search Service (CSS). Elasticsearch is a popular enterprise-class Lucene-powered search server and provides the distributed multi-user capabilities. It delivers multiple functions, including full-text retrieval, structured search, analytics, aggregation, and highlighting. With Elasticsearch, you can achieve stable, reliable, real-time search. Elasticsearch applies to diversified scenarios, such as log analysis and site search.

CSS is a fully managed, distributed search service. It is fully compatible with open-source Elasticsearch and provides DLI with structured and unstructured data search, statistics, and report capabilities. For more information about CSS, see .

Prerequisites

- Ensure that you have created a cluster on CSS using your account.
If you need to access Elasticsearch using the cluster username and password, enable the security mode and disable HTTPS for the created CSS cluster.

- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with CSS. You can also set the security group rules as required.

Precautions

- Currently, only CSS 7.X and later versions are supported. Version 7.6.2 is recommended.
- Do not enable the security mode for the CSS cluster if **connector.username** and **connector.password** are not configured.
- ICMP must be enabled for the security group inbound rule of the CSS cluster.

Syntax

```
create table esSink (  
  attr_name attr_type  
  (,' attr_name attr_type)*  
  (,'PRIMARY KEY (attr_name, ...) NOT ENFORCED)  
)  
with (  
  'connector.type' = 'elasticsearch',  
  'connector.version' = '7',  
  'connector.hosts' = 'http://xxx:9200',  
  'connector.index' = '',  
  'connector.document-type' = '',  
  'update-mode' = '',  
  'format.type' = 'json'  
);
```

Parameters

Table 2-18 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to elasticsearch .
connector.version	Yes	Elasticsearch version Currently, only version 7 can be used. That is, the value of this parameter can only be 7 .
connector.hosts	Yes	Host name of the cluster where Elasticsearch locates. Use semicolons (;) to separate multiple host names. Ensure that the host name starts with http , for example, http://x.x.x.x:9200 .
connector.index	Yes	Elasticsearch index name
connector.document-type	Yes	Elasticsearch type name This attribute is invalid because Elasticsearch 7 uses the default _doc type.
update-mode	Yes	Data update mode of the sink. The value can be append or upsert .

Parameter	Mandatory	Description
connector.key-delimiter	No	Delimiter of compound primary keys. The default value is <code>_</code> .
connector.key-null-literal	No	Character used to replace null in keys.
connector.failure-handler	No	Policy used when an Elasticsearch request fails. The default value is fail . fail : An exception is thrown when the request fails and the job fails. ignore : The failed request is ignored. retry-rejected : If the request fails because the queue running the Elasticsearch node is full, the request is resent and no failure is reported. custom : A custom policy is used.
connector.failure-handler-class	No	Custom processing mode used to handle a failure
connector.flush-on-checkpoint	No	Whether the connector waits for all pending action requests to be acknowledged by Elasticsearch on checkpoints. The default value true indicates that wait for all pending action requests on checkpoints. If you set this parameter to false, the connector will not wait for the requests.
connector.bulk-flush.max-actions	No	Maximum number of records that can be written in a batch
connector.bulk-flush.max-size	No	Maximum total amount of data to be written in batches. Specify the unit when you configure this parameter. The unit is MB.
connector.bulk-flush.interval	No	Update interval for batch writing. The unit is milliseconds and is not required.
format.type	Yes	Data format. Currently, only JSON is supported.
connector.username	No	Account of the cluster where Elasticsearch locates. This parameter and must be configured in pair with connector.password . If the account and password are used, the security mode must be enabled and HTTPS must be disabled for the created CSS cluster.
connector.password	No	Password of the cluster where Elasticsearch locates. This parameter must be configured in pair with connector.username .

Example

```
create table sink1(  
  attr1 string,  
  attr2 int  
) with (  
  'connector.type' = 'elasticsearch',  
  'connector.version' = '7',  
  'connector.hosts' = 'http://xxx:9200',  
  'connector.index' = 'es',  
  'connector.document-type' = 'one',  
  'update-mode' = 'append',  
  'format.type' = 'json'  
);
```

2.3.2.11 OpenTSDB Result Table

Function

OpenTSDB is a distributed, scalable time series database based on HBase. OpenTSDB is designed to collect monitoring information of a large-scale cluster and query data in seconds, facilitating querying and storing massive amounts of monitoring data in common databases. OpenTSDB can be used for system monitoring and measurement as well as collection and monitoring of IoT data, financial data, and scientific experimental results.

DLI uses enhanced datasource connections to write the output of Flink jobs to OpenTSDB.

Prerequisites

- The OpenTSDB service has been enabled.
- An enhanced datasource connection has been created for DLI to connect to OpenTSDB, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.

Syntax

```
create table tsdbSink (  
  attr_name attr_type  
  (' attr_name attr_type)*  
)  
with (  
  'connector.type' = 'opentsdb',  
  'connector.region' = "",  
  'connector.tsdb-metrics' = "",  
  'connector.tsdb-timestamps' = "",  
  'connector.tsdb-values' = "",  
  'connector.tsdb-tags' = "",  
  'connector.tsdb-link-address' = ""  
);
```

Parameters

Table 2-19 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to opentsdb .
connector.region	Yes	Region where OpenTSDB locates
connector.tsdb-metrics	Yes	Metrics of data points, which can be specified through parameter configurations. The number of metrics must be 1 or the same as the number of connector.tsdb-values . Use semicolons (;) to separate multiple metrics.
connector.tsdb-timestamps	Yes	Timestamps of data points. Only dynamic columns are supported. The data type can be int, bigint, or string. Only numbers are supported. The number of metrics must be 1 or the same as the number of connector.tsdb-values . Use semicolons (;) to separate multiple timestamps.
connector.tsdb-values	Yes	Values of data points. You can specify dynamic columns or constant values. Separate multiple values with semicolons (;).
connector.tsdb-tags	Yes	Tags of data points. Each tag contains at least one tag value and a maximum of eight tag values. Separate multiple tags by commas (,). You can specify the tags by parameters. The number of metrics must be 1 or the same as the number of connector.tsdb-values . Separate multiple tags with semicolons (;).
connector.batch-insert-data-num	No	Number of data records to be written in batches at a time. The value must be a positive integer. The default value is 8 .
connector.tsdb-link-address	Yes	OpenTSDB address for connecting to the cluster where the data to be inserted belongs.

Precautions

- If your OpenTSDB runs in an MRS cluster, ensure that:

- a. The IP address and port number of OpenTSDB must be obtained from **tsd.network.bind** and **tsd.network.port** in the OpenTSDB service configuration.
 - b. If **tsd.https.enabled** is set to **true**, the value format of **connector.tsd-link-address** in the SQL statement is **https://ip:port**. If **tsd.https.enabled** is set to **false**, the value of **connector.tsd-link-address** can be in the format of **http://ip:port** or **ip:port**.
 - c. When establishing an enhanced datasource connection, you need to add the mapping between MRS cluster hosts and IP addresses in **/etc/hosts** to the Host Information parameter.
- If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_\${car_brand}** and the value of **car_brand** in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.
 - If dynamic columns are supported, the format must be **\${columnName}**, where **columnName** indicates a field name.

Example

```
create table sink1(  
  attr1 bigint,  
  attr2 int,  
  attr3 int  
) with (  
  'connector.type' = 'opentsdb',  
  'connector.region' = '',  
  'connector.tsd-metrics' = '',  
  'connector.tsd-timestamps' = '${attr1}',  
  'connector.tsd-values' = '${attr2};10',  
  'connector.tsd-tags' = 'key1:value1,key2:value2;key3:value3',  
  'connector.tsd-link-address' = ''  
);
```

2.3.2.12 User-defined Result Table

Function

Write your Java code to insert the processed data into a specified database supported by your cloud service.

Prerequisites

Implement the custom sink class :

The custom sink class is inherited from Flink open-source class **RichSinkFunction**. The data type is **Tuple2<Boolean, Row>**.

For example, define the **MySink** class by **public class MySink extends RichSinkFunction< Tuple2<Boolean, Row>> {}**, and implement the **open**, **invoke**, and **close** functions. A code example is as follows:

```
public class MySink extends RichSinkFunction<Tuple2<Boolean, Row>> {  
  // Initialize the object.  
  @Override  
  public void open(Configuration parameters) throws Exception {}  
  
  @Override
```

```
// Implement the data processing logic.  
/* The in parameter contains two values. The first value is of the Boolean type. The value true indicates  
the insert or update operation, and the value false indicates the delete operation. If the interconnected sink  
does not support the delete operation, the deletion will not be executed. The second value indicates the  
data to be operated.*/  
public void invoke(Tuple2<Boolean, Row> in, Context context) throws Exception {}  
  
@Override  
public void close() throws Exception {}  
}
```

Content of the dependent pom configuration file is as follows:

```
<dependency>  
  <groupId>org.apache.flink</groupId>  
  <artifactId>flink-streaming-java_2.11</artifactId>  
  <version>${flink.version}</version>  
  <scope>provided</scope>  
</dependency>  
  
<dependency>  
  <groupId>org.apache.flink</groupId>  
  <artifactId>flink-core</artifactId>  
  <version>${flink.version}</version>  
  <scope>provided</scope>  
</dependency>
```

Pack the implemented class and compile it in a JAR file, and upload it using the UDF Jar parameter on the editing page of your Flink OpenSource SQL job.

Syntax

```
create table userDefinedSink (  
  attr_name attr_type  
  (' attr_name attr_type)*  
)  
with (  
  'connector.type' = 'user-defined',  
  'connector.class-name' = "  
);
```

Parameters

Table 2-20 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. The value can only be a user-defined sink.
connector.class-name	Yes	Fully qualified class name of the sink class. For details about the implementation of the sink class, see Prerequisites .
connector.class-parameter	No	Parameter of the constructor of the sink class. Only one parameter of the string type is supported.

Precautions

connector.class-name must be a fully qualified class name.

Example

```
create table userDefinedSink (
  attr1 int,
  attr2 int
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'xx.xx.MySink'
);
```

2.3.2.13 Print Result Table

Function

The print connector exports your data output to the **error** file or the **out** file of TaskManager. It is mainly used for code debugging and output viewing.

Syntax

```
create table printSink (
  attr_name attr_type (',' attr_name attr_type) * (',' PRIMARY KEY (attr_name,...) NOT ENFORCED)
) with (
  'connector' = 'print',
  'print-identifier' = "",
  'standard-error' = ""
);
```

Parameters

Table 2-21 Parameter description

Parameter	Mandatory	Description
connector	Yes	The value is fixed to print .
print-identifier	No	Message that identifies print and is prefixed to the output of the value.
standard-error	No	The value can be only true or false . The default value is false . <ul style="list-style-type: none"> If the value is true, data is output to the error file of the TaskManager. If the value is false, data is output to the out file of the TaskManager.

Example

Read data from Kafka and export the data to the **out** file of TaskManager. You can view the output in the exported file.

```
create table kafkaSource(  
  attr0 string,  
  attr1 boolean,  
  attr3 decimal(38, 18),  
  attr4 TINYINT,  
  attr5 smallint,  
  attr6 int,  
  attr7 bigint,  
  attr8 float,  
  attr9 double,  
  attr10 date,  
  attr11 time,  
  attr12 timestamp(3)  
) with (  
  'connector.type' = 'kafka',  
  'connector.version' = '0.11',  
  'connector.topic' = 'test_json',  
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',  
  'connector.properties.group.id' = 'test_print',  
  'connector.startup-mode' = 'latest-offset',  
  'format.type' = 'csv'  
);  
  
create table printTable(  
  attr0 string,  
  attr1 boolean,  
  attr3 decimal(38,18),  
  attr4 TINYINT,  
  attr5 smallint,  
  attr6 int,  
  attr7 bigint,  
  attr8 float,  
  attr9 double,  
  attr10 date,  
  attr11 time,  
  attr12 timestamp(3),  
  attr13 array<string>,  
  attr14 row<attr15 float, attr16 timestamp(3)>,  
  attr17 map<int, bigint>  
) with (  
  "connector" = "print"  
);  
  
insert into  
  printTable  
select  
  attr0,  
  attr1,  
  attr3,  
  attr4,  
  attr5,  
  attr6,  
  attr7,  
  attr8,  
  attr9,  
  attr10,  
  attr11,  
  attr12,  
  array [cast(attr0 as string), cast(attr0 as string)],  
  row(  
    cast(attr8 as float),  
    cast(attr12 as timestamp(3))  
  ),  
  map [cast(attr6 as int), cast(attr7 as bigint)]
```

```
from  
kafkaSource;
```

2.3.2.14 File System Result Table

Function

You can create a file system result table to export data to a file system such as HDFS or OBS. After the data is generated, a non-DLI table can be created directly according to the generated directory. The table can be processed through DLI SQL, and the output data directory can be stored in partition tables. It is applicable to scenarios such as data dumping, big data analysis, data backup, and active, deep, or cold archiving.

Syntax

```
create table filesystemSink (  
  attr_name attr_type (',' attr_name attr_type) *  
) with (  
  'connector.type' = 'filesystem',  
  'connector.file-path' = "",  
  'format.type' = ""  
);
```

Important Notes

- If the data output directory in the table creation syntax is OBS, the directory must be a parallel file system and cannot be an OBS bucket.
- When using a file system table, you must enable checkpointing to ensure job consistency.
- When **format.type** is **parquet**, the supported data type is string, boolean, tinyint, smallint, int, bigint, float, double, map<string, string>, timestamp(3), and time.
- To avoid data loss or data coverage, you need to enable automatic restart upon job exceptions. Enable the **Restore Job from Checkpoint**.
- Set the checkpoint interval after weighing between real-time output file, file size, and recovery time, such as 10 minutes.
- When using HDFS, you need to bind the data source and enter the host information.
- When using HDFS, you need to configure information about the node where the active NameNode locates.

Parameter

Table 2-22 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	The value is fixed to filesystem .
connector.file-path	Yes	Data output directory. The format is <i>schema://file.path</i> . NOTE Currently, Schema supports only OBS and HDFS. <ul style="list-style-type: none"> If schema is set to obs, data is stored to OBS. Note that OBS directory must be a parallel file system and must not be an OBS bucket. For example, obs://bucketName/fileName indicates that data is exported to the fileName directory in the bucketName bucket. If schema is set to hdfs, data is exported to HDFS. Example: hdfs://node-master1sYAx:9820/user/car_infos, where node-master1sYAx:9820 is the name of the node where the NameNode locates.
format.type	Yes	Output data encoding format. Only parquet and csv are supported. <ul style="list-style-type: none"> When schema is set to obs, the encoding format of the output data can only be parquet. When schema is set to hdfs, the output data can be encoded in Parquet or CSV format.
format.field-delimiter	No	Delimiter used to separate every two attributes. This parameter needs to be configured if the CSV encoding format is adopted. It can be user-defined, for example, a comma (,).
connector.ak	No	Access key for accessing OBS This parameter is mandatory when data is written to OBS.
connector.sk	No	Secret key for accessing OBS This parameter is mandatory when data is written to OBS.
connector.partitioned-by	No	Partitioning field. Use commas (,) to separate multiple fields.

Example

Read data from Kafka and write the data in Parquet format to the **fileName** directory in the **bucketName** bucket.

```
create table kafkaSource(  
  attr0 string,  
  attr1 boolean,  
  attr2 TINYINT,  
  attr3 smallint,  
  attr4 int,  
  attr5 bigint,  
  attr6 float,  
  attr7 double,  
  attr8 timestamp(3),  
  attr9 time  
) with (  
  'connector.type' = 'kafka',  
  'connector.version' = '0.11',  
  'connector.topic' = 'test_json',  
  'connector.properties.bootstrap.servers' = 'xx.xx.xx:9092',  
  'connector.properties.group.id' = 'test_filesystem',  
  'connector.startup-mode' = 'latest-offset',  
  'format.type' = 'csv'  
);  
  
create table filesystemSink(  
  attr0 string,  
  attr1 boolean,  
  attr2 TINYINT,  
  attr3 smallint,  
  attr4 int,  
  attr5 bigint,  
  attr6 float,  
  attr7 double,  
  attr8 map < string, string >,  
  attr9 timestamp(3),  
  attr10 time  
) with (  
  "connector.type" = "filesystem",  
  "connector.file-path" = "obs://bucketName/fileName",  
  "format.type" = "parquet",  
  "connector.ak" = "xxxx",  
  "connector.sk" = "xxxxxx"  
);  
  
insert into  
  filesystemSink  
select  
  attr0,  
  attr1,  
  attr2,  
  attr3,  
  attr4,  
  attr5,  
  attr6,  
  attr7,  
  map [attr0,attr0],  
  attr8,  
  attr9  
from  
  kafkaSource;
```

2.3.3 Creating a Dimension Table

2.3.3.1 JDBC Dimension Table

Create a JDBC dimension table to connect to the source stream.

Prerequisites

- You have created a JDBC instance for your account.

Syntax

```
CREATE TABLE table_id (  
  attr_name attr_type  
  (' attr_name attr_type)*  
)  
WITH (  
  'connector.type' = 'jdbc',  
  'connector.url' = "",  
  'connector.table' = "",  
  'connector.username' = "",  
  'connector.password' = ""  
);
```

Parameters

Table 2-23 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Data source type. Set this parameter to jdbc .
connector.url	Yes	Database URL
connector.table	Yes	Name of the table where the data to be read from the database is located
connector.driver	No	Driver required for connecting to the database. If you do not set this parameter, the automatically extracted URL will be used.
connector.username	No	Database authentication user name. This parameter must be configured in pair with connector.password .
connector.password	No	Database authentication password. This parameter must be configured in pair with connector.username .
connector.read.partition.column	No	Name of the column used to partition the input This parameter is mandatory if connector.read.partition.lower-bound , connector.read.partition.upper-bound , and connector.read.partition.num are configured.

Parameter	Mandatory	Description
connector.read.partition.lower-bound	No	Lower bound of values to be fetched for the first partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.upper-bound	No	Upper bound of values to be fetched for the last partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.lower-bound , and connector.read.partition.num are configured.
connector.read.partition.num	No	Number of partitions This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.upper-bound are configured.
connector.read.fetch-size	No	Number of rows fetched from the database each time. The default value is 0 , indicating the hint is ignored.
connector.lookup.cache.max-rows	No	Maximum number of cached rows in a dimension table. If the number of cached rows exceeds the value, old data will be deleted. The value -1 indicates that data cache disabled.
connector.lookup.cache.ttl	No	Time To Live (TTL) of dimension table cache. Caches exceeding the TTL will be deleted. The format is {length value}{time unit label}, for example, 123ms , 321s . The supported time units include d, h, min, s, and ms (default unit).
connector.lookup.max-retries	No	Maximum number of attempts to obtain data from the dimension table. The default value is 3 .

Example

The RDS table is used to connect to the source stream.

```
CREATE TABLE car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
```

```
proctime as PROCTIME()
)
WITH (
  'connector.type' = 'dis',
  'connector.region' = '',
  'connector.channel' = 'disInput',
  'format.type' = 'csv'
);

CREATE TABLE db_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  'connector.type' = 'jdbc',
  'connector.url' = 'jdbc:mysql://xx.xx.xx.xx:3306/xx',
  'connector.table' = 'jdbc_table_name',
  'connector.driver' = 'com.mysql.jdbc.Driver',
  'connector.username' = 'xxx',
  'connector.password' = 'xxxxx'
);

CREATE TABLE audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  'connector.type' = 'dis',
  'connector.region' = '',
  'connector.channel' = 'disOutput',
  'connector.partition-key' = 'car_id,car_owner',
  'format.type' = 'csv'
);

INSERT INTO audi_cheaper_than_30w
SELECT a.car_id, b.car_owner, b.car_brand, b.car_price
FROM car_infos as a join db_info FOR SYSTEM_TIME AS OF a.proctime AS b on a.car_id = b.car_id;
```

2.3.3.2 GaussDB(DWS) Dimension Table

Create a GaussDB(DWS) dimension table to connect to the input stream.

Prerequisites

- You have created a GaussDB(DWS) instance for your account.

Syntax

```
create table dwsSource (
  attr_name attr_type
  (' attr_name attr_type)*
)
with (
  'connector.type' = 'gaussdb',
  'connector.url' = '',
  'connector.table' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

Parameters

Table 2-24 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to gaussdb .
connector.url	Yes	JDBC connection address. The format is jdbc:postgresql://\${ip}:\${port}/\${dbName} .
connector.table	Yes	Name of the table where the data to be read from the database is located
connector.driver	No	JDBC connection driver. The default value is org.postgresql.Driver .
connector.username	No	Database authentication user name. This parameter must be configured in pair with connector.password .
connector.password	No	Database authentication password. This parameter must be configured in pair with connector.username .
connector.read.partition.column	No	Name of the column used to partition the input This parameter is mandatory if connector.read.partition.lower-bound , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.lower-bound	No	Lower bound of values to be fetched for the first partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.num are configured.
connector.read.partition.upper-bound	No	Upper bound of values to be fetched for the last partition This parameter is mandatory if connector.read.partition.column , connector.read.partition.lower-bound , and connector.read.partition.num are configured.
connector.read.partition.num	No	Number of partitions This parameter is mandatory if connector.read.partition.column , connector.read.partition.upper-bound , and connector.read.partition.upper-bound are configured.

Parameter	Mandatory	Description
connector.read.fetch-size	No	Number of rows fetched from the database each time. The default value is 0 , indicating the hint is ignored.
connector.lookup.cache.max-rows	No	Maximum number of cached rows in a dimension table. If the number of cached rows exceeds the value, old data will be deleted. The value -1 indicates that data cache disabled.
connector.lookup.cache.ttl	No	Time To Live (TTL) of dimension table cache. Caches exceeding the TTL will be deleted. The format is {length value}{time unit label}, for example, 123ms , 321s . The supported time units include d, h, min, s, and ms (default unit).
connector.lookup.max-retries	No	Maximum number of attempts to obtain data from the dimension table. The default value is 3 .

Example

Use an RDS table to connect to the source stream.

```
CREATE TABLE car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  proctime as PROCTIME()
)
WITH (
  'connector.type' = 'dis',
  'connector.region' = '',
  'connector.channel' = 'disInput',
  'format.type' = 'csv'
);

CREATE TABLE db_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  'connector.type' = 'gaussdb',
  'connector.driver' = 'org.postgresql.Driver',
  'connector.url' = 'jdbc:gaussdb://xx.xx.xx.xx:8000/xx',
  'connector.table' = 'car_info',
  'connector.username' = 'xx',
  'connector.password' = 'xx',
  'connector.lookup.cache.max-rows' = '10000',
  'connector.lookup.cache.ttl' = '24h'
);

CREATE TABLE audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
```

```

car_price INT
)
WITH (
'connector.type' = 'dis',
'connector.region' = "",
'connector.channel' = 'disOutput',
'connector.partition-key' = 'car_id,car_owner',
'format.type' = 'csv'
);

INSERT INTO audi_cheaper_than_30w
SELECT a.car_id, b.car_owner, b.car_brand, b.car_price
FROM car_infos as a join db_info FOR SYSTEM_TIME AS OF a.proctime AS b on a.car_id = b.car_id;

```

2.3.3.3 HBase Dimension Table

Function

Create a Hbase dimension table to connect to the source stream.

Prerequisites

- An enhanced datasource connection has been created for DLI to connect to HBase, so that jobs can run on the dedicated queue of DLI and you can set the security group rules as required.
- **If MRS HBase is used, IP addresses of all hosts in the MRS cluster have been added to host information of the enhanced datasource connection.**

Syntax

```

create table hbaseSource (
attr_name attr_type
(,' attr_name attr_type)*
)
with (
'connector.type' = 'hbase',
'connector.version' = '1.4.3',
'connector.table-name' = "",
'connector.zookeeper.quorum' = ""
);

```

Parameters

Table 2-25 Parameter description

Parameter	Mandatory	Description
connector.type	Yes	Connector type. Set this parameter to hbase .
connector.version	Yes	The value must be 1.4.3 .
connector. table-name	Yes	Table name in HBase

Parameter	Mandatory	Description
connector.zookeeper.quorum	Yes	ZooKeeper address
connector.zookeeper.znode.parent	No	Root directory for ZooKeeper. The default value is / hbase .

Example

```

create table hbaseSource(
  id string,
  i Row<score string>
) with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = 'user',
  'connector.zookeeper.quorum' = 'xxx:2181'
);

create table source1(
  id string,
  name string,
  geneder string,
  age int,
  address string,
  proctime as PROCTIME()
) with (
  "connector.type" = "dis",
  "connector.region" = "",
  "connector.channel" = "read",
  "connector.ak" = "xxxxxx",
  "connector.sk" = "xxxxxx",
  "format.type" = 'csv'
);

create table hbaseSink(
  rowkey string,
  i Row<name string, geneder string, age int, address string>,
  j ROW<score string>
) with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = 'score',
  'connector.write.buffer-flush.max-rows' = '1',
  'connector.zookeeper.quorum' = 'xxx:2181'
);
insert into hbaseSink select d.id, ROW(name, geneder,age,address), ROW(score) from source1 as d join
hbaseSource for system_time as of d.proctime as h on d.id = h.id;

```

2.4 Data Manipulation Language (DML)

2.4.1 SELECT

SELECT

Syntax


```
SELECT [ ALL | DISTINCT ]  
{ * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ]
```

Description

This clause is used to select data from a table.

ALL indicates that all results are returned.

DISTINCT indicates that the duplicated results are removed.

Precautions

- The to-be-queried table must exist. Otherwise, an error is reported.
- WHERE is used to specify the filtering condition, which can be the arithmetic operator, relational operator, or logical operator.
- GROUP BY is used to specify the grouping field, which can be one or more multiple fields.

Example

Select the order which contains more than 3 pieces of data.

```
insert into temp SELECT * FROM Orders WHERE units > 3;
```

Insert a group of constant data.

```
insert into temp select 'Lily', 'male', 'student', 17;
```

WHERE Filtering Clause

Syntax

```
SELECT { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]
```

Description

This clause is used to filter the query results using the WHERE clause.

Precautions

- The to-be-queried table must exist.
- WHERE filters the records that do not meet the requirements.

Example

Filter orders which contain more than 3 pieces and fewer than 10 pieces of data.

```
insert into temp SELECT * FROM Orders  
WHERE units > 3 and units < 10;
```

HAVING Filtering Clause

Function

This clause is used to filter the query results using the HAVING clause.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ]
```

Description

Generally, HAVING and GROUP BY are used together. GROUP BY applies first for grouping and HAVING then applies for filtering. The arithmetic operation and aggregate function are supported by the HAVING clause.

Precautions

If the filtering condition is subject to the query results of GROUP BY, the HAVING clause, rather than the WHERE clause, must be used for filtering.

Example

Group the **student** table according to the **name** field and filter the records in which the maximum score is higher than 95 based on groups.

```
insert into temp SELECT name, max(score) FROM student  
GROUP BY name  
HAVING max(score) >95;
```

Column-Based GROUP BY

Function

This clause is used to group a table based on columns.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]
```

Description

Column-based GROUP BY can be categorized into single-column GROUP BY and multi-column GROUP BY.

- Single-column GROUP BY indicates that the GROUP BY clause contains only one column.
- Multi-column GROUP BY indicates that the GROUP BY clause contains multiple columns. The table will be grouped according to all fields in the GROUP BY clause. The records whose fields are the same are grouped into one group.

Precautions

GroupBy generates update results in the stream processing table.

Example

Group the **student** table according to the score and name fields and return the grouping results.

```
insert into temp SELECT name,score, max(score) FROM student
GROUP BY name,score;
```

Expression-Based GROUP BY

Function

This clause is used to group a table according to expressions.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
```

Description

groupItem can have one or more fields. The fields can be called by string functions, but cannot be called by aggregate functions.

Precautions

None

Example

Use the substring function to obtain the character string from the name field, group the **student** table according to the obtained character string, and return each sub string and the number of records.

```
insert into temp SELECT substring(name,6),count(name) FROM student
GROUP BY substring(name,6);
```

Grouping sets, Rollup, Cube

Function

- The GROUP BY GROUPING SETS generates a result set equivalent to that generated by multiple simple GROUP BY UNION ALL statements. Using GROUPING SETS is more efficient.
- The ROLLUP and CUBE generate multiple groups based on certain rules and then collect statistics by group.
- The result set generated by CUBE contains all the combinations of values in the selected columns.
- The result set generated by ROLLUP contains the combinations of a certain layer structure in the selected columns.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY groupingItem]
```

Description

Values of groupingItem can be **Grouping sets(columnName [, columnName]*), Rollup(columnName [, columnName]*), and Cube(columnName [, columnName]*).**

Precautions

None

Example

Return the results generated based on **user** and **product**.

```
INSERT INTO temp SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product));
```

GROUP BY Using HAVING

Function

This statement filters a table after grouping it using the HAVING clause.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ]
```

Description

Generally, HAVING and GROUP BY are used together. GROUP BY applies first for grouping and HAVING then applies for filtering.

Precautions

- If the filtering condition is subject to the query results of GROUP BY, the HAVING clause, rather than the WHERE clause, must be used for filtering. HAVING and GROUP BY are used together. GROUP BY applies first for grouping and HAVING then applies for filtering.
- Fields used in HAVING, except for those used for aggregate functions, must exist in GROUP BY.
- The arithmetic operation and aggregate function are supported by the HAVING clause.

Example

Group the **transactions** according to **num**, use the HAVING clause to filter the records in which the maximum value derived from multiplying **price** with **amount** is higher than 5000, and return the filtered results.

```
insert into temp SELECT num, max(price*amount) FROM transactions
WHERE time > '2016-06-01'
GROUP BY num
HAVING max(price*amount)>5000;
```

2.4.2 Set Operations

UNION/UNION ALL/INTERSECT/EXCEPT

Syntax

```
query UNION [ ALL ] | Intersect | Except query
```

Description

- UNION is used to return the union set of multiple query results.
- INTERSECT is used to return the intersection of multiple query results.
- EXCEPT is used to return the difference set of multiple query results.

Precautions

- Set operation is to join tables from head to tail under certain conditions. The quantity of columns returned by each SELECT statement must be the same. Column types must be the same. Column names can be different.
- By default, the duplicate records returned by UNION are removed. The duplicate records returned by UNION ALL are not removed.

Example

Output the union set of Orders1 and Orders2 without duplicate records.

```
insert into temp SELECT * FROM Orders1
UNION SELECT * FROM Orders2;
```

IN

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }
FROM tableExpression
WHERE column_name IN (value [, value]* ) | query
```

Description

The IN operator allows multiple values to be specified in the WHERE clause. It returns true if the expression exists in the given table subquery.

Precautions

The subquery table must consist of a single column, and the data type of the column must be the same as that of the expression.

Example

Return **user** and **amount** information of the products in **NewProducts** of the **Orders** table.

```
insert into temp SELECT user, amount
FROM Orders
WHERE product IN (
  SELECT product FROM NewProducts
);
```

2.4.3 Window

GROUP WINDOW

Description

Group Window is defined in GROUP BY. One record is generated from each group. Group Window involves the following functions:

- Array functions

Table 2-26 Array functions

Grouping Window Function	Description
TUMBLE(time_attr, interval)	Defines a tumbling time window. A tumbling time window assigns rows to non-overlapping, continuous windows with a fixed duration (interval). For example, a tumbling window of 5 minutes groups rows in 5 minutes intervals. Tumbling windows can be defined on event-time (stream + batch) or processing-time (stream).
HOP(time_attr, interval, interval)	Defines a hopping time window (called sliding window in the Table API). A hopping time window has a fixed duration (second interval parameter) and hops by a specified hop interval (first interval parameter). If the hop interval is smaller than the window size, hopping windows are overlapping. Thus, rows can be assigned to multiple windows. For example, a hopping window of 15 minutes size and 5 minute hop interval assigns each row to 3 different windows of 15 minute size, which are evaluated in an interval of 5 minutes. Hopping windows can be defined on event-time (stream + batch) or processing-time (stream).
SESSION(time_attr, interval)	Defines a session time window. Session time windows do not have a fixed duration but their bounds are defined by a time interval of inactivity, i.e., a session window is closed if no event appears for a defined gap period. For example a session window with a 30 minute gap starts when a row is observed after 30 minutes inactivity (otherwise the row would be added to an existing window) and is closed if no row is added within 30 minutes. Session windows can work on event-time (stream + batch) or processing-time (stream).

Notes:

In streaming mode, the **time_attr** argument of the group window function must refer to a valid time attribute that specifies the processing time or event time of rows.

In batch mode, the **time_attr** argument of the group window function must be an attribute of type **TIMESTAMP**.

- Window auxiliary functions

The start and end timestamps of group windows as well as time attributes can be selected with the following auxiliary functions.

Table 2-27 Window auxiliary functions

Auxiliary Function	Description
TUMBLE_START(time_attr, interval) HOP_START(time_attr, interval, interval) SESSION_START(time_attr, interval)	Returns the timestamp of the inclusive lower bound of the corresponding tumbling, hopping, or session window.
TUMBLE_END(time_attr, interval) HOP_END(time_attr, interval, interval) SESSION_END(time_attr, interval)	Returns the timestamp of the exclusive upper bound of the corresponding tumbling, hopping, or session window. Note: The exclusive upper bound timestamp cannot be used as a rowtime attribute in subsequent time-based operations, such as interval joins and group window or over window aggregations.
TUMBLE_ROWTIME(time_attr, interval) HOP_ROWTIME(time_attr, interval, interval) SESSION_ROWTIME(time_attr, interval)	Returns the timestamp of the inclusive upper bound of the corresponding tumbling, hopping, or session window. The resulting attribute is a rowtime attribute that can be used in subsequent time-based operations such as interval joins and group window or over window aggregations.
TUMBLE_PROCTIME(time_attr, interval) HOP_PROCTIME(time_attr, interval, interval) SESSION_PROCTIME(time_attr, interval)	Returns a proctime attribute that can be used in subsequent time-based operations such as interval joins and group window or over window aggregations.

Note: Auxiliary functions must be called with exactly same arguments as the group window function in the GROUP BY clause.

Example

```
// Calculate the SUM every day (event time).
insert into temp SELECT name,
  TUMBLE_START(ts, INTERVAL '1' DAY) as wStart,
  SUM(amount)
FROM Orders
GROUP BY TUMBLE(ts, INTERVAL '1' DAY), name;

//Calculate the SUM every day (processing time).
insert into temp SELECT name,
  SUM(amount)
FROM Orders
GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;
```

```
//Calculate the SUM over the recent 24 hours every hour (event time).
insert into temp SELECT product,
    SUM(amount)
FROM Orders
GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

//Calculate the SUM of each session and an inactive interval every 12 hours (event time).
insert into temp SELECT name,
    SESSION_START(ts, INTERVAL '12' HOUR) AS sStart,
    SESSION_END(ts, INTERVAL '12' HOUR) AS sEnd,
    SUM(amount)
FROM Orders
GROUP BY SESSION(ts, INTERVAL '12' HOUR), name;
```

TUMBLE WINDOW Extension

Function

The extension functions of the DLI tumbling window are as follows:

- Periodical tumbling windows for lower latency
Before the tumbling window ends, the window can be periodically triggered based on the configured frequency. The compute result from the start to the current time is output, which does not affect the final output. The latest result can be viewed in each period before the window ends.
- Custom latency for higher data accuracy
You can set a latency for the end of the window. The output of the window is updated according to the configured latency each time a piece of late data reaches.

Precautions

If you use **insert** to write results into the sink, the sink must support the upsert mode.

Syntax

```
TUMBLE(time_attr, window_interval, period_interval, lateness_interval)
```

Example

If the current **time_attr** attribute column is **testtime** and the window interval is 10 seconds, the statement is as follows:

```
TUMBLE(testtime, INTERVAL '10' SECOND, INTERVAL '10' SECOND, INTERVAL '10' SECOND)
```

Description

Table 2-28 Parameter description

Parameter	Description	Format
time_attr	Event time or processing time attribute column	-
window_interval	Duration of the window	<ul style="list-style-type: none"> • Format 1: INTERVAL '10' SECOND

Parameter	Description	Format
period_interval	Frequency of periodic triggering within the window range. That is, before the window ends, the output result is updated at an interval specified by period_interval from the time when the window starts. If this parameter is not set, the periodic triggering policy is not used by default.	<p>The window interval is 10 seconds. You can change the value as needed.</p> <ul style="list-style-type: none"> Format 2: INTERVAL '10' MINUTE The window interval is 10 minutes. You can change the value as needed.
lateness_interval	<p>Time to postpone the end of the window. The system continues to collect the data that reaches the window within lateness_interval after the window ends. The output is updated for each data that reaches the window within lateness_interval.</p> <p>NOTE If the time window is for processing time, lateness_interval does not take effect.</p>	<ul style="list-style-type: none"> Format 3: INTERVAL '10' DAY The window interval is 10 days. You can change the value as needed.

 **NOTE**

Values of **period_interval** and **lateness_interval** cannot be negative numbers.

- If **period_interval** is set to **0**, periodic triggering is disabled for the window.
- If **lateness_interval** is set to **0**, the latency after the window ends is disabled.
- If neither of the two parameters is set, both periodic triggering and latency are disabled and only the regular tumbling window functions are available .
- If only the latency function needs to be used, set period_interval **INTERVAL '0' SECOND**.

OVER WINDOW

The difference between Over Window and Group Window is that one record is generated from one row in Over Window.

Syntax

```
SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  ROWS
  BETWEEN (UNBOUNDED|rowCount) PRECEDING AND CURRENT ROW FROM TABLENAME
```

```
SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  RANGE
  BETWEEN (UNBOUNDED|timeInterval) PRECEDING AND CURRENT ROW FROM TABLENAME
```

Description

Table 2-29 Parameter description

Parameter	Parameter Description
PARTITION BY	Indicates the primary key of the specified group. Each group separately performs calculation.
ORDER BY	Indicates the processing time or event time as the timestamp for data.
ROWS	Indicates the count window.
RANGE	Indicates the time window.

Precautions

- All aggregates must be defined in the same window, that is, in the same partition, sort, and range.
- Currently, only windows from PRECEDING (unbounded or bounded) to CURRENT ROW are supported. The range described by FOLLOWING is not supported.
- ORDER BY must be specified for a single time attribute.

Example

```
// Calculate the count and total number from syntax rules enabled to now (in proctime).
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt2
FROM Orders;

//Calculate the count and total number of the recent four records (in proctime).
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) as cnt2
FROM Orders;

//Calculate the count and total number last 60s (in eventtime). Process the events based on event time,
which is the timeattr field in Orders.
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND PRECEDING AND CURRENT ROW) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND PRECEDING AND CURRENT ROW) as cnt2
FROM Orders;
```

2.4.4 JOIN

Equi-join

Syntax

```
FROM tableExpression INNER | LEFT | RIGHT | FULL JOIN tableExpression
ON value11 = value21 [ AND value12 = value22]
```

Precautions

- Currently, only equi-joins are supported, for example, joins that have at least one conjunctive condition with an equality predicate. Arbitrary cross or theta joins are not supported.
- Tables are joined in the order in which they are specified in the FROM clause. Make sure to specify tables in an order that does not yield a cross join (Cartesian product), which are not supported and would cause a query to fail.
- For streaming queries the required state to compute the query result might grow infinitely depending on the type of aggregation and the number of distinct grouping keys. Provide a query configuration with valid retention interval to prevent excessive state size.

Example

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

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;
```

Time-Windowed Join

Function

Each piece of data in a stream is joined with data in different time zones in another stream.

Syntax

```
from t1 JOIN t2 ON t1.key = t2.key AND TIMEBOUND_EXPRESSION
```

Description

TIMEBOUND_EXPRESSION can be in either of the following formats:

- L.time between LowerBound(R.time) and UpperBound(R.time)
- R.time between LowerBound(L.time) and UpperBound(L.time)
- Comparison expression with the time attributes (L.time/R.time)

Precautions

A time window join requires at least one equi join predicate and a join condition that limits the time of both streams.

For example, use two range predicates (<, <=, >=, or >), a BETWEEN predicate, or an equal predicate that compares the same type of time attributes (such as processing time and event time) in two input tables.

For example, the following predicate is a valid window join condition:

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

Example

Join all orders shipped within 4 hours with their associated shipments.

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

Array Expansion

Precautions

This clause is used to return a new row for each element in the given array. Unnesting WITH ORDINALITY is not yet supported.

Example

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

User-Defined Table Functions

Function

This clause is used to join a table with the results of a table function. Each row of the left (outer) table is joined with all rows produced by the corresponding call of the table function.

Precautions

A left outer join against a lateral table requires a TRUE literal in the ON clause.

Example

The row of the left (outer) table is dropped, if its table function call returns an empty result.

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

If a table function call returns an empty result, the corresponding outer row is preserved, and the result padded with null values.

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

Temporal Table Function Join

Function

Precautions

Currently only inner join and left outer join with temporal tables are supported.

Example

Assuming Rates is a temporal table function, the join can be expressed in SQL as follows:

```
SELECT
  o_amount, r_rate
FROM
```

```
Orders,  
LATERAL TABLE (Rates(o_proctime))  
WHERE  
r_currency = o_currency;
```

Join Temporal Tables

Function

This clause is used to join the Temporal table.

Syntax

```
SELECT column-names  
FROM table1 [AS <alias1>]  
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.proctime [AS <alias2>]  
ON table1.column-name1 = table2.key-name1
```

Description

- **table1.proctime** indicates the processing time attribute (computed column) of **table1**.
- **FOR SYSTEM_TIME AS OF table1.proctime** indicates that when the records in the left table are joined with the dimension table on the right, only the snapshot data is used for matching the current processing time dimension table.

Precautions

Only inner and left joins are supported for temporal tables with processing time attributes.

Example

LatestRates is a temporal table that is materialized with the latest rate.

```
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;
```

2.4.5 OrderBy & Limit

OrderBy

Function

This clause is used to sort data in ascending order on a time attribute.

Precautions

Currently, only sorting by time attribute is supported.

Example

Sort data in ascending order on the time attribute.

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

Limit

Function

This clause is used to constrain the number of rows returned.

Precautions

This clause is used in conjunction with ORDER BY to ensure that the results are deterministic.

Example

```
SELECT *  
FROM Orders  
ORDER BY orderTime  
LIMIT 3;
```

2.4.6 Top-N

Function

Top-N queries ask for the N smallest or largest values ordered by columns. Both smallest and largest values sets are considered Top-N queries. Top-N queries are useful in cases where the need is to display only the N bottom-most or the N top-most records from batch/streaming table on a condition.

Syntax

```
SELECT [column_list]  
FROM (  
  SELECT [column_list],  
    ROW_NUMBER() OVER ([PARTITION BY col1 [, col2...]]  
    ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum  
  FROM table_name)  
WHERE rownum <= N [AND conditions]
```

Description

- ROW_NUMBER(): Allocate a unique and consecutive number to each line starting from the first line in the current partition. Currently, we only support ROW_NUMBER as the over window function. In the future, we will support RANK() and DENSE_RANK().
- PARTITION BY col1 [, col2...]: Specifies the partition columns. Each partition will have a Top-N result.
- ORDER BY col1 [asc|desc][, col2 [asc|desc]...]: Specifies the ordering columns. The ordering directions can be different on different columns.
- WHERE rownum <= N: The rownum <= N is required for Flink to recognize this query is a Top-N query. The N represents the N smallest or largest records will be retained.
- [AND conditions]: It is free to add other conditions in the where clause, but the other conditions can only be combined with rownum <= N using AND conjunction.

Important Notes

- The TopN query is Result Updating.

- Flink SQL will sort the input data stream according to the order key,
- so if the top N records have been changed, the changed ones will be sent as retraction/update records to downstream.
- If the top N records need to be stored in external storage, the result table should have the same unique key with the Top-N query.

Example

This is an example to get the top five products per category that have the maximum sales in realtime.

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) as row_num
  FROM ShopSales)
WHERE row_num <= 5;
```

2.4.7 Deduplication

Function

Deduplication removes rows that duplicate over a set of columns, keeping only the first one or the last one.

Syntax

```
SELECT [column_list]
FROM (
  SELECT [column_list],
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
      ORDER BY time_attr [asc|desc]) AS rownum
  FROM table_name)
WHERE rownum = 1
```

Description

- `ROW_NUMBER()`: Assigns a unique, sequential number to each row, starting with one.
- `PARTITION BY col1[, col2...]`: Specifies the partition columns, for example, the deduplicate key.
- `ORDER BY time_attr [asc|desc]`: Specifies the ordering column, it must be a time attribute. Currently Flink supports proctime only. Ordering by ASC means to keep the first row, ordering by DESC means to keep the last row.
- `WHERE rownum = 1`: The rownum = 1 is required for Flink to recognize this query is deduplication.

Precautions

None

Example

The following examples show how to remove duplicate rows on `order_id`. The proctime is an event time attribute.

```
SELECT order_id, user, product, number
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY proctime ASC) as row_num
  FROM Orders)
WHERE row_num = 1;
```

2.5 Functions

2.5.1 User-Defined Functions

Overview

DLI supports the following three types of user-defined functions (UDFs):

- Regular UDF: takes in one or more input parameters and returns a single result.
- User-defined table-generating function (UDTF): takes in one or more input parameters and returns multiple rows or columns.
- User-defined aggregate function (UDAF): aggregates multiple records into one value.

NOTE

UDFs can only be used in dedicated queues.

POM Dependency

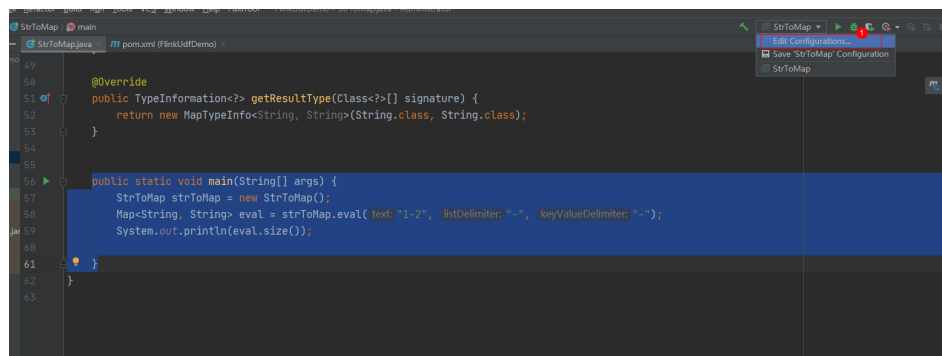
```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-table-common</artifactId>
  <version>1.10.0</version>
  <scope>provided</scope>
</dependency>
```

Important Notes

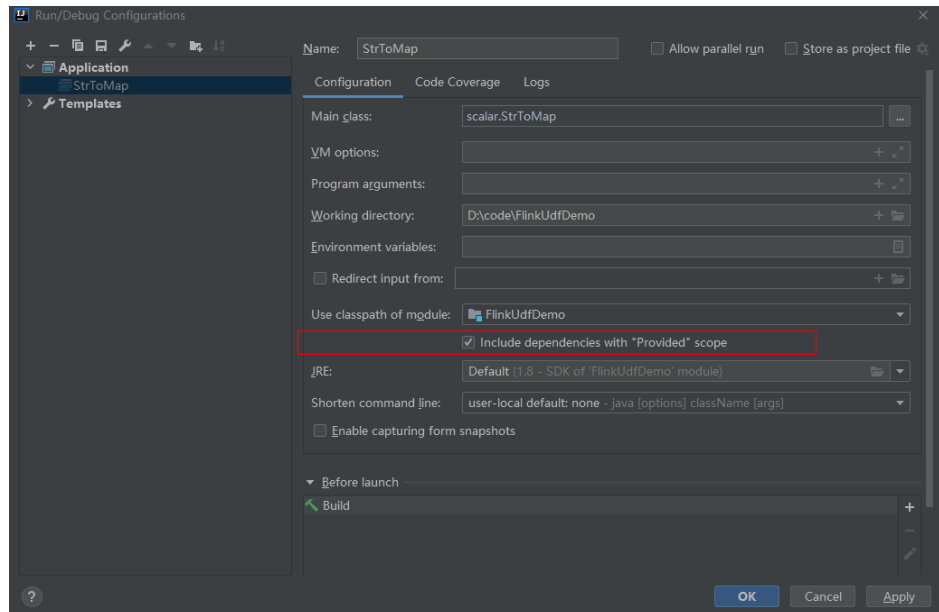
- Currently, Python is not supported for programming UDFs, UDTFs, and UDAFs.
- If you use IntelliJ IDEA to debug the created UDF, select **include dependencies with "Provided" scope**. Otherwise, the dependency packages in the POM file cannot be loaded for local debugging.

The following uses IntelliJ IDEA 2020.2 as an example:

- a. On the IntelliJ IDEA page, select the configuration file you need to debug and click **Edit Configurations**.



- b. On the **Run/Debug Configurations** page, select **include dependencies with "Provided" scope**.



- c. Click **OK**.

Using UDFs

1. Encapsulate the implemented UDFs into a JAR package and upload the package to OBS.
2. In the navigation pane of the DLI management console, choose **Data Management > Package Management**. On the displayed page, click **Create** and use the JAR package uploaded to OBS to create a package.
3. In the left navigation, choose **Job Management** and click **Flink Jobs**. Locate the row where the target resides and click **Edit** in the **Operation** column to switch to the page where you can edit the job.
4. Click the **Running Parameters** tab of your job, select the UDF JAR and click **Save**.
5. Add the following statement to the SQL statements to use the functions:

UDF

The regular UDF must inherit the `ScalarFunction` function and implement the `eval` method. The open and close functions are optional.

Example code

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class UdfScalarFunction extends ScalarFunction {
    private int factor = 12;
    public UdfScalarFunction() {
        this.factor = 12;
    }
    /**
     * (optional) Initialization
     * @param context
     */
}
```

```
@Override
public void open(FunctionContext context) {}
/**
 * Custom logic
 * @param s
 * @return
 */
public int eval(String s) {
    return s.hashCode() * factor;
}
/**
 * Optional
 */
@Override
public void close() {}
}
```

Example

```
CREATE FUNCTION udf_test AS 'com.company.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

The UDTF must inherit the TableFunction function and implement the eval method. The open and close functions are optional. If the UDTF needs to return multiple columns, you only need to declare the returned value as **Tuple** or **Row**. If **Row** is used, you need to overload the getResultType method to declare the returned field type.

Example code

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class UdfTableFunction extends TableFunction<Row> {
    private Logger log = LoggerFactory.getLogger(TableFunction.class);
    /**
     * (optional) Initialization
     * @param context
     */
    @Override
    public void open(FunctionContext context) {}
    public void eval(String str, String split) {
        for (String s : str.split(split)) {
            Row row = new Row(2);
            row.setField(0, s);
            row.setField(1, s.length());
            collect(row);
        }
    }
    /**
     * Declare the type returned by the function
     * @return
     */
    @Override
    public TypeInformation<Row> getResultType() {
        return Types.ROW(Types.STRING, Types.INT);
    }
    /**
     * Optional
     */
    @Override
```

```
public void close() {}  
}
```

Example

The UDTF supports CROSS JOIN and LEFT JOIN. When the UDTF is used, the **LATERAL** and **TABLE** keywords must be included.

- **CROSS JOIN**: does not output the data of a row in the left table if the UDTF does not output the result for the data of the row.
- **LEFT JOIN**: outputs the data of a row in the left table even if the UDTF does not output the result for the data of the row, but pads null with UDTF-related fields.

```
CREATE FUNCTION udtf_test AS 'com.company.udf.TableFunction';  
// CROSS JOIN  
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL  
TABLE(udtf_test(attr, ',')) as T(subValue, length);  
// LEFT JOIN  
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN LATERAL  
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

UDAF

The UDAF must inherit the AggregateFunction function. You need to create an accumulator for storing the computing result, for example, **WeightedAvgAccum** in the following example code.

Example code

```
public class WeightedAvgAccum {  
    public long sum = 0;  
    public int count = 0;  
}
```

```
import org.apache.flink.table.functions.AggregateFunction;  
import java.util.Iterator;  
/**  
 * The first type variable is the type returned by the aggregation function, and the second type variable is of  
 * the Accumulator type.  
 * Weighted Average user-defined aggregate function.  
 */  
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {  
    // Initialize the accumulator.  
    @Override  
    public WeightedAvgAccum createAccumulator() {  
        return new WeightedAvgAccum();  
    }  
    // Return the intermediate computing value stored in the accumulator.  
    @Override  
    public Long getValue(WeightedAvgAccum acc) {  
        if (acc.count == 0) {  
            return null;  
        } else {  
            return acc.sum / acc.count;  
        }  
    }  
    // Update the intermediate computing value according to the input.  
    public void accumulate(WeightedAvgAccum acc, long iValue) {  
        acc.sum += iValue;  
        acc.count += 1;  
    }  
    // Perform the retraction operation, which is opposite to the accumulate operation.  
    public void retract(WeightedAvgAccum acc, long iValue) {  
        acc.sum -= iValue;  
        acc.count -= 1;  
    }  
}
```

```
}  
// Combine multiple accumulator values.  
public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {  
    Iterator<WeightedAvgAccum> iter = it.iterator();  
    while (iter.hasNext()) {  
        WeightedAvgAccum a = iter.next();  
        acc.count += a.count;  
        acc.sum += a.sum;  
    }  
}  
// Reset the intermediate computing value.  
public void resetAccumulator(WeightedAvgAccum acc) {  
    acc.count = 0;  
    acc.sum = 0L;  
}  
}
```

Example

```
CREATE FUNCTION udaf_test AS 'com.company.udf.UdfAggFunction';  
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

2.5.2 Built-In Functions

2.5.2.1 Mathematical Operation Functions

Relational Operators

All data types can be compared by using relational operators and the result is returned as a **BOOLEAN** value.

Relationship operators are binary operators. Two compared data types must be of the same type or they must support implicit conversion.

Table 2-30 lists all relational operators supported by Flink SQL.

Table 2-30 Relational Operators

Operator	Returned Data Type	Description
A = B	BOOLEAN	If A is equal to B, then TRUE is returned. Otherwise, FALSE is returned. This operator is used for value assignment.
A <> B	BOOLEAN	If A is not equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned. This operator follows the standard SQL syntax.
A < B	BOOLEAN	If A is less than B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A <= B	BOOLEAN	If A is less than or equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.

Operator	Returned Data Type	Description
A > B	BOOLEAN	If A is greater than B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A >= B	BOOLEAN	If A is greater than or equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A IS NULL	BOOLEAN	If A is NULL , then TRUE is returned. Otherwise, FALSE is returned.
A IS NOT NULL	BOOLEAN	If A is not NULL , then TRUE is returned. Otherwise, FALSE is returned.
A IS DISTINCT FROM B	BOOLEAN	If A is not equal to B, TRUE is returned. NULL indicates A equals B.
A IS NOT DISTINCT FROM B	BOOLEAN	If A is equal to B, TRUE is returned. NULL indicates A equals B.
A BETWEEN [ASYMMETRIC SYMMETRIC] B AND C	BOOLEAN	If A is greater than or equal to B but less than or equal to C, TRUE is returned. <ul style="list-style-type: none"> ASYMMETRIC: indicates that B and C are location-related. For example, "A BETWEEN ASYMMETRIC B AND C" is equivalent to "A BETWEEN B AND C". SYMMETRIC: indicates that B and C are not location-related. For example, "A BETWEEN SYMMETRIC B AND C" is equivalent to "A BETWEEN B AND C) OR (A BETWEEN C AND B)".
A NOT BETWEEN B [ASYMMETRIC SYMMETRIC] AND C	BOOLEAN	If A is less than B or greater than C, TRUE is returned. <ul style="list-style-type: none"> ASYMMETRIC: indicates that B and C are location-related. For example, "A NOT BETWEEN ASYMMETRIC B AND C" is equivalent to "A NOT BETWEEN B AND C". SYMMETRIC: indicates that B and C are not location-related. For example, "A NOT BETWEEN SYMMETRIC B AND C" is equivalent to "(A NOT BETWEEN B AND C) OR (A NOT BETWEEN C AND B)".
A LIKE B [ESCAPE C]	BOOLEAN	If A matches pattern B, TRUE is returned. The escape character C can be defined as required.

Operator	Returned Data Type	Description
A NOT LIKE B [ESCAPE C]	BOOLEAN	If A does not match pattern B, TRUE is returned. The escape character C can be defined as required.
A SIMILAR TO B [ESCAPE C]	BOOLEAN	If A matches regular expression B, TRUE is returned. The escape character C can be defined as required.
A NOT SIMILAR TO B [ESCAPE C]	BOOLEAN	If A does not match regular expression B, TRUE is returned. The escape character C can be defined as required.
value IN (value [, value]*)	BOOLEAN	If the value is equal to any value in the list, TRUE is returned.
value NOT IN (value [, value]*)	BOOLEAN	If the value is not equal to any value in the list, TRUE is returned.
EXISTS (sub-query)	BOOLEAN	If sub-query returns at least one row, TRUE is returned.
value IN (sub-query)	BOOLEAN	If value is equal to a row returned by subquery, TRUE is returned.
value NOT IN (sub-query)	BOOLEAN	If value is not equal to a row returned by subquery, TRUE is returned.

Precautions

- Values of the double, real, and float types may be different in precision. The equal sign (=) is not recommended for comparing two values of the double type. You are advised to obtain the absolute value by subtracting these two values of the double type and determine whether they are the same based on the absolute value. If the absolute value is small enough, the two values of the double data type are regarded equal. For example:
`abs(0.9999999999 - 1.0000000000) < 0.0000000001` //The precision decimal places of 0.9999999999 and 1.0000000000 are 10, while the precision decimal place of 0.0000000001 is 9. Therefore, 0.9999999999 can be regarded equal to 1.0000000000.
- Comparison between data of the numeric type and character strings is allowed. During comparison using relational operators, including >, <, ≤, and ≥, data of the string type is converted to numeric type by default. No characters other than numeric characters are allowed.
- Character strings can be compared using relational operators.

Logical Operators

Common logical operators are AND, OR, and NOT. Their priority order is NOT > AND > OR.

Table 2-31 lists the calculation rules. A and B indicate logical expressions.

Table 2-31 Logical Operators

Operator	Result Type	Description
A OR B	BOOLEAN	If A or B is TRUE, TRUE is returned. Three-valued logic is supported.
A AND B	BOOLEAN	If both A and B are TRUE, TRUE is returned. Three-valued logic is supported.
NOT A	BOOLEAN	If A is not TRUE, TRUE is returned. If A is UNKNOWN, UNKNOWN is returned.
A IS FALSE	BOOLEAN	If A is TRUE, TRUE is returned. If A is UNKNOWN, FALSE is returned.
A IS NOT FALSE	BOOLEAN	If A is not FALSE, TRUE is returned. If A is UNKNOWN, TRUE is returned.
A IS TRUE	BOOLEAN	If A is TRUE, TRUE is returned. If A is UNKNOWN, FALSE is returned.
A IS NOT TRUE	BOOLEAN	If A is not TRUE, TRUE is returned. If A is UNKNOWN, TRUE is returned.
A IS UNKNOWN	BOOLEAN	If A is UNKNOWN, TRUE is returned.
A IS NOT UNKNOWN	BOOLEAN	If A is not UNKNOWN, TRUE is returned.

Precautions

Only data of the Boolean type can be used for calculation using logical operators. Implicit type conversion is not supported.

Arithmetic Operators

Arithmetic operators include binary operators and unary operators, for all of which, the returned results are of the numeric type. [Table 2-32](#) lists arithmetic operators supported by Flink SQL.

Table 2-32 Arithmetic Operators

Operator	Result Type	Description
+ numeric	All numeric types	Returns numbers.

Operator	Result Type	Description
- numeric	All numeric types	Returns negative numbers.
A + B	All numeric types	A plus B. The result type is associated with the operation data type. For example, if floating-point number is added to an integer, the result will be a floating-point number.
A - B	All numeric types	A minus B. The result type is associated with the operation data type.
A * B	All numeric types	Multiply A and B. The result type is associated with the operation data type.
A / B	All numeric types	Divide A by B. The result is a double-precision number.
POWER(A, B)	All numeric types	Returns the value of A raised to the power B.
ABS(numeric)	All numeric types	Returns the absolute value of a specified value.
MOD(A, B)	All numeric types	Returns the remainder (modulus) of A divided by B. A negative value is returned only when A is a negative value.
SQRT(A)	All numeric types	Returns the square root of A.
LN(A)	All numeric types	Returns the nature logarithm of A (base e).

Operator	Result Type	Description
LOG10(A)	All numeric types	Returns the base 10 logarithms of A.
LOG2(A)	All numeric types	Returns the base 2 logarithm of A.
LOG(B) LOG(A, B)	All numeric types	When called with one argument, returns the natural logarithm of B. When called with two arguments, this function returns the logarithm of B to the base A. B must be greater than 0 and A must be greater than 1.
EXP(A)	All numeric types	Return the value of e raised to the power of a .
CEIL(A) CEILING(A)	All numeric types	Return the smallest integer that is greater than or equal to a . For example: <code>ceil(21.2) = 22</code> .
FLOOR(A)	All numeric types	Return the largest integer that is less than or equal to a . For example: <code>floor(21.2) = 21</code> .
SIN(A)	All numeric types	Returns the sine value of A.
COS(A)	All numeric types	Returns the cosine value of A.
TAN(A)	All numeric types	Returns the tangent value of A.
COT(A)	All numeric types	Returns the cotangent value of A.

Operator	Result Type	Description
ASIN(A)	All numeric types	Returns the arc sine value of A.
ACOS(A)	All numeric types	Returns the arc cosine value of A.
ATAN(A)	All numeric types	Returns the arc tangent value of A.
ATAN2(A, B)	All numeric types	Returns the arc tangent of a coordinate (A, B).
COSH(A)	All numeric types	Returns the hyperbolic cosine of A. Return value type is DOUBLE.
DEGREES(A)	All numeric types	Convert the value of a from radians to degrees.
RADIANS(A)	All numeric types	Convert the value of a from degrees to radians.
SIGN(A)	All numeric types	Returns the sign of A. 1 is returned if A is positive. -1 is returned if A is negative. Otherwise, 0 is returned.
ROUND(A, d)	All numeric types	Returns a number rounded to d decimal places for A. For example: round(21.263,2) = 21.26.
PI	All numeric types	Returns the value of pi .

Operator	Result Type	Description
E()	All numeric types	Returns the value of e .
RAND()	All numeric types	Returns a pseudorandom double value in the range [0.0, 1.0)
RAND(A)	All numeric types	Returns a pseudorandom double value in the range [0.0, 1.0) with an initial seed A. Two RAND functions will return identical sequences of numbers if they have the same initial seed.
RAND_INTEG ER(A)	All numeric types	Returns a pseudorandom double value in the range [0.0, A)
RAND_INTEG ER(A, B)	All numeric types	Returns a pseudorandom double value in the range [0.0, B) with an initial seed A.
UUID()	All numeric types	Returns a UUID string.
BIN(A)	All numeric types	Returns a string representation of integer A in binary format. Returns NULL if A is NULL.
HEX(A) HEX(B)	All numeric types	Returns a string representation of an integer A value or a string B in hex format. Returns NULL if the A or B is NULL.
TRUNCATE(A, d)	All numeric types	Returns a number of truncated to d decimal places. Returns NULL if A or d is NULL. Example: truncate (42.345, 2) = 42.340 truncate(42.345) = 42.000
PI()	All numeric types	Returns the value of pi .

Precautions

Data of the string type is not allowed in arithmetic operations.

2.5.2.2 String Functions

Table 2-33 String functions

SQL Function	Return Type	Description
string1 string2	STRING	Returns the concatenation of string1 and string2.
CHAR_LENGTH(string) CHARACTER_LENGTH(string)	INT	Returns the number of characters in the string.
UPPER(string)	STRING	Returns the string in uppercase.
LOWER(string)	STRING	Returns the string in lowercase.
POSITION(string1 IN string2)	INT	Returns the position (start 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)	STRING	Returns a string that removes leading and/or trailing characters string2 from string1.
LTRIM(string)	STRING	Returns a string that removes the left whitespaces from the specified string. For example, LTRIM(' This is a test String.') returns "This is a test String."
RTRIM(string)	STRING	Returns a string that removes the right whitespaces from the specified string. For example, RTRIM('This is a test String. ') returns "This is a test String."
REPEAT(string, integer)	STRING	Returns a string that repeats the base string integer times. For example, REPEAT('This is a test String.', 2) returns "This is a test String.This is a test String."

SQL Function	Return Type	Description
REGEXP_REPLACE(string1, string2, string3)	STRING	Returns a string from string1 with all the substrings that match a regular expression string2 consecutively being replaced with string3. For example, REGEXP_REPLACE('foobar', 'oo ar', '') returns "fb" . REGEXP_REPLACE('ab\ab', '\\', 'e') returns "abeab" .
OVERLAY(string1 PLACING string2 FROM integer1 [FOR integer2])	STRING	Returns a string that replaces integer2 characters of STRING1 with STRING2 from position integer1. The default value of integer2 is the length of string2. For example, OVERLAY('This is an old string' PLACING ' new' FROM 10 FOR 5) returns "This is a new string" .
SUBSTRING(string FROM integer1 [FOR integer2])	STRING	Returns a substring of the specified string starting from position integer1 with length integer2 (to the end by default). If integer2 is not configured, the substring from integer1 to the end is returned by default.
REPLACE(string1, string2, string3)	STRING	Returns a new string which replaces all the occurrences of string2 with string3 (non-overlapping) from string1. For example, REPLACE('hello world', 'world', 'flink') returns "hello flink" ; REPLACE('ababab', 'abab', 'z') returns "zab" . REPLACE('ab\\ab', '\\', 'e') returns "abeab" .
REGEXP_EXTRACT(string1, string2[, integer])	STRING	Returns a string from string1 which extracted with a specified regular expression string2 and a regex match group index integer. Returns NULL, if the parameter is NULL or the regular expression is invalid. For example, REGEXP_EXTRACT('foothebar', 'foo.(?)(bar)', 2) returns "bar" .
INITCAP(string)	STRING	Returns a new form of STRING with the first character of each word converted to uppercase and the rest characters to lowercase.
CONCAT(string1, string2,...)	STRING	Returns a string that concatenates string1, string2, For example, CONCAT('AA', 'BB', 'CC') returns "AABBCC" .

SQL Function	Return Type	Description
CONCAT_WS(string1, string2, string3,...)	STRING	Returns a string that concatenates string2, string3, ... with a separator string1. The separator is added between the strings to be concatenated. Returns NULL if string1 is NULL. If other arguments are NULL, this function automatically skips NULL arguments. For example, CONCAT_WS('~', 'AA', NULL, 'BB', ' ', 'CC') returns "AA~BB~ ~CC".
LPAD(string1, integer, string2)	STRING	Returns a new string from string1 left-padded with string2 to a length of integer characters. If any argument is NULL, NULL is returned. If integer is negative, NULL is returned. If the length of string1 is shorter than integer, returns string1 shortened to integer characters. For example, LPAD(Symbol,4,Symbol) returns "Symbol hi". LPAD('hi',1,'?') returns "h".
RPAD(string1, integer, string2)	STRING	Returns a new string from string1 right-padded with string2 to a length of integer characters. If any argument is NULL, NULL is returned. If integer is negative, NULL is returned. If the length of string1 is shorter than integer, returns string1 shortened to integer characters. For example, RPAD('hi',4,'?') returns "hi???". RPAD('hi',1,'?') returns "h".
FROM_BASE64(string)	STRING	Returns the base64-decoded result from string. Returns NULL if string is NULL. For example, FROM_BASE64('aGVsbG8gd29ybGQ=') returns "hello world".
TO_BASE64(string)	STRING	Returns the base64-encoded result from string; if string is NULL. Returns NULL if string is NULL. For example, TO_BASE64(hello world) returns "aGVsbG8gd29ybGQ=".

SQL Function	Return Type	Description
ASCII(string)	INT	Returns the numeric value of the first character of string. Returns NULL if string is NULL. For example, ascii('abc') returns 97 . ascii(CAST(NULL AS VARCHAR)) returns NULL .
CHR(integer)	STRING	Returns the ASCII character having the binary equivalent to integer. If integer is larger than 255, we will get the modulus of integer divided by 255 first, and returns CHR of the modulus. Returns NULL if integer is NULL. chr(97) returns a . chr(353) Return a .
DECODE(binary, string)	STRING	Decodes the first argument into a String using the provided character set (one of 'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16'). If either argument is NULL, the result will also be NULL.
ENCODE(string1, string2)	STRING	Encodes the string1 into a BINARY using the provided string2 character set (one of 'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16'). If either argument is NULL, the result will also be NULL.
INSTR(string1, string2)	INT	Returns the position of the first occurrence of string2 in string1. Returns NULL if any argument is NULL.
LEFT(string, integer)	STRING	Returns the leftmost integer characters from the string. Returns EMPTY String if integer is negative. Returns NULL if any argument is NULL.
RIGHT(string, integer)	STRING	Returns the rightmost integer characters from the string. Returns EMPTY String if integer is negative. Returns NULL if any argument is NULL.

SQL Function	Return Type	Description
LOCATE(string1, string2[, integer])	INT	Returns the position of the first occurrence of string1 in string2 after position integer. Returns 0 if not found. The value of integer defaults to 0 . Returns NULL if any argument is NULL.
PARSE_URL(string1, string2[, string3])	STRING	Returns the specified part from the URL. Valid values for string2 include 'HOST', 'PATH', 'QUERY', 'REF', 'PROTOCOL', 'AUTHORITY', 'FILE', and 'USERINFO'. Returns NULL if any argument is NULL. If string2 is QUERY, the key in QUERY can be specified as string3. Example: The parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') returns 'facebook.com'. parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') returns 'v1'.
REGEXP(string1, string2)	BOOLEAN	Returns TRUE if any (possibly empty) substring of string1 matches the regular expression string2, otherwise FALSE. If the information is found, TRUE is returned. string1 indicates the specified string, and string2 indicates the regular expression. Returns NULL if any argument is NULL.
REVERSE(string)	STRING	Returns the reversed string. Returns NULL if string is NULL.
SPLIT_INDEX(string1, string2, integer1)	STRING	Splits string1 by the delimiter string2, returns the integer1-th (zero-based) string of the split strings. Returns NULL if integer is negative. Returns NULL if any argument is NULL.
STR_TO_MAP(string1[, string2, string3])	MAP	Returns a map after splitting the string1 into key/value pairs using delimiters. The default value of string2 is ','. The default value of string3 is '='.
SUBSTR(string[, integer1[, integer2]])	STRING	Returns a substring of string starting from position integer1 with length integer2. If integer2 is not specified, the string is truncated to the end.

SQL Function	Return Type	Description
JSON_VAL(STRING json_string, STRING json_path)	STRING	<p>Returns the value of the specified json_path from the json_string. For details about how to use the functions, see JSON_VAL Function.</p> <p>NOTE The following rules are listed in descending order of priority.</p> <ol style="list-style-type: none"> 1. The two arguments json_string and json_path cannot be NULL. 2. The value of json_string must be a valid JSON string. Otherwise, the function returns NULL. 3. If json_string is an empty string, the function returns an empty string. 4. If json_path is an empty string or the path does not exist, the function returns NULL.

JSON_VAL Function

- Syntax

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

Table 2-34 Parameter description

Parameter	Type	Description
json_string	STRING	JSON object to be parsed
json_path	STRING	Path expression for parsing the JSON string For the supported expressions, see Table 2-35 .

Table 2-35 Expressions supported

Expression	Description
\$	Root node in the path
[]	Access array elements
*	Array wildcard
.	Access child elements

- Example
 - a. Test input data.
Test the data source kafka. The message content is as follows:

```
"{name:James,age:24,sex:male,grade:{math:95,science:[80,85],english:100}}"  
"{name:James,age:24,sex:male,grade:{math:95,science:[80,85],english:100}}"
```

b. Use JSON_VAL in SQL statements.

```
create table kafkaSource(  
  message STRING  
)  
with (  
  'connector.type' = 'kafka',  
  'connector.version' = '0.11',  
  'connector.topic' = 'topic-swq',  
  'connector.properties.bootstrap.servers' = 'xxx.xxx.xxx.xxx:9092,yyy.yyy.yyy:9092,zzz.zzz.zzz.zzz:9092',  
  'connector.startup-mode' = 'earliest-offset',  
  'format.field-delimiter' = '|',  
  'format.type' = 'csv'  
);  
  
create table kafkaSink(  
  message1 STRING,  
  message2 STRING,  
  message3 STRING,  
  message4 STRING,  
  message5 STRING,  
  message6 STRING  
)  
with (  
  'connector.type' = 'kafka',  
  'connector.version' = '0.11',  
  'connector.topic' = 'topic-swq-out',  
  'connector.properties.bootstrap.servers' = 'xxx.xxx.xxx.xxx:9092,yyy.yyy.yyy:9092,zzz.zzz.zzz.zzz:9092',  
  'format.type' = 'json'  
);  
  
INSERT INTO kafkaSink  
SELECT  
  JSON_VAL(message,""),  
  JSON_VAL(message,"$.name"),  
  JSON_VAL(message,"$.grade.science"),  
  JSON_VAL(message,"$.grade.science[*]"),  
  JSON_VAL(message,"$.grade.science[1]"),  
  JSON_VAL(message,"$.grade.dddd")  
FROM kafkaSource;
```

c. View output.

```
{"message1":null,"message2":"swq","message3":"[80,85]","message4":"[80,85]","message5":"85",  
,"message6":null}  
{"message1":null,"message2":null,"message3":null,"message4":null,"message5":null,"message6":  
null}
```

2.5.2.3 Temporal Functions

[Table 2-36](#) lists the temporal functions supported by Flink OpenSource SQL.

Function Description

Table 2-36 Temporal functions

Function	Return Type	Description
DATE string	DATE	Parse the date string (yyyy-MM-dd) to a SQL date.

Function	Return Type	Description
TIME string	TIME	Parse the time string (HH:mm:ss[.fff]) to a SQL time.
TIMESTAMP string	TIMESTAMP	Convert the time string into a timestamp. The time string format is yyyy-MM-dd HH:mm:ss[.fff] .
INTERVAL string range	INTERVAL	<p>Parse an interval string in the following two forms:</p> <ul style="list-style-type: none"> • yyyy-MM for SQL intervals of months. An interval range might be YEAR or YEAR TO MONTH. • dd hh:mm:ss.fff for SQL intervals of milliseconds. An interval range might be DAY, MINUTE, DAY TO HOUR, or DAY TO SECOND. <p>Example:</p> <p>INTERVAL '10 00:00:00.004' DAY TO second indicates that the interval is 10 days and 4 milliseconds.</p> <p>INTERVAL '10' DAY: indicates that the interval is 10 days.</p> <p>INTERVAL '2-10' YEAR TO MONTH indicates that the interval is two years and ten months.</p>
CURRENT_DATE	DATE	Return the SQL date of UTC time zone.
CURRENT_TIME	TIME	Return the SQL time of UTC time zone.
CURRENT_TIMESTAMP	TIMESTAMP	Return the SQL timestamp of UTC time zone.
LOCALTIME	TIME	Return the SQL time of the local time zone.
LOCALTIMESTAMP	TIMESTAMP	Return the SQL timestamp of the local time zone.
EXTRACT(timeintervalunit FROM temporal)	BIGINT	<p>Extract part of the time point or interval. Return the part in the int type.</p> <p>For example, extract the date 2006-06-05 and return 5.</p> <p>EXTRACT(DAY FROM DATE '2006-06-05') returns 5.</p>
YEAR(date)	BIGINT	<p>Return the year from a SQL date.</p> <p>For example, YEAR(DATE'1994-09-27') returns 1994.</p>

Function	Return Type	Description
QUARTER(date)	BIGINT	Return the quarter of a year from a SQL date.
MONTH(date)	BIGINT	Return the month of a year from a SQL date. For example, MONTH(DATE '1994-09-27') returns 9 .
WEEK(date)	BIGINT	Return the week of a year from a SQL date. For example, WEEK(DATE'1994-09-27') returns 39 .
DAYOFYEAR(date)	BIGINT	Return the day of a year from a SQL date. For example, DAYOFYEAR(DATE '1994-09-27') is 270 .
DAYOFMONTH(date)	BIGINT	Return the day of a month from a SQL date. For example, DAYOFMONTH(DATE'1994-09-27') returns 27 .
DAYOFWEEK(date)	BIGINT	Return the day of a week from a SQL date. Sunday is set to 1 . For example, DAYOFWEEK(DATE'1994-09-27') returns 3 .
HOUR(timestamp)	BIGINT	Return the hour of a day (an integer between 0 and 23) from a SQL timestamp. For example, HOUR(TIMESTAMP '1994-09-27 13:14:15') returns 13 .
MINUTE(timestamp)	BIGINT	Return the minute of an hour (an integer between 0 and 59) from a SQL timestamp. For example, MINUTE(TIMESTAMP '1994-09-27 13:14:15') returns 14 .
SECOND(timestamp)	BIGINT	Returns the second of a minute (an integer between 0 and 59) from a SQL timestamp. For example, SECOND(TIMESTAMP '1994-09-27 13:14:15') returns 15 .
FLOOR(timepoint TO timeintervalunit)	TIME	Round a time point down to the given unit. For example, 12:44:00 is returned from FLOOR(TIME '12:44:31' TO MINUTE) .
CEIL(timepoint TO timeintervalunit)	TIME	Round a time point up to the given unit. For example, CEIL(TIME '12:44:31' TO MINUTE) returns 12:45:00 .

Function	Return Type	Description
(timepoint1, temporal1) OVERLAPS (timepoint2, temporal2)	BOOLEAN	Return TRUE if two time intervals overlap. Example: (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) returns TRUE . (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) returns FALSE .
DATE_FORMAT(time stamp, string)	STRING	Convert a timestamp to a value of string in the format specified by the date format string.
TIMESTAMPADD(timeintervalunit, interval, timepoint)	TIMESTAMP/ DATE/ TIME	Return the date and time added to timepoint based on the result of interval and timeintervalunit . For example, TIMESTAMPADD(WEEK, 1, DATE '2003-01-02') returns 2003-01-09 .
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	INT	Return the (signed) number of timepointunit between timepoint1 and timepoint2 . The unit for the interval is given by the first argument, which should be one of the following values: SECOND, MINUTE, HOUR, DAY, MONTH, and YEAR . For example, TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00') returns 1 .
CONVERT_TZ(string1, string2, string3)	TIMESTAMP	Convert a datetime string1 from time zone string2 to time zone string3 . For example, CONVERT_TZ('1970-01-01 00:00:00', 'UTC', 'America/Los_Angeles') returns '1969-12-31 16:00:00' .
FROM_UNIXTIME(numeric[, string])	STRING	Return a representation of the numeric argument as a value in string format. The default string format is YYYY-MM-DD hh:mm:ss. For example, FROM_UNIXTIME(44) returns 1970-01-01 09:00:44 .
UNIX_TIMESTAMP()	BIGINT	Get current Unix timestamp in seconds.

Function	Return Type	Description
UNIX_TIMESTAMP(string1[, string2])	BIGINT	Convert date time string string1 in format string2 to Unix timestamp (in seconds), using the specified timezone in table config. The default format of string2 is yyyy-MM-dd HH:mm:ss.
TO_DATE(string1[, string2])	DATE	Convert a date string string1 with format string2 to a date. The default format of string2 is yyyy-MM-dd.
TO_TIMESTAMP(string1[, string2])	TIMESTAMP	Convert date time string string1 with format string2 to a timestamp. The default format of string2 is yyyy-MM-dd HH:mm:ss.

DATE

- **Function**
Returns a date parsed from string in form of **yyyy-MM-dd**.
- **Description**
DATE DATE string
- **Input parameters**

Parameter	Type	Description
string	STRING	String in the SQL date format. Note that the string must be in the yyyy-MM-dd format. Otherwise, an error will be reported.

- **Example**
 - Test statement

```
SELECT
  DATE "2021-08-19" AS `result`
FROM
  testtable;
```

- Test result

result
2021-08-19

TIME

- **Function**
Returns a SQL time parsed from string in form of **HH:mm:ss[.fff]**.

- **Description**
TIME TIME string

- **Input parameters**

Parameter	Type	Description
string	STRING	Time Note that the string must be in the format of HH:mm:ss[.fff] . Otherwise, an error will be reported.

- **Example**

- Test statement

```
SELECT
  TIME "10:11:12" AS `result`,
  TIME "10:11:12.032" AS `result2`
FROM
  testtable;
```

- Test result

result	result2
10:11:12	10:11:12.032

TIMESTAMP

- **Function**
Converts the time string into timestamp. The time string format is **yyyy-MM-dd HH:mm:ss[.fff]**. The return value is of the **TIMESTAMP(3)** type.

- **Description**
TIMESTAMP(3) TIMESTAMP string

- **Input parameters**

Parameter	Type	Description
string	STRING	Time Note that the string must be in the format of yyyy-MM-dd HH:mm:ss[.fff] . Otherwise, an error will be reported.

- **Example**

- Test statement

```
SELECT
  TIMESTAMP "1997-04-25 13:14:15" AS `result`,
```

```
TIMESTAMP "1997-04-25 13:14:15.032" AS `result2`
FROM
  testtable;
```

- Test result

result	result2
1997-04-25 13:14:15	1997-04-25 13:14:15.032

INTERVAL

- **Function**

Parses an interval string.

- **Description**

INTERVAL **INTERVAL** string range

- **Input parameters**

Parameter	Type	Description
string	STRING	Timestamp string used together with the range parameter. The string is in either of the following two formats: <ul style="list-style-type: none"> • yyyy-MM for SQL intervals of months. An interval range might be YEAR or YEAR TO MONTH for intervals of months. • dd hh:mm:ss.fff for SQL intervals of milliseconds. An interval range might be DAY, MINUTE, DAY TO HOUR, or DAY TO SECOND.
range	INTERVAL	Interval range. This parameter is used together with the string parameter. Available values are as follows: YEAR , YEAR To Month , DAY , MINUTE , DAY TO HOUR and DAY TO SECOND .

- **Example**

Test statement

```
-- The interval is 10 days and 4 milliseconds.
INTERVAL '10 00:00:00.004' DAY TO second
-- The interval is 10 days.
INTERVAL '10'
-- The interval is 2 years and 10 months.
INTERVAL '2-10' YEAR TO MONTH
```

CURRENT_DATE

- **Function**

Returns the current SQL time (**yyyy-MM-dd**) in the local time zone. The return value is of the **DATE** type.

- **Description**
DATE CURRENT_DATE
- **Input parameters**
N/A
- **Example**
 - Test statement

```
SELECT
  CURRENT_DATE AS `result`
FROM
  testtable;
```

- Test result

result
2021-10-28

CURRENT_TIME

- **Function**
Returns the current SQL time (**HH:mm:ss.fff**) in the local time zone. The return value is of the **TIME** type.

- **Description**
TIME CURRENT_TIME

- **Input parameters**
N/A

- **Example**
 - Test statement

```
SELECT
  CURRENT_TIME AS `result`
FROM
  testtable;
```

- Test result

result
08:29:19.289

CURRENT_TIMESTAMP

- **Function**
Returns the current SQL timestamp in the local time zone. The return value is of the **TIMESTAMP(3)** type.

- **Description**
TIMESTAMP(3) CURRENT_TIMESTAMP

- **Input parameters**
N/A

- **Example**
 - Test statement

```
SELECT
  CURRENT_TIMESTAMP AS `result`
FROM
  testtable;
```

- Test result

result
2021-10-28 08:33:51.606

LOCALTIME

- **Function**

Returns the current SQL time in the local time zone. The return value is of the **TIME** type.

- **Description**

TIME LOCALTIME

- **Input parameters**

N/A

- **Example**

- Test statement

```
SELECT
  LOCALTIME AS `result`
FROM
  testtable;
```

- Test result

result
16:39:37.706

LOCALTIMESTAMP

- **Function**

Returns the current SQL timestamp in the local time zone. The return value is of the **TIMESTAMP(3)** type.

- **Description**

TIMESTAMP(3) LOCALTIMESTAMP

- **Input parameters**

N/A

- **Example**

- Test statement

```
SELECT
  LOCALTIMESTAMP AS `result`
FROM
  testtable;
```

- Test result

result
2021-10-28 16:43:17.625

EXTRACT

- Function**
 Returns a value extracted from the **timeintervalunit** part of temporal. The return value is of the **BIGINT** type.
- Description**
BIGINT **EXTRACT**(timeinteravlunit **FROM** temporal)
- Input parameters**

Parameter	Type	Description
timeinteravlunit	TIMEUNIT	Time unit to be extracted from a time point or interval. The value can be YEAR, QUARTER, MONTH, WEEK, DAY, DOY, HOUR, MINUTE, SECOND.
temporal	DATE/TIME/TIMESTAMP/INTERVAL	Time point or interval.

CAUTION

Do not specify a time unit that is not of any time points or intervals. Otherwise, the job fails to be submitted.

For example, an error message is displayed when the following statement is executed because **YEAR** cannot be extracted from **TIME**.

```
SELECT
  EXTRACT(YEAR FROM TIME '12:44:31' ) AS `result`
FROM
  testtable;
```

- Example**

- Test statement

```
SELECT
  EXTRACT(YEAR FROM DATE '1997-04-25' ) AS `result`,
  EXTRACT(MINUTE FROM TIME '12:44:31') AS `result2`,
  EXTRACT(SECOND FROM TIMESTAMP '1997-04-25 13:14:15') AS `result3`,
  EXTRACT(YEAR FROM INTERVAL '2-10' YEAR TO MONTH) AS `result4`,
FROM
  testtable;
```

- Test result

result	result2	result3	result4
1997	44	15	2

YEAR

- **Function**

Returns the year from a SQL date. The return value is of the **BIGINT** type.

- **Description**

BIGINT YEAR(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  YEAR(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

result
1997

QUARTER

- **Function**

Returns the quarter of a year (an integer between 1 and 4) from a SQL date. The return value is of the **BIGINT** type.

- **Description**

BIGINT QUARTER(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  QUARTER(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

Result
2

MONTH

- **Function**

Returns the month of a year (an integer between 1 and 12) from a SQL date. The return value is of the **BIGINT** type.

- **Description**

BIGINT MONTH(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  MONTH(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

result
4

WEEK

- **Function**

Returns the week of a year from a SQL date. The return value is of the **BIGINT** type.

- **Description**

BIGINT WEEK(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  WEEK(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

result
17

DAYOFYEAR

- **Function**

Returns the day of a year (an integer between 1 and 366) from SQL date. The return value is of the **BIGINT** type.

- **Description**

BIGINT DAYOFYEAR(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  DAYOFYEAR(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

result
115

DAYOFMONTH

- **Function**

Returns the day of a month (an integer between 1 and 31) from a SQL date. The return value is of the **BIGINT** type.

- **Description**

BIGINT DAYOFMONTH(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  DAYOFMONTH(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- Test result

Result
25

DAYOFWEEK

- **Function**

Returns the day of a week (an integer between 1 and 7) from a SQL date. The return value is of the **BIGINT** type.

 **NOTE**

Note that the start day of a week is Sunday.

- **Description**

BIGINT DAYOFWEEK(date)

- **Input parameters**

Parameter	Type	Description
date	DATE	SQL date

- **Example**

- Test statement

```
SELECT
  DAYOFWEEK(DATE '1997-04-25') AS `result`
FROM
  testtable;
```

- Test result

result
6

HOUR

- **Function**

Returns the hour of a day (an integer between 0 and 23) from SQL timestamp. The return value is of the **BIGINT** type.

- **Description**

BIGINT HOUR(timestamp)

- **Input parameters**

Parameter	Type	Description
timestamp	TIMESTAMP	SQL timestamp

- **Example**

- Test statement

```
SELECT
  HOUR(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
  testtable;
```

- Test result

result
10

MINUTE

- **Function**

Returns the minute of an hour (an integer between 0 and 59) from a SQL timestamp. The return value is of the **BIGINT** type.

- **Description**

BIGINT **MINUTE**(timestamp)

- **Input parameters**

Parameter	Type	Description
timestamp	TIMESTAMP	SQL timestamp

- **Example**

- Test statement

```
SELECT
  MINUTE(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
  testtable;
```

- Test result

result
11

SECOND

- **Function**

Returns the second of an hour (an integer between 0 and 59) from a SQL timestamp. The return value is of the **BIGINT** type.

- **Description**

BIGINT **SECOND**(timestamp)

- **Input parameters**

Parameter	Type	Description
timestamp	TIMESTAMP	SQL timestamp

- **Example**

- Test statement

```
SELECT
  SECOND(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
  testtable;
```

- Test result

result
12

FLOOR

- **Function**

Returns a value that rounds **timepoint** down to the time unit **timeintervalunit**.

- **Description**

TIME/TIMESTAMP(3) **FLOOR**(timepoint TO timeintervalunit)

- **Input parameters**

Parameter	Type	Description
timepoint	TIMESTAMP /TIME	SQL time or SQL timestamp
timeintervalunit	TIMEUNIT	Time unit. The value can be YEAR, QUARTER, MONTH, WEEK, DAY, DOY, HOUR, MINUTE, or SECOND.

- **Example**

- Test statement For details about the syntax of the userDefined result table, see [User-defined Result Table](#).

```
create table PrintSink (
  message TIME,
  message2 TIME,
  message3 TIMESTAMP(3)
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'com.swqtest.flink.sink.PrintSink'--Replace the class with a user-
defined class. For details, see the syntax description in the userDefined result table.
);

INSERT INTO
  PrintSink
SELECT
  FLOOR(TIME '13:14:15' TO MINUTE) AS `result`
  FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
  FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`;
```

- Test result

The values of the fields in the PrintSink table are as follows:

Message	Message 2	Message 3
13:14	13:14	1997-04-25T13:14

CEIL

- **Function**

Returns a value that rounds **timepoint** up to the time unit **timeintervalunit**.

- **Description**
TIME/TIMESTAMP(3) **CEIL**(timepoint TO timeintervalunit)

- **Input parameters**

Parameter	Type	Description
timepoint	TIMESTAMP /TIME	SQL time or SQL timestamp
timeintervalunit	TIMEUNIT	Time unit. The value can be YEAR, QUARTER, MONTH, WEEK, DAY, DOY, HOUR, MINUTE, or SECOND.

- **Example**

- Test statement For details about the syntax of the userDefined result table, see [User-defined Result Table](#).

```
create table PrintSink (
  message TIME,
  message2 TIME,
  message3 TIMESTAMP(3)
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'com.swqtest.flink.sink.PrintSink'--Replace the class with a user-
defined class. For details, see the syntax description in the userDefined result table.
);

INSERT INTO
  PrintSink
SELECT
  CEIL(TIME '13:14:15' TO MINUTE) AS `result`
  CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
  CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`;
```

- Test result

result	result2	result3
13:15	13:15	1997-04-25T13:15

OVERLAPS

- **Function**
Returns **TRUE** if two time intervals overlap; returns **FALSE** otherwise.

- **Description**
BOOLEAN (timepoint1, temporal1) **OVERLAPS** (timepoint2, temporal2)

- **Input parameters**

Parameter	Type	Description
timepoint1/ timepoint2	DATE/TIME/ TIMESTAMP	Time point
temporal1/ temporal2	DATE/TIME/ TIMESTAMP/ INTERVAL	Time point or interval

 **NOTE**

- **(timepoint, temporal)** is a closed interval.
- The temporal can be of the **DATE, TIME, TIMESTAMP, or INTERVAL** type.
 - When the temporal is **DATE, TIME, or TIMESTAMP, (timepoint, temporal)** indicates an interval between **timepoint** and **temporal**. The temporal can be earlier than the value of **timepoint**, for example, **(DATE '1997-04-25', DATE '1997-04-23')**.
 - When the temporal is **INTERVAL, (timepoint, temporal)** indicates an interval between **timepoint** and **timepoint + temporal**.
- Ensure that **(timepoint1, temporal1)** and **(timepoint2, temporal2)** are intervals of the same data type.

• **Example**

- Test statement

```
SELECT
  (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS `result`,
  (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS `result2`,
  (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:31:00', INTERVAL '2' HOUR) AS `result3`,
  (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:00:00', INTERVAL '3' HOUR) AS `result4`,
  (TIMESTAMP '1997-04-25 12:00:00', TIMESTAMP '1997-04-25 12:20:00') OVERLAPS
  (TIMESTAMP '1997-04-25 13:00:00', INTERVAL '2' HOUR) AS `result5`,
  (DATE '1997-04-23', INTERVAL '2' DAY) OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
  AS `result6`,
  (DATE '1997-04-25', DATE '1997-04-23') OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
  AS `result7`
FROM
  testtable;
```

- Test result

res ult	res ult 2	res ult 3	res ult 4	resu lt5	resu lt6	result7
tru e	tru e	fals e	tru e	fals e	true	true

DATE_FORMAT

• **Function**

Converts a timestamp to a value of string in the format specified by the date format string.

• **Description**

STRING **DATE_FORMAT**(timestamp, dateformat)

• **Input parameters**

Parameter	Type	Description
timestamp	TIMESTAMP/ STRING	Time point
dateformat	STRING	String in the date format

- **Example**

- Test statement

```
SELECT
  DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd HH:mm:ss') AS `result`,
  DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result2`,
  DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yy/MM/dd HH:mm') AS `result3`,
  DATE_FORMAT('1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result4`
FROM testtable;
```

- Test result

result	result2	result3	result4
1997-04-25 10:11:12	1997-04-25	97/04/25 10:11	1997-04-25

TIMESTAMPADD

- **Function**

Returns the date and time by combining **interval** and **timeintervalunit** and adding the combination to **timepoint**.

 **NOTE**

The return value of **TIMESTAMPADD** is the value of **timepoint**. An exception is that if the input **timepoint** is of the **TIMESTAMP** type, the return value can be inserted into a table field of the **DATE** type.

- **Description**

TIMESTAMP(3)/DATE/TIME **TIMESTAMPADD**(timeintervalunit, interval, timepoint)

- **Input parameters**

Parameter	Type	Description
timeintervalunit	TIMEUNIT	Time unit
interval	INT	Interval
timepoint	TIMESTAMP/ DATE/TIME	Time point

- **Example**

- Test statement

```
SELECT
  TIMESTAMPADD(WEEK, 1, DATE '1997-04-25') AS `result`,
  TIMESTAMPADD(QUARTER, 1, TIMESTAMP '1997-04-25 10:11:12') AS `result2`,
  TIMESTAMPADD(SECOND, 2, TIME '10:11:12') AS `result3`
FROM testtable;
```

- Test result

result	result2	result3
1997-05-02	<ul style="list-style-type: none"> If this field is inserted into a table field of the TIMESTAMP type, 1997-07-25T10:11:12 is returned. If this field is inserted into a table field of the DATE type, 1997-07-25 is returned. 	10:11:14

TIMESTAMPDIFF

- **Function**

Returns the (signed) number of **timepointunit** between **timepoint1** and **timepoint2**. The unit for the interval is given by the first argument.

- **Description**

INT **TIMESTAMPDIFF**(timepointunit, timepoint1, timepoint2)

- **Input parameters**

Parameter	Type	Description
timepointunit	TIMEUNIT	Time unit. The value can be SECOND, MINUTE, HOUR, DAY, MONTH or YEAR .
timepoint1/ timepoint2	TIMESTAMP/ DATE	Time point

- **Example**

- Test statement

```
SELECT
  TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-25 10:00:00', TIMESTAMP '1997-04-28 10:00:00')
  AS `result`,
  TIMESTAMPDIFF(DAY, DATE '1997-04-25', DATE '1997-04-28') AS `result2`,
  TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-27 10:00:20', TIMESTAMP '1997-04-25 10:00:00')
  AS `result3`
FROM testtable;
```

- Test result

result	result2	result3
3	3	-2

CONVERT_TZ

- **Function**

Converts a datetime **string1** (with default ISO timestamp format '**yyyy-MM-dd HH:mm:ss**') from time zone **string2** to time zone **string3**.

- **Description**
STRING `CONVERT_TZ(string1, string2, string3)`

- **Input parameters**

Parameter	Type	Description
string1	STRING	SQL timestamp. If the value does not meet the format requirements, NULL is returned.
string2	STRING	Time zone before conversion. The format of time zone should be either an abbreviation such as PST , a full name such as America/Los_Angeles , or a custom ID such as GMT-08:00 .
string3	STRING	Time zone after conversion. The format of time zone should be either an abbreviation such as PST , a full name such as America/Los_Angeles , or a custom ID such as GMT-08:00 .

- **Example**

- Test statement

```
SELECT
  CONVERT_TZ(1970-01-01 00:00:00, UTC, America/Los_Angeles) AS `result`,
  CONVERT_TZ(1997-04-25 10:00:00, UTC, GMT-08:00) AS `result2`
FROM testtable;
```

- Test result

result	result2
1969-12-31 16:00:00	1997-04-25 02:00:00

FROM_UNIXTIME

- **Function**

Returns a representation of the **numeric** argument as a value in string format.

- **Description**

STRING `FROM_UNIXTIME(numeric[, string])`

- **Input parameters**

Parameter	Type	Description
numeric	BIGINT	An internal timestamp representing the number of seconds since 1970-01-01 00:00:00 UTC. The value can be generated by the UNIX_TIMESTAMP() function.

Parameter	Type	Description
string	STRING	Time. If this parameter is not specified, the default time format is yyyy-MM-dd HH:mm:ss format.

- **Example**

- Test statement

```
SELECT
  FROM_UNIXTIME(44) AS `result`,
  FROM_UNIXTIME(44, 'yyy:MM:dd') AS `result2`
FROM testtable;
```

- Test result

result	result2
1970-01-01 08:00:44	1970:01:01

UNIX_TIMESTAMP

- **Function**

Gets current Unix timestamp in seconds. The return value is of the **BIGINT** type.

- **Description**

BIGINT UNIX_TIMESTAMP()

- **Input parameters**

N/A

- **Example**

- Test statement

```
SELECT
  UNIX_TIMESTAMP() AS `result`
FROM
  table;
```

- Test result

result
1635401982

UNIX_TIMESTAMP(string1[, string2])

- **Function**

Converts date time **string1** in format **string2** to Unix timestamp (in seconds). The return value is of the **BIGINT** type.

- **Description**

BIGINT UNIX_TIMESTAMP(string1[, string2])

- **Input parameters**

Parameter	Type	Description
string1	STRING	SQL timestamp string. An error is reported if the value does not comply with the string2 format.
string2	STRING	Time. If this parameter is not specified, the default time format is yyyy-MM-dd HH:mm:ss .

- **Example**

- Test statement

```
SELECT
  UNIX_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
  UNIX_TIMESTAMP('1997-04-25 00:00:10', 'yyyy-MM-dd HH:mm:ss') AS `result2`,
  UNIX_TIMESTAMP('1997-04-25 00:00:00') AS `result3`
FROM
  testtable;
```

- Test result

result	result2	result3
861897600	861897610	861897600

TO_DATE

- **Function**

Converts a date **string1** with format **string2** to a date.

- **Description**

DATE TO_DATE(string1[, string2])

- **Input parameters**

Parameter	Type	Description
string1	STRING	SQL timestamp string. If the value is not in the required format, an error is reported.
string2	STRING	Format. If this parameter is not specified, the default time format is yyyy-MM-dd .

- **Example**

- Test statement

```
SELECT
  TO_DATE('1997-04-25') AS `result`,
  TO_DATE('1997:04:25', 'yyyy-MM-dd') AS `result2`,
  TO_DATE('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
  testtable;
```

- Test result

result	result2	result3
1997-04-25	1997-04-25	1997-04-25

TO_TIMESTAMP

- **Function**

Converts date time **string1** with format **string2** to a timestamp.

- **Description**

`TIMESTAMP TO_TIMESTAMP(string1[, string2])`

- **Input parameters**

Parameter	Type	Description
string1	STRING	SQL timestamp string. If the value is not in the required format, NULL is returned.
string2	STRING	Date format. If this parameter is not specified, the default format is yyyy-MM-dd HH:mm:ss .

- **Example**

- Test statement

```
SELECT
  TO_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
  TO_TIMESTAMP('1997-04-25 00:00:00') AS `result2`,
  TO_TIMESTAMP('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
  testtable;
```

- Test result

result	result2	result3
1997-04-25 00:00	1997-04-25 00:00	1997-04-25 00:00

2.5.2.4 Conditional Functions

Description

Table 2-37 Conditional functions

Function	Description
CASE value WHEN value1_1 [, value1_2]* THEN result1 [WHEN value2_1 [, value2_2]* THEN result2]* [ELSE resultZ] END	Returns resultX when the value is contained in (valueX_1, valueX_2, ...). Only the first matched value is returned. When no value matches, returns resultZ if it is provided and returns NULL otherwise.
CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2]* [ELSE resultZ] END	Returns resultX when the first conditionX is met. Only the first matched value is returned. When no condition is met, returns resultZ if it is provided and returns NULL otherwise.
NULLIF(value1, value2)	Returns NULL if value1 is equal to value2; returns value1 otherwise. For example, NullIF (5, 5) returns NULL . NULLIF(5, 0) returns 5 .
COALESCE(value1, value2 [, value3]*)	Returns the first value (from left to right) that is not NULL from value1, value2, For example, COALESCE(NULL, 5) returns 5 .
IF(condition, true_value, false_value)	Returns the true_value if condition is met, otherwise false_value . For example, IF(5 > 3, 5, 3) returns 5 .
IS_ALPHA(string)	Returns TRUE if all characters in the string are letters, otherwise FALSE .
IS_DECIMAL(string)	Returns TRUE if string can be parsed to a valid numeric, otherwise FALSE .
IS_DIGIT(string)	Returns TRUE if all characters in the string are digits, otherwise FALSE .

2.5.2.5 Type Conversion Function

Syntax

```
CAST(value AS type)
```

Syntax Description

This function is used to forcibly convert types.

Precautions

If the input is **NULL**, **NULL** is returned.

Example

The following example converts the **amount** value to an integer.

```
insert into temp select cast(amount as INT) from source_stream;
```

Table 2-38 Examples of type conversion functions

Example	Description	Example
cast(v1 as string)	Converts v1 to a string. The value of v1 can be of the numeric type or of the timestamp, date, or time type.	<p>Table T1:</p> <pre> content (INT) ----- 5 </pre> <p>Statement:</p> <pre>SELECT cast(content as varchar) FROM T1;</pre> <p>Result:</p> <pre>"5"</pre>
cast (v1 as int)	Converts v1 to the int type. The value of v1 can be a number or a character.	<p>Table T1:</p> <pre> content (STRING) ----- "5" </pre> <p>Statement:</p> <pre>SELECT cast(content as int) FROM T1;</pre> <p>Result:</p> <pre>5</pre>

Example	Description	Example
cast(v1 as timestamp)	Converts v1 to the timestamp type. The value of v1 can be of the string , date , or time type.	<p>Table T1:</p> <pre> content (STRING) ----- "2018-01-01 00:00:01" </pre> <p>Statement:</p> <pre>SELECT cast(content as timestamp) FROM T1;</pre> <p>Result:</p> <pre>1514736001000</pre>
cast(v1 as date)	Converts v1 to the date type. The value of v1 can be of the string or timestamp type.	<p>Table T1:</p> <pre> content (TIMESTAMP) ----- 1514736001000 </pre> <p>Statement:</p> <pre>SELECT cast(content as date) FROM T1;</pre> <p>Result:</p> <pre>"2018-01-01"</pre>

 NOTE

Flink jobs do not support the conversion of **bigint** to **timestamp** using CAST. You can convert it using **to_timestamp**.

Detailed Sample Code

```

/** source */
CREATE
TABLE car_infos (cast_int_to_string int, cast_String_to_int string,
case_string_to_timestamp string, case_timestamp_to_date timestamp(3)) WITH (
'connector.type' = 'dis',
'connector.region' = 'xxxxx',
'connector.channel' = 'dis-input',
'format.type' = 'json'
);
/** sink */
CREATE
TABLE cars_infos_out (cast_int_to_string string, cast_String_to_int
int, case_string_to_timestamp timestamp(3), case_timestamp_to_date date) WITH (
'connector.type' = 'dis',
'connector.region' = 'xxxxx',
'connector.channel' = 'dis-output',
'format.type' = 'json'
);
/** Statistics on static car information*/
INSERT
INTO
cars_infos_out
SELECT
cast(cast_int_to_string as string),
cast(cast_String_to_int as int),
cast(case_string_to_timestamp as timestamp),
cast(case_timestamp_to_date as date)
FROM
car_infos;

```

2.5.2.6 Collection Functions

Description

Table 2-39 Collection functions

Function	Description
CARDINALITY(array)	Returns the number of elements in array.
array '[' integer ']'	Returns the element at position INT 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 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.

2.5.2.7 Value Construction Functions

Description

Table 2-40 Value construction functions

Function	Description
ROW(value1, [, value2]*) (value1, [, value2]*)	Returns a row created from a list of values (value1, value2,...).
ARRAY '[' value1 [, value2]* ']'	Returns an array created from a list of values (value1, value2, ...).
MAP '[' key1, value1 [, key2, value2]* ']'	Returns a map created from a list of key-value pairs ((value1, value2), (value3, value4), ...). The key-value pair is (key1, value1), (key2, value2).

2.5.2.8 Value Access Functions

Description

Table 2-41 Value access functions

Function	Description
tableName.compositeType.field	Returns the value of a field from a Flink composite type (e.g., Tuple, POJO) by name.
tableName.compositeType.*	Returns a flat representation of a Flink composite type (e.g., Tuple, POJO) that converts each of its direct subtype into a separate field.

2.5.2.9 Hash Functions

Description

Table 2-42 Hash functions

Function	Description
MD5(string)	Returns the MD5 hash as a string that contains 32 hexadecimal digits. Returns NULL if string is NULL .
SHA1(string)	Returns the SHA-1 hash as a string that contains 40 hexadecimal digits. Returns NULL if string is NULL .
SHA224(string)	Returns the SHA-224 hash as a string that contains 56 hexadecimal digits. Returns NULL if string is NULL .
SHA256(string)	Returns the SHA-256 hash as a string that contains 64 hexadecimal digits. Returns NULL if string is NULL .
SHA384(string)	Returns the SHA-384 hash as a string that contains 96 hexadecimal digits. Returns NULL if string is NULL .
SHA512(string)	Returns the SHA-512 hash as a string that contains 128 hexadecimal digits. Returns NULL if string is NULL .

Function	Description
SHA2(string, hashLength)	Returns the hash using the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, or SHA-512). The first argument string is the string to be hashed and the second argument hashLength is the bit length of the result (224, 256, 384, or 512). Returns NULL if string or hashLength is NULL .

2.5.2.10 Aggregate Function

An aggregate function performs a calculation operation on a set of input values and returns a value. For example, the COUNT function counts the number of rows retrieved by an SQL statement. [Table 2-43](#) lists aggregate functions.

Table 2-43 Aggregate functions

Function	Return Data Type	Description
COUNT([ALL] expression DISTINCT expression1 [, expression2]*)	BIGINT	Returns the number of input rows for which the expression is not NULL. Use DISTINCT for one unique instance of each value.
COUNT(*) COUNT(1)	BIGINT	Returns the number of input rows.
AVG([ALL DISTINCT] expression)	DOUBLE	Returns the average (arithmetic mean) of expression across all input rows. Use DISTINCT for one unique instance of each value.
SUM([ALL DISTINCT] expression)	DOUBLE	Returns the sum of expression across all input rows. Use DISTINCT for one unique instance of each value.
MAX([ALL DISTINCT] expression)	DOUBLE	Returns the maximum value of expression across all input rows.
MIN([ALL DISTINCT] expression)	DOUBLE	Returns the minimum value of expression across all input rows.
STDDEV_POP([ALL DISTINCT] expression)	DOUBLE	Returns the population standard deviation of expression across all input rows.
STDDEV_SAMP([ALL DISTINCT] expression)	DOUBLE	Returns the sample standard deviation of expression across all input rows.

Function	Return Data Type	Description
VAR_POP([ALL DISTINCT] expression)	DOUBLE	Returns the population variance (square of the population standard deviation) of expression across all input rows.
VAR_SAMP([ALL DISTINCT] expression)	DOUBLE	Returns the sample variance (square of the sample standard deviation) of expression across all input rows.
COLLECT([ALL DISTINCT] expression)	MULTISET	Returns a multiset of expression across all input rows.
VARIANCE([ALL DISTINCT] expression)	DOUBLE	Returns the sample variance (square of the sample standard deviation) of expression across all input rows.
FIRST_VALUE(expression)	Actual type	Returns the first value in an ordered set of values.
LAST_VALUE(expression)	Actual type	Returns the last value in an ordered set of values.

2.5.2.11 Table-Valued Functions

2.5.2.11.1 split_cursor

The **split_cursor** function can convert one row of records into multiple rows or convert one column of records into multiple columns. Table-valued functions can only be used in JOIN LATERAL TABLE.

Table 2-44 split_cursor function

Function	Return Type	Description
split_cursor(value, delimiter)	cursor	Separates the "value" string into multiple rows of strings by using the delimiter.

Example

Input one record ("student1", "student2, student3") and output two records ("student1", "student2") and ("student1", "student3").

```
create table s1(attr1 string, attr2 string) with (.....);
insert into s2 select attr1, b1 from s1 left join lateral table(split_cursor(attr2, ',')) as T(b1) on true;
```


2.5.2.11.2 string_split

The **string_split** function splits a target string into substrings based on the specified separator and returns a substring list.

Description

```
string_split(target, separator)
```

Table 2-45 string_split parameters

Parameter	Type	Description
target	STRING	Target string to be processed NOTE <ul style="list-style-type: none">• If target is NULL, an empty line is returned.• If target contains two or more consecutive separators, an empty substring is returned.• If target does not contain a specified separator, the original string passed to target is returned.
separator	VARCHAR	Delimiter. Currently, only single-character delimiters are supported.

Example

1. Prepare test input data.

Table 2-46 Source table disSource

target (STRING)	separator (VARCHAR)
test-flink	-
flink	-
one-two-ww-three	-

2. Write test SQL statements.

```
create table disSource(  
  target STRING,  
  separator VARCHAR  
) with (  
  "connector.type" = "dis",  
  "connector.region" = "xxx",  
  "connector.channel" = "ygj-dis-in",  
  "format.type" = 'csv'
```

```
);
create table disSink(
  target STRING,
  item STRING
) with (
  'connector.type' = 'dis',
  'connector.region' = 'xxx',
  'connector.channel' = 'ygj-dis-out',
  'format.type' = 'csv'
);

insert into
  disSink
select
  target,
  item
from
  disSource,
lateral table(string_split(target, separator)) as T(item);
```

3. Check test results.

Table 2-47 disSink result table

target (STRING)	item (STRING)
test-flink	test
test-flink	flink
flink	flink
one-two-ww-three	one
one-two-ww-three	two
one-two-ww-three	ww
one-two-ww-three	three

3 Historical Version

3.1 Flink SQL Syntax (This Syntax Will Not Evolve. Use FlinkOpenSource SQL Instead.)

3.1.1 Constraints and Definitions

Syntax Constraints

- Currently, Flink SQL only supports the following operations: SELECT, FROM, WHERE, UNION, aggregation, window, JOIN between stream and table data, and JOIN between streams.
- Data cannot be inserted into the source stream.
- The sink stream cannot be used to perform query operations.

Data Types Supported by Syntax

- Basic data types: VARCHAR, STRING, BOOLEAN, TINYINT, SMALLINT, INTEGER/INT, BIGINT, REAL/FLOAT, DOUBLE, DECIMAL, DATE, TIME, and TIMESTAMP
- Array: Square brackets ([]) are used to quote fields. The following is an example:

```
insert into temp select CARDINALITY(ARRAY[1,2,3]) FROM OrderA;
```

Syntax Definition

```
INSERT INTO stream_name query;  
query:  
  values  
  | {  
    select  
    | selectWithoutFrom  
    | query UNION [ ALL ] query  
  }  
orderItem:  
  expression [ ASC | DESC ]  
select:
```

```

SELECT
{ * | projectItem [, projectItem ]* }
FROM tableExpression [ JOIN tableExpression ]
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ]

selectWithoutFrom:
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }

projectItem:
expression [ [ AS ] columnAlias ]
| tableAlias . *

tableExpression:
tableReference

tableReference:
tablePrimary
[ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]

tablePrimary:
[ TABLE ] [ [ catalogName . ] schemaName . ] tableName
| LATERAL TABLE '(' functionName '(' expression [, expression ]* ')' ')'
| UNNEST '(' expression ')'

values:
VALUES expression [, expression ]*

groupItem:
expression
| '(' ')'
| '(' expression [, expression ]* ')'
| CUBE '(' expression [, expression ]* ')'
| ROLLUP '(' expression [, expression ]* ')'
| GROUPING SETS '(' groupItem [, groupItem ]* ')'

```

3.1.2 Overview

This section describes the Flink SQL syntax list provided by DLI. For details about the parameters and examples, see the syntax description.

Table 3-1 SQL Syntax of stream jobs

Classification	Function
Creating a Source Stream	CloudTable HBase Source Stream
Creating a Source Stream	DIS Source Stream
	DMS Source Stream
Creating a Source Stream	MRS Kafka Source Stream
	Open-Source Kafka Source Stream
	OBS Source Stream
Creating a Sink Stream	CloudTable HBase Sink Stream
Creating a Sink Stream	CloudTable OpenTSDB Sink Stream
Creating a Sink Stream	CSS Elasticsearch Sink Stream

Classification	Function
	DCS Sink Stream
	DDS Sink Stream
	DIS Sink Stream
	DMS Sink Stream
	DWS Sink Stream (JDBC Mode)
	DWS Sink Stream (OBS-based Dumping)
Creating a Sink Stream	MRS HBase Sink Stream
	MRS Kafka Sink Stream
	Open-Source Kafka Sink Stream
	OBS Sink Stream
	RDS Sink Stream
Creating a Sink Stream	SMN Sink Stream
	File System Sink Stream (Recommended)
Creating a Temporary Stream	Creating a Temporary Stream
Creating a Dimension Table	Creating a Redis Table
	Creating an RDS Table
Custom Stream Ecosystem	Custom Source Stream
	Custom Sink Stream

3.1.3 Creating a Source Stream

3.1.3.1 CloudTable HBase Source Stream

Function

Create a source stream to obtain data from HBase of CloudTable as input data of the job. HBase is a column-oriented distributed cloud storage system that features enhanced reliability, excellent performance, and elastic scalability. It applies to the storage of massive amounts of data and distributed computing. You can use HBase to build a storage system capable of storing TB- or even PB-level data. With HBase, you can filter and analyze data with ease and get responses in milliseconds, rapidly mining data value. DLI can read data from HBase for filtering, analysis, and data dumping.

CloudTable is a distributed, scalable, and fully-hosted key-value data storage service based on Apache HBase. It provides DLI with high-performance random

read and write capabilities, which are helpful when applications need to store and query a massive amount of structured data, semi-structured data, and time series data. CloudTable applies to IoT scenarios and storage and query of massive volumes of key-value data. For more information about CloudTable, see the *CloudTable Service User Guide*.

Prerequisites

In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with CloudTable HBase. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Syntax

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH (
  type = "cloudtable",
  region = "",
  cluster_id = "",
  table_name = "",
  table_columns = ""
);
```

Keyword

Table 3-2 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. CloudTable indicates that the data source is CloudTable.
region	Yes	Region to which CloudTable belongs.
cluster_id	Yes	ID of the cluster to which the data table to be read belongs. For details about how to view the ID of the CloudTable cluster, see section " Viewing Basic Cluster Information " in the .
table_name	Yes	Name of the table from which data is to be read. If a namespace needs to be specified, set it to namespace_name:table_name .
table_columns	Yes	Column to be read. The format is rowKey,f1:c1,f1:c2,f2:c1 . The number of columns must be the same as the number of attributes specified in the source stream.

Precautions

When creating a source stream, you can specify a time model for subsequent calculation. Currently, DLI supports two time models: Processing Time and Event Time. For details about the syntax, see [Configuring Time Models](#).

Example

Read the `car_infos` table from HBase of CloudTable.

```
CREATE SOURCE STREAM car_infos (  
  car_id STRING,  
  car_owner STRING,  
  car_age INT,  
  average_speed INT,  
  total_miles INT  
)  
WITH (  
  type = "cloudtable",  
  region = "xxx",  
  cluster_id = "209ab1b6-de25-4c48-8e1e-29e09d02de28",  
  table_name = "carinfo",  
  table_columns = "rowKey,info:owner,info:age,car:speed,car:miles"  
);
```

3.1.3.2 DIS Source Stream

Function

Create a source stream to read data from DIS. DIS accesses user data and Flink job reads data from the DIS stream as input data for jobs. Flink jobs can quickly remove data from producers using DIS source sources for continuous processing. Flink jobs are applicable to scenarios where data outside the cloud service is imported to the cloud service for filtering, real-time analysis, monitoring reports, and dumping.

DIS addresses the challenge of transmitting data outside cloud services to cloud services. DIS builds data intake streams for custom applications capable of processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, Internet of Things (IoT) data, social media feeds, website clickstreams, and location-tracking events. For more information about DIS, see the *Data Ingestion Service User Guide*.

Syntax

```
CREATE SOURCE STREAM stream_id (attr_name attr_type ('|' attr_name attr_type)* )  
WITH (  
  type = "dis",  
  region = "",  
  channel = "",  
  partition_count = "",  
  encode = "",  
  field_delimiter = "",  
  offset= "");
```

Keyword

Table 3-3 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. dis indicates that the data source is DIS.
region	Yes	Region where DIS for storing the data is located.
ak	No	Access Key ID (AK). For details about how to obtain the access key, see My Credentials .
sk	No	Specifies the secret access key used together with the ID of the access key. For details about how to obtain the access key, see My Credentials .
channel	Yes	Name of the DIS stream where data is located.
partition_count	No	Number of partitions of the DIS stream where data is located. This parameter and partition_range cannot be configured at the same time. If this parameter is not specified, data of all partitions is read by default.
partition_range	No	Range of partitions of a DIS stream, data in which is ingested by the DLI job. This parameter and partition_count cannot be configured at the same time. If this parameter is not specified, data of all partitions is read by default. If you set this parameter to [0:2] , data will be read from partitions 1, 2, and 3.
encode	Yes	Data encoding format. The value can be csv , json , xml , email , blob , or user_defined . <ul style="list-style-type: none">• field_delimiter must be specified if this parameter is set to csv.• json_config must be specified if this parameter is set to json.• xml_config must be specified if this parameter is set to xml.• email_key must be specified if this parameter is set to email.• If this parameter is set to blob, the received data is not parsed, only one stream attribute exists, and the data format is ARRAY[TINYINT].• encode_class_name and encode_class_parameter must be specified if this parameter is set to user_defined.
field_delimiter	No	Attribute delimiter. This parameter is mandatory only when the CSV encoding format is used. You can set this parameter, for example, to a comma (,).

Parameter	Mandatory	Description
quote	No	<p>Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters.</p> <ul style="list-style-type: none"> If double quotation marks are used as the quoted symbol, set this parameter to <code>\u005c\u0022</code> for character conversion. If a single quotation mark is used as the quoted symbol, set this parameter to a single quotation mark ('). <p>NOTE</p> <ul style="list-style-type: none"> Currently, only the CSV format is supported. After this parameter is specified, ensure that each field does not contain quoted symbols or contains an even number of quoted symbols. Otherwise, parsing will fail.
json_config	No	<p>When the encoding format is JSON, you need to use this parameter to specify the mapping between JSON fields and stream definition fields. The format is field1=data_json.field1; field2=data_json.field2; field3=\$, where field3=\$ indicates that the content of field3 is the entire JSON string.</p>
xml_config	No	<p>If encode is set to xml, you need to set this parameter to specify the mapping between the xml field and the stream definition field. An example of the format is as follows: field1=data_xml.field1; field2=data_xml.field2.</p>
email_key	No	<p>If encode is set to email, you need to set the parameter to specify the information to be extracted. You need to list the key values that correspond to stream definition fields. Multiple key values are separated by commas (,), for example, "Message-ID, Date, Subject, body". There is no keyword in the email body and DLI specifies "body" as the keyword.</p>
encode_class_name	No	<p>If encode is set to user_defined, you need to set this parameter to the name of the user-defined decoding class (including the complete package path). The class must inherit the DeserializationSchema class.</p>
encode_class_parameter	No	<p>If encode is set to user_defined, you can set this parameter to specify the input parameter of the user-defined decoding class. Only one parameter of the string type is supported.</p>
offset	No	<ul style="list-style-type: none"> If data is imported to the DIS stream after the job is started, this parameter will become invalid. If the job is started after data is imported to the DIS stream, you can set the parameter as required. For example, if offset is set to 100, DLI starts from the 100th data record in DIS.

Parameter	Mandatory	Description
start_time	No	Start time for reading DIS data. <ul style="list-style-type: none"> If this parameter is specified, DLI reads data read from the specified time. The format is yyyy-MM-dd HH:mm:ss. If neither start_time nor offset is specified, DLI reads the latest data. If start_time is not specified but offset is specified, DLI reads data from the data record specified by offset.
enable_checkpoint	No	Whether to enable the checkpoint function. The value can be true (enabled) or false (disabled). The default value is false .
checkpoint_app_name	No	ID of a DIS consumer. If a DIS stream is consumed by different jobs, you need to configure the consumer ID for each job to avoid checkpoint confusion.
checkpoint_interval	No	Interval of checkpoint operations on the DIS source operator. The value is in the unit of seconds. The default value is 60 .

Precautions

When creating a source stream, you can specify a time model for subsequent calculation. Currently, DLI supports two time models: Processing Time and Event Time. For details about the syntax, see [Configuring Time Models](#).

Example

- In CSV encoding format, DLI reads data from the DIS stream and records it as codes in CSV format. The codes are separated by commas (.).

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT,
  car_timestamp LONG
)
WITH (
  type = "dis",
  region = "xxx",
  channel = "dliinput",
  encode = "csv",
  field_delimiter = ","
);
```

- In JSON encoding format, DLI reads data from the DIS stream and records it as codes in JSON format. For example, {"car":{"car_id":"ZJA710XC", "car_owner":"coco", "car_age":5, "average_speed":80, "total_miles":15000, "car_timestamp":1526438880}}

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
```

```
car_owner STRING,  
car_age INT,  
average_speed INT,  
total_miles INT,  
car_timestamp LONG  
)  
WITH (  
  type = "dis",  
  region = "xxx",  
  channel = "dliinput",  
  encode = "json",  
  json_config = "car_id=car.car_id;car_owner =car.car_owner;car_age=car.car_age;average_speed  
=car.average_speed ;total_miles=car.total_miles;"  
);
```

- In XML encoding format, DLI reads data from the DIS stream and records it as codes in XML format.

```
CREATE SOURCE STREAM person_infos (  
  pid BIGINT,  
  pname STRING,  
  page int,  
  plocation STRING,  
  pbir DATE,  
  phealthy BOOLEAN,  
  pgrade ARRAY[STRING]  
)  
WITH (  
  type = "dis",  
  region = "xxx",  
  channel = "dis-dli-input",  
  encode = "xml",  
  field_delimiter = ",",  
  xml_config =  
"pid=person.pid;page=person.page;pname=person.pname;plocation=person.plocation;pbir=person.pbir;  
pgrade=person.pgrade;phealthy=person.phealthy"  
);
```

An example of XML data is as follows:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<root>  
  <person>  
    <pid>362305199010025042</pid>  
    <pname>xiaoming</pname>  
    <page>28</page>  
    <plocation>xxx</plocation>  
    <pbir>1990-10-02</pbir>  
    <phealthy>true</phealthy>  
    <pgrade>[A,B,C]</pgrade>  
  </person>  
</root>
```

- In EMAIL encoding format, DLI reads data from the DIS stream and records it as a complete Email.

```
CREATE SOURCE STREAM email_infos (  
  Event_ID String,  
  Event_Time Date,  
  Subject String,  
  From_Email String,  
  To_EMAIL String,  
  CC_EMAIL Array[String],  
  BCC_EMAIL String,  
  MessageBody String,  
  Mime_Version String,  
  Content_Type String,  
  charset String,  
  Content_Transfer_Encoding String  
)  
WITH (  
  type = "dis",
```

```
region = "xxx",  
channel = "dliinput",  
encode = "email",  
email_key = "Message-ID, Date, Subject, From, To, CC, BCC, Body, Mime-Version, Content-Type,  
charset, Content_Transfer_Encoding"  
);
```

An example of email data is as follows:

```
Message-ID: <200906291839032504254@sample.com>  
Date: Fri, 11 May 2001 09:54:00 -0700 (PDT)  
From: zhangsan@sample.com  
To: lisi@sample.com, wangwu@sample.com  
Subject: "Hello World"  
Cc: lilei@sample.com, hanmei@sample.com  
Mime-Version: 1.0  
Content-Type: text/plain; charset=us-ascii  
Content-Transfer-Encoding: 7bit  
Bcc: jack@sample.com, lily@sample.com  
X-From: Zhang San  
X-To: Li Si, Wang Wu  
X-cc: Li Lei, Han Mei  
X-bcc:  
X-Folder: \Li_Si_June2001\Notes Folders\Notes inbox  
X-Origin: Lucy  
X-FileName: sample.nsf
```

Dear Associate / Analyst Committee:

Hello World!

Thank you,

Associate / Analyst Program
zhangsan

3.1.3.3 DMS Source Stream

DMS (Distributed Message Service) is a message middleware service based on distributed, high-availability clustering technology. It provides reliable, scalable, fully managed queues for sending, receiving, and storing messages. DMS for Kafka is a message queuing service based on Apache Kafka. This service provides Kafka premium instances.

The source stream can read data from a Kafka instance as the input data of jobs. The syntax for creating a Kafka source stream is the same as that for creating an open source Apache Kafka source stream. For details, see [Open-Source Kafka Source Stream](#).

3.1.3.4 MRS Kafka Source Stream

Function

Create a source stream to obtain data from Kafka as input data for jobs.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages. Kafka clusters are deployed and hosted on MRS that is powered on Apache Kafka.

Prerequisites

- If the Kafka server listens on the port using hostname, you need to add the mapping between the hostname and IP address of the Kafka Broker node to the DLI queue. Contact the Kafka service deployment personnel to obtain the hostname and IP address of the Kafka Broker node. For details about how to add an IP-domain mapping, see [Modifying the Host Information](#) in the *Data Lake Insight User Guide*.
- Kafka is an offline cluster. You need to use the enhanced datasource connection function to connect Flink jobs to Kafka. You can also set security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH (
  type = "kafka",
  kafka_bootstrap_servers = "",
  kafka_group_id = "",
  kafka_topic = "",
  encode = "json"
);
```

Keyword

Table 3-4 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. Kafka indicates that the data source is Kafka.
kafka_bootstrap_servers	Yes	Port that connects DLI to Kafka. Use enhanced datasource connections to connect DLI queues with Kafka clusters.
kafka_group_id	No	Group ID
kafka_topic	Yes	Kafka topic to be read. Currently, only one topic can be read at a time.

Parameter	Mandatory	Description
encode	Yes	<p>Data encoding format. The value can be csv, json, blob, or user_defined.</p> <ul style="list-style-type: none"> • field_delimiter must be specified if this parameter is set to csv. • json_config must be specified if this parameter is set to json. • If this parameter is set to blob, the received data is not parsed, only one stream attribute exists, and the stream attribute is of the Array[TINYINT] type. • encode_class_name and encode_class_parameter must be specified if this parameter is set to user_defined.
encode_class_name	No	<p>If encode is set to user_defined, you need to set this parameter to the name of the user-defined decoding class (including the complete package path). The class must inherit the DeserializationSchema class.</p>
encode_class_parameter	No	<p>If encode is set to user_defined, you can set this parameter to specify the input parameter of the user-defined decoding class. Only one parameter of the string type is supported.</p>
krb_auth	No	<p>The authentication name for creating a datasource connection authentication. This parameter is mandatory when Kerberos authentication is enabled.</p> <p>NOTE Ensure that the /etc/hosts information of the master node in the MRS cluster is added to the host file of the DLI queue.</p>
json_config	No	<p>If encode is set to json, you can use this parameter to specify the mapping between JSON fields and stream attributes.</p> <p>The format is field1=json_field1;field2=json_field2.</p> <p>field1 and field2 indicate the names of the created table fields. json_field1 and json_field2 are key fields of the JSON strings in the Kafka input data.</p> <p>For details, see the example.</p>
field_delimiter	No	<p>If encode is set to csv, you can use this parameter to specify the separator between CSV fields. By default, the comma (,) is used.</p>

Parameter	Mandatory	Description
quote	No	<p>Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters.</p> <ul style="list-style-type: none"> If double quotation marks are used as the quoted symbol, set this parameter to <code>\u005c\u0022</code> for character conversion. If a single quotation mark is used as the quoted symbol, set this parameter to a single quotation mark (<code>'</code>). <p>NOTE</p> <ul style="list-style-type: none"> Currently, only the CSV format is supported. After this parameter is specified, ensure that each field does not contain quoted symbols or contains an even number of quoted symbols. Otherwise, parsing will fail.
start_time	No	<p>Start time when Kafka data is ingested.</p> <p>If this parameter is specified, DLI reads data read from the specified time. The format is yyyy-MM-dd HH:mm:ss. Ensure that the value of start_time is not later than the current time. Otherwise, no data will be obtained.</p>
kafka_properties	No	<p>This parameter is used to configure the native attributes of Kafka. The format is key1=value1;key2=value2.</p>
kafka_certificate_name	No	<p>Specifies the name of the datasource authentication information. This parameter is valid only when the datasource authentication type is set to Kafka_SSL.</p> <p>NOTE</p> <ul style="list-style-type: none"> If this parameter is specified, the service loads only the specified file and password under the authentication. The system automatically sets this parameter to kafka_properties. Other configuration information required for Kafka SSL authentication needs to be manually configured in the kafka_properties attribute.

Precautions

When creating a source stream, you can specify a time model for subsequent calculation. Currently, DLI supports two time models: Processing Time and Event Time. For details about the syntax, see [Configuring Time Models](#).

Example

- Read data from the Kafka topic **test**.

```
CREATE SOURCE STREAM kafka_source (  
  name STRING,  
  age int  
)  
WITH (  
  type = "kafka",  
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",  
  kafka_group_id = "sourcegroup1",  
  kafka_topic = "test",  
  encode = "json"  
);
```

- Read the topic whose object is **test** from Kafka and use **json_config** to map JSON data to table fields.

The data encoding format is non-nested JSON.

```
{"attr1": "lilei", "attr2": 18}
```

The table creation statement is as follows:

```
CREATE SOURCE STREAM kafka_source (name STRING, age int)  
WITH (  
  type = "kafka",  
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",  
  kafka_group_id = "sourcegroup1",  
  kafka_topic = "test",  
  encode = "json",  
  json_config = "name=attr1;age=attr2"  
);
```

3.1.3.5 Open-Source Kafka Source Stream

Function

Create a source stream to obtain data from Kafka as input data for jobs.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

- If the Kafka server listens on the port using hostname, you need to add the mapping between the hostname and IP address of the Kafka Broker node to the DLI queue. Contact the Kafka service deployment personnel to obtain the hostname and IP address of the Kafka Broker node. For details about how to add an IP-domain mapping, see [Modifying the Host Information](#) in the *Data Lake Insight User Guide*.
- Kafka is an offline cluster. You need to use the enhanced datasource connection function to connect Flink jobs to Kafka. You can also set security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (';' attr_name attr_type)* )
WITH (
  type = "kafka",
  kafka_bootstrap_servers = "",
  kafka_group_id = "",
  kafka_topic = "",
  encode = "json",
  json_config=""
);
```

Keyword

Table 3-5 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. Kafka indicates that the data source is Kafka.
kafka_bootstrap_servers	Yes	Port that connects DLI to Kafka. Use enhanced datasource connections to connect DLI queues with Kafka clusters.
kafka_group_id	No	Group ID.
kafka_topic	Yes	Kafka topic to be read. Currently, only one topic can be read at a time.
encode	Yes	Data encoding format. The value can be csv , json , blob , or user_defined . <ul style="list-style-type: none"> • field_delimiter must be specified if this parameter is set to csv. • json_config must be specified if this parameter is set to json. • If this parameter is set to blob, the received data will not be parsed, and only one Array[TINYINT] field exists in the table. • encode_class_name and encode_class_parameter must be specified if this parameter is set to user_defined.
encode_class_name	No	If encode is set to user_defined , you need to set this parameter to the name of the user-defined decoding class (including the complete package path). The class must inherit the Deserialization-Schema class.
encode_class_parameter	No	If encode is set to user_defined , you can set this parameter to specify the input parameter of the user-defined decoding class. Only one parameter of the string type is supported.

Parameter	Mandatory	Description
json_config	No	<p>If encode is set to json, you can use this parameter to specify the mapping between JSON fields and stream attributes.</p> <p>The format is <code>field1=json_field1;field2=json_field2</code>.</p> <p>field1 and field2 indicate the names of the created table fields. json_field1 and json_field2 are key fields of the JSON strings in the Kafka input data.</p> <p>For details, see Example.</p> <p>NOTE</p> <p>If the attribute names in the source stream are the same as those in JSON fields, you do not need to set this parameter.</p>
field_delimiter	No	<p>If encode is set to csv, you can use this parameter to specify the separator between CSV fields. By default, the comma (,) is used.</p>
quote	No	<p>Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters.</p> <ul style="list-style-type: none"> If double quotation marks are used as the quoted symbol, set this parameter to <code>\u005c\u0022</code> for character conversion. If a single quotation mark is used as the quoted symbol, set this parameter to a single quotation mark ('). <p>NOTE</p> <ul style="list-style-type: none"> Currently, only the CSV format is supported. After this parameter is specified, ensure that each field does not contain quoted symbols or contains an even number of quoted symbols. Otherwise, parsing will fail.
start_time	No	<p>Start time when Kafka data is ingested.</p> <p>If this parameter is specified, DLI reads data read from the specified time. The format is yyyy-MM-dd HH:mm:ss. Ensure that the value of start_time is not later than the current time. Otherwise, no data will be obtained.</p> <p>If you set this parameter, only the data generated after the specified time for the Kafka topic will be read.</p>
kafka_properties	No	<p>Native properties of Kafka. The format is key1=value1;key2=value2. For details about the property values, see the description in Apache Kafka.</p>

Parameter	Mandatory	Description
kafka_certificate_name	No	<p>Name of the datasource authentication information. This parameter is valid only when the datasource authentication type is set to Kafka_SSL.</p> <p>NOTE</p> <ul style="list-style-type: none"> If this parameter is specified, the service loads only the specified file and password under the authentication. The system automatically sets this parameter to kafka_properties. Other configuration information required for Kafka SSL authentication needs to be manually configured in the kafka_properties attribute.

Precautions

When creating a source stream, you can specify a time model for subsequent calculation. Currently, DLI supports two time models: Processing Time and Event Time. For details about the syntax, see [Configuring Time Models](#).

Example

- Read Kafka topic **test**. The data encoding format is non-nested JSON, for example, {"attr1": "lilei", "attr2": 18}.

```
CREATE SOURCE STREAM kafka_source (name STRING, age int)
WITH (
  type = "kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_group_id = "sourcegroup1",
  kafka_topic = "test",
  encode = "json",
  json_config = "name=attr1;age=attr2"
);
```

- Read Kafka topic **test**. The data is encoded in JSON format and nested. This example uses the complex data type ROW. For details about the syntax of ROW, see [Data Types](#).

The test data is as follows:

```
{
  "id": "1",
  "type2": "online",
  "data": {
    "patient_id": "1234",
    "name": "bob1234"
  }
}
```

An example of the table creation statements is as follows:

```
CREATE SOURCE STREAM kafka_source
(
  id STRING,
  type2 STRING,
  data ROW<
    patient_id STRING,
    name STRING>
)
WITH (
  type = "kafka",
```

```

kafka_bootstrap_servers = "ip1:port1,ip2:port2",
kafka_group_id = "sourcegroup1",
kafka_topic = "test",
encode = "json"
);

CREATE SINK STREAM kafka_sink
(
  id STRING,
  type2 STRING,
  patient_id STRING,
  name STRING
)
WITH (
  type="kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_topic = "testsink",
  encode = "csv"
);

INSERT INTO kafka_sink select id, type2, data.patient_id, data.name from kafka_source;

```

3.1.3.6 OBS Source Stream

Function

Create a source stream to obtain data from OBS. DLI reads data stored by users in OBS as input data for jobs. OBS applies to various scenarios, such as big data analysis, cloud-native application program data, static website hosting, backup/active archive, and deep/cold archive.

OBS is an object-based storage service. It provides massive, secure, highly reliable, and low-cost data storage capabilities. For more information about OBS, see the [Object Storage Service Console Operation Guide](#)

Syntax

```

CREATE SOURCE STREAM stream_id (attr_name attr_type (';' attr_name attr_type)*)
WITH (
  type = "obs",
  region = "",
  bucket = "",
  object_name = "",
  row_delimiter = "\n",
  field_delimiter = ",
  version_id = ""
);

```

Keyword

Table 3-6 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. obs indicates that the data source is OBS.
region	Yes	Region to which OBS belongs.

Parameter	Mandatory	Description
encode	No	Data encoding format. The value can be csv or json . The default value is csv .
ak	No	Access Key ID (AK). For details about how to obtain the access key, see My Credentials .
sk	No	Secret access key used together with the ID of the access key. For details about how to obtain the access key, see My Credentials .
bucket	Yes	Name of the OBS bucket where data is located.
object_name	Yes	Name of the object stored in the OBS bucket where data is located. If the object is not in the OBS root directory, you need to specify the folder name, for example, test/test.csv . For the object file format, see the encode parameter.
row_delimiter	Yes	Separator used to separate every two rows.
field_delimiter	No	Separator used to separate every two attributes. <ul style="list-style-type: none"> This parameter is mandatory when encode is csv. You use custom attribute separators. If encode is json, you do not need to set this parameter.
quote	No	Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters. <ul style="list-style-type: none"> If double quotation marks are used as the quoted symbol, set this parameter to <code>\u005c\u0022</code> for character conversion. If a single quotation mark is used as the quoted symbol, set this parameter to a single quotation mark (<code>'</code>). <p>NOTE</p> <ul style="list-style-type: none"> Currently, only the CSV format is supported. After this parameter is specified, ensure that each field does not contain quoted symbols or contains an even number of quoted symbols. Otherwise, parsing will fail.
version_id	No	Version number. This parameter is optional and required only when the OBS bucket or object has version settings.

Precautions

When creating a source stream, you can specify a time model for subsequent calculation. Currently, DLI supports two time models: Processing Time and Event Time. For details about the syntax, see [Configuring Time Models](#).

Example

- The **input.csv** file is read from the OBS bucket. Rows are separated by '\n' and columns are separated by ','.

To use the test data, create an **input.txt** file, copy and paste the following text data, and save the file as **input.csv**. Upload the **input.csv** file to the target OBS bucket directory. For example, upload the file to the **dli-test-obs01** bucket directory.

```
1,2,3,4,1403149534
5,6,7,8,1403149535
```

The following is an example for creating the table:

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
WITH (
  type = "obs",
  bucket = "dli-test-obs01",
  region = "xxx",
  object_name = "input.csv",
  row_delimiter = "\n",
  field_delimiter = ","
);
```

- The **input.json** file is read from the OBS bucket. Rows are separated by '\n'.

```
CREATE SOURCE STREAM obs_source (
  str STRING
)
WITH (
  type = "obs",
  bucket = "obssource",
  region = "xxx",
  encode = "json",
  row_delimiter = "\n",
  object_name = "input.json"
);
```

3.1.4 Creating a Sink Stream

3.1.4.1 CloudTable HBase Sink Stream

Function

DLI exports the job output data to HBase of CloudTable. HBase is a column-oriented distributed cloud storage system that features enhanced reliability, excellent performance, and elastic scalability. It applies to the storage of massive amounts of data and distributed computing. You can use HBase to build a storage system capable of storing TB- or even PB-level data. With HBase, you can filter and analyze data with ease and get responses in milliseconds, rapidly mining data value. Structured and semi-structured key-value data can be stored, including messages, reports, recommendation data, risk control data, logs, and orders. With DLI, you can write massive volumes of data to HBase at a high speed and with low latency.

CloudTable is a distributed, scalable, and fully-hosted key-value data storage service based on Apache HBase. It provides DLI with high-performance random

read and write capabilities, which are helpful when applications need to store and query a massive amount of structured data, semi-structured data, and time series data. CloudTable applies to IoT scenarios and storage and query of massive volumes of key-value data. For more information about CloudTable, see the *CloudTable Service User Guide*.

Prerequisites

In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with CloudTable HBase. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
  type = "cloudtable",
  region = "",
  cluster_id = "",
  table_name = "",
  table_columns = "",
  create_if_not_exist = ""
)
```

Keyword

Table 3-7 Keyword description

Parameter	Man datory	Description
type	Yes	Output channel type. cloudtable indicates that data is exported to CloudTable (HBase).
region	Yes	Region to which CloudTable belongs.
cluster_id	Yes	ID of the cluster to which the data you want to insert belongs.
table_name	Yes	Name of the table, into which data is to be inserted. It can be specified through parameter configurations. For example, if you want one or more certain columns as part of the table name, use car_pass_inspect_with_age_{car_age} , where car_age is the column name.
table_columns	Yes	Columns to be inserted. The format is rowKey, f1:c1, f1:c2, f2:c1 , where rowKey must be specified. If you do not want to add a column (for example, the third column) to the database, set this parameter to rowKey,f1:c1,,f2:c1 .

Parameter	Mandatory	Description
illegal_data_table	No	If this parameter is specified, abnormal data (for example, rowKey does not exist) will be written into the table. If not specified, abnormal data will be discarded. The rowKey value is a timestamp followed by six random digits, and the schema is info:data, info:reason.
create_if_not_exist	No	Whether to create a table or column into which the data is written when this table or column does not exist. The value can be true or false . The default value is false .
batch_insert_data_num	No	Number of data records to be written in batches at a time. The value must be a positive integer. The upper limit is 100 . The default value is 10 .

Precautions

- If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_\${car_brand}** and the value of **car_brand** in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.
- In this way, data is written to HBase of CloudTable. The speed is limited. The dedicated resource mode is recommended.

For details about how to create a dedicated resource mode, see [Creating a Queue](#) in the *Data Lake Insight User Guide*.

Example

Output data of stream **qualified_cars** to CloudTable (HBase).

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
WITH (
  type = "cloudtable",
  region = "xxx",
  cluster_id = "209ab1b6-de25-4c48-8e1e-29e09d02de28",
  table_name = "car_pass_inspect_with_age_${car_age}",
  table_columns = "rowKey,info:owner,,car:speed,car:miles",
  illegal_data_table = "illegal_data",
  create_if_not_exist = "true",
  batch_insert_data_num = "20"
);
```


3.1.4.2 CloudTable OpenTSDB Sink Stream

Function

DLI exports the job output data to OpenTSDB of CloudTable. OpenTSDB is a distributed, scalable time series database based on HBase. It stores time series data. Time series data refers to the data collected at different time points. This type of data reflects the change status or degree of an object over time. OpenTSDB supports data collection and monitoring in seconds, permanent storage, index, and queries. It can be used for system monitoring and measurement as well as collection and monitoring of IoT data, financial data, and scientific experimental results.

CloudTable is a distributed, scalable, and fully-hosted key-value data storage service based on Apache HBase. It provides DLI with high-performance random read and write capabilities, which are helpful when applications need to store and query a massive amount of structured data, semi-structured data, and time series data. CloudTable applies to IoT scenarios and storage and query of massive volumes of key-value data. For more information about CloudTable, see the *CloudTable Service User Guide*.

Prerequisites

- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with CloudTable HBase. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
  type = "opentsdb",
  region = "",
  cluster_id = "",
  tsdb_metrics = "",
  tsdb_timestamps = "",
  tsdb_values = "",
  tsdb_tags = "",
  batch_insert_data_num = ""
)
```

Keyword

Table 3-8 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. opentsdb indicates that data is exported to CloudTable (OpenTSDB).

Parameter	Mandatory	Description
region	Yes	Region to which CloudTable belongs.
cluster_id	No	ID of the cluster to which the data to be inserted belongs. Either this parameter or tsdb_link_address must be specified.
tsdb_metrics	Yes	Metric of a data point, which can be specified through parameter configurations.
tsdb_timestamps	Yes	Timestamp of a data point. The data type can be LONG, INT, SHORT, or STRING. Only dynamic columns are supported.
tsdb_values	Yes	Value of a data point. The data type can be SHORT, INT, LONG, FLOAT, DOUBLE, or STRING. Dynamic columns or constant values are supported.
tsdb_tags	Yes	Tags of a data point. Each of tags contains at least one tag value and up to eight tag values. Tags of the data point can be specified through parameter configurations.
batch_insert_data_num	No	Number of data records to be written in batches at a time. The value must be a positive integer. The upper limit is 65536 . The default value is 8 .
tsdb_link_address	No	OpenTSDB link of the cluster to which the data to be inserted belongs. If this parameter is used, the job must run in a dedicated DLI queue, and the DLI queue must be connected to the CloudTable cluster through an enhanced datasource connection. Either this parameter or cluster_id must be specified. NOTE For details about how to configure security group rules, see Security Group in the Virtual Private Cloud User Guide .

Precautions

If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_\${car_brand}** and the value of **car_brand** in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.

Example

Output data of stream **weather_out** to CloudTable (OpenTSDB).

```
CREATE SINK STREAM weather_out (
  timestamp_value LONG, /* Time */
  temperature FLOAT, /* Temperature value */
  humidity FLOAT, /* Humidity */
```

```
location STRING /* Location */
)
WITH (
  type = "opentsdb",
  region = "xxx",
  cluster_id = "e05649d6-00e2-44b4-b0ff-7194adaeab3f",
  tsdb_metrics = "weather",
  tsdb_timestamps = "${timestamp_value}",
  tsdb_values = "${temperature}; ${humidity}",
  tsdb_tags = "location:${location},signify:temperature; location:${location},signify:humidity",
  batch_insert_data_num = "10"
);
```

3.1.4.3 MRS OpenTSDB Sink Stream

Function

DLI exports the output data of the Flink job to OpenTSDB of MRS.

Prerequisites

- OpenTSDB has been installed in the MRS cluster.
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with MRS clusters. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
  type = "opentsdb",
  region = "",
  tsdb_metrics = "",
  tsdb_timestamps = "",
  tsdb_values = "",
  tsdb_tags = "",
  batch_insert_data_num = ""
)
```

Keyword

Table 3-9 Keyword description

Parameter	Mandatory	Description
type	Yes	Sink channel type. opentsdb indicates that data is exported to OpenTSDB of MRS.
region	Yes	Region where MRS resides.

Parameter	Mandatory	Description
tsdb_link_address	Yes	Service address of the OpenTSDB instance in MRS. The format is http://ip:port or https://ip:port . NOTE If tsd.https.enabled is set to true , HTTPS must be used. Note that HTTPS does not support certificate authentication.
tsdb_metrics	Yes	Metric of a data point, which can be specified through parameter configurations.
tsdb_timestamps	Yes	Timestamp of a data point. The data type can be LONG, INT, SHORT, or STRING. Only dynamic columns are supported.
tsdb_values	Yes	Value of a data point. The data type can be SHORT, INT, LONG, FLOAT, DOUBLE, or STRING. Dynamic columns or constant values are supported.
tsdb_tags	Yes	Tags of a data point. Each of tags contains at least one tag value and up to eight tag values. Tags of the data point can be specified through parameter configurations.
batch_insert_data_num	No	Number of data records to be written in batches at a time. The value must be a positive integer. The upper limit is 65536 . The default value is 8 .

Precautions

If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_\${car_brand}** and the value of **car_brand** in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.

Example

Output data of stream **weather_out** to OpenTSDB of MRS.

```
CREATE SINK STREAM weather_out (
  timestamp_value LONG, /* Time */
  temperature FLOAT, /* Temperature value */
  humidity FLOAT, /* Humidity */
  location STRING /* Location */
)
WITH (
  type = "opentsdb",
  region = "xxx",
  tsdb_link_address = "https://x.x.x.x:4242",
  tsdb_metrics = "weather",
  tsdb_timestamps = "${timestamp_value}",
  tsdb_values = "${temperature}; ${humidity}",
  tsdb_tags = "location:${location},signify:temperature; location:${location},signify:humidity",
  batch_insert_data_num = "10"
);
```

3.1.4.4 CSS Elasticsearch Sink Stream

Function

DLI exports Flink job output data to Elasticsearch of Cloud Search Service (CSS). Elasticsearch is a popular enterprise-class Lucene-powered search server and provides the distributed multi-user capabilities. It delivers multiple functions, including full-text retrieval, structured search, analytics, aggregation, and highlighting. With Elasticsearch, you can achieve stable, reliable, real-time search. Elasticsearch applies to diversified scenarios, such as log analysis and site search.

CSS is a fully managed, distributed search service. It is fully compatible with open-source Elasticsearch and provides DLI with structured and unstructured data search, statistics, and report capabilities.

For more information about CSS, see the *Cloud Search Service User Guide*.

NOTE

If the security mode is enabled when you create a CSS cluster, it cannot be undone.

Prerequisites

- Ensure that you have created a cluster on CSS using your account. For details about how to create a cluster on CSS, see [Creating a Cluster](#) in the *Cloud Search Service User Guide*.
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with CSS. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (';' attr_name attr_type)* )
WITH (
  type = "es",
  region = "",
  cluster_address = "",
  es_index = "",
  es_type= "",
  es_fields= "",
  batch_insert_data_num= ""
);
```

Keyword

Table 3-10 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. es indicates that data is exported to CSS.
region	Yes	Region where CSS is located.
cluster_addresses	Yes	Private access address of the CSS cluster, for example: x.x.x.x:x . Use commas (,) to separate multiple addresses.
es_index	Yes	Index of the data to be inserted. This parameter corresponds to CSS index. For details, see Cloud Search Service Overview .
es_type	Yes	Type of the document to which data is to be inserted. This parameter corresponds to the CSS type. For details, see Cloud Search Service Overview . If the Elasticsearch version is 6.x, the value cannot start with an underscore (_). If the Elasticsearch version is 7.x and the type of CSS is preset, the value must be _doc . Otherwise, the value must comply with CSS specifications.
es_fields	Yes	Key of the data field to be inserted. The format is id,f1,f2,f3,f4 . Ensure that the key corresponds to the data column in the sink. If a random attribute field instead of a key is used, the keyword id does not need to be used, for example, f1,f2,f3,f4,f5 . This parameter corresponds to the CSS field. For details, see Cloud Search Service Overview .
batch_insert_data_num	Yes	Amount of data to be written in batches at a time. The value must be a positive integer. The unit is 10 records. The maximum value allowed is 65536 , and the default value is 10 .
action	No	If the value is add , data is forcibly overwritten when the same ID is encountered. If the value is upsert , data is updated when the same ID is encountered. (If upsert is selected, id in the es_fields field must be specified.) The default value is add .
enable_output_null	No	This parameter is used to configure whether to generate an empty field. If this parameter is set to true , an empty field (the value is null) is generated. If set to false , no empty field is generated. The default value is false .

Parameter	Mandatory	Description
max_record_num_cache	No	Maximum number of records that can be cached.
es_certificate_name	No	<p>Name of the datasource authentication information</p> <p>For details about how to create cross-source authentication, see Creating and Managing Datasource Authentication.</p> <p>If the security mode is enabled and HTTPS is used by the Elasticsearch cluster, the certificate is required for access. In this case, set the datasource authentication type to CSS.</p> <p>If the security mode is enabled for the Elasticsearch cluster but HTTPS is disabled, the certificate and username and password are required for access. In this case, set the datasource authentication type to Password.</p>

Precautions

If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_ \$ {car_brand}** and the value of **car_brand** in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.

Example

Data of stream **qualified_cars** is exported to the cluster on CSS.

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
WITH (
  type = "es",
  region = "xxx",
  cluster_address = "192.168.0.212:9200",
  es_index = "car",
  es_type = "information",
  es_fields = "id,owner,age,speed,miles",
  batch_insert_data_num = "10"
);
```

3.1.4.5 DCS Sink Stream

Function

DLI exports the Flink job output data to Redis of DCS. Redis is a storage system that supports multiple types of data structures such as key-value. It can be used in

scenarios such as caching, event pub/sub, and high-speed queuing. Redis supports direct read/write of strings, hashes, lists, queues, and sets. Redis works with in-memory dataset and provides persistence. For more information about Redis, visit <https://redis.io/>.

DCS provides Redis-compatible, secure, reliable, out-of-the-box, distributed cache capabilities allowing elastic scaling and convenient management. It meets users' requirements for high concurrency and fast data access.

For more information about DCS, see the [Distributed Cache Service User Guide](#).

Prerequisites

- Ensure that You have created a Redis cache instance on DCS using your account.
For details about how to create a Redis cache instance, see **Creating a DCS Instance** in the [Distributed Cache Service User Guide](#).
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must be interconnected with the DCS clusters. You can also set the security group rules as required.
For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).
- If you use a VPC peering connection to access a DCS instance, the following restrictions also apply:
 - If network segment 172.16.0.0/12~24 is used during DCS instance creation, the DLI queue cannot be in any of the following network segments: 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24.
 - If network segment 192.168.0.0/16~24 is used during DCS instance creation, the DLI queue cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.
 - If network segment 10.0.0.0/8~24 is used during DCS instance creation, the DLI queue cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH (
  type = "dcs_redis",
  region = "",
  cluster_address = "",
  password = "",
  value_type= "",key_value= ""
);
```


Keyword

Table 3-11 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. dcx_redis indicates that data is exported to DCS Redis.
region	Yes	Region where DCS for storing the data is located.
cluster_address	Yes	Redis instance connection address.
password	No	Redis instance connection password. This parameter is not required if password-free access is used.
value_type	Yes	This parameter can be set to any or the combination of the following options: <ul style="list-style-type: none"> Data types, including string, list, hash, set, and zset Commands used to set the expiration time of a key, including expire, pexpire, expireAt, and pexpireAt Commands used to delete a key, including del and hdel Use commas (,) to separate multiple commands.
key_value	Yes	Key and value. The number of key_value pairs must be the same as the number of types specified by value_type, and key_value pairs are separated by semicolons (;). Both key and value can be specified through parameter configurations. The dynamic column name is represented by <code>\${column name}</code> .

Precautions

- If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_\${car_brand}** and the value of **car_brand** in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.
- Characters ":", ";", "\$", "{", and "}" have been used as special separators without the escape function. These characters cannot be used in key and value as common characters. Otherwise, parsing will be affected and the program exceptions will occur.

Example

Data of stream **qualified_cars** is exported to the Redis cache instance on DCS.

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
```

```
average_speed DOUBLE,  
total_miles DOUBLE  
)  
WITH (  
  type = "dcs_redis",  
  cluster_address = "192.168.0.34:6379",  
  password = "xxxxxxx",  
  value_type = "string; list; hash; set; zset",  
  key_value = "${car_id}_str: ${car_owner}; name_list: ${car_owner}; ${car_id}_hash: {name:${car_owner},  
age: ${car_age}}; name_set: ${car_owner}; math_zset: ${car_owner}:${average_speed}}"  
);
```

3.1.4.6 DDS Sink Stream

Function

DLI outputs the job output data to Document Database Service (DDS).

DDS is compatible with the MongoDB protocol and is secure, highly available, reliable, scalable, and easy to use. It provides DB instance creation, scaling, redundancy, backup, restoration, monitoring, and alarm reporting functions with just a few clicks on the DDS console.

Prerequisites

- Ensure that you have created a DDS instance on DDS using your account.
For details about how to create a DDS instance, see **Buying a DDS DB Instance** in the *Document Database Service Getting Started*.
- Currently, only cluster instances with SSL authentication disabled are supported. Replica set and single node instances are not supported.
- In this scenario, jobs must run on the dedicated queue of DLI. Ensure that the dedicated queue of DLI has been created.

To create a dedicated DLI queue, select **Pay-per-use** for **Billing Mode** and click **Dedicated Resource Mode** for **Queue Type** when creating a queue. For details, see [Creating a Queue](#) in the *Data Lake Insight User Guide*.

- Ensure that a datasource connection has been set up between the DLI dedicated queue and the DDS cluster, and security group rules have been configured based on the site requirements.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )  
WITH (  
  type = "dds",  
  username = "",  
  password = "",  
  db_url = "",  
  field_names = ""  
);
```

Keyword

Table 3-12 Keyword description

Parameter	Man dato ry	Description
type	Yes	Output channel type. dds indicates that data is exported to DDS.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.
db_url	Yes	DDS instance access address, for example, ip1:port,ip2:port/database/collection .
field_names	Yes	Key of the data field to be inserted. The format is f1,f2,f3 . Ensure that the key corresponds to the data column in the sink stream.
batch_insert_data_num	No	Amount of data to be written in batches at a time. The value must be a positive integer. The default value is 10 .

Example

Output data in the **qualified_cars** stream to the **collectionTest** DDS DB.

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
WITH (
  type = "dds",
  region = "xxx",
  db_url = "192.168.0.8:8635,192.168.0.130:8635/dbtest/collectionTest",
  username = "xxxxxxxxxx",
  password = "xxxxxxxxxx",
  field_names = "car_id,car_owner,car_age,average_speed,total_miles",
  batch_insert_data_num = "10"
);
```

3.1.4.7 DIS Sink Stream

Function

DLI writes the Flink job output data into DIS. This cloud ecosystem is applicable to scenarios where data is filtered and imported to the DIS stream for future processing.

DIS addresses the challenge of transmitting data outside cloud services to cloud services. DIS builds data intake streams for custom applications capable of

processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, Internet of Things (IoT) data, social media feeds, website clickstreams, and location-tracking events. For more information about DIS, see the *Data Ingestion Service User Guide*.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
  type = "dis",
  region = "",
  channel = "",
  partition_key = "",
  encode= "",
  field_delimiter= ""
);
```

Keyword

Table 3-13 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. dis indicates that data is exported to DIS.
region	Yes	Region where DIS for storing the data is located.
ak	No	Access Key ID (AK).
sk	No	Specifies the secret access key used together with the ID of the access key.
channel	Yes	DIS stream.
partition_key	No	Group primary key. Multiple primary keys are separated by commas (,). If this parameter is not specified, data is randomly written to DIS partitions.
encode	Yes	Data encoding format. The value can be csv , json , or user_defined . NOTE <ul style="list-style-type: none"> field_delimiter must be specified if this parameter is set to csv. If the encoding format is json, you need to configure enable_output_null to determine whether to generate an empty field. For details, see the examples. encode_class_name and encode_class_parameter must be specified if this parameter is set to user_defined.

Parameter	Mandatory	Description
field_delimiter	Yes	Separator used to separate every two attributes. <ul style="list-style-type: none"> This parameter needs to be configured if the CSV encoding format is adopted. It can be user-defined, for example, a comma (,). This parameter is not required if the JSON encoding format is adopted.
json_config	No	If encode is set to json , you can set this parameter to specify the mapping between the JSON field and the stream definition field. An example of the format is as follows: field1=data_json.field1; field2=data_json.field2.
enable_output_null	No	If encode is set to json , you need to specify this parameter to determine whether to generate an empty field. If this parameter is set to true , an empty field (the value is null) is generated. If set to false , no empty field is generated. The default value is true .
encode_class_name	No	If encode is set to user_defined , you need to set this parameter to the name of the user-defined decoding class (including the complete package path). The class must inherit the Deserialization-Schema class.
encode_class_parameter	No	If encode is set to user_defined , you can set this parameter to specify the input parameter of the user-defined decoding class. Only one parameter of the string type is supported.

Precautions

None

Example

- CSV: Data is written to the DIS stream and encoded using CSV. CSV fields are separated by commas (,). If there are multiple partitions, car_owner is used as the key to distribute data to different partitions. An example is as follows:
"ZJA710XC", "lilei", "BMW", 700000.

```
CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "dis",
  region = "xxx",
  channel = "dlioutput",
```

```
    encode = "csv",  
    field_delimiter = ","  
);
```

- JSON: Data is written to the DIS stream and encoded using JSON. If there are multiple partitions, `car_owner` and `car_brand` are used as the keys to distribute data to different partitions. If **enableOutputNull** is set to **true**, an empty field (the value is **null**) is generated. If set to **false**, no empty field is generated. An example is as follows: `"car_id ":"ZJA710XC", "car_owner ":"lilei", "car_brand ":"BMW", "car_price ":700000`.

```
CREATE SINK STREAM audi_cheaper_than_30w (  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_price INT  
)  
WITH (  
  type = "dis",  
  channel = "dlioutput",  
  region = "xxx",  
  partition_key = "car_owner,car_brand",  
  encode = "json",  
  enable_output_null = "false"  
);
```

3.1.4.8 DMS Sink Stream

DMS (Distributed Message Service) is a message middleware service based on distributed, high-availability clustering technology. It provides reliable, scalable, fully managed queues for sending, receiving, and storing messages. DMS for Kafka is a message queuing service based on Apache Kafka. This service provides Kafka premium instances.

DLI can write the job output data into the Kafka instance. The syntax for creating a Kafka sink stream is the same as that for creating an open source Apache Kafka sink stream. For details, see [MRS Kafka Sink Stream](#).

3.1.4.9 DWS Sink Stream (JDBC Mode)

Function

DLI outputs the Flink job output data to Data Warehouse Service (DWS). DWS database kernel is compliant with PostgreSQL. The PostgreSQL database can store data of more complex types and delivers space information services, multi-version concurrent control (MVCC), and high concurrency. It applies to location applications, financial insurance, and e-commerce.

DWS is an online data processing database based on the cloud infrastructure and platform and helps you mine and analyze massive sets of data. For more information about DWS, see the *Data Warehouse Service Management Guide*.

Prerequisites

- Ensure that you have created a DWS cluster on DWS using your account. For details about how to create a DWS cluster, see **Creating a Cluster** in the *Data Warehouse Service Management Guide*.
- Ensure that a DWS database table has been created.

- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with DWS clusters. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
  type = "rds",
  username = "",
  password = "",
  db_url = "",
  table_name = ""
);
```

Keyword

Table 3-14 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. rds indicates that data is exported to RDS or DWS.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.
db_url	Yes	Database connection address, for example, postgresql://ip:port/database .
table_name	Yes	Name of the table where data will be inserted. You need to create the database table in advance.

Parameter	Mandatory	Description
db_columns	No	<p>Mapping between attributes in the output stream and those in the database table. This parameter must be configured based on the sequence of attributes in the output stream.</p> <p>Example:</p> <pre>create sink stream a3(student_name string, student_age int) with (type = "rds", username = "root", password = "xxxxxxx", db_url = "postgresql://192.168.0.102:8000/test1", db_columns = "name,age", table_name = "t1");</pre> <p>In the example, student_name corresponds to the name attribute in the database, and student_age corresponds to the age attribute in the database.</p> <p>NOTE</p> <ul style="list-style-type: none"> If db_columns is not configured, it is normal that the number of attributes in the output stream is less than that of attributes in the database table and the extra attributes in the database table are all nullable or have default values.
primary_key	No	<p>To update data in the table in real time by using the primary key, add the primary_key configuration item (c_timeminute in the following example) when creating a table. During the data writing operation, data is updated if the specified primary_key exists. Otherwise, data is inserted.</p> <p>Example:</p> <pre>CREATE SINK STREAM test(c_timeminute LONG, c_cnt LONG) WITH (type = "rds", username = "root", password = "xxxxxxx", db_url = "postgresql://192.168.0.12:8000/test", table_name = "test", primary_key = "c_timeminute");</pre>

Precautions

The stream format defined by **stream_id** must be the same as the table format.

Example

Data of stream **audi_cheaper_than_30w** is exported to the **audi_cheaper_than_30w** table in the **test** database.

```
CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
```



```
WITH (  
  type = "rds",  
  username = "root",  
  password = "xxxxxx",  
  db_url = "postgresql://192.168.1.1:8000/test",  
  table_name = "audi_cheaper_than_30w"  
);  
  
insert into audi_cheaper_than_30w select "1","2","3",4;
```

3.1.4.10 DWS Sink Stream (OBS-based Dumping)

Function

Create a sink stream to export Flink job data to DWS through OBS-based dumping, specifically, output Flink job data to OBS and then import data from OBS to DWS. For details about how to import OBS data to DWS, see **Concurrently Importing Data from OBS** in the *Data Warehouse Service Development Guide*.

DWS is an online data processing database based on the cloud infrastructure and platform and helps you mine and analyze massive sets of data. For more information about DWS, see the *Data Warehouse Service Management Guide*.

Precautions

- OBS-based dumping supports intermediate files of the following two types:
 - ORC: The ORC format does not support array data type. If the ORC format is used, create a foreign server in DWS. For details, see **Creating a Foreign Server** in the *Data Warehouse Development Guide*.
 - CSV: By default, the line break is used as the record separator. If the line break is contained in the attribute content, you are advised to configure quote. For details, see [Table 3-15](#).
- If the target table does not exist, a table is automatically created. DLI data of the SQL type does not support **text**. If a long text exists, you are advised to create a table in the database.
- When **encode** uses the ORC format to create a DWS table, if the field attribute of the SQL stream is defined as the **String** type, the field attribute of the DWS table cannot use the **varchar** type. Instead, a specific text type must be used. If the SQL stream field attribute is defined as the **Integer** type, the DWS table field must use the **Integer** type.

Prerequisites

- Ensure that OBS buckets and folders have been created.
For details about how to create an OBS bucket, see **Creating a Bucket** in the [Object Storage Service Console Operation Guide](#).
For details about how to create a folder, see **Creating a Folder** in the [Object Storage Service Console Operation Guide](#).
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with DWS clusters. You can also set the security group rules as required.
For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
  type = "dws",
  region = "",
  ak = "",
  sk = "",
  encode = "",
  field_delimiter = "",
  quote = "",
  db_obs_server = "",
  obs_dir = "",
  username = "",
  password = "",
  db_url = "",
  table_name = "",
  max_record_num_per_file = "",
  dump_interval = ""
);
```

Keyword

Table 3-15 Keyword description

Parameter	Man dato ry	Description
type	Yes	Output channel type. dws indicates that data is exported to DWS.
region	Yes	Region where DWS is located.
ak	Yes	Access Key ID (AK). For details about how to obtain the access key, see My Credentials .
sk	Yes	Secret access key used together with the ID of the AK. For details about how to obtain the access key, see My Credentials .
encode	Yes	Encoding format. Currently, CSV and ORC are supported.
field_delimiter	No	Separator used to separate every two attributes. This parameter needs to be configured if the CSV encoding mode is used. It is recommended that you use invisible characters as separators, for example, <code>\u0006\u0002</code> .
quote	No	Single byte. It is recommended that invisible characters be used, for example, <code>\u0007</code> .

Parameter	Man dato ry	Description
db_obs_server	No	Foreign server (for example, obs_server) that has been created in the database. For details about how to create a foreign server, see Creating a Foreign Server in the <i>Data Warehouse Service Database Development Guide</i> . You need to specify this parameter if the ORC encoding mode is adopted.
obs_dir	Yes	Directory for storing intermediate files. The directory is in the format of {Bucket name}/{Directory name}, for example, obs-a1/dir1/subdir.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.
db_url	Yes	Database connection address. The format is /ip:port/database, for example, 192.168.1.21:8000/test1 .
table_name	Yes	Data table name. If no table is available, a table is automatically created.
max_record_n um_per_file	Yes	Maximum number of records that can be stored in a file. If the number of records in a file is less than the maximum value, the file will be dumped to OBS after one dumping period.
dump_interval	Yes	Dumping period. The unit is second.
delete_obs_te mp_file	No	Whether to delete temporary files on OBS. The default value is true . If this parameter is set to false , files on OBS will not be deleted. You need to manually clear the files.
max_dump_fil e_num	No	Maximum number of files that can be dumped at a time. If the number of files to be dumped is less than the maximum value, the files will be dumped to OBS after one dumping period.

Example

- Dump files in CSV format.

```
CREATE SINK STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
WITH (
  type = "dws",
  region = "xxx",
```

```
ak = "",
sk = "",
encode = "csv",
field_delimiter = "\u0006\u0006\u0002",
quote = "\u0007",
obs_dir = "dli-append-2/dws",
username = "",
password = "",
db_url = "192.168.1.12:8000/test1",
table_name = "table1",
max_record_num_per_file = "100",
dump_interval = "10"
);
```

- Dump files in ORC format.

```
CREATE SINK STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
WITH (
  type = "dws",
  region = "xxx",
  ak = "",
  sk = "",
  encode = "orc",
  db_obs_server = "obs_server",
  obs_dir = "dli-append-2/dws",
  username = "",
  password = "",
  db_url = "192.168.1.12:8000/test1",
  table_name = "table1",
  max_record_num_per_file = "100",
  dump_interval = "10"
);
```

3.1.4.11 MRS HBase Sink Stream

Function

DLI exports the output data of the Flink job to HBase of MRS.

Prerequisites

- An MRS cluster has been created by using your account. DLI can interconnect with HBase clusters with Kerberos enabled.
- In this scenario, jobs must run on the dedicated queue of DLI. Ensure that the dedicated queue of DLI has been created.

To create a dedicated DLI queue, select **Pay-per-use** for **Billing Mode** and click **Dedicated Resource Mode** for **Queue Type** when creating a queue. For details, see [Creating a Queue](#) in the *Data Lake Insight User Guide*.

- Ensure that a datasource connection has been set up between the DLI dedicated queue and the MRS cluster, and security group rules have been configured based on the site requirements.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see **Security Group** in the [Virtual Private Cloud User Guide](#).

- If you use MRS HBase, ensure that you have added IP addresses of all hosts in the MRS cluster for the enhanced datasource connection.

For details about how to add an IP-domain mapping, see [Modifying the Host Information](#) in the *Data Lake Insight User Guide*.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )  
WITH (  
  type = "mrs_hbase",  
  region = "",  
  cluster_address = "",  
  table_name = "",  
  table_columns = "",  
  illegal_data_table = "",  
  batch_insert_data_num = "",  
  action = ""  
)
```

Keyword

Table 3-16 Keyword description

Parameter	Mandator y	Description
type	Yes	Output channel type. mrs_hbase indicates that data is exported to HBase of MRS.
region	Yes	Region where MRS resides.
cluster_address	Yes	ZooKeeper address of the cluster to which the data table to be inserted belongs. The format is ip1,ip2:port .
table_name	Yes	Name of the table where data is to be inserted. It can be specified through parameter configurations. For example, if you want one or more certain columns as part of the table name, use car_pass_inspect_with_age_\${car_age} , where car_age is the column name.
table_columns	Yes	Columns to be inserted. The format is rowKey, f1:c1, f1:c2, f2:c1 , where rowKey must be specified. If you do not want to add a column (for example, the third column) to the database, set this parameter to rowKey,f1:c1,,f2:c1 .
illegal_data_table	No	If this parameter is specified, abnormal data (for example, rowKey does not exist) will be written into the table. If not specified, abnormal data will be discarded. The rowKey value is taskNo_Timestamp followed by six random digits, and the schema is info:data, info:reason.

Parameter	Mandatory	Description
batch_insert_data_num	No	Number of data records to be written in batches at a time. The value must be a positive integer. The upper limit is 1000 . The default value is 10 .
action	No	Whether data is added or deleted. Available options include add and delete . The default value is add .
krb_auth	No	Authentication name for creating a datasource connection authentication. This parameter is mandatory when Kerberos authentication is enabled. Set this parameter to the corresponding cross-source authentication name. For details, see Datasource Authentication . NOTE Ensure that the /etc/hosts information of the master node in the MRS cluster is added to the host file of the DLI queue.

Precautions

None

Example

Output data to HBase of MRS.

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
WITH (
  type = "mrs_hbase",
  region = "xxx",
  cluster_address = "192.16.0.88,192.87.3.88:2181",
  table_name = "car_pass_inspect_with_age_${car_age}",
  table_columns = "rowKey,info:owner,car:speed,car:miles",
  illegal_data_table = "illegal_data",
  batch_insert_data_num = "20",
  action = "add",
  krb_auth = "KRB_AUTH_NAME"
);
```

3.1.4.12 MRS Kafka Sink Stream

Function

DLI exports the output data of the Flink job to Kafka.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and

provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages. Kafka clusters are deployed and hosted on MRS that is powered on Apache Kafka.

Prerequisites

- If the Kafka server listens on the port using hostname, you need to add the mapping between the hostname and IP address of the Kafka Broker node to the DLI queue. Contact the Kafka service deployment personnel to obtain the hostname and IP address of the Kafka Broker node. For details about how to add an IP-domain mapping, see [Modifying the Host Information](#) in the *Data Lake Insight User Guide*.
- Kafka is an offline cluster. You need to use the enhanced datasource connection function to connect Flink jobs to Kafka. You can also set security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH(
  type = "kafka",
  kafka_bootstrap_servers = "",
  kafka_topic = "",
  encode = "json"
)
```

Keyword

Table 3-17 Keyword description

Parameter	Man dat ory	Description
type	Yes	Output channel type. kafka indicates that data is exported to Kafka.
kafka_bootstrap_servers	Yes	Port that connects DLI to Kafka. Use enhanced datasource connections to connect DLI queues with Kafka clusters.
kafka_topic	Yes	Kafka topic into which DLI writes data.
encode	Yes	Encoding format. Currently, json and user_defined are supported. encode_class_name and encode_class_parameter must be specified if this parameter is set to user_defined .

Parameter	Mandatory	Description
encode_class_name	No	If encode is set to user_defined , you need to set this parameter to the name of the user-defined decoding class (including the complete package path). The class must inherit the DeserializationSchema class.
encode_class_parameter	No	If encode is set to user_defined , you can set this parameter to specify the input parameter of the user-defined decoding class. Only one parameter of the string type is supported.
krb_auth	No	Authentication name for creating a datasource connection authentication. This parameter is mandatory when Kerberos authentication is enabled. If Kerberos authentication is not enabled for the created MRS cluster, ensure that the /etc/hosts information of the master node in the MRS cluster is added to the host file of the DLI queue.
kafka_properties	No	This parameter is used to configure the native attributes of Kafka. The format is key1=value1;key2=value2 .
kafka_certificate_name	No	Specifies the name of the datasource authentication information. This parameter is valid only when the datasource authentication type is set to Kafka_SSL . NOTE <ul style="list-style-type: none"> If this parameter is specified, the service loads only the specified file and password under the authentication. The system automatically sets this parameter to kafka_properties. Other configuration information required for Kafka SSL authentication needs to be manually configured in the kafka_properties attribute.

Precautions

None

Example

Output data to Kafka.

- Example 1:

```
CREATE SINK STREAM kafka_sink (name STRING)
WITH (
  type="kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_topic = "testsink",
  encode = "json"
);
```
- Example 2:


```
CREATE SINK STREAM kafka_sink (  
  a1 string,  
  a2 string,  
  a3 string,  
  a4 INT  
) // Output Field  
WITH (  
  type="kafka",  
  kafka_bootstrap_servers = "192.x.x.x:9093, 192.x.x.x:9093, 192.x.x.x:9093",  
  kafka_topic = "testflink", // Written topic  
  encode = "csv", // Encoding format, which can be JSON or CSV.  
  kafka_certificate_name = "Flink",  
  kafka_properties_delimiter = ",",  
  kafka_properties = "sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule  
required username=\"xxx\" password=\"xxx\",sasl.mechanism=PLAIN,security.protocol=SASL_SSL"  
);
```

3.1.4.13 Open-Source Kafka Sink Stream

Function

DLI exports the output data of the Flink job to Kafka.

Apache Kafka is a fast, scalable, and fault-tolerant distributed message publishing and subscription system. It delivers high throughput and built-in partitions and provides data replicas and fault tolerance. Apache Kafka is applicable to scenarios of handling massive messages.

Prerequisites

- If the Kafka server listens on the port using hostname, you need to add the mapping between the hostname and IP address of the Kafka Broker node to the DLI queue. Contact the Kafka service deployment personnel to obtain the hostname and IP address of the Kafka Broker node. For details about how to add an IP-domain mapping, see [Modifying the Host Information](#) in the *Data Lake Insight User Guide*.
- Kafka is an offline cluster. You need to use the enhanced datasource connection function to connect Flink jobs to Kafka. You can also set security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)*  
WITH(  
  type = "kafka",  
  kafka_bootstrap_servers = "",  
  kafka_topic = "",  
  encode = "json"  
)
```

Keyword

Table 3-18 Keyword description

Parameter	Man dato ry	Description
type	Yes	Output channel type. kafka indicates that data is exported to Kafka.
kafka_bootstrap_servers	Yes	Port that connects DLI to Kafka. Use enhanced datasource connections to connect DLI queues with Kafka clusters.
kafka_topic	Yes	Kafka topic into which DLI writes data.
encode	Yes	Data encoding format. The value can be csv , json , or user_defined . <ul style="list-style-type: none"> • field_delimiter must be specified if this parameter is set to csv. • encode_class_name and encode_class_parameter must be specified if this parameter is set to user_defined.
filed_delimiter	No	If encode is set to csv , you can use this parameter to specify the separator between CSV fields. By default, the comma (,) is used.
encode_class_name	No	If encode is set to user_defined , you need to set this parameter to the name of the user-defined decoding class (including the complete package path). The class must inherit the DeserializationSchema class.
encode_class_parameter	No	If encode is set to user_defined , you can set this parameter to specify the input parameter of the user-defined decoding class. Only one parameter of the string type is supported.
kafka_properties	No	This parameter is used to configure the native attributes of Kafka. The format is key1=value1;key2=value2 .
kafka_certificate_name	No	Name of the datasource authentication information. This parameter is valid only when the datasource authentication type is set to Kafka_SSL . <p>NOTE</p> <ul style="list-style-type: none"> • If this parameter is specified, the service loads only the specified file and password under the authentication. The system automatically sets this parameter to kafka_properties. • Other configuration information required for Kafka SSL authentication needs to be manually configured in the kafka_properties attribute.

Precautions

None

Example

Output the data in the kafka_sink stream to Kafka.

```
CREATE SINK STREAM kafka_sink (name STRING)
WITH (
  type="kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_topic = "testsink",
  encode = "json"
);
```

3.1.4.14 File System Sink Stream (Recommended)

Function

You can create a sink stream to export data to a file system such as HDFS or OBS. After the data is generated, a non-DLI table can be created directly according to the generated directory. The table can be processed through DLI SQL, and the output data directory can be stored in partitioned tables. It is applicable to scenarios such as data dumping, big data analysis, data backup, and active, deep, or cold archiving.

OBS is an object-based storage service. It provides massive, secure, highly reliable, and low-cost data storage capabilities.

For more information about OBS, see the [Object Storage Service Console Operation Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
[PARTITIONED BY (attr_name (',' attr_name)*]
WITH (
  type = "filesystem",
  file.path = "obs://bucket/xx",
  encode = "parquet",
  ak = "",
  sk = ""
);
```

Keyword

Table 3-19 Keyword description

Parameter	Mandatory	Description
type	Yes	Output stream type. If type is set to filesystem , data is exported to the file system.

Parameter	Mandatory	Description
file.path	Yes	Output directory in the form: schema://file.path . Currently, Schema supports only OBS and HDFS. <ul style="list-style-type: none"> If schema is set to obs, data is stored to OBS. If schema is set to hdfs, data is exported to HDFS. A proxy user needs to be configured for HDFS. For details, see HDFS Proxy User Configuration. Example: hdfs://node-master1sYAx:9820/user/car_infos, where node-master1sYAx:9820 is the name of the node where the NameNode is located.
encode	Yes	Output data encoding format. Currently, only the parquet and csv formats are supported. <ul style="list-style-type: none"> When schema is set to obs, the encoding format of the output data can only be parquet. When schema is set to hdfs, the output data can be encoded in Parquet or CSV format.
ak	No	Access key. This parameter is mandatory when data is exported to OBS. Global variables can be used to mask the access key used for OBS authentication. For details about how to use global variables on the console, see the Data Lake Insight User Guide .
sk	No	Secret access key. This parameter is mandatory when data is exported to OBS. Secret key for accessing OBS authentication. Global variables can be used to mask sensitive information. For details about how to use global variables on the console, see the Data Lake Insight User Guide .
krb_auth	No	Authentication name for creating a datasource connection authentication. This parameter is mandatory when Kerberos authentication is enabled. If Kerberos authentication is not enabled for the created MRS cluster, ensure that the /etc/hosts information of the master node in the MRS cluster is added to the host file of the DLI queue.
field_delimiter	No	Separator used to separate every two attributes. This parameter needs to be configured if the CSV encoding format is adopted. It can be user-defined, for example, a comma (,).

Precautions

- To ensure job consistency, enable checkpointing if the Flink job uses the file system output stream.
- To avoid data loss or data coverage, you need to enable automatic or manual restart upon job exceptions. Enable the **Restore Job from Checkpoint**.

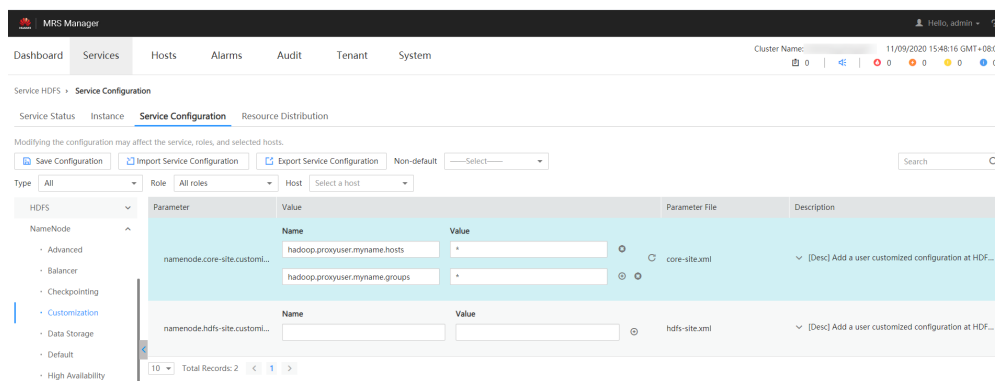
- Set the checkpoint interval after weighing between real-time output file, file size, and recovery time, such as 10 minutes.
- Two modes are supported.
 - **At least once**: Events are processed at least once.
 - **Exactly once**: Events are processed only once.
- When you use sink streams of a file system to write data into OBS, do not use multiple jobs for the same directory.
 - The default behavior of an OBS bucket is overwriting, which may cause data loss.
 - The default behavior of the OBS parallel file system bucket is appending, which may cause data confusion.

You should carefully select the OBS bucket because of the preceding behavior differences. Data exceptions may occur after abnormal job restart.

HDFS Proxy User Configuration

1. Log in to the MRS management page.
2. Select the HDFS NameNode configuration of MRS and add configuration parameters in the **Customization** area.

Figure 3-1 HDFS service configuration



In the preceding information, **myname** in the **core-site** values **hadoop.proxyuser.myname.hosts** and **hadoop.proxyuser.myname.groups** is the name of the krb authentication user.

NOTE

Ensure that the permission on the HDFS data write path is **777**.

3. After the configuration is complete, click **Save**.

Example

- Example 1:
The following example dumps the **car_info** data to OBS, with the **buyday** field as the partition field and **parquet** as the encoding format.

```
create sink stream car_infos (  
  carId string,  
  carOwner string,  
  average_speed double,
```

```
buyday string
) partitioned by (buyday)
with (
  type = "filesystem",
  file.path = "obs://obs-sink/car_infos",
  encode = "parquet",
  ak = "{{myAk}}",
  sk = "{{mySk}}")
);
```

The data is ultimately stored in OBS. Directory: **obs://obs-sink/car_infos/buyday=xx/part-x-x**.

After the data is generated, the OBS partitioned table can be established for subsequent batch processing through the following SQL statements:

- a. Create an OBS partitioned table.

```
create table car_infos (
  carId string,
  carOwner string,
  average_speed double
)
partitioned by (buyday string)
stored as parquet
location 'obs://obs-sink/car_infos';
```

- b. Restore partition information from the associated OBS path.
alter table car_infos recover partitions;

- Example 2:

The following example dumps the **car_info** data to HDFS, with the **buyday** field as the partition field and **csv** as the encoding format.

```
create sink stream car_infos (
  carId string,
  carOwner string,
  average_speed double,
  buyday string
) partitioned by (buyday)
with (
  type = "filesystem",
  file.path = "hdfs://node-master1sYAx:9820/user/car_infos",
  encode = "csv",
  field_delimiter = ",")
);
```

The data is ultimately stored in HDFS. Directory: **/user/car_infos/buyday=xx/part-x-x**.

3.1.4.15 OBS Sink Stream

Function

Create a sink stream to export DLI data to OBS. DLI can export the job analysis results to OBS. OBS applies to various scenarios, such as big data analysis, cloud-native application program data, static website hosting, backup/active archive, and deep/cold archive.

OBS is an object-based storage service. It provides massive, secure, highly reliable, and low-cost data storage capabilities. For more information about OBS, see the [Object Storage Service Console Operation Guide](#)

NOTE

You are advised to use the [File System Sink Stream \(Recommended\)](#).

Prerequisites

Before data exporting, check the version of the OBS bucket. The OBS sink stream supports data exporting to an OBS bucket running OBS 3.0 or a later version.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH (
  type = "obs",
  region = "",
  encode = "",
  field_delimiter = "",
  row_delimiter = "",
  obs_dir = "",
  file_prefix = "",
  rolling_size = "",
  rolling_interval = "",
  quote = "",
  array_bracket = "",
  append = "",
  max_record_num_per_file = "",
  dump_interval = "",
  dis_notice_channel = ""
)
```

Keyword

Table 3-20 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. obs indicates that data is exported to OBS.
region	Yes	Region to which OBS belongs.
ak	No	Access Key ID (AK). For details about how to obtain the access key, see My Credentials .
sk	No	Secret access key used together with the ID of the access key. For details about how to obtain the access key, see My Credentials .
encode	Yes	Encoding format. Currently, formats CSV, JSON, ORC, Avro, Avro-Merge, and Parquet are supported.
field_delimiter	No	Separator used to separate every two attributes. This parameter is mandatory only when the CSV encoding format is adopted. If this parameter is not specified, the default separator comma (,) is used.
row_delimiter	No	Row delimiter. This parameter does not need to be configured if the CSV or JSON encoding format is adopted.

Parameter	Mandatory	Description
json_config	No	If encode is set to json , you can set this parameter to specify the mapping between the JSON field and the stream definition field. An example of the format is as follows: field1=data_json.field1;field2=data_json.field2.
obs_dir	Yes	Directory for storing files. The directory is in the format of {Bucket name}/{Directory name}, for example, obs-a1/dir1/subdir. If encode is set to csv (append is false) , json (append is false) , avro_merge , or parquet , parameterization is supported.
file_prefix	No	Prefix of the data export file name. The generated file is named in the format of file_prefix.x, for example, file_prefix.1 and file_prefix.2. If this parameter is not specified, the file prefix is temp by default.
rolling_size	No	Maximum size of a file. NOTE <ul style="list-style-type: none"> One or both of rolling_size and rolling_interval must be configured. When the size of a file exceeds the specified size, a new file is generated. The unit can be KB, MB, or GB. If no unit is specified, the byte unit is used. This parameter does not need to be configured if the ORC encoding format is adopted.
rolling_interval	No	Time mode, in which data is saved to the corresponding directory. NOTE <ul style="list-style-type: none"> One or both of rolling_size and rolling_interval must be configured. After this parameter is specified, data is written to the corresponding directories according to the output time. The parameter value can be in the format of yyyy/MM/dd/HH/mm, which is case sensitive. The minimum unit is minute. If this parameter is set to yyyy/MM/dd/HH, data is written to the directory that is generated at the hour time. For example, data generated at 2018-09-10 16:00 will be written to the {obs_dir}/2018-09-10_16 directory. If both rolling_size and rolling_interval are set, a new file is generated when the size of a single file exceeds the specified size in the directory corresponding to each time point.
quote	No	Modifier, which is added before and after each attribute only when the CSV encoding format is adopted. You are advised to use invisible characters, such as u0007 , as the parameter value.

Parameter	Mandatory	Description
array_bracket	No	Array bracket, which can be configured only when the CSV encoding format is adopted. The available options are <code>()</code> , <code>{}</code> , and <code>[]</code> . For example, if you set this parameter to <code>{}</code> , the array output format is <code>{a1, a2}</code> .
append	No	The value can be true or false . The default value is true . If OBS does not support the append mode and the encoding format is CSV or JSON, set this parameter to false . If Append is set to false , max_record_num_per_file and dump_interval must be set.
max_record_num_per_file	No	Maximum number of records in a file. This parameter needs to be set if encode is csv (append is false) , json (append is false) , orc , avro , avro_merge , or parquet . If the maximum number of records has been reached, a new file is generated.
dump_interval	No	Triggering period. This parameter needs to be configured when the ORC encoding format is adopted or notification to DIS is enabled. <ul style="list-style-type: none"> If the ORC encoding format is specified, this parameter indicates that files will be uploaded to OBS when the triggering period arrives even if the number of file records does not reach the maximum value. In notification to DIS is enabled, this parameter specifies that a notification is sent to DIS every period to indicate that no more files will be generated in the directory.
dis_notice_channel	No	DIS channel where DLI sends the record that contains the OBS directory. DLI periodically sends the DIS channel a record, which contains the OBS directory, indicating that no more new files will be generated in the directory.
encoded_data	No	Data to be encoded. This parameter is set if encode is json (append is false) , avro_merge , or parquet . The format is `\${field_name}` , indicating that the stream field content is encoded as a complete record.

Precautions

If a configuration item can be specified through parameter configurations, one or more columns in the record can be used as part of the configuration item. For example, if the configuration item is set to **car_\${car_brand}** and the value of

car_brand in a record is **BMW**, the value of this configuration item is **car_BMW** in the record.

Example

- Export the **car_infos** data to the **obs-sink** bucket in OBS. The output directory is **car_infos**. The output file uses **greater_30** as the file name prefix. The maximum size of a single file is 100 MB. If the data size exceeds 100 MB, another new file is generated. The data is encoded in CSV format, the comma (,) is used as the attribute delimiter, and the line break is used as the line separator.

```
CREATE SINK STREAM car_infos (  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_price INT,  
  car_timestamp LONG  
)  
WITH (  
  type = "obs",  
  encode = "csv",  
  region = "xxx",  
  field_delimiter = ",",  
  row_delimiter = "\n",  
  obs_dir = "obs-sink/car_infos",  
  file_prefix = "greater_30",  
  rolling_size = "100m"  
);
```

- Example of the ORC encoding format

```
CREATE SINK STREAM car_infos (  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_price INT,  
  car_timestamp LONG  
)  
WITH (  
  type = "obs",  
  region = "xxx",  
  encode = "orc",  
  obs_dir = "dli-append-2/obsorc",  
  FILE_PREFIX = "es_info",  
  max_record_num_per_file = "100000",  
  dump_interval = "60"  
);
```

- For details about the parquet encoding example, see the example in [File System Sink Stream \(Recommended\)](#).

3.1.4.16 RDS Sink Stream

Function

DLI outputs the Flink job output data to RDS. Currently, PostgreSQL and MySQL databases are supported. The PostgreSQL database can store data of more complex types and delivers space information services, multi-version concurrent control (MVCC), and high concurrency. It applies to location applications, financial insurance, and e-commerce. The MySQL database reduces IT deployment and maintenance costs in various scenarios, such as web applications, e-commerce, enterprise applications, and mobile applications.

RDS is a cloud-based web service.

For more information about RDS, see the [Relational Database Service User Guide](#).

Prerequisites

- Ensure that you have created a PostgreSQL or MySQL RDS instance in RDS. For details about how to create an RDS instance, see [Buying an Instance](#) in the [Relational Database Service Getting Started](#).
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with RDS instance. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (';' attr_name attr_type)* )
WITH (
  type = "rds",
  username = "",
  password = "",
  db_url = "",
  table_name = ""
);
```

Keyword

Table 3-21 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. rds indicates that data is exported to RDS.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.
db_url	Yes	Database connection address, for example, {database_type}://ip:port/database . Currently, two types of database connections are supported: MySQL and PostgreSQL. <ul style="list-style-type: none"> • MySQL: 'mysql://ip:port/database' • PostgreSQL: 'postgresql://ip:port/database'

Parameter	Mandatory	Description
table_name	Yes	Name of the table where data will be inserted.
db_columns	No	<p>Mapping between attributes in the output stream and those in the database table. This parameter must be configured based on the sequence of attributes in the output stream.</p> <p>Example:</p> <pre>create sink stream a3(student_name string, student_age int) with (type = "rds", username = "root", password = "xxxxxxx", db_url = "mysql://192.168.0.102:8635/test1", db_columns = "name,age", table_name = "t1");</pre> <p>In the example, student_name corresponds to the name attribute in the database, and student_age corresponds to the age attribute in the database.</p> <p>NOTE</p> <ul style="list-style-type: none"> If db_columns is not configured, it is normal that the number of attributes in the output stream is less than that of attributes in the database table and the extra attributes in the database table are all nullable or have default values.
primary_key	No	<p>To update data in the table in real time by using the primary key, add the primary_key configuration item (c_timeminute in the following example) when creating a table. During the data writing operation, data is updated if the specified primary_key exists. Otherwise, data is inserted.</p> <p>Example:</p> <pre>CREATE SINK STREAM test(c_timeminute LONG, c_cnt LONG) WITH (type = "rds", username = "root", password = "xxxxxxx", db_url = "mysql://192.168.0.12:8635/test", table_name = "test", primary_key = "c_timeminute");</pre>
operation_field	No	<p>Processing method of specified data in the format of <code>{field_name}</code>. The value of field_name must be a string. If field_name indicates D or DELETE, this record is deleted from the database and data is inserted by default.</p>

Precautions

The stream format defined by **stream_id** must be the same as the table format.

Example

Data of stream **audi_cheaper_than_30w** is exported to the **audi_cheaper_than_30w** table in the **test** database.

```
CREATE SINK STREAM audi_cheaper_than_30w (  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_price INT  
)  
WITH (  
  type = "rds",  
  username = "root",  
  password = "xxxxxx",  
  db_url = "mysql://192.168.1.1:8635/test",  
  table_name = "audi_cheaper_than_30w"  
);
```

3.1.4.17 SMN Sink Stream

Function

DLI exports Flink job output data to SMN.

SMN provides reliable and flexible large-scale message notification services to DLI. It significantly simplifies system coupling and pushes messages to subscription endpoints based on requirements. SMN can be connected to other cloud services or integrated with any application that uses or generates message notifications to push messages over multiple protocols.

For more information about SMN, see the [Simple Message Notification User Guide](#).

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )  
WITH(  
  type = "smn",  
  region = "",  
  topic_urn = "",  
  urn_column = "",  
  message_subject = "",  
  message_column = ""  
)
```

Keyword

Table 3-22 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. smn indicates that data is exported to SMN.
region	Yes	Region to which SMN belongs.

Parameter	Mandatory	Description
topic_urn	No	URN of an SMN topic, which is used for the static topic URN configuration. The SMN topic serves as the destination for short message notification and needs to be created in SMN. One of topic_urn and urn_column must be configured. If both of them are configured, the topic_urn setting takes precedence.
urn_column	No	Field name of the topic URN content, which is used for the dynamic topic URN configuration. One of topic_urn and urn_column must be configured. If both of them are configured, the topic_urn setting takes precedence.
message_subject	Yes	Message subject sent to SMN. This parameter can be user-defined.
message_column	Yes	Field name in the sink stream. Contents of the field name serve as the message contents, which are user-defined. Currently, only text messages (default) are supported.

Precautions

None

Example

Data of stream **over_speed_warning** is exported to SMN.

```
//Static topic configuration
CREATE SINK STREAM over_speed_warning (
  over_speed_message STRING /* over speed message */
)
WITH (
  type = "smn",
  region = "xxx",
  topic_urn = "xxx",
  message_subject = "message title",
  message_column = "over_speed_message"
);
//Dynamic topic configuration
CREATE SINK STREAM over_speed_warning2 (
  over_speed_message STRING, /* over speed message */
  over_speed_urn STRING
)
WITH (
  type = "smn",
  region = "xxx",
  urn_column = "over_speed_urn",
  message_subject = "message title",
  message_column = "over_speed_message"
);
```

3.1.5 Creating a Temporary Stream

Function

The temporary stream is used to simplify SQL logic. If complex SQL logic is followed, write SQL statements concatenated with temporary streams. The temporary stream is just a logical concept and does not generate any data.

Syntax

```
CREATE TEMP STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
```

Example

```
create temp stream a2(attr1 int, attr2 string);
```

3.1.6 Creating a Dimension Table

3.1.6.1 Creating a Redis Table

Create a Redis table to connect to the source stream.

For more information about DCS, see the [Distributed Cache Service User Guide](#).

For details about the JOIN syntax, see [JOIN Between Stream Data and Table Data](#).

Syntax

```
CREATE TABLE table_id (key_attr_name STRING(, hash_key_attr_name STRING)?, value_attr_name STRING)  
WITH (  
  type = "dcs_redis",  
  cluster_address = ""(,password = "")?,  
  value_type= "",  
  key_column= ""(,hash_key_column=""?)?);
```

Keyword

Table 3-23 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value dcs_redis indicates that data is exported to DCS Redis.
cluster_address	Yes	Redis instance connection address.
password	No	Redis instance connection password. This parameter is not required if password-free access is used.
value_type	Yes	Indicates the field data type. Supported data types include string, list, hash, set, and zset.

Parameter	Mandatory	Description
key_column	Yes	Indicates the column name of the Redis key attribute.
hash_key_column	No	If value_type is set to hash , this field must be specified as the column name of the level-2 key attribute.
cache_max_num	No	Indicates the maximum number of cached query results. The default value is 32768 .
cache_time	No	Indicates the maximum duration for caching database query results in the memory. The unit is millisecond. The default value is 10000 . The value 0 indicates that caching is disabled.

Precautions

- Redis clusters are not supported.
- Ensure that You have created a Redis cache instance on DCS using your account.
For details about how to create a Redis cache instance, see the [Distributed Cache Service User Guide](#).
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with DCS instance. You can also set the security group rules as required.
For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.
For details about how to configure security group rules, see [Security Group](#) in the [Virtual Private Cloud User Guide](#).

Example

The Redis table is used to connect to the source stream.

```
CREATE TABLE table_a (attr1 string, attr2 string, attr3 string)
WITH (
  type = "dcs_redis",
  value_type = "hash",
  key_column = "attr1",
  hash_key_column = "attr2",
  cluster_address = "192.168.1.238:6379",
  password = "xxxxxxx"
);
```

3.1.6.2 Creating an RDS Table

Create an RDS/DWS table to connect to the source stream.

For details about the JOIN syntax, see [JOIN](#).

Prerequisites

- Ensure that you have created a PostgreSQL or MySQL RDS instance in RDS.
- In this scenario, jobs must run on the dedicated queue of DLI. Therefore, DLI must interconnect with the enhanced datasource connection that has been connected with RDS instance. You can also set the security group rules as required.

For details about how to create an enhanced datasource connection, see [Enhanced Datasource Connections](#) in the *Data Lake Insight User Guide*.

For details about how to configure security group rules, see [Security Group](#) in the *Virtual Private Cloud User Guide*.

Syntax

```
CREATE TABLE table_id (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "rds",
  username = "",
  password = "",
  db_url = "",
  table_name = ""
);
```

Keyword

Table 3-24 Keyword description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value rds indicates that data is stored to RDS.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.
db_url	Yes	Database connection address, for example, {database_type}://ip:port/database . Currently, two types of database connections are supported: MySQL and PostgreSQL. <ul style="list-style-type: none"> • MySQL: 'mysql://ip:port/database' • PostgreSQL: 'postgresql://ip:port/database' NOTE To create a DWS dimension table, set the database connection address to a DWS database address. If the DWS database version is later than 8.1.0, the open-source PostgreSQL driver cannot be used for connection. You need to use the GaussDB driver for connection.

Parameter	Mandatory	Description
table_name	Yes	Indicates the name of the database table for data query.
db_columns	No	Indicates the mapping of stream attribute fields between the sink stream and database table. This parameter is mandatory when the stream attribute fields in the sink stream do not match those in the database table. The parameter value is in the format of dbtable_attr1,dbtable_attr2,dbtable_attr3.
cache_max_num	No	Indicates the maximum number of cached query results. The default value is 32768 .
cache_time	No	Indicates the maximum duration for caching database query results in the memory. The unit is millisecond. The default value is 10000 . The value 0 indicates that caching is disabled.

Example

The RDS table is used to connect to the source stream.

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "dis",
  region = "",
  channel = "dliinput",
  encode = "csv",
  field_delimiter = ","
);

CREATE TABLE db_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "rds",
  username = "root",
  password = "*****",
  db_url = "postgresql://192.168.0.0:2000/test1",
  table_name = "car"
);

CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "dis",
  region = "",
```

```
channel = "dlioutput",
partition_key = "car_owner",
encode = "csv",
field_delimiter = ","
);

INSERT INTO audi_cheaper_than_30w
SELECT a.car_id, b.car_owner, b.car_brand, b.car_price
FROM car_infos as a join db_info as b on a.car_id = b.car_id;
```

NOTE

To create a DWS dimension table, set the database connection address to a DWS database address. If the DWS database version is later than 8.1.0, the open-source PostgreSQL driver cannot be used for connection. You need to use the GaussDB driver for connection.

3.1.7 Custom Stream Ecosystem

3.1.7.1 Custom Source Stream

Compile code to obtain data from the desired cloud ecosystem or open-source ecosystem as the input data of Flink jobs.

Syntax

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH (
  type = "user_defined",
  type_class_name = "",
  type_class_parameter = ""
)
(TIMESTAMP BY timeindicator (' timeindicator?);timeindicator:PROCTIME '! PROCTIME| ID '! ROWTIME
```

Keyword

Table 3-25 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. The value user_defined indicates that the data source is a user-defined data source.
type_class_name	Yes	Name of the source class for obtaining source data. The value must contain the complete package path.
type_class_parameter	Yes	Input parameter of the user-defined source class. Only one parameter of the string type is supported.

Precautions

The user-defined source class needs to inherit the **RichParallelSourceFunction** class and specify the data type as Row. For example, define MySource class: **public class MySource extends RichParallelSourceFunction<Row>{}**. It aims to implement the **open**, **run**, and **close** functions.

Dependency pom:

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_2.11</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-core</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
```

Example

A data record is generated in each period. The data record contains only one field of the INT type. The initial value is 1 and the period is 60 seconds. The period is specified by an input parameter.

```
CREATE SOURCE STREAM user_in_data (
  count INT
)
WITH (
  type = "user_defined",
  type_class_name = "mySourceSink.MySource",
  type_class_parameter = "60"
)
TIMESTAMP BY car_timestamp.rowtime;
```

NOTE

To customize the implementation of the source class, you need to pack the class in a JAR package and upload the UDF function on the SQL editing page.

3.1.7.2 Custom Sink Stream

Compile code to write the data processed by DLI to a specified cloud ecosystem or open-source ecosystem.

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (' attr_name attr_type)* )
WITH (
  type = "user_defined",
  type_class_name = "",
  type_class_parameter = ""
);
```

Keyword

Table 3-26 Keyword description

Parameter	Mandatory	Description
type	Yes	Data source type. The value user_defined indicates that the data source is a user-defined data source.

Parameter	Mandatory	Description
type_class_name	Yes	Name of the sink class for obtaining source data. The value must contain the complete package path.
type_class_parameter	Yes	Input parameter of the user-defined sink class. Only one parameter of the string type is supported.

Precautions

The user-defined sink class needs to inherit the **RichSinkFunction** class and specify the data type as Row. For example, define MySink class: **public class MySink extends RichSinkFunction<Row>{}**. It aims to implement the **open**, **invoke**, and **close** functions.

Dependency pom:

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_2.11</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-core</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
```

Example

Writing data encoded in CSV format to a DIS stream is used as an example.

```
CREATE SINK STREAM user_out_data (
  count INT
)
WITH (
  type = "user_defined",
  type_class_name = "mySourceSink.MySink",
  type_class_parameter = ""
);
```

NOTE

To customize the implementation of the sink class, you need to pack the class in a JAR package and upload the UDF function on the SQL editing page.

3.1.8 Data Manipulation Language (DML)

3.1.8.1 SELECT

SELECT

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ]
```

Description

The SELECT statement is used to select data from a table or insert constant data into a table.

Precautions

- The table to be queried must exist. Otherwise, an error is reported.
- WHERE is used to specify the filtering condition, which can be the arithmetic operator, relational operator, or logical operator.
- GROUP BY is used to specify the grouping field, which can be one or more multiple fields.

Example

Select the order which contains more than 3 pieces of data.

```
insert into temp SELECT * FROM Orders WHERE units > 3;
```

Insert a group of constant data.

```
insert into temp select 'Lily', 'male', 'student', 17;
```

WHERE Filtering Clause

Syntax

```
SELECT { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]
```

Description

This statement is used to filter the query results using the WHERE clause.

Precautions

- The to-be-queried table must exist.
- WHERE filters the records that do not meet the requirements.

Example

Filter orders which contain more than 3 pieces and fewer than 10 pieces of data.

```
insert into temp SELECT * FROM Orders  
WHERE units > 3 and units < 10;
```

HAVING Filtering Clause

Function

This statement is used to filter the query results using the HAVING clause.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression
```

```
[ WHERE booleanExpression ]  
[ GROUP BY { groupltem [, groupltem ]* } ]  
[ HAVING booleanExpression ]
```

Description

Generally, HAVING and GROUP BY are used together. GROUP BY applies first for grouping and HAVING then applies for filtering. The arithmetic operation and aggregate function are supported by the HAVING clause.

Precautions

If the filtering condition is subject to the query results of GROUP BY, the HAVING clause, rather than the WHERE clause, must be used for filtering.

Example

Group the **student** table according to the **name** field and filter the records in which the maximum score is higher than 95 based on groups.

```
insert into temp SELECT name, max(score) FROM student  
GROUP BY name  
HAVING max(score) >95
```

Column-Based GROUP BY

Function

This statement is used to group a table based on columns.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupltem [, groupltem ]* } ]
```

Description

Column-based GROUP BY can be categorized into single-column GROUP BY and multi-column GROUP BY.

- Single-column GROUP BY indicates that the GROUP BY clause contains only one column.
- Multi-column GROUP BY indicates that the GROUP BY clause contains multiple columns. The table will be grouped according to all fields in the GROUP BY clause. The records whose fields are the same are grouped into one group.

Precautions

None

Example

Group the **student** table according to the score and name fields and return the grouping results.

```
insert into temp SELECT name,score, max(score) FROM student  
GROUP BY name,score;
```

Expression-Based GROUP BY

Function

This statement is used to group a table according to expressions.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]
```

Description

groupItem can have one or more fields. The fields can be called by string functions, but cannot be called by aggregate functions.

Precautions

None

Example

Use the substring function to obtain the character string from the name field, group the **student** table according to the obtained character string, and return each sub character string and the number of records.

```
insert into temp SELECT substring(name,6),count(name) FROM student  
GROUP BY substring(name,6);
```

GROUP BY Using HAVING

Function

This statement filters a table after grouping it using the HAVING clause.

Syntax

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ]
```

Description

Generally, HAVING and GROUP BY are used together. GROUP BY applies first for grouping and HAVING then applies for filtering.

Precautions

- If the filtering condition is subject to the query results of GROUP BY, the HAVING clause, rather than the WHERE clause, must be used for filtering. HAVING and GROUP BY are used together. GROUP BY applies first for grouping and HAVING then applies for filtering.
- Fields used in HAVING, except for those used for aggregate functions, must exist in GROUP BY.
- The arithmetic operation and aggregate function are supported by the HAVING clause.

Example

Group the **transactions** according to **num**, use the HAVING clause to filter the records in which the maximum value derived from multiplying **price** with **amount** is higher than 5000, and return the filtered results.

```
insert into temp SELECT num, max(price*amount) FROM transactions
WHERE time > '2016-06-01'
GROUP BY num
HAVING max(price*amount)>5000;
```

UNION

Syntax

```
query UNION [ ALL ] query
```

Description

This statement is used to return the union set of multiple query results.

Precautions

- Set operation is to join tables from head to tail under certain conditions. The quantity of columns returned by each SELECT statement must be the same. Column types must be the same. Column names can be different.
- By default, the repeated records returned by UNION are removed. The repeated records returned by UNION ALL are not removed.

Example

Output the union set of Orders1 and Orders2 without duplicate records.

```
insert into temp SELECT * FROM Orders1
UNION SELECT * FROM Orders2;
```

3.1.8.2 Condition Expression

CASE Expression

Syntax

```
CASE value WHEN value1 [, value11 ]* THEN result1
[ WHEN valueN [, valueN1 ]* THEN resultN ]* [ ELSE resultZ ]
END
```

or

```
CASE WHEN condition1 THEN result1
[ WHEN conditionN THEN resultN ]* [ ELSE resultZ ]
END
```

Description

- If the value of **value** is **value1**, **result1** is returned. If the value is not any of the values listed in the clause, **resultZ** is returned. If no else statement is specified, **null** is returned.
- If the value of **condition1** is **true**, **result1** is returned. If the value does not match any condition listed in the clause, **resultZ** is returned. If no else statement is specified, **null** is returned.

Precautions

- All results must be of the same type.
- All conditions must be of the Boolean type.
- If the value does not match any condition, the value of **ELSE** is returned when the else statement is specified, and **null** is returned when no else statement is specified.

Example

If the value of **units** equals **5**, **1** is returned. Otherwise, **0** is returned.

Example 1:

```
insert into temp SELECT CASE units WHEN 5 THEN 1 ELSE 0 END FROM Orders;
```

Example 2:

```
insert into temp SELECT CASE WHEN units = 5 THEN 1 ELSE 0 END FROM Orders;
```

NULLIF Expression

Syntax

```
NULLIF(value, value)
```

Description

If the values are the same, **NULL** is returned. For example, **NULL** is returned from **NULLIF (5,5)** and **5** is returned from **NULLIF (5,0)**.

Precautions

None

Example

If the value of **units** equals **3**, **null** is returned. Otherwise, the value of **units** is returned.

```
insert into temp SELECT NULLIF(units, 3) FROM Orders;
```

COALESCE Expression

Syntax

```
COALESCE(value, value [, value ]* )
```

Description

Return the first value that is not **NULL**, counting from left to right.

Precautions

All values must be of the same type.

Example

5 is returned from the following example:

```
insert into temp SELECT COALESCE(NULL, 5) FROM Orders;
```

3.1.8.3 Window

GROUP WINDOW

Description

Group Window is defined in GROUP BY. One record is generated from each group. Group Window involves the following functions:

NOTE

- **time_attr** can be **processing-time** or **event-time**.
 - **event-time**: Specify the data type to **bigint** or **timestamp**.
 - **processing-time**: No need to specify the type.
- **interval** specifies the window period.
- Array functions

Table 3-27 Array functions

Function Name	Description
TUMBLE(time_attr, interval)	Indicates the tumble window.
HOP(time_attr, interval, interval)	Indicates the extended tumble window (similar to the datastream sliding window). You can set the output triggering cycle and window period.
SESSION(time_attr, interval)	Indicates the session window. A session window will be closed if no response is returned within a duration specified by interval .

- Window functions

Table 3-28 Window functions

Function Name	Description
TUMBLE_START(time_attr, interval)	Indicates the start time of returning to the tumble window. The parameter is a UTC time zone.
TUMBLE_END(time_attr, interval)	Indicates the end time of returning to the tumble window. The parameter is a UTC time zone.
HOP_START(time_attr, interval, interval)	Indicates the start time of returning to the extended tumble window. The parameter is a UTC time zone.

Function Name	Description
HOP_END(time_attr, interval, interval)	Indicates the end time of returning to the extended tumble window. The parameter is a UTC time zone.
SESSION_START(time_attr, interval)	Indicates the start time of returning to the session window. The parameter is a UTC time zone.
SESSION_END(time_attr, interval)	Indicates the end time of returning to the session window. The parameter is a UTC time zone.

Example

```
//Calculate the SUM every day (event time).
insert into temp SELECT name,
  TUMBLE_START(ts, INTERVAL '1' DAY) as wStart,
  SUM(amount)
FROM Orders
GROUP BY TUMBLE(ts, INTERVAL '1' DAY), name;

//Calculate the SUM every day (processing time).
insert into temp SELECT name,
  SUM(amount)
FROM Orders
GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;

//Calculate the SUM over the recent 24 hours every hour (event time).
insert into temp SELECT product,
  SUM(amount)
FROM Orders
GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

//Calculate the SUM of each session and an inactive interval every 12 hours (event time).
insert into temp SELECT name,
  SESSION_START(ts, INTERVAL '12' HOUR) AS sStart,
  SESSION_END(ts, INTERVAL '12' HOUR) AS sEnd,
  SUM(amount)
FROM Orders
GROUP BY SESSION(ts, INTERVAL '12' HOUR), name;
```

OVER WINDOW

The difference between Over Window and Group Window is that one record is generated from one row in Over Window.

Syntax

```
OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime(ROWS number PRECEDING) |(RANGE (BETWEEN INTERVAL '1' SECOND
  PRECEDING AND CURRENT ROW | UNBOUNDED preceding))
)
```

Description

Table 3-29 Parameter description

Parameter	Parameter Description
PARTITION BY	Indicates the primary key of the specified group. Each group separately performs calculation.
ORDER BY	Indicates the processing time or event time as the timestamp for data.
ROWS	Indicates the count window.
RANGE	Indicates the time window.

Precautions

- In the same SELECT statement, windows defined by aggregate functions must be the same.
- Currently, Over Window only supports forward calculation (preceding).
- The value of **ORDER BY** must be specified as **processing time** or **event time**.
- Constants do not support aggregation, such as sum(2).

Example

```
//Calculate the count and total number from syntax rules enabled to now (in proctime).
insert into temp SELECT name,
  count(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as
cnt1,
  sum(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt2
FROM Orders;

//Calculate the count and total number of the recent four records (in proctime).
insert into temp SELECT name,
  count(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt1,
  sum(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt2
FROM Orders;

//Calculate the count and total number last 60s (in eventtime). Process the events based on event time,
which is the timeattr field in Orders.
insert into temp SELECT name,
  count(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60'
SECOND PRECEDING AND CURRENT ROW) as cnt1,
  sum(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND
PRECEDING AND CURRENT ROW) as cnt2
FROM Orders;
```

3.1.8.4 JOIN Between Stream Data and Table Data

The JOIN operation allows you to query data from a table and write the query result to the sink stream. Currently, only RDSs and DCS Redis tables are supported. The ON keyword describes the Key used for data query and then writes the **Value** field to the sink stream.

For details about the data definition statements of RDS tables, see [Creating an RDS Table](#).

For details about the data definition statements of Redis tables, see [Creating a Redis Table](#).

Syntax

```
FROM tableExpression JOIN tableExpression  
ON value11 = value21 [ AND value12 = value22]
```

Syntax Description

The ON keyword only supports equivalent query of table attributes. If level-2 keys exist (specifically, the Redis value type is HASH), the AND keyword needs to be used to express the equivalent query between Key and Hash Key.

Precautions

None

Example

Perform equivalent JOIN between the vehicle information source stream and the vehicle price table, get the vehicle price data, and write the price data into the vehicle information sink stream.

```
CREATE SOURCE STREAM car_infos (  
  car_id STRING,  
  car_owner STRING,  
  car_brand STRING,  
  car_detail_type STRING  
)  
WITH (  
  type = "dis",  
  region = "",  
  channel = "dliinput",  
  partition_count = "1",  
  encode = "csv",  
  field_delimiter = ","  
);  
  
/** Create a data dimension table to connect to the source stream to fulfill field backfill.  
 *  
 * Reconfigure the following options according to actual conditions:  
 * value_type: indicates the value type of the Redis key value. The value can be STRING, HASH, SET, ZSET,  
 or LIST. For the HASH type, you need to specify hash_key_column as the layer-2 primary key. For the SET  
 type, you need to concatenate all queried values using commas (.).  
 * key_column: indicates the column name corresponding to the primary key of the dimension table.  
 * hash_key_column: indicates the column name corresponding to the KEY of the HASHMAP when  
 value_type is HASH. If value_type is not HASH, you do not need to set this option.  
 * cluster_address: indicates the DCS Redis cluster address.  
 * password: indicates the DCS Redis cluster password.  
 **/  
CREATE TABLE car_price_table (  
  car_brand STRING,  
  car_detail_type STRING,  
  car_price STRING  
)  
WITH (  
  type = "dcs_redis",  
  value_type = "hash",  
  key_column = "car_brand",  
  hash_key_column = "car_detail_type",  
  cluster_address = "192.168.1.238:6379",  
  password = "xxxxxxx"  
);
```

```
CREATE SINK STREAM audi_car_owner_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_detail_type STRING,
  car_price STRING
)
WITH (
  type = "dis",
  region = "",
  channel = "dlioutput",
  partition_key = "car_owner",
  encode = "csv",
  field_delimiter = ","
);

INSERT INTO audi_car_owner_info
SELECT t1.car_id, t1.car_owner, t2.car_brand, t1.car_detail_type, t2.car_price
FROM car_infos as t1 join car_price_table as t2
ON t2.car_brand = t1.car_brand and t2.car_detail_type = t1.car_detail_type
WHERE t1.car_brand = "audi";
```

3.1.9 Data Types

Overview

Data type is a basic attribute of data and used to distinguish different types of data. Different data types occupy different storage space and support different operations. Data is stored in data tables in the database. Each column of a data table defines the data type. During storage, data must be stored according to data types.

Similar to the open source community, Flink SQL of the Huawei big data platform supports both native data types and complex data types.

Primitive Data Types

[Table 3-30](#) lists native data types supported by Flink SQL.

Table 3-30 Primitive Data Types

Data Type	Description	Storage Space	Value Range
VARCHAR	Character with a variable length	-	-
BOOLEAN	Boolean	-	TRUE/FALSE
TINYINT	Signed integer	1 byte	-128-127
SMALLINT	Signed integer	2 bytes	-32768-32767
INT	Signed integer	4 bytes	-2147483648 to 2147483647
INTEGER	Signed integer	4 bytes	-2147483648 to 2147483647

Data Type	Description	Storage Space	Value Range
BIGINT	Signed integer	8 bytes	-9223372036854775808 to 9223372036854775807
REAL	Single-precision floating point	4 bytes	-
FLOAT	Single-precision floating point	4 bytes	-
DOUBLE	Double-precision floating-point	8 bytes	-
DECIMAL	Data type of valid fixed places and decimal places	-	-
DATE	Date type in the format of yyyy-MM-dd, for example, 2014-05-29	-	DATE does not contain time information. Its value ranges from 0000-01-01 to 9999-12-31.
TIME	Time type in the format of HH:MM:SS For example, 20:17:40	-	-
TIMESTAMP(3)	Timestamp of date and time For example, 1969-07-20 20:17:40	-	-
INTERVAL timeUnit [TO timeUnit]	Time interval For example, INTERVAL '1:5' YEAR TO MONTH, INTERVAL '45' DAY	-	-

Complex Data Types

Flink SQL supports complex data types and complex type nesting. [Table 3-31](#) describes complex data types.

Table 3-31 Complex Data Types

Data Type	Description	Declaration Method	Reference Method	Construction Method
ARRAY	Indicates a group of ordered fields that are of the same data type.	ARRAY[TYPE]	Variable name [subscript] . The subscript starts from 1, for example, v1[1] .	Array[value1, value2, ...] as v1
MAP	Indicates a group of unordered key/value pairs. The key must be native data type, but the value can be either native data type or complex data type. The type of the same MAP key, as well as the MAP value, must be the same.	MAP [TYPE, TYPE]	Variable name [key] , for example, v1[key]	Map[key, value, key2, value2, key3, value3.....] as v1
ROW	Indicates a group of named fields. The data types of the fields can be different.	ROW<a1 TYPE1, a2 TYPE2>	Variable name. Field name, for example, v1.a1 .	Row('1',2) as v1

Here is a sample code:

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  address ROW<city STRING, province STRING, country STRING>,
  average_speed MAP[STRING, LONG],
  speeds ARRAY[LONG]
)
WITH (
  type = "dis",
  region = "xxx",
  channel = "dliinput",
  encode = "json"
);

CREATE temp STREAM car_speed_infos (
  car_id STRING,
  province STRING,
  average_speed LONG,
  start_speed LONG
);

INSERT INTO car_speed_infos SELECT
  car_id,
  address.province,
  average_speed[address.city],
  speeds[1]
FROM car_infos;
```

Complex Type Nesting

- JSON format enhancement

The following uses Source as an example. The method of using Sink is the same.

- **json_schema** can be configured.

After **json_schema** is configured, fields in DDL can be automatically generated from **json_schema** without declaration. Here is a sample code:

```
CREATE SOURCE STREAM data_with_schema WITH (  
    type = "dis",  
    region = "xxx",  
    channel = "dis-in",  
    encode = "json",  
    json_schema = '{"definitions":{"address":{"type":"object","properties":{"street_address":{"type":"string"},"city":{"type":"string"},"state":{"type":"string"}}, "required":["street_address","city","state"]},"type":"object","properties":{"billing_address":{"$ref":"#/definitions/address"},"shipping_address":{"$ref":"#/definitions/address"},"optional_address":{"oneOf":[{"type":"null"}, {"$ref":"#/definitions/address"}]}}}'  
);  
  
CREATE SINK STREAM buy_infos (  
    billing_address_city STRING,  
    shipping_address_state string  
) WITH (  
    type = "obs",  
    encode = "csv",  
    region = "xxx",  
    field_delimiter = ",",  
    row_delimiter = "\n",  
    obs_dir = "bucket/car_infos",  
    file_prefix = "over",  
    rolling_size = "100m"  
);  
  
insert into buy_infos select billing_address.city, shipping_address.state from  
data_with_schema;
```

Example data

```
{  
  "billing_address":  
  {  
    "street_address": "xxx",  
    "city": "xxx",  
    "state": "xxx"  
  },  
  "shipping_address":  
  {  
    "street_address": "xxx",  
    "city": "xxx",  
    "state": "xxx"  
  }  
}
```

- The **json_schema** and **json_config** parameters can be left empty. For details about how to use **json_config**, see the example in [Open-Source Kafka Source Stream](#).

In this case, the attribute name in the DDL is used as the JSON key for parsing by default.

The following is example data. It contains nested JSON fields, such as **billing_address** and **shipping_address**, and non-nested fields **id** and **type2**.

```
{  
  "id": "1",  
  "type2": "online",
```

```
"billing_address":
{
  "street_address":"xxx",
  "city":"xxx",
  "state":"xxx"
},
"shipping_address":
{
  "street_address":"xxx",
  "city":"xxx",
  "state":"xxx"
}
}
```

The table creation and usage examples are as follows:

```
CREATE SOURCE STREAM car_info_data (
  id STRING,
  type2 STRING,
  billing_address Row<street_address string, city string, state string>,
  shipping_address Row<street_address string, city string, state string>,
  optional_address Row<street_address string, city string, state string>
) WITH (
  type = "dis",
  region = "xxx",
  channel = "dis-in",
  encode = "json"
);

CREATE SINK STREAM buy_infos (
  id STRING,
  type2 STRING,
  billing_address_city STRING,
  shipping_address_state string
) WITH (
  type = "obs",
  encode = "csv",
  region = "xxx",
  field_delimiter = ",",
  row_delimiter = "\n",
  obs_dir = "bucket/car_infos",
  file_prefix = "over",
  rolling_size = "100m"
);

insert into buy_infos select id, type2, billing_address.city, shipping_address.state from
car_info_data;
```

- Complex data types supported by sink serialization
 - Currently, only the CSV and JSON formats support complex data types.
 - For details about the JSON format, see [Json format enhancement](#).
 - There is no standard format for CSV files. Therefore, only sink parsing is supported.
 - Output format: It is recommended that the output format be the same as that of the native Flink.
Map: {key1=Value1, key2=Value2}
Row: Attributes are separated by commas (,), for example, **Row(1,'2')** => 1,'2'.

3.1.10 User-Defined Functions

Overview

DLI supports the following three types of user-defined functions (UDFs):

- Regular UDF: takes in one or more input parameters and returns a single result.
- User-defined table-generating function (UDTF): takes in one or more input parameters and returns multiple rows or columns.
- User-defined aggregate function (UDAF): aggregates multiple records into one value.

NOTE

UDFs can only be used in dedicated queues.

POM Dependency

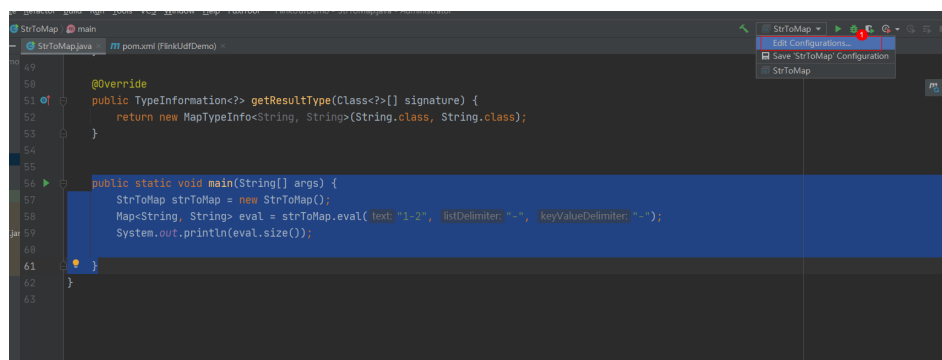
```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-table_2.11</artifactId>
  <version>1.7.2</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_2.11</artifactId>
  <version>1.7.2</version>
  <scope>provided</scope>
</dependency>
```

Precautions

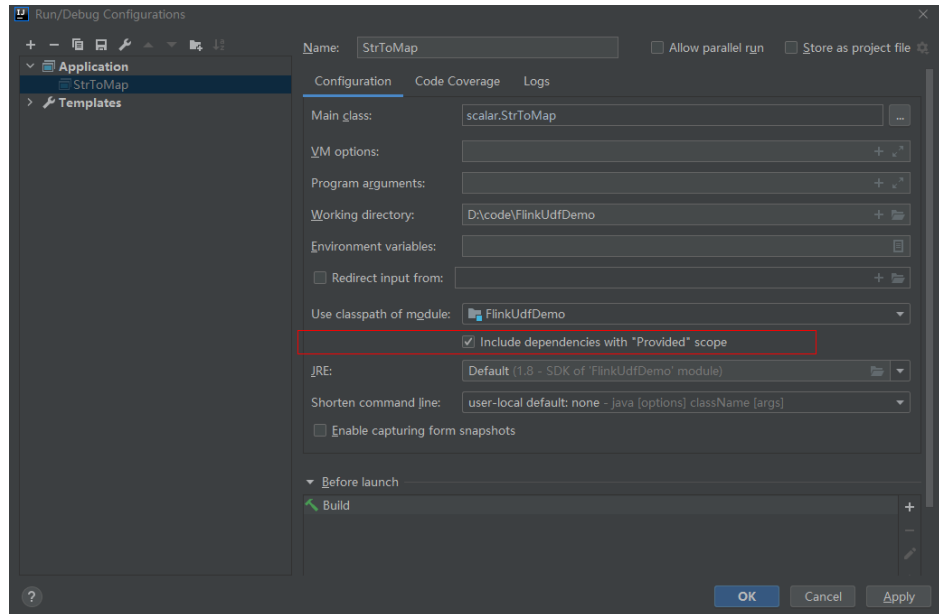
- **Currently, Python is not supported for programming UDFs, UDTFs, and UDAFs.**
- If you use IntelliJ IDEA to debug the created UDF, select **include dependencies with "Provided" scope**. Otherwise, the dependency packages in the POM file cannot be loaded for local debugging.

The following uses IntelliJ IDEA 2020.2 as an example:

- a. On the IntelliJ IDEA page, select the configuration file you need to debug and click **Edit Configurations**.



- b. On the **Run/Debug Configurations** page, select **include dependencies with "Provided" scope**.



c. Click **OK**.

Using UDFs

1. Write the code of custom functions. For details about the code examples, see [UDF](#), [UDTF](#), or [UDAF](#).
2. Compile the UDF code, pack it into a JAR package, and upload the package to OBS.
3. In the left navigation pane of the DLI management console, click **Job Management > Flink Jobs**. Locate the row where the target resides and click **Edit** in the **Operation** column to switch to the page where you can edit the job.
4. On the **Running Parameters** tab page, select an exclusive queue for **Queue**. The **UDF Jar** parameter is displayed. Select the JAR file stored on OBS and click **Save**.

NOTE

Before selecting a user-defined function JAR package, upload the JAR package to the created OBS bucket.

After the JAR package is selected, add the UDF statement to the SQL statement. The following is an example:

```
CREATE FUNCTION udf_test AS 'com.xxx.udf.UdfScalarFunction';
```

UDF

The regular UDF must inherit the `ScalarFunction` function and implement the `eval` method. The open and close functions are optional.

Example code

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class UdfScalarFunction extends ScalarFunction {
    private int factor = 12;
    public UdfScalarFunction() {
```

```
    this.factor = 12;
  }
  /**
   * (optional) Initialization
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  /**
   * Custom logic
   * @param s
   * @return
   */
  public int eval(String s) {
    return s.hashCode() * factor;
  }
  /**
   * Optional
   */
  @Override
  public void close() {}
}
```

Example

```
CREATE FUNCTION udf_test AS 'com.xxx.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

The UDTF must inherit the TableFunction function and implement the eval method. The open and close functions are optional. If the UDTF needs to return multiple columns, you only need to declare the returned value as **Tuple** or **Row**. If **Row** is used, you need to overload the getResultType method to declare the returned field type.

Example code

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class UdfTableFunction extends TableFunction<Row> {
  private Logger log = LoggerFactory.getLogger(TableFunction.class);
  /**
   * (optional) Initialization
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  public void eval(String str, String split) {
    for (String s : str.split(split)) {
      Row row = new Row(2);
      row.setField(0, s);
      row.setField(1, s.length());
      collect(row);
    }
  }
  /**
   * Declare the type returned by the function
   * @return
   */
  @Override
  public TypeInformation<Row> getResultType() {
```

```
return Types.ROW(Types.STRING, Types.INT);
}
/**
 * Optional
 */
@Override
public void close() {}
}
```

Example

The UDTF supports CROSS JOIN and LEFT JOIN. When the UDTF is used, the **LATERAL** and **TABLE** keywords must be included.

- **CROSS JOIN**: does not output the data of a row in the left table if the UDTF does not output the result for the data of the row.
- **LEFT JOIN**: outputs the data of a row in the left table even if the UDTF does not output the result for the data of the row, but pads null with UDTF-related fields.

```
CREATE FUNCTION udtf_test AS 'com.xxx.udf.TableFunction';
// CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);
// LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

UDAF

The UDAF must inherit the AggregateFunction function. You need to create an accumulator for storing the computing result, for example, **WeightedAvgAccum** in the following example code.

Example code

```
public class WeightedAvgAccum {
    public long sum = 0;
    public int count = 0;
}
```

```
import org.apache.flink.table.functions.AggregateFunction;
import java.util.Iterator;
/**
 * The first type variable is the type returned by the aggregation function, and the second type variable is of
 * the Accumulator type.
 * Weighted Average user-defined aggregate function.
 */
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {
    // Initialize the accumulator.
    @Override
    public WeightedAvgAccum createAccumulator() {
        return new WeightedAvgAccum();
    }
    // Return the intermediate computing value stored in the accumulator.
    @Override
    public Long getValue(WeightedAvgAccum acc) {
        if (acc.count == 0) {
            return null;
        } else {
            return acc.sum / acc.count;
        }
    }
    // Update the intermediate computing value according to the input.
    public void accumulate(WeightedAvgAccum acc, long iValue) {
        acc.sum += iValue;
    }
}
```

```
acc.count += 1;
}
// Perform the retraction operation, which is opposite to the accumulate operation.
public void retract(WeightedAvgAccum acc, long iValue) {
acc.sum -= iValue;
acc.count -= 1;
}
// Combine multiple accumulator values.
public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {
Iterator<WeightedAvgAccum> iter = it.iterator();
while (iter.hasNext()) {
WeightedAvgAccum a = iter.next();
acc.count += a.count;
acc.sum += a.sum;
}
}
// Reset the intermediate computing value.
public void resetAccumulator(WeightedAvgAccum acc) {
acc.count = 0;
acc.sum = 0L;
}
}
```

Example

```
CREATE FUNCTION udaf_test AS 'com.xxx.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

3.1.11 Built-In Functions

3.1.11.1 Mathematical Operation Functions

Relational Operators

All data types can be compared by using relational operators and the result is returned as a **BOOLEAN** value.

Relationship operators are binary operators. Two compared data types must be of the same type or they must support implicit conversion.

[Table 3-32](#) lists all relational operators supported by Flink SQL.

Table 3-32 Relational Operators

Operator	Returned Data Type	Description
A = B	BOOLEAN	If A is equal to B, then TRUE is returned. Otherwise, FALSE is returned. This operator is used for value assignment.
A <> B	BOOLEAN	If A is not equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned. This operator follows the standard SQL syntax.
A < B	BOOLEAN	If A is less than B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.

Operator	Returned Data Type	Description
A <= B	BOOLEAN	If A is less than or equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A > B	BOOLEAN	If A is greater than B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A >= B	BOOLEAN	If A is greater than or equal to B, then TRUE is returned. Otherwise, FALSE is returned. If A or B is NULL , then NULL is returned.
A IS NULL	BOOLEAN	If A is NULL , then TRUE is returned. Otherwise, FALSE is returned.
A IS NOT NULL	BOOLEAN	If A is not NULL , then TRUE is returned. Otherwise, FALSE is returned.
A IS DISTINCT FROM B	BOOLEAN	If A is not equal to B, TRUE is returned. NULL indicates A equals B.
A IS NOT DISTINCT FROM B	BOOLEAN	If A is equal to B, TRUE is returned. NULL indicates A equals B.
A BETWEEN [ASYMMETRIC SYMMETRIC] B AND C	BOOLEAN	If A is greater than or equal to B but less than or equal to C, TRUE is returned. <ul style="list-style-type: none"> • ASYMMETRIC: indicates that B and C are location-related. For example, "A BETWEEN ASYMMETRIC B AND C" is equivalent to "A BETWEEN B AND C". • SYMMETRIC: indicates that B and C are not location-related. For example, "A BETWEEN SYMMETRIC B AND C" is equivalent to "A BETWEEN B AND C) OR (A BETWEEN C AND B".
A NOT BETWEEN B AND C	BOOLEAN	If A is less than B or greater than C, TRUE is returned.
A LIKE B [ESCAPE C]	BOOLEAN	If A matches pattern B, TRUE is returned. The escape character C can be defined as required.
A NOT LIKE B [ESCAPE C]	BOOLEAN	If A does not match pattern B, TRUE is returned. The escape character C can be defined as required.

Operator	Returned Data Type	Description
A SIMILAR TO B [ESCAPE C]	BOOLEAN	If A matches regular expression B, TRUE is returned. The escape character C can be defined as required.
A NOT SIMILAR TO B [ESCAPE C]	BOOLEAN	If A does not match regular expression B, TRUE is returned. The escape character C can be defined as required.
value IN (value [, value]*)	BOOLEAN	If the value is equal to any value in the list, TRUE is returned.
value NOT IN (value [, value]*)	BOOLEAN	If the value is not equal to any value in the list, TRUE is returned.

 **NOTE**

- Values of the double, real, and float types may be different in precision. The equal sign (=) is not recommended for comparing two values of the double type. You are advised to obtain the absolute value by subtracting these two values of the double type and determine whether they are the same based on the absolute value. If the absolute value is small enough, the two values of the double data type are regarded equal. For example:
`abs(0.9999999999 - 1.0000000000) < 0.000000001 //The precision decimal places of 0.9999999999 and 1.0000000000 are 10, while the precision decimal place of 0.000000001 is 9. Therefore, 0.9999999999 can be regarded equal to 1.0000000000.`
- Comparison between data of the numeric type and character strings is allowed. During comparison using relational operators, including >, <, ≤, and ≥, data of the string type is converted to numeric type by default. No characters other than numeric characters are allowed.
- Character strings can be compared using relational operators.

Logical Operators

Common logical operators are AND, OR, and NOT. Their priority order is NOT > AND > OR.

Table 3-33 lists the calculation rules. A and B indicate logical expressions.

Table 3-33 Logical Operators

Operator	Result Type	Description
A OR B	BOOLEAN	If A or B is TRUE, TRUE is returned. Three-valued logic is supported.

Operator	Result Type	Description
A AND B	BOOLEAN	If both A and B are TRUE, TRUE is returned. Three-valued logic is supported.
NOT A	BOOLEAN	If A is not TRUE, TRUE is returned. If A is UNKNOWN, UNKNOWN is returned.
A IS FALSE	BOOLEAN	If A is TRUE, TRUE is returned. If A is UNKNOWN, FALSE is returned.
A IS NOT FALSE	BOOLEAN	If A is not FALSE, TRUE is returned. If A is UNKNOWN, TRUE is returned.
A IS TRUE	BOOLEAN	If A is TRUE, TRUE is returned. If A is UNKNOWN, FALSE is returned.
A IS NOT TRUE	BOOLEAN	If A is not TRUE, TRUE is returned. If A is UNKNOWN, TRUE is returned.
A IS UNKNOWN	BOOLEAN	If A is UNKNOWN, TRUE is returned.
A IS NOT UNKNOWN	BOOLEAN	If A is not UNKNOWN, TRUE is returned.

 **NOTE**

Only data of the Boolean type can be used for calculation using logical operators. Implicit type conversion is not supported.

Arithmetic Operators

Arithmetic operators include binary operators and unary operators, for all of which, the returned results are of the numeric type. [Table 3-34](#) lists arithmetic operators supported by Flink SQL.

Table 3-34 Arithmetic Operators

Operator	Result Type	Description
+ numeric	All numeric types	Returns numbers.
- numeric	All numeric types	Returns negative numbers.

Operator	Result Type	Description
A + B	All numeric types	A plus B. The result type is associated with the operation data type. For example, if floating-point number is added to an integer, the result will be a floating-point number.
A - B	All numeric types	A minus B. The result type is associated with the operation data type.
A * B	All numeric types	Multiply A and B. The result type is associated with the operation data type.
A / B	All numeric types	Divide A by B. The result is a number of the double type (double-precision number).
POWER(A, B)	All numeric types	Returns the value of A raised to the power B.
ABS(numeric)	All numeric types	Returns the absolute value of a specified value.
MOD(A, B)	All numeric types	Returns the remainder (modulus) of A divided by B. A negative value is returned only when A is a negative value.
SQRT(A)	All numeric types	Returns the square root of A.
LN(A)	All numeric types	Returns the nature logarithm of A (base e).
LOG10(A)	All numeric types	Returns the base 10 logarithms of A.
EXP(A)	All numeric types	Return the value of e raised to the power of a.
CEIL(A) CEILING(A)	All numeric types	Return the smallest integer that is greater than or equal to a. For example: ceil(21.2) = 22.
FLOOR(A)	All numeric types	Return the largest integer that is less than or equal to a. For example: floor(21.2) = 21.
SIN(A)	All numeric types	Returns the sine value of A.
COS(A)	All numeric types	Returns the cosine value of A.
TAN(A)	All numeric types	Returns the tangent value of A.

Operator	Result Type	Description
COT(A)	All numeric types	Returns the cotangent value of A.
ASIN(A)	All numeric types	Returns the arc sine value of A.
ACOS(A)	All numeric types	Returns the arc cosine value of A.
ATAN(A)	All numeric types	Returns the arc tangent value of A.
DEGREES(A)	All numeric types	Convert the value of a from radians to degrees.
RADIANS(A)	All numeric types	Convert the value of a from degrees to radians.
SIGN(A)	All numeric types	Returns the sign of A. 1 is returned if A is positive. -1 is returned if A is negative. Otherwise, 0 is returned.
ROUND(A, d)	All numeric types	Round A to d places right to the decimal point. d is an int type. For example: round(21.263,2) = 21.26.
PI()	All numeric types	Return the value of pi .

 **NOTE**

Data of the string type is not allowed in arithmetic operations.

3.1.11.2 String Functions

The common character string functions of DLI are as follows:

Table 3-35 String Operators

Operator	Returned Data Type	Description
 	VARCHAR	Concatenates two strings.
CHAR_LENGTH	INT	Returns the number of characters in a string.
CHARACTER_LENGTH	INT	Returns the number of characters in a string.

Operator	Returned Data Type	Description
CONCAT	VARC HAR	Concatenates two or more string values to form a new string. If the value of any parameter is NULL , skip this parameter.
CONCAT_WS	VARC HAR	Concatenates each parameter value and the separator specified by the first parameter separator to form a new string. The length and type of the new string depend on the input value.
HASH_CODE	INT	Returns the absolute value of HASH_CODE() of a string. In addition to string , int , bigint , float , and double are also supported.
INITCAP	VARC HAR	Returns a string whose first letter is in uppercase and the other letters in lowercase. Words are sequences of alphanumeric characters separated by non-alphanumeric characters.
IS_ALPHA	BOOLEA N	Checks whether a string contains only letters.
IS_DIGITS	BOOLEA N	Checks whether a string contains only digits.
IS_NUMBER	BOOLEA N	Checks whether a string is numeric.
IS_URL	BOOLEA N	Checks whether a string is a valid URL.
JSON_VALUE	VARC HAR	Obtains the value of a specified path in a JSON string.
KEY_VALUE	VARC HAR	Obtains the value of a key in a key-value pair string.
LOWER	VARC HAR	Returns a string of lowercase characters.
LPAD	VARC HAR	Concatenates the pad string to the left of the string until the length of the new string reaches the specified length len.
MD5	VARC HAR	Returns the MD5 value of a string. If the parameter is an empty string (that is, the parameter is ""), an empty string is returned.
OVERLAY	VARC HAR	Replaces the substring of x with y . Replace length +1 characters starting from start_position .

Operator	Returned Data Type	Description
POSITION	INT	Returns the position of the first occurrence of the target string x in the queried string y . If the target string x does not exist in the queried string y , 0 is returned.
REPLACE	VARCHAR	Replaces all str2 in the str1 string with str3 . <ul style="list-style-type: none"> • str1: original character. • str2: target character. • str3: replacement character.
RPAD	VARCHAR	Concatenates the pad string to the right of the str string until the length of the new string reaches the specified length len .
SHA1	STRING	Returns the SHA1 value of the expr string.
SHA256	STRING	Returns the SHA256 value of the expr string.
STRING_TO_ARRAY	ARRAY[STRING]	Separates the value string as string arrays by using the delimiter.
SUBSTRING	VARCHAR	Returns the substring starting from a fixed position of A . The start position starts from 1.
TRIM	STRING	Removes A at the start position, or end position, or both the start and end positions from B . By default, string expressions A at both the start and end positions are removed.
UPPER	VARCHAR	Returns a string converted to uppercase characters.

||

- Function
Concatenates two character strings.
- Syntax
VARCHAR VARCHAR a || VARCHAR b
- Parameter description
 - **a**: character string.
 - **b**: character string.
- Example
 - Test statement
SELECT "hello" || "world";
 - Test result
"helloworld"

CHAR_LENGTH

- Function
Returns the number of characters in a string.
- Syntax
INT CHAR_LENGTH(a)
- Parameter description
 - **a**: character string.
- Example
 - Test statement
SELECT CHAR_LENGTH(var1) as aa FROM T1;
 - Test data and result

Table 3-36 Test data and result

Test Data (var1)	Test Result (aa)
abcde123	8

CHARACTER_LENGTH

- Function
Returns the number of characters in a string.
- Syntax
INT CHARACTER_LENGTH(a)
- Parameter description
 - **a**: character string.
- Example
 - Test statement
SELECT CHARACTER_LENGTH(var1) as aa FROM T1;
 - Test data and result

Table 3-37 Test data and result

Test Data (var1)	Test Result (aa)
abcde123	8

CONCAT

- Function
Concatenates two or more string values to form a new string. If the value of any parameter is NULL, skip this parameter.
- Syntax
VARCHAR CONCAT(VARCHAR var1, VARCHAR var2, ...)
- Parameter description
 - **var1**: character string

- **var2**: character string
- Example
 - Test statement

```
SELECT CONCAT("abc", "def", "ghi", "jkl");
```
 - Test result

```
"abcdefghijkl"
```

CONCAT_WS

- Function

Concatenates each parameter value and the separator specified by the first parameter separator to form a new string. The length and type of the new string depend on the input value.
- 📖 NOTE

If the value of **separator** is **null**, **separator** is combined with an empty string. If other parameters are set to null, the parameters whose values are null are skipped during combination.
- Syntax

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```
- Parameter description
 - **separator**: separator.
 - **var1**: character string
 - **var2**: character string
- Example
 - Test statement

```
SELECT CONCAT_WS("-", "abc", "def", "ghi", "jkl");
```
 - Test result

```
"abc-def-ghi-jkl"
```

HASH_CODE

- Function

Returns the absolute value of **HASH_CODE()** of a string. In addition to **string**, **int**, **bigint**, **float**, and **double** are also supported.
- Syntax

```
INT HASH_CODE(VARCHAR str)
```
- Parameter description
 - **str**: character string.
- Example
 - Test statement

```
SELECT HASH_CODE("abc");
```
 - Test result

```
96354
```

INITCAP

- Function

Return the string whose first letter is in uppercase and the other letters in lowercase. Strings are sequences of alphanumeric characters separated by non-alphanumeric characters.

- Syntax
VARCHAR INITCAP(a)
- Parameter description
 - a: character string.
- Example
 - Test statement
SELECT INITCAP(var1)as aa FROM T1;
 - Test data and result

Table 3-38 Test data and result

Test Data (var1)	Test Result (aa)
aBCde	Abcde

IS_ALPHA

- Function
Checks whether a character string contains only letters.
- Syntax
BOOLEAN IS_ALPHA(VARCHAR content)
- Parameter description
 - **content**: Enter a character string.
- Example
 - Test statement
SELECT IS_ALPHA(content) AS case_result FROM T1;
 - Test data and results

Table 3-39 Test data and results

Test Data (content)	Test Result (case_result)
Abc	true
abc1#\$	false
null	false
Empty string	false

IS_DIGITS

- Function
Checks whether a character string contains only digits.
- Syntax

BOOLEAN IS_DIGITS(VARCHAR content)

- Parameter description
 - **content:** Enter a character string.
- Example
 - Test statement
SELECT IS_DIGITS(content) AS case_result FROM T1;
 - Test data and results

Table 3-40 Test data and results

Test Data (content)	Test Result (case_result)
78	true
78.0	false
78a	false
null	false
Empty string	false

IS_NUMBER

- Function
This function is used to check whether a character string is a numeric string.
- Syntax
BOOLEAN IS_NUMBER(VARCHAR content)
- Parameter description
 - **content:** Enter a character string.
- Example
 - Test statement
SELECT IS_NUMBER(content) AS case_result FROM T1;
 - Test data and results

Table 3-41 Test data and results

Test Data (content)	Test Result (case_result)
78	true
78.0	true
78a	false
null	false
Empty string	false

IS_URL

- Function
This function is used to check whether a character string is a valid URL.
- Syntax
BOOLEAN IS_URL(VARCHAR content)
- Parameter description
 - **content:** Enter a character string.
- Example
 - Test statement
SELECT IS_URL(content) AS case_result FROM T1;
 - Test data and results

Table 3-42 Test data and results

Test Data (content)	Test Result (case_result)
https://www.testweb.com	true
https://www.testweb.com:443	true
www.testweb.com:443	false
null	false
Empty string	false

JSON_VALUE

- Function
Obtains the value of a specified path in a JSON character string.
- Syntax
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
- Parameter description
 - **content:** Enter a character string.
 - **path:** path to be obtained.
- Example
 - Test statement
SELECT JSON_VALUE(content, path) AS case_result FROM T1;
 - Test data and results

Table 3-43 Test data and results

Test Data (content and path)	Test Result (case_result)
{ "a1": "v1", "a2": 7, "a3": 8.0, "a4": {"a41": "v41", "a42": ["v1", "v2"]}}	\$ { "a1": "v1", "a2": 7, "a3": 8.0, "a4": {"a41": "v41", "a42": ["v1", "v2"]}}

Test Data (content and path)		Test Result (case_result)
{ "a1": "v1", "a2": 7, "a3": 8.0, "a4": { "a41": "v41", "a42": ["v1", "v2"] } }	\$.a1	v1
{ "a1": "v1", "a2": 7, "a3": 8.0, "a4": { "a41": "v41", "a42": ["v1", "v2"] } }	\$.a4	{ "a41": "v41", "a42": ["v1", "v2"] }
{ "a1": "v1", "a2": 7, "a3": 8.0, "a4": { "a41": "v41", "a42": ["v1", "v2"] } }	\$.a4. a42	["v1", "v2"]
{ "a1": "v1", "a2": 7, "a3": 8.0, "a4": { "a41": "v41", "a42": ["v1", "v2"] } }	\$.a4. a42[0]	v1

KEY_VALUE

- Function
This function is used to obtain the value of a key in a key-value pair string.
- Syntax
VARCHAR KEY_VALUE(VARCHAR content, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
- Parameter description
 - **content**: Enter a character string.
 - **split1**: separator of multiple key-value pairs.
 - **split2**: separator between the key and value.
 - **key_name**: name of the key to be obtained.
- Example
 - Test statement
SELECT KEY_VALUE(content, split1, split2, key_name) AS case_result FROM T1;
 - Test data and results

Table 3-44 Test data and results

Test Data (content, split1, split2, and key_name)				Test Result (case_result)
k1=v1;k2=v2	;	=	k1	v1
null	;	=	k1	null
k1=v1;k2=v2	nul l	=	k1	null

LOWER

- Function
Returns a string of lowercase characters.

- Syntax
VARCHAR LOWER(A)
- Parameter description
 - **A**: character string.
- Example
 - Test statement
SELECT LOWER(var1) AS aa FROM T1;
 - Test data and result

Table 3-45 Test data and result

Test Data (var1)	Test Result (aa)
ABc	abc

LPAD

- Function
Concatenates the pad string to the left of the str string until the length of the new string reaches the specified length len.
- Syntax
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
- Parameter description
 - **str**: character string before concatenation.
 - **len**: length of the concatenated character string.
 - **pad**: character string to be concatenated.

NOTE

- If any parameter is null, **null** is returned.
 - If the value of len is a negative number, value **null** is returned.
 - If the value of **len** is less than the length of **str**, the first chunk of **str** characters in **len** length is returned.
- Example
 - Test statement
SELECT
LPAD("adc", 2, "hello"),
LPAD("adc", -1, "hello"),
LPAD("adc", 10, "hello");
 - Test result
"ad",,"hellohead"

MD5

- Function
Returns the MD5 value of a string. If the parameter is an empty string (that is, the parameter is ""), an empty string is returned.
- Syntax
VARCHAR MD5(VARCHAR str)

- Parameter description
 - **str**: character string
- Example
 - Test statement
`SELECT MD5("abc");`
 - Test result
`"900150983cd24fb0d6963f7d28e17f72"`

OVERLAY

- Function
Replaces the substring of **x** with **y**. Replaces length+1 characters starting from **start_position**.
- Syntax
`VARCHAR OVERLAY ((VARCHAR x PLACING VARCHAR y FROM INT start_position [FOR INT length]))`
- Parameter description
 - **x**: character string
 - **y**: character string.
 - **start_position**: start position.
 - **length (optional)**: indicates the character length.
- Example
 - Test statement
`OVERLAY('abcdefg' PLACING 'xyz' FROM 2 FOR 2) AS result FROM T1;`
 - Test result

Table 3-46 Test result

result
axyzdefg

POSITION

- Function
Returns the position of the first occurrence of the target string **x** in the queried string **y**. If the target character string **x** does not exist in the queried character string **y**, **0** is returned.
- Syntax
`INTEGER POSITION(x IN y)`
- Parameter description
 - **x**: character string
 - **y**: character string.
- Example
 - Test statement
`POSITION('in' IN 'chin') AS result FROM T1;`
 - Test result

Table 3-47 Test result

result
3

REPLACE

- **Function**
The character string replacement function is used to replace all **str2** in the **str1** string with **str3**.
- **Syntax**
VARCHAR REPLACE(VARCHAR str1, VARCHAR str2, VARCHAR str3)
- **Parameter description**
 - **str1**: original character.
 - **str2**: target character.
 - **str3**: replacement character.
- **Example**
 - **Test statement**

```
SELECT
  replace(
    "hello world hello world hello world",
    "world",
    "hello"
  );
```
 - **Test result**
"hello hello hello hello hello hello"

RPAD

- **Function**
Concatenates the pad string to the right of the str string until the length of the new string reaches the specified length len.
 - If any parameter is null, **null** is returned.
 - If the value of len is a negative number, value **null** is returned.
 - The value of **pad** is an empty string. If the value of **len** is less than the length of **str**, the string whose length is the same as the length of **str** is returned.
- **Syntax**
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
- **Parameter description**
 - **str**: start character string.
 - **len**: indicates the length of the new character string.
 - **pad**: character string that needs to be added repeatedly.
- **Example**
 - **Test statement**

```
SELECT
  RPAD("adc", 2, "hello"),
  RPAD("adc", -1, "hello"),
  RPAD("adc", 10, "hello");
```


- Test result
"ad", "adchellohe"

SHA1

- Function
Returns the SHA1 value of the **expr** string.
- Syntax
STRING SHA1 (STRING expr)
- Parameter description
 - **expr**: character string.
- Example
 - Test statement
SELECT SHA1("abc");
 - Test result
"a9993e364706816aba3e25717850c26c9cd0d89d"

SHA256

- Function
Returns the SHA256 value of the **expr** string.
- Syntax
STRING SHA256 (STRING expr)
- Parameter description
 - **expr**: character string.
- Example
 - Test statement
SELECT SHA256("abc");
 - Test result
"ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad"

STRING_TO_ARRAY

- Function
Separates the **value** string as character string arrays by using the delimiter.

NOTE

delimiter uses the Java regular expression. If special characters are used, they need to be escaped.

- Syntax
ARRAY[String] STRING_TO_ARRAY (STRING value, VARCHAR delimiter)
- Parameter description
 - **value**: character string.
 - **delimiter**: specifies the delimiter.
- Example
 - Test statement
SELECT
string_to_array("127.0.0.1", "\\."),
string_to_array("red-black-white-blue", "-");

- Test result
`[127,0,0,1],[red,black,white,blue]`

SUBSTRING

- Function
Returns the substring that starts from a fixed position of A. The start position starts from 1.
 - If **len** is not specified, the substring from the start position to the end of the string is truncated.
 - If **len** is specified, the substring starting from the position specified by **start** is truncated. The length is specified by **len**.

NOTE

The value of **start** starts from **1**. If the value is **0**, it is regarded as **1**. If the value of **start** is a negative number, the position is calculated from the end of the character string in reverse order.

- Syntax
`VARCHAR SUBSTRING(String A FROM INT start)`
Or
`VARCHAR SUBSTRING(String A FROM INT start FOR INT len)`
- Parameter description
 - **A**: specified character string.
 - **start**: start position for truncating the character string **A**.
 - **len**: intercepted length.
- Example
 - Test statement 1
`SELECT SUBSTRING("123456" FROM 2);`
 - Test result 1
`"23456"`
 - Test statement 2
`SELECT SUBSTRING("123456" FROM 2 FOR 4);`
 - Test result 2
`"2345"`

TRIM

- Function
Remove A at the start position, or end position, or both the start and end positions from B. By default, string expressions A at both the start and end positions are removed.
- Syntax
`STRING TRIM({ BOTH | LEADING | TRAILING } STRING a FROM STRING b)`
- Parameter description
 - **a**: character string.
 - **b**: character string.
- Example
 - Test statement
`SELECT TRIM(BOTH " " FROM " hello world ");`

- Test result
"hello world"

UPPER

- Function
Returns a string converted to an uppercase character.
- Syntax
VARCHAR UPPER(A)
- Parameter description
 - **A**: character string.
- Example
 - Test statement
SELECT UPPER("hello world");
 - Test result
"HELLO WORLD"

3.1.11.3 Temporal Functions

[Table 3-48](#) lists the time functions supported by Flink SQL.

Function Description

Table 3-48 Time Function

Function	Return Type	Description
DATE string	DATE	Parse the date string (yyyy-MM-dd) to a SQL date.
TIME string	TIME	Parse the time string (HH:mm:ss) to the SQL time.
TIMESTAMP string	TIMESTAMP	Convert the time string into timestamp. The time string format is yyyy-MM-dd HH:mm:ss.fff .

Function	Return Type	Description
INTERVAL string range	INTERVAL	<p>There are two types of intervals: yyyy-MM and dd HH:mm:sss.fff'. The range of yyyy-MM can be YEAR or YEAR TO MONTH, with the precision of month. The range of dd HH:mm:sss.fff' can be DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, or DAY TO MILLISECONDS, with the precision of millisecond. For example, if the range is DAY TO SECOND, the day, hour, minute, and second are all valid and the precision is second. DAY TO MINUTE indicates that the precision is minute.</p> <p>The following is an example:</p> <p>INTERVAL '10 00:00:00.004' DAY TO milliseconds indicates that the interval is 10 days and 4 milliseconds.</p> <p>INTERVAL '10' DAY indicates that the interval is 10 days and INTERVAL '2-10' YEAR TO MONTH indicates that the interval is 2 years and 10 months.</p>
CURRENT_DATE	DATE	Return the SQL date of UTC time zone.
CURRENT_TIME	TIME	Return the SQL time of UTC time zone.
CURRENT_TIMESTAMP	TIMESTAMP	Return the SQL timestamp of UTC time zone.
LOCALTIME	TIME	Return the SQL time of the current time zone.
LOCALTIMESTAMP	TIMESTAMP	Return the SQL timestamp of the current time zone.
EXTRACT(timeintervalunit FROM temporal)	INT	<p>Extract part of the time point or interval. Return the part in the int type.</p> <p>For example, 5 is returned from EXTRACT(DAY FROM DATE "2006-06-05").</p>
FLOOR(timepoint TO timeintervalunit)	TIME	<p>Round a time point down to the given unit.</p> <p>For example, 12:44:00 is returned from FLOOR(TIME '12:44:31' TO MINUTE).</p>
CEIL(timepoint TO timeintervalunit)	TIME	<p>Round a time point up to the given unit.</p> <p>For example, 12:45:00 is returned from CEIL(TIME '12:44:31' TO MINUTE).</p>
QUARTER(date)	INT	Return the quarter from the SQL date.

Function	Return Type	Description
(timepoint, temporal) OVERLAPS (timepoint, temporal)	BOOLEAN	<p>Check whether two intervals overlap. The time points and time are converted into a time range with a start point and an end point. The function is <i>leftEnd >= rightStart && rightEnd >= leftStart</i>. If leftEnd is greater than or equal to rightStart and rightEnd is greater than or equal to leftStart, true is returned. Otherwise, false is returned.</p> <p>The following is an example:</p> <ul style="list-style-type: none"> If leftEnd is 3:55:00 (2:55:00+1:00:00), rightStart is 3:30:00, rightEnd is 5:30:00 (3:30:00+2:00:00), and leftStart is 2:55:00, true will be returned. Specifically, true is returned from (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR). If leftEnd is 10:00:00, rightStart is 10:15:00, rightEnd is 13:15:00 (10:15:00+3:00:00), and leftStart is 9:00:00, false will be returned. Specifically, false is returned from (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR).
TO_TIMESTAMP(long expr)	TIMESTAMP	<p>Convert a timestamp to time.</p> <p>The input parameter this function must be of the BIGINT type. Other data types, such as VARCHAR and STRING, are not supported.</p> <p>For example, TO_TIMESTAMP (1628765159000) is converted to 2021-08-12 18:45:59.</p>

Function	Return Type	Description
UNIX_TIMESTAMP	BIGINT	<p>Returns the timestamp of a specified parameter. The timestamp type is BIGINT and the unit is second.</p> <p>The following methods are supported:</p> <ul style="list-style-type: none"> • UNIX_TIMESTAMP(): returns the timestamp of the current time if no parameter is specified. • UNIX_TIMESTAMP(String datestr): returns the timestamp indicated by the parameter if only one parameter is contained. The format of datestr must be yyyy-MM-dd HH:mm:ss. • UNIX_TIMESTAMP(String datestr, String format): returns the timestamp indicated by the first parameter if two parameters are contained. The second parameter can specify the format of datestr.
UNIX_TIMESTAMP_MS	BIGINT	<p>Returns the timestamp of a specified parameter. The timestamp type is BIGINT and the unit is millisecond.</p> <p>The following methods are supported:</p> <ul style="list-style-type: none"> • UNIX_TIMESTAMP_MS(): returns the timestamp of the current time if no parameter is specified. • UNIX_TIMESTAMP_MS(String datestr): returns the timestamp indicated by the parameter if only one parameter is contained. The format of datestr must be yyyy-MM-dd HH:mm:ss.SSS. • UNIX_TIMESTAMP_MS(String datestr, String format): returns the timestamp indicated by the first parameter if two parameters are contained. The second parameter can specify the format of datestr.

Precautions

None

Example

```
insert into temp SELECT Date '2015-10-11' FROM OrderA;//Date is returned
insert into temp1 SELECT Time '12:14:50' FROM OrderA;//Time is returned
insert into temp2 SELECT Timestamp '2015-10-11 12:14:50' FROM OrderA;//Timestamp is returned
```

3.1.11.4 Type Conversion Functions

Syntax

```
CAST(value AS type)
```

Syntax Description

This function is used to forcibly convert types.

Precautions

- If the input is **NULL**, **NULL** is returned.
- Flink jobs do not support the conversion of **bigint** to **timestamp** using **CAST**. You can convert it using **to_timestamp** or **to_localtimestamp**.

Example

Convert amount into a character string. The specified length of the string is invalid after the conversion.

```
insert into temp select cast(amount as VARCHAR(10)) from source_stream;
```

Common Type Conversion Functions

Table 3-49 Common type conversion functions

Function	Description
cast(v1 as varchar)	Converts v1 to a string. The value of v1 can be of the numeric type or of the timestamp, date, or time type.
cast (v1 as int)	Converts v1 to the int type. The value of v1 can be a number or a character.
cast(v1 as timestamp)	Converts v1 to the timestamp type. The value of v1 can be of the string , date , or time type.
cast(v1 as date)	Converts v1 to the date type. The value of v1 can be of the string or timestamp type.

- cast(v1 as varchar)
 - Test statement

```
SELECT cast(content as varchar) FROM T1;
```
 - Test data and result

Table 3-50 T1

content (INT)	varchar
5	"5"

- cast (v1 as int)
 - Test statement
SELECT cast(content as int) FROM T1;
 - Test data and result

Table 3-51 T1

content (STRING)	int
"5"	5

- cast(v1 as timestamp)
 - Test statement
SELECT cast(content as timestamp) FROM T1;
 - Test data and result

Table 3-52 T1

content (STRING)	timestamp
"2018-01-01 00:00:01"	1514736001000

- cast(v1 as date)
 - Test statement
SELECT cast(content as date) FROM T1;
 - Test data and result

Table 3-53 T1

content (TIMESTAMP)	date
1514736001000	"2018-01-01"

Detailed Sample Code

```

/** source */
CREATE
SOURCE STREAM car_infos (cast_int_to_varchar int, cast_String_to_int string,
case_string_to_timestamp string, case_timestamp_to_date timestamp) WITH (
  type = "dis",
  region = "xxxxx",
  channel = "dis-input",
  partition_count = "1",
  encode = "json",
  offset = "13",
  json_config =
"cast_int_to_varchar=cast_int_to_varchar;cast_String_to_int=cast_String_to_int;case_string_to_timestamp=case_string_to_timestamp;case_timestamp_to_date=case_timestamp_to_date"
);
/** sink */
CREATE
SINK STREAM cars_infos_out (cast_int_to_varchar varchar, cast_String_to_int

```



```
int, case_string_to_timestamp timestamp, case_timestamp_to_date date) WITH (
  type = "dis",
  region = "xxxxx",
  channel = "dis-output",
  partition_count = "1",
  encode = "json",
  offset = "4",
  json_config =
"cast_int_to_varchar=cast_int_to_varchar;cast_string_to_int=cast_string_to_int;case_string_to_timestamp=cas
e_string_to_timestamp;case_timestamp_to_date=case_timestamp_to_date",
  enable_output_null="true"
);
/** Statistics on static car information**/
INSERT
INTO
cars_infos_out
SELECT
cast(cast_int_to_varchar as varchar),
cast(cast_string_to_int as int),
cast(case_string_to_timestamp as timestamp),
cast(case_timestamp_to_date as date)
FROM
car_infos;
```

Returned data

```
{"case_string_to_timestamp":
1514736001000,"cast_int_to_varchar":"5","case_timestamp_to_date":"2018-01-01","cast_string_to_int":100}
```

3.1.11.5 Aggregate Functions

An aggregate function performs a calculation operation on a set of input values and returns a value. For example, the COUNT function counts the number of rows retrieved by an SQL statement. [Table 3-54](#) lists aggregate functions.

Sample data: Table T1

```
|score|
|81 |
|100 |
|60 |
|95 |
|86 |
```

Common Aggregate Functions

Table 3-54 Common aggregation functions

Function	Return Data Type	Description
COUNT(*)	BIGINT	Return count of tuples.
COUNT([ALL] expression...)	BIGINT	Returns the number of input rows for which the expression is not NULL. Use DISTINCT for a unique instance of each value.
AVG(numeric)	DOUBLE	Return average (arithmetic mean) of all input values.
SUM(numeric)	DOUBLE	Return the sum of all input numerical values.

Function	Return Data Type	Description
MAX(value)	DOUBLE	Return the maximum value of all input values.
MIN(value)	DOUBLE	Return the minimum value of all input values.
STDDEV_POP(value)	DOUBLE	Return the population standard deviation of all numeric fields of all input values.
STDDEV_SAMP(value)	DOUBLE	Return the sample standard deviation of all numeric fields of all input values.
VAR_POP(value)	DOUBLE	Return the population variance (square of population standard deviation) of numeral fields of all input values.
VAR_SAMP(value)	DOUBLE	Return the sample variance (square of the sample standard deviation) of numeric fields of all input values.

Example

- COUNT(*)
 - Test statement
SELECT COUNT(score) FROM T1;
 - Test data and results

Table 3-55 T1

Test Data (score)	Test Result
81	5
100	
60	
95	
86	

- COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)
 - Test statement
SELECT COUNT(DISTINCT content) FROM T1;
 - Test data and results

Table 3-56 T1

content (STRING)	Test Result
"hello1 "	2
"hello2 "	
"hello2"	
null	
86	

- AVG(numeric)
 - Test statement
`SELECT AVG(score) FROM T1;`
 - Test data and results

Table 3-57 T1

Test Data (score)	Test Result
81	84.0
100	
60	
95	
86	

- SUM(numeric)
 - Test statement
`SELECT SUM(score) FROM T1;`
 - Test data and results

Table 3-58 T1

Test Data (score)	Test Result
81	422.0
100	
60	
95	
86	

- MAX(value)
 - Test statement

```
SELECT MAX(score) FROM T1;
```

- Test data and results

Table 3-59 T1

Test Data (score)	Test Result
81	100.0
100	
60	
95	
86	

- MIN(value)

- Test statement

```
SELECT MIN(score) FROM T1;
```

- Test data and results

Table 3-60 T1

Test Data (score)	Test Result
81	60.0
100	
60	
95	
86	

- STDDEV_POP(value)

- Test statement

```
SELECT STDDEV_POP(score) FROM T1;
```

- Test data and results

Table 3-61 T1

Test Data (score)	Test Result
81	13.0
100	
60	
95	
86	

- STDDEV_SAMP(value)
 - Test statement
`SELECT STDDEV_SAMP(score) FROM T1;`
 - Test data and results

Table 3-62 T1

Test Data (score)	Test Result
81	15.0
100	
60	
95	
86	

- VAR_POP(value)
 - Test statement
`SELECT VAR_POP(score) FROM T1;`
 - Test data and results

Table 3-63 T1

Test Data (score)	Test Result
81	193.0
100	
60	
95	
86	

- VAR_SAMP(value)
 - Test statement
`SELECT VAR_SAMP(score) FROM T1;`
 - Test data and results

Table 3-64 T1

Test Data (score)	Test Result
81	241.0
100	
60	
95	

Test Data (score)	Test Result
86	

3.1.11.6 Table-Valued Functions

Table-valued functions can convert one row of records into multiple rows or convert one column of records into multiple columns. Table-valued functions can only be used in JOIN LATERAL TABLE.

Table 3-65 Table-valued functions

Function	Return Data Type	Description
split_cursor(value, delimiter)	cursor	Separates the "value" string into multiple rows of strings by using the delimiter.

Example

Input one record ("student1", "student2, student3") and output two records ("student1", "student2") and ("student1", "student3").

```
create source stream s1 (attr1 string, attr2 string) with (.....);
insert into s2 select attr1, b1 from s1 left join lateral table(split_cursor(attr2, ',')) as T(b1) on true;
```

3.1.11.7 Other Functions

Array Functions

Table 3-66 Array functions

Function	Return Data Type	Description
CARDINALITY(ARRAY)	INT	Return the element count of an array.
ELEMENT(ARRAY)	-	Return the sole element of an array with a single element. If the array contains no elements, null is returned. If the array contains multiple elements, an exception is reported.

Example:

The returned number of elements in the array is 3.

```
insert into temp select CARDINALITY(ARRAY[TRUE, TRUE, FALSE]) from source_stream;
```

HELLO WORLD is returned.

```
insert into temp select ELEMENT(ARRAY['HELLO WORLD']) from source_stream;
```

Attribute Access Functions

Table 3-67 Attribute access functions

Function	Return Data Type	Description
tableName.compositeType.field	-	Select a single field, use the name to access the field of Apache Flink composite types, such as Tuple and POJO, and return the value.
tableName.compositeType.*	-	Select all fields, and convert Apache Flink composite types, such as Tuple and POJO, and all their direct subtypes into a simple table. Each subtype is a separate field.

3.1.12 Geographical Functions

Function description

Table 3-68 describes the basic geospatial geometric elements.

Table 3-68 Basic geospatial geometric element table

Geospatial geometric elements	Description	Example Value
ST_POINT(latitude, longitude)	Indicates a geographical point, including the longitude and latitude.	ST_POINT(1.12012, 1.23401)
ST_LINE(array[point1...pointN])	Indicates a geographical line formed by connecting multiple geographical points (ST_POINT) in sequence. The line can be a polygonal line or a straight line.	ST_LINE(ARRAY[ST_POINT(1.12, 2.23), ST_POINT(1.13, 2.44), ST_POINT(1.13, 2.44)])
ST_POLYGON(array[point1...point1])	Indicates a geographical polygon, which is a closed polygon area formed by connecting multiple geographical points (ST_POINT) with the same start and end points in sequence.	ST_POLYGON(ARRAY[ST_POINT(1.0, 1.0), ST_POINT(2.0, 1.0), ST_POINT(2.0, 2.0), ST_POINT(1.0, 1.0)])

Geospatial geometric elements	Description	Example Value
ST_CIRCLE(point, radius)	Indicates a geographical circle that consists of ST_POINT and a radius.	ST_CIRCLE(ST_POINT(1.0, 1.0), 1.234)

You can build complex geospatial geometries based on basic geospatial geometric elements. [Table 3-69](#) describes the related transformation methods.

Table 3-69 Transformation methods for building complex geometric elements based on basic geospatial geometric elements

Transformation Method	Description	Example Value
ST_BUFFER(geometry, distance)	Creates a polygon that surrounds the geospatial geometric elements at a given distance. Generally, this function is used to build the road area of a certain width for yaw detection.	ST_BUFFER(ST_LINE(ARRAY[ST_POINT(1.12, 2.23), ST_POINT(1.13, 2.44), ST_POINT(1.13, 2.44)]),1.0)
ST_INTERSECTION(geometry, geometry)	Creates a polygon that delimits the overlapping area of two given geospatial geometric elements.	ST_INTERSECTION(ST_CIRCLE(ST_POINT(1.0, 1.0), 2.0), ST_CIRCLE(ST_POINT(3.0, 1.0), 1.234))
ST_ENVELOPE(geometry)	Creates the minimal rectangle polygon including the given geospatial geometric elements.	ST_ENVELOPE(ST_CIRCLE(ST_POINT(1.0, 1.0), 2.0))

DLI provides multiple functions used for performing operations on and determining locations of geospatial geometric elements. [Table 3-70](#) describes the SQL scalar functions.

Table 3-70 SQL scalar function table

Function	Return Type	Description
ST_DISTANCE(point_1, point_2)	DOUBLE	Calculates the Euclidean distance between the two geographical points. The following provides an example: Select ST_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input

Function	Return Type	Description
ST_GEODESIC_DISTANCE(point_1, point_2)	DOUBLE	Calculates the shortest distance along the surface between two geographical points. The following provides an example: Select ST_GEODESIC_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input
ST_PERIMETER(polygon)	DOUBLE	Calculates the circumference of a polygon. The following provides an example: Select ST_PERIMETER(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)])) FROM input
ST_AREA(polygon)	DOUBLE	Calculates the area of a polygon. The following provides an example: Select ST_AREA(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)])) FROM input
ST_OVERLAPS(polygon_1, polygon_2)	BOOLEAN	Checks whether one polygon overlaps with another. The following provides an example: SELECT ST_OVERLAPS(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_INTERSECT(line1, line2)	BOOLEAN	Checks whether two line segments, rather than the two straight lines where the two line segments are located, intersect each other. The following provides an example: SELECT ST_INTERSECT(ST_LINE(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12)]), ST_LINE(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23)])) FROM input

Function	Return Type	Description
ST_WITHIN(point, polygon)	BOOLEAN	Checks whether one point is contained inside a geometry (polygon or circle). The following provides an example: SELECT ST_WITHIN(ST_POINT(x11, y11), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_CONTAINS(polygon_1, polygon_2)	BOOLEAN	Checks whether the first geometry contains the second geometry. The following provides an example: SELECT ST_CONTAINS(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_COVERS(polygon_1, polygon_2)	BOOLEAN	Checks whether the first geometry covers the second geometry. This function is similar to ST_CONTAINS except the situation when judging the relationship between a polygon and the boundary line of polygon, for which ST_COVER returns TRUE and ST_CONTAINS returns FALSE. The following provides an example: SELECT ST_COVERS(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON([ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_DISJOINT(polygon_1, polygon_2)	BOOLEAN	Checks whether one polygon is disjoint (not overlapped) with the other polygon. The following provides an example: SELECT ST_DISJOINT(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input

The World Geodetic System 1984 (WGS84) is used as the reference coordinate system for geographical functions. Due to offsets, the GPS coordinates cannot be directly used in the Baidu Map (compliant with BD09) and the Google Map

(compliant with GCJ02). To implement switchover between different geographical coordinate systems, DLI provides a series of functions related to coordinate system conversion as well as functions related to conversion between geographical distances and the unit meter. For details, see [Table 3-71](#).

Table 3-71 Functions for geographical coordinate system conversion and distance-unit conversion

Function	Return Type	Description
WGS84_TO_BD09(geometry)	Geospatial geometric elements in the Baidu Map coordinate system	Converts the geospatial geometric elements in the GPS coordinate system into those in the Baidu Map coordinate system. The following provides an example: WGS84_TO_BD09(ST_CIRCLE(ST_POINT(x, y), r))
WGS84_TO_CJ02(geometry)	Geospatial geometric elements in the Google Map coordinate system	Converts the geospatial geometric elements in the GPS coordinate system into those in the Google Map coordinate system. The following provides an example: WGS84_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))
BD09_TO_WGS84(geometry)	Geospatial geometric elements in the GPS coordinate system	Converts the geospatial geometric elements in the Baidu Map coordinate system into those in the GPS coordinate system. The following provides an example: BD09_TO_WGS84(ST_CIRCLE(ST_POINT(x, y), r))
BD09_TO_CJ02(geometry)	Geospatial geometric elements in the Google Map coordinate system	Converts the geospatial geometric elements in the Baidu Map coordinate system into those in the Google Map coordinate system. The following provides an example: BD09_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))

Function	Return Type	Description
CJ02_TO_WGS84(geometry)	Geospatial geometric elements in the GPS coordinate system	Converts the geospatial geometric elements in the Google Map coordinate system into those in the GPS coordinate system. The following provides an example: CJ02_TO_WGS84(ST_CIRCLE(ST_POINT(x, y), r))
CJ02_TO_BD09(geometry)	Geospatial geometric elements in the Baidu Map coordinate system	Converts the geospatial geometric elements in the Google Map coordinate system into those in the Baidu Map coordinate system. The following provides an example: CJ02_TO_BD09(ST_CIRCLE(ST_POINT(x, y), r))
DEGREE_TO_METER(distance)	DOUBLE	Converts the distance value of the geographical function to a value in the unit of meter. In the following example, you calculate the circumference of a triangle in the unit of meter. DEGREE_TO_METER(ST_PERIMETER(ST_POLYGON(ARRAY[ST_POINT(x1,y1), ST_POINT(x2,y2), ST_POINT(x3,y3), ST_POINT(x1,y1)])))

Function	Return Type	Description
METER_TO_DEGREE(numerical_value)	DOUBLE	Convert the value in the unit of meter to the distance value that can be calculated using the geographical function. In the following example, you draw a circle which takes a specified geographical point as the center and has a radius of 1 km. ST_CIRCLE(ST_POINT(x,y), METER_TO_DEGREE(1000))

DLI also provides window-based SQL geographical aggregation functions specific for scenarios where SQL logic involves windows and aggregation. For details about the functions, see [Table 3-72](#).

Table 3-72 Time-related SQL geographical aggregation function table

Function	Description	Example Value
AGG_DISTANCE(point)	Distance aggregation function, which is used to calculate the total distance of all adjacent geographical points in the window.	SELECT AGG_DISTANCE(ST_POINT(x,y)) FROM input GROUP BY HOP(rowtime, INTERVAL '1' HOUR, INTERVAL '1' DAY)
AVG_SPEED(point)	Average speed aggregation function, which is used to calculate the average speed of moving tracks formed by all geographical points in a window. The average speed is in the unit of m/s.	SELECT AVG_SPEED(ST_POINT(x,y)) FROM input GROUP BY TUMBLE(proctime, INTERVAL '1' DAY)

Precautions

None

Example

Example of yaw detection:

```
INSERT INTO yaw_warning
SELECT "The car is yawing"
FROM driver_behavior
WHERE NOT ST_WITHIN(ST_POINT(cast(Longitude as DOUBLE), cast(Latitude as DOUBLE)),
ST_BUFFER(ST_LINE(ARRAY[ST_POINT(34.585555,105.725221),ST_POINT(34.586729,105.735974),ST_POINT(
34.586492,105.740538),ST_POINT(34.586388,105.741651),ST_POINT(34.586135,105.748712),ST_POINT(34.5
88691,105.74997)]),0.001));
```

IP Functions

 **NOTE**

Currently, only IPv4 addresses are supported.

Table 3-73 IP functions

Function	Return Type	Description
IP_TO_COUNTRY	STRING	Obtains the name of the country where the IP address is located.
IP_TO_PROVINCE	STRING	Obtains the province where the IP address is located. Usage: <ul style="list-style-type: none"> IP_TO_PROVINCE(STRING ip): Determines the province where the IP address is located and returns the province name. IP_TO_PROVINCE(STRING ip, STRING lang): Determines the province where the IP is located and returns the province name of the specified language. NOTE <ul style="list-style-type: none"> If the province where the IP address is located cannot be obtained through IP address parsing, the country where the IP address is located is returned. If the IP address cannot be parsed, Unknown is returned. The name returned by the function for the province is the short name.
IP_TO_CITY	STRING	Obtains the name of the city where the IP address is located. NOTE If the city where the IP address is located cannot be obtained through IP address parsing, the province or the country where the IP address is located is returned. If the IP address cannot be parsed, Unknown is returned.

Function	Return Type	Description
IP_TO_CITY_GEO	STRING	Obtains the longitude and latitude of the city where the IP address is located. The parameter value is in the following format: <i>Latitude, Longitude</i> . Usage: IP_TO_CITY_GEO(String ip): Returns the longitude and latitude of the city where the IP address is located.

3.1.13 Configuring Time Models

Flink provides two time models: processing time and event time.

DLI allows you to specify the time model during creation of the source stream and temporary stream.

Configuring Processing Time

Processing time refers to the system time, which is irrelevant to the data timestamp.

Syntax

```
CREATE SOURCE STREAM stream_name(...) WITH (...)
TIMESTAMP BY proctime.proctime;
CREATE TEMP STREAM stream_name(...)
TIMESTAMP BY proctime.proctime;
```

Description

To set the processing time, you only need to add `proctime.proctime` following `TIMESTAMP BY`. You can directly use the `proctime` field later.

Precautions

None

Example

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* Student ID */
  student_name STRING, /* Name */
  subject STRING, /* Subject */
  score INT /* Score */
)
WITH (
  type = "dis",
  region = "",
  channel = "dliinput",
  partition_count = "1",
  encode = "csv",
  field_delimiter=","
)TIMESTAMP BY proctime.proctime;
```

```
INSERT INTO score_greate_90
SELECT student_name, sum(score) over (order by proctime RANGE UNBOUNDED PRECEDING)
FROM student_scores;
```

Configuring Event Time

Event Time refers to the time when an event is generated, that is, the timestamp generated during data generation.

Syntax

```
CREATE SOURCE STREAM stream_name(...) WITH (...)
TIMESTAMP BY {attr_name}.rowtime
SET WATERMARK (RANGE {time_interval} | ROWS {literal}, {time_interval});
```

Description

To set the event time, you need to select a certain attribute in the stream as the timestamp and set the watermark policy.

Out-of-order events or late events may occur due to network faults. The watermark must be configured to trigger the window for calculation after waiting for a certain period of time. Watermarks are mainly used to process out-of-order data before generated events are sent to DLI during stream processing.

The following two watermark policies are available:

- By time interval
SET WATERMARK(range interval {time_unit}, interval {time_unit})
- By event quantity
SET WATERMARK(rows literal, interval {time_unit})

NOTE

Parameters are separated by commas (,). The first parameter indicates the watermark sending interval and the second indicates the maximum event delay.

Precautions

None

Example

- Send a watermark every 10s the **time2** event is generated. The maximum event latency is 20s.

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* Student ID */
  student_name STRING, /* Name */
  subject STRING, /* Subject */
  score INT, /* Score */
  time2 TIMESTAMP
)
WITH (
  type = "dis",
  region = "",
  channel = "dliinput",
  partition_count = "1",
  encode = "csv",
  field_delimiter=","
)
TIMESTAMP BY time2.rowtime
SET WATERMARK (RANGE interval 10 second, interval 20 second);

INSERT INTO score_greate_90
```



```
SELECT student_name, sum(score) over (order by time2 RANGE UNBOUNDED PRECEDING)
FROM student_scores;
```

- Send the watermark every time when 10 pieces of data are received, and the maximum event latency is 20s.

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* Student ID */
  student_name STRING, /* Name */
  subject STRING, /* Subject */
  score INT, /* Score */
  time2 TIMESTAMP
)
WITH (
  type = "dis",
  region = "",
  channel = "dliinput",
  partition_count = "1",
  encode = "csv",
  field_delimiter=","
)
TIMESTAMP BY time2.rowtime
SET WATERMARK (ROWS 10, interval 20 second);

INSERT INTO score_greate_90
SELECT student_name, sum(score) over (order by time2 RANGE UNBOUNDED PRECEDING)
FROM student_scores;
```

3.1.14 Pattern Matching

Complex event processing (CEP) is used to detect complex patterns in endless data streams so as to identify and search patterns in various data rows. Pattern matching is a powerful aid to complex event handling.

CEP is used in a collection of event-driven business processes, such as abnormal behavior detection in secure applications and the pattern of searching for prices, transaction volume, and other behavior in financial applications. It also applies to fraud detection and sensor data analysis.

Syntax

```
MATCH_RECOGNIZE (
  [ PARTITION BY expression [, expression ]* ]
  [ ORDER BY orderItem [, orderItem ]* ]
  [ MEASURES measureColumn [, measureColumn ]* ]
  [ ONE ROW PER MATCH | ALL ROWS PER MATCH ]
  [ AFTER MATCH
    ( SKIP TO NEXT ROW
      | SKIP PAST LAST ROW
      | SKIP TO FIRST variable
      | SKIP TO LAST variable
      | SKIP TO variable )
  ]
  PATTERN ( pattern )
  [ WITHIN intervalLiteral ]
  DEFINE variable AS condition [, variable AS condition ]*
) MR
```

 **NOTE**

Pattern matching in SQL is performed using the MATCH_RECOGNIZE clause. MATCH_RECOGNIZE enables you to do the following tasks:

- Logically partition and order the data that is used in the MATCH_RECOGNIZE clause with its PARTITION BY and ORDER BY clauses.
- Define patterns of rows to seek using the PATTERN clause of the MATCH_RECOGNIZE clause. These patterns use regular expression syntax.
- Specify the logical conditions required to map a row to a row pattern variable in the DEFINE clause.
- Define measures, which are expressions usable in other parts of the SQL query, in the MEASURES clause.

Syntax description

Table 3-74 Syntax description

Parameter	Mandatory	Description
PARTITION BY	No	Logically divides the rows into groups.
ORDER BY	No	Logically orders the rows in a partition.
[ONE ROW ALL ROWS] PER MATCH	No	<p>Chooses summaries or details for each match.</p> <ul style="list-style-type: none"> • ONE ROW PER MATCH: Each match produces one summary row. • ALL ROWS PER MATCH: A match spanning multiple rows will produce one output row for each row in the match. <p>The following provides an example:</p> <pre>SELECT * FROM MyTable MATCH_RECOGNIZE (MEASURES AVG(B.id) as Bid ALL ROWS PER MATCH PATTERN (A B C) DEFINE A AS A.name = 'a', B AS B.name = 'b', C AS C.name = 'c') MR</pre> <p>Example description</p> <p>Assume that the format of MyTable is (id, name) and there are three data records: (1, a), (2, b), and (3, c).</p> <p>ONE ROW PER MATCH outputs the average value 2 of B.</p> <p>ALL ROWS PER MATCH outputs each record and the average value of B, specifically, (1,a, null), (2,b,2), (3,c,2).</p>
MEASURES	No	Defines calculations for export from the pattern matching.

Parameter	Mandatory	Description
PATTERN	Yes	<p>Defines the row pattern that will be matched.</p> <ul style="list-style-type: none"> • PATTERN (A B C) indicates to detect concatenated events A, B, and C. • PATTERN (A B) indicates to detect A or B. • Modifiers <ul style="list-style-type: none"> - *: 0 or more iterations. For example, A* indicates to match A for 0 or more times. - +: 1 or more iterations. For example, A+ indicates to match A for 1 or more times. - ?: 0 or 1 iteration. For example, A? indicates to match A for 0 times or once. - {n}: n iterations (n > 0). For example, A{5} indicates to match A for five times. - {n,}: n or more iterations (n ≥ 0). For example, A{5,} indicates to match A for five or more times. - {n, m}: between n and m (inclusive) iterations (0 ≤ n ≤ m, 0 < m). For example, A{3,6} indicates to match A for 3 to 6 times. - {, m}: between 0 and m (inclusive) iterations (m > 0). For example, A{,4} indicates to match A for 0 to 4 times.
DEFINE	Yes	Defines primary pattern variables.
AFTER MATCH SKIP	No	<p>Defines where to restart the matching process after a match is found.</p> <ul style="list-style-type: none"> • SKIP TO NEXT ROW: Resumes pattern matching at the row after the first row of the current match. • SKIP PAST LAST ROW: Resumes pattern matching at the next row after the last row of the current match. • SKIP TO FIRST variable: Resumes pattern matching at the first row that is mapped to the pattern variable. • SKIP TO LAST variable: Resumes pattern matching at the last row that is mapped to the pattern variable. • SKIP TO variable: Same as SKIP TO LAST variable.

Functions Supported by CEP

Table 3-75 Function description

Function	Description
MATCH_NUMBER()	Finds which rows are in which match. It can be used in the MEASURES and DEFINE clauses.
CLASSIFIER()	Finds which pattern variable applies to which rows. It can be used in the MEASURES and DEFINE clauses.
FIRST()/LAST()	FIRST returns the value of an expression evaluated in the first row of the group of rows mapped to a pattern variable. LAST returns the value of an expression evaluated in the last row of the group of rows mapped to a pattern variable. In PATTERN (A B+ C), FIRST (B.id) indicates the ID of the first B in the match, and LAST (B.id) indicates the ID of the last B in the match.
NEXT()/PREV()	Relative offset, which can be used in DEFINE. For example, PATTERN (A B+) DEFINE B AS B.price > PREV(B.price)
RUNNING/ FINAL	RUNNING indicates to match the middle value, while FINAL indicates to match the final result value. Generally, RUNNING/FINAL is valid only in ALL ROWS PER MATCH. For example, if there are three records (a, 2), (b, 6), and (c, 12), then the values of RUNNING AVG (A.price) and FINAL AVG (A.price) are (2,6), (4,6), (6,6).
Aggregate functions (COUNT, SUM, AVG, MAX, MIN)	Aggregation operations. These functions can be used in the MEASURES and DEFINE clauses. For details, see Aggregate Functions .

Example

- Fake plate vehicle detection

CEP conducts pattern matching based on license plate switchover features on the data of vehicles collected by cameras installed on urban roads or high-speed roads in different areas within 5 minutes.

```
INSERT INTO fake_licensed_car
SELECT * FROM camera_license_data MATCH_RECOGNIZE
(
  PARTITION BY car_license_number
  ORDER BY proctime
  MEASURES A.car_license_number as car_license_number, A.camera_zone_number as first_zone,
  B.camera_zone_number as second_zone
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST C
  PATTERN (A B+ C)
  WITHIN interval '5' minute
  DEFINE
  B AS B.camera_zone_number <> A.camera_zone_number,
```

```
C AS C.camera_zone_number = A.camera_zone_number  
) MR;
```

According to this rule, if a vehicle of a license plate number drives from area A to area B but another vehicle of the same license plate number is detected in area A within 5 minutes, then the vehicle in area A is considered to carry a fake license plate.

Input data:

```
Zhejiang B88888, zone_A  
Zhejiang AZ626M, zone_A  
Zhejiang B88888, zone_A  
Zhejiang AZ626M, zone_A  
Zhejiang AZ626M, zone_A  
Zhejiang B88888, zone_B  
Zhejiang B88888, zone_B  
Zhejiang AZ626M, zone_B  
Zhejiang AZ626M, zone_B  
Zhejiang AZ626M, zone_C  
Zhejiang B88888, zone_A  
Zhejiang B88888, zone_A
```

The output is as follows:

```
Zhejiang B88888, zone_A, zone_B
```

3.1.15 StreamingML

3.1.15.1 Anomaly Detection

Anomaly detection applies to various scenarios, including intrusion detection, financial fraud detection, sensor data monitoring, medical diagnosis, natural data detection, and more. The typical algorithms for anomaly detection include the statistical modeling method, distance-based calculation method, linear model, and nonlinear model.

DLI uses an anomaly detection method based on the random forest, which has the following characteristics:

- The one-pass algorithm is used with $O(1)$ amortized time complexity and $O(1)$ space complexity.
- The random forest structure is constructed only once. The model update operation only updates the node data distribution values.
- The node stores data distribution information of multiple windows, and the algorithm can detect data distribution changes.
- Anomaly detection and model updates are completed in the same code framework.

Syntax

```
SRF_UNSUP(ARRAY[Field 1, Field 2, ...], 'Optional parameter list')
```

 NOTE

- The anomaly score returned by the function is a DOUBLE value in the range of [0, 1].
- The field names must be of the same type. If the field types are different, you can use the CAST function to escape the field names, for example, [a, CAST(b as DOUBLE)].
- The syntax of the optional parameter list is as follows: "key1=value,key2=value2,..."

Parameter Description

Table 3-76 Parameter Description

Parameter	Mandatory	Description	Default Value
transientThreshold	No	Threshold for which the histogram change is indicating a change in the data.	5
numTrees	No	Number of trees composing the random forest.	15
maxLeafCount	No	Maximum number of leaf nodes one tree can have.	15
maxTreeHeight	No	Maximum height of the tree.	12
seed	No	Random seed value used by the algorithm.	4010
numClusters	No	Number of types of data to be detected. By default, the following two data types are available: anomalous and normal data.	2
dataViewMode	No	Algorithm learning mode. <ul style="list-style-type: none"> • Value history indicates that all historical data is considered. • Value horizon indicates that only historical data of a recent time period (typically a size of 4 windows) is considered. 	history

Example

Anomaly detection is conducted on the **c** field in data stream **MyTable**. If the anomaly score is greater than 0.8, then the detection result is considered to be anomaly.

```
SELECT c,
CASE WHEN SRF_UNSUP(ARRAY[c], "numTrees=15,seed=4010") OVER (ORDER BY proctime RANGE
BETWEEN INTERVAL '99' SECOND PRECEDING AND CURRENT ROW) > 0.8
THEN 'anomaly'
ELSE 'not anomaly'
```

```
END
FROM MyTable
```

3.1.15.2 Time Series Forecasting

Modeling and forecasting time series is a common task in many business verticals. Modeling is used to extract meaningful statistics and other characteristics of the data. Forecasting is the use of a model to predict future data. DLI provides a series of stochastic linear models to help users conduct online modeling and forecasting in real time.

ARIMA (Non-Seasonal)

Auto-Regressive Integrated Moving Average (ARIMA) is a classical model used for time series forecasting and is closely correlated with the AR, MA, and ARMA models.

- The AR, MA, and ARMA models are applicable to **stationary** sequences.
 - AR(p) is an autoregressive model. An AR(p) is a linear combination of p consecutive values from immediate past. The model can predict the next value by using the weight of linear combination.
 - MA(q) is a moving average model. An MA(q) is a linear combination of q white noise values from the past plus the average value. The model can also predict the next value by using the weight of linear combination.
 - ARMA(p, q) is an autoregressive moving average model, which integrates the advantages of both AR and MA models. In the ARMA model, the autoregressive process is responsible for quantizing the relationship between the current data and the previous data, and the moving average process is responsible for solving problems of random variables. Therefore, the ARMA model is more effective than AR/MA.
- ARIMA is suitable for **non-stationary** series. In ARIMA(p, q, d), **p** indicates the autoregressive order, **q** indicates the moving average order, and **d** indicates the difference order.

Syntax

```
AR_PRED(field, degree): Use the AR model to forecast new data.
AR_COEF(field, degree): Return the weight of the AR model.
ARMA_PRED(field, degree): Use the ARMA model to forecast new data.
ARMA_COEF(field, degree): Return the weight of the ARMA model.
ARIMA_PRED(field, degree, derivativeOrder): Use ARIMA to forecast new data.
```

Table 3-77 Parameter Description

Parameter	Mandatory	Description	Default Value
field	Yes	Name of the field, data in which is used for prediction, in the data stream.	-
degree	No	Defines how many steps in the past are going to be considered for the next prediction. Currently, only "p = q = degree" is allowed.	5

Parameter	Mandatory	Description	Default Value
derivativeOrder	No	Derivative order. Generally, this parameter is set to 1 or 2 .	1

Example

Separately use AR, ARMA, and ARIMA to forecast the time series ordered by rowtime.

```
SELECT b,
  AR_PRED(b) OVER (ORDER BY rowtime ROWS BETWEEN 5 PRECEDING AND CURRENT ROW) AS ar,
  ARMA_PRED(b) OVER (ORDER BY rowtime ROWS BETWEEN 5 PRECEDING AND CURRENT ROW) AS
arma,
  ARIMA_PRED(b) OVER (ORDER BY rowtime ROWS BETWEEN 5 PRECEDING AND CURRENT ROW) AS
arima
FROM MyTable
```

Holt Winters

The Holt-Winters algorithm is one of the Exponential smoothing methods used to forecast **seasonal** data in time series.

Syntax

```
HOLT_WINTERS(field, seasonality, forecastOrder)
```

Table 3-78 Parameter Description

Parameter	Mandatory	Description
field	Yes	Name of the field, data in which is used for prediction, in the data stream.
seasonality	Yes	Seasonality space used to perform the prediction. For example, if data samples are collected daily, and the season space to consider is a week, then seasonality is 7 .
forecastOrder	No	Value to be forecast, specifically, the number of steps to be considered in the future for producing the forecast. If forecastOrder is set to 1 , the algorithm forecasts the next value. If forecastOrder is set to 2 , the algorithm forecasts the value of 2 steps ahead in the future. The default value is 1 . When using this parameter, ensure that the OVER window size is greater than the value of this parameter.

Example

Use Holt-Winters to forecast time series ordered by rowtime.

```
SELECT b,  
       HOLT_WINTERS(b, 5) OVER (ORDER BY rowtime ROWS BETWEEN 5 PRECEDING AND CURRENT ROW)  
AS a1,  
       HOLT_WINTERS(b, 5, 2) OVER (ORDER BY rowtime ROWS BETWEEN 5 PRECEDING AND CURRENT  
ROW) AS a2  
FROM MyTable
```

3.1.15.3 Real-Time Clustering

Clustering algorithms belong to unsupervised algorithms. K-Means, a clustering algorithm, partitions data points into related clusters by calculating the distance between data points based on the predefined cluster quantity. For offline static datasets, we can determine the clusters based on field knowledge and run K-Means to achieve a better clustering effect. However, online real-time streaming data is always changing and evolving, and the cluster quantity is likely to change. To address clustering issues on online real-time streaming data, DLI provides a low-delay online clustering algorithm that does not require predefined cluster quantity.

The algorithm works as follows: Given a distance function, if the distance between two data points is less than a threshold, both data points will be partitioned into the same cluster. If the distances between a data point and the central data points in several cluster centers are less than the threshold, then related clusters will be merged. When data in a data stream arrives, the algorithm computes the distances between each data point and the central data points of all clusters to determine whether the data point can be partitioned into to an existing or new cluster.

Syntax

CENTROID(ARRAY[field_names], distance_threshold): Compute the centroid of the cluster where the current data point is assigned.
CLUSTER_CENTROIDS(ARRAY[field_names], distance_threshold): Compute all centroids after the data point is assigned.
ALL_POINTS_OF_CLUSTER(ARRAY[field_names], distance_threshold): Compute all data points in the cluster where the current data point is assigned.
ALL_CLUSTERS_POINTS(ARRAY[field_names], distance_threshold): Computers all data points in each cluster after the current data point is assigned.

NOTE

- Clustering algorithms can be applied in **unbounded streams**.

Parameter Description

Table 3-79 Parameter Description

Parameter	Mandatory	Description
field_names	Yes	Name of the field where the data is located in the data stream. Multiple fields are separated by commas (,). For example, ARRAY[a, b, c] .
distance_threshold	Yes	Distance threshold. When the distance between two data points is less than the threshold, both data points are placed in the same cluster.

Example

Use four functions to compute information related to clusters over windows.

```
SELECT
  CENTROID(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE UNBOUNDED PRECEDING) AS centroid,
  CLUSTER_CENTROIDS(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE UNBOUNDED PRECEDING) AS
  centroids
FROM MyTable

SELECT
  CENTROID(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE BETWEEN INTERVAL '60' MINUTE
  PRECEDING AND CURRENT ROW) AS centroidCE,
  ALL_POINTS_OF_CLUSTER(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE BETWEEN INTERVAL '60'
  MINUTE PRECEDING AND CURRENT ROW) AS itemList,
  ALL_CLUSTERS_POINTS(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE BETWEEN INTERVAL '60'
  MINUTE PRECEDING AND CURRENT ROW) AS listoflistofpoints
FROM MyTable
```

3.1.15.4 Deep Learning Model Prediction

Deep learning has a wide range of applications in many industries, such as image classification, image recognition, and speech recognition. DLI provides several functions to load deep learning models for prediction.

Currently, models DeepLearning4j and Keras are supported. In Keras, TensorFlow, CNTK, or Theano can serve as the backend engine. With importing of the neural network model from Keras, models of mainstream learning frameworks such as Theano, TensorFlow, Caffe, and CNTK can be imported.

Syntax

```
-- Image classification: returns the predicted category IDs used for image classification.
DL_IMAGE_MAX_PREDICTION_INDEX(field_name, model_path, is_dl4j_model)
DL_IMAGE_MAX_PREDICTION_INDEX(field_name, keras_model_config_path, keras_weights_path) --
Suitable for the Keras model

--Text classification: returns the predicted category IDs used for text classification.
DL_TEXT_MAX_PREDICTION_INDEX(field_name, model_path, is_dl4j_model) -- Use the default word2vec
model.
DL_TEXT_MAX_PREDICTION_INDEX(field_name, word2vec_path, model_path, is_dl4j_model)
```

 NOTE

Models and configuration files must be stored on OBS. The path format is `obs://your_ak:your_sk@obs.your_obs_region.xxx.com:443/your_model_path`. For example, if your model is stored on OBS, the bucket name is `dl_model`, and the file name is `model.h5`, set the path to `obs://your_ak:your_sk@obs.xxx.com:443/dl_model/model.h5`.

Parameter Description

Table 3-80 Parameter description

Parameter	Man dato ry	Description
field_name	Yes	Name of the field, data in which is used for prediction, in the data stream. In image classification, this parameter needs to declare ARRAY[TINYINT]. In image classification, this parameter needs to declare String.
model_path	Yes	Complete save path of the model on OBS, including the model structure and model weight.
is_dl4j_model	Yes	Whether the model is a Deeplearning4j model Value true indicates that the model is a Deeplearning4j model, while value false indicates that the model is a Keras model.
keras_model_config_path	Yes	Complete save path of the model structure on OBS. In Keras, you can obtain the model structure by using model.to_json() .
keras_weights_path	Yes	Complete save path of the model weight on OBS. In Keras, you can obtain the model weight by using model.save_weights(filepath) .
word2vec_path	Yes	Complete save path of the word2vec model on OBS.

Example

For prediction in image classification, use the Mnist dataset as the input and load the pre-trained Deeplearning4j model or Keras model to predict the digit representing each image in real time.

```
CREATE SOURCE STREAM Mnist(
  image Array[TINYINT]
)
SELECT DL_IMAGE_MAX_PREDICTION_INDEX(image, 'your_dl4j_model_path', false) FROM Mnist
SELECT DL_IMAGE_MAX_PREDICTION_INDEX(image, 'your_keras_model_path', true) FROM Mnist
SELECT DL_IMAGE_MAX_PREDICTION_INDEX(image, 'your_keras_model_config_path', 'keras_weights_path')
FROM Mnist
```

For prediction in text classification, use data of a group of news titles as the input and load the pre-trained Deeplearning4j model or Keras model to predict the category of each news title in real time, such as economy, sports, and entertainment.

```
CREATE SOURCE STREAM News(  
  title String  
)  
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title, 'your_dl4j_word2vec_model_path', 'your_dl4j_model_path',  
false) FROM News  
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title,  
'your_keras_word2vec_model_path', 'your_keras_model_path', true) FROM News  
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title, 'your_dl4j_model_path', false) FROM New  
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title, 'your_keras_model_path', true) FROM New
```

3.1.16 Reserved Keywords

Flink SQL reserves some strings as keywords. If you want to use the following character strings as field names, ensure that they are enclosed by back quotes, for example, `value` and `count`.

A

- A
- ABS
- ABSOLUTE
- ACTION
- ADA
- ADD
- ADMIN
- AFTER
- AK
- ALL
- ALLOCATE
- ALLOW
- ALTER
- ALWAYS
- AND
- ANY
- APPEND
- APP_ID
- ARE
- ARRAY
- ARRAY_BRACKET
- AS
- ASC
- ASENSITIVE
- ASSERTION

- ASSIGNMENT
- ASYMMETRIC
- AT
- AT_LEAST_ONCE
- ATOMIC
- ATTRIBUTE
- ATTRIBUTES
- AUTHORIZATION
- AVG
- AVRO_CONFIG
- AVRO_DATA
- AVRO_SCHEMA

B

- BATCH_INSERT_DATA_NUM
- BEFORE
- BEGIN
- BERNOULLI
- BETWEEN
- BIGINT
- BINARY
- BIT
- BLOB
- BOOL
- BOOLEAN
- BOTH
- BREADTH
- BUCKET
- BY

C

- C
- CACHE_MAX_NUM
- CACHE_TIME
- CALL
- CALLED
- CARDINALITY
- CASCADE
- CASCADED
- CASE
- CAST

- CATALOG
- CATALOG_NAME
- CEIL
- CEILING
- CENTURY
- CHAIN
- CHANNEL
- CHAR
- CHARACTER
- CHARACTERISTICS
- CHARACTERS
- CHARACTER_LENGTH
- CHARACTER_SET_CATALOG
- CHARACTER_SET_NAME
- CHARACTER_SET_SCHEMA
- CHAR_LENGTH
- CHECK
- CHECKPOINT_APP_NAME
- CHECKPOINT_INTERVAL
- CHECKPOINTINTERVAL
- CLASS_ORIGIN
- CLOB
- CLOSE
- CLUSTER_ADDRESS
- CLUSTER_ID
- CLUSTER_NAME
- COALESCE
- COBOL
- COLLATE
- COLLATION
- COLLATION_CATALOG
- COLLATION_NAME
- COLLATION_SCHEMA
- COLLECT
- COLUMN
- COLUMN_NAME
- COLUMN_NAME_MAP
- COMMAND_FUNCTION
- COMMAND_FUNCTION_CODE
- COMMIT

- COMMITTED
- CONDITION
- CONDITION_NUMBER
- CONFIGURATION
- CONFLUENT_CERTIFICATE_NAME
- CONFLUENT_PROPERTIES
- CONFLUENT_SCHEMA_FIELD
- CONFLUENT_URL
- CONNECT
- CONNECTION_NAME
- CONSTRAINT
- CONSTRAINTS
- CONSTRAINT_CATALOG
- CONSTRAINT_NAME
- CONSTRAINT_SCHEMA
- CONSTRUCTOR
- CONTAINS
- CONTINUE
- CONVERT
- CORR
- CORRESPONDING
- COUNT
- COVAR_POP
- COVAR_SAMP
- CREATE
- CREATE_IF_NOT_EXIST
- CROSS
- CUBE
- CUME_DIST
- CURRENT
- CURRENT_CATALOG
- CURRENT_DATE
- CURRENT_DEFAULT_TRANSFORM_GROUP
- CURRENT_PATH
- CURRENT_ROLE
- CURRENT_SCHEMA
- CURRENT_TIMESTAMP
- CURRENT_TRANSFORM_GROUP_FOR_TYPE
- CURRENT_USER
- CURSOR

- CURSOR_NAME
- CYCLE

D

- DATE
- DATABASE
- DATE
- DATETIME_INTERVAL_CODE
- DATETIME_INTERVAL_PRECISION
- DAY
- DB_COLUMNS
- DB_URL
- DB_OBS_SERVER
- DB_TYPE
- DEALLOCATE
- DEC
- DECADE
- DECIMAL
- DECLARE
- DEFAULTS
- DEFERRABLE
- DEFERRED
- DEFINER
- DEGREE
- DELETE
- DELETE_OBS_TEMP_FILE
- DENSE_RANK
- DEPTH
- Deref
- DERIVED
- DESC
- DESCRIBE
- DESCRIPTION
- DESCRIPTOR
- DETERMINISTIC
- DIAGNOSTICS
- DISALLOW
- DISCONNECT
- DIS_NOTICE_CHANNEL
- DISPATCH

- DISTINCT
- DOMAIN
- DOUBLE
- DOW
- DOY
- DRIVER
- DROP
- DUMP_INTERVAL
- DYNAMIC
- DYNAMIC_FUNCTION
- DYNAMIC_FUNCTION_CODE

E

- EACH
- ELEMENT
- ELSE
- EMAIL_KEY
- ENABLECHECKPOINT
- ENABLE_CHECKPOINT
- ENABLE_OUTPUT_NULL
- ENCODE
- ENCODE_CLASS_NAME
- ENCODE_CLASS_PARAMETER
- ENCODED_DATA
- END
- ENDPOINT
- END_EXEC
- EPOCH
- EQUALS
- ESCAPE
- ES_FIELDS
- ES_INDEX
- ES_TYPE
- ESTIMATEMEM
- ESTIMATEPARALLELISM
- EXACTLY_ONCE
- EXCEPT
- EXCEPTION
- EXCLUDE
- EXCLUDING

- EXEC
- EXECUTE
- EXISTS
- EXP
- EXPLAIN
- EXTEND
- EXTERNAL
- EXTRACT
- EVERY

F

- FALSE
- FETCH
- FIELD_DELIMITER
- FIELD_NAMES
- FILE_PREFIX
- FILTER
- FINAL
- FIRST
- FIRST_VALUE
- FLOAT
- FLOOR
- FOLLOWING
- FOR
- FUNCTION
- FOREIGN
- FORTRAN
- FOUND
- FRAC_SECOND
- FREE
- FROM
- FULL
- FUSION

G

- G
- GENERAL
- GENERATED
- GET
- GLOBAL
- GO

- GOTO
- GRANT
- GRANTED
- GROUP
- GROUPING
- GW_URL

H

- HASH_KEY_COLUMN
- HAVING
- HIERARCHY
- HOLD
- HOUR
- HTTPS_PORT

I

- IDENTITY
- ILLEGAL_DATA_TABLE
- IMMEDIATE
- IMPLEMENTATION
- IMPORT
- IN
- INCLUDING
- INCREMENT
- INDICATOR
- INITIALLY
- INNER
- INOUT
- INPUT
- INSENSITIVE
- INSERT
- INSTANCE
- INSTANTIABLE
- INT
- INTEGER
- INTERSECT
- INTERSECTION
- INTERVAL
- INTO
- INVOKER
- IN_WITH_SCHEMA

- IS
- ISOLATION

J

- JAVA
- JOIN
- JSON_CONFIG
- JSON_SCHEMA

K

- K
- KAFKA_BOOTSTRAP_SERVERS
- KAFKA_CERTIFICATE_NAME
- KAFKA_GROUP_ID
- KAFKA_PROPERTIES
- KAFKA_PROPERTIES_DELIMITER
- KAFKA_TOPIC
- KEY
- KEY_COLUMN
- KEY_MEMBER
- KEY_TYPE
- KEY_VALUE
- KRB_AUTH

L

- LABEL
- LANGUAGE
- LARGE
- LAST
- LAST_VALUE
- LATERAL
- LEADING
- LEFT
- LENGTH
- LEVEL
- LIBRARY
- LIKE
- LIMIT
- LONG

M

- M
- MAP
- MATCH
- MATCHED
- MATCHING_COLUMNS
- MATCHING_REGEX
- MAX
- MAXALLOWEDCPU
- MAXALLOWEDMEM
- MAXALLOWEDPARALLELISM
- MAX_DUMP_FILE_NUM
- MAX_RECORD_NUM_CACHE
- MAX_RECORD_NUM_PER_FILE
- MAXVALUE
- MEMBER
- MERGE
- MESSAGE_COLUMN
- MESSAGE_LENGTH
- MESSAGE_OCTET_LENGTH
- MESSAGE_SUBJECT
- MESSAGE_TEXT
- METHOD
- MICROSECOND
- MILLENNIUM
- MIN
- MINUTE
- MINVALUE
- MOD
- MODIFIES
- MODULE
- MONTH
- MORE
- MS
- MULTISSET
- MUMPS

N

- NAME
- NAMES

- NATIONAL
- NATURAL
- NCHAR
- NCLOB
- NESTING
- NEW
- NEXT
- NO
- NONE
- NORMALIZE
- NORMALIZED
- NOT
- NULL
- NULLABLE
- NULLIF
- NULLS
- NUMBER
- NUMERIC

O

- OBJECT
- OBJECT_NAME
- OBS_DIR
- OCTETS
- OCTET_LENGTH
- OF
- OFFSET
- OLD
- ON
- ONLY
- OPEN
- OPERATION_FIELD
- OPTION
- OPTIONS
- OR
- ORDER
- ORDERING
- ORDINALITY
- OTHERS
- OUT

- OUTER
- OUTPUT
- OVER
- OVERLAPS
- OVERLAY
- OVERRIDING

P

- PAD
- PARALLELISM
- PARAMETER
- PARAMETER_MODE
- PARAMETER_NAME
- PARAMETER_ORDINAL_POSITION
- PARAMETER_SPECIFIC_CATALOG
- PARAMETER_SPECIFIC_NAME
- PARAMETER_SPECIFIC_SCHEMA
- PARTIAL
- PARTITION
- PARTITION_COUNT
- PARTITION_KEY
- PARTITION_RANGE
- PASCAL
- PASSTHROUGH
- PASSWORD
- PATH
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- PERSIST_SCHEMA
- PIPELINE_ID
- PLACING
- PLAN
- PLI
- POSITION
- POWER
- PRECEDING
- PRECISION
- PREPARE
- PRESERVE

- PRIMARY
- PRIMARY_KEY
- PRIOR
- PRIVILEGES
- PROCEDURE
- PROCTIME
- PROJECT_ID
- PUBLIC

Q

- QUARTER
- QUOTE

R

- RANGE
- RANK
- RAW
- READ
- READS
- READ_ONCE
- REAL
- RECURSIVE
- REF
- REFERENCES
- REFERENCING
- REGION
- REGR_AVGX
- REGR_AVGY
- REGR_COUNT
- REGR_INTERCEPT
- REGR_R2
- REGR_SLOPE
- REGR_SXX
- REGR_SXY
- REGR_SYY
- RELATIVE
- RELEASE
- REPEATABLE
- RESET
- RESTART
- RESTRICT

- RESULT
- RETURN
- RETURNED_CARDINALITY
- RETURNED_LENGTH
- RETURNED_OCTET_LENGTH
- RETURNED_SQLSTATE
- RETURNS
- REVOKE
- RIGHT
- ROLE
- ROLLBACK
- ROLLING_INTERVAL
- ROLLING_SIZE
- ROLLUP
- ROUTINE
- ROUTINE_CATALOG
- ROUTINE_NAME
- ROUTINE_SCHEMA
- ROW
- ROW_COUNT
- ROW_DELIMITER
- ROW_NUMBER
- ROWS
- ROWTIME

S

- SAVEPOINT
- SCALE
- SCHEMA
- SCHEMA_CASE_SENSITIVE
- SCHEMA_NAME
- SCOPE
- SCOPE_CATALOGS
- SCOPE_NAME
- SCOPE_SCHEMA
- SCROLL
- SEARCH
- SECOND
- SECTION
- SECURITY

- SELECT
- SELF
- SENSITIVE
- SEQUENCE
- SERIALIZABLE
- SERVER
- SERVER_NAME
- SESSION
- SESSION_USER
- SET
- SETS
- SIMILAR
- SIMPLE
- SINK
- SIZE
- SK
- SMALLINT
- SOME
- SOURCE
- SPACE
- SPECIFIC
- SPECIFICTYPE
- SPECIFIC_NAME
- SQL
- SQLEXCEPTION
- SQLSTATE
- SQLWARNING
- SQL_TSI_DAY
- SQL_TSI_FRAC_SECOND
- SQL_TSI_HOUR
- SQL_TSI_MICROSECOND
- SQL_TSI_MINUTE
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_SECOND
- SQL_TSI_WEEK
- SQL_TSI_YEAR
- SQRT
- START
- START_TIME

- STATE
- STATEMENT
- STATIC
- STDDEV_POP
- STDDEV_SAMP
- STREAM
- STRING
- STRUCTURE
- STYLE
- SUBCLASS_ORIGIN
- SUBMULTISET
- SUBSTITUTE
- SUBSTRING
- SUM
- SYMMETRIC
- SYSTEM
- SYSTEM_USER

T

- TABLE
- TABLESAMPLE
- TABLE_COLUMNS
- TABLE_NAME
- TABLE_NAME_MAP
- TEMP
- TEMPORARY
- THEN
- TIES
- TIME
- TIMESTAMP
- TIMESTAMPADD
- TIMESTAMPDIFF
- TIMEZONE_HOUR
- TIMEZONE_MINUTE
- TINYINT
- TO
- TOP_LEVEL_COUNT
- TOPIC
- TOPIC_URN
- TRAILING

- TRANSACTION
- TRANSACTIONAL_TABLE
- TRANSACTIONS_ACTIVE
- TRANSACTIONS_COMMITTED
- TRANSACTIONS_ROLLED_BACK
- TRANSFORM
- TRANSFORMS
- TRANSLATE
- TRANSLATION
- TRANX_ID
- TREAT
- TRIGGER
- TRIGGER_CATALOG
- TRIGGER_NAME
- TRIGGER_SCHEMA
- TRIM
- TRUE
- TSDB_LINK_ADDRESS
- TSDB_METRICS
- TSDB_TIMESTAMPS
- TSDB_TAGS
- TSDB_VALUES
- TYPE
- TYPE_CLASS_NAME
- TYPE_CLASS_PARAMETER

U

- UESCAPE
- UNBOUNDED
- UNCOMMITTED
- UNDER
- UNION
- UNIQUE
- UNKNOWN
- UNNAMED
- UNNEST
- UPDATE
- UPPER
- UPSERT
- URN_COLUMN

- USAGE
- USER
- USER_DEFINED_TYPE_CATALOG
- USER_DEFINED_TYPE_CODE
- USER_DEFINED_TYPE_NAME
- USER_DEFINED_TYPE_SCHEMA
- USERNAME
- USING

V

- VALUE
- VALUES
- VALUE_TYPE
- VARBINARY
- VARCHAR
- VARYING
- VAR_POP
- VAR_SAMP
- VERSION
- VERSION_ID
- VIEW

W

- WATERMARK
- WEEK
- WHEN
- WHENEVER
- WHERE
- WIDTH_BUCKET
- WINDOW
- WITH
- WITHIN
- WITHOUT
- WORK
- WRAPPER
- WRITE

X

- XML
- XML_CONFIG

Y

- YEAR

Z

- ZONE