Cloud Stream Service

SQL Syntax Reference

Issue 02

Date 2020-09-29

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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 Syntax Constraints

- Currently, Stream 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. For example:
  ```
  insert into temp select CARDINALITY(ARRAY[1,2,3]) FROM OrderA;
  ```


2 Data Type

Overview

Data type is a basic attribute of data and used to distinguish different types of data. Data of different types occupies different storage space and supports 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, StreamSQL of the Huawei big data platform supports both native data types and complex data types.

Native Data Types

Table 2-1 lists native data types supported by Stream SQL.

Table 2-1 Native 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 to 127
SMALLINT	Signed integer	2 bytes	–32768 to 32767
INT	Signed integer	4 bytes	–2147483648 to 2147483647
INTEGER	Signed integer	4 bytes	–2147483648 to 2147483647
BIGINT	Signed integer	8 bytes	–9223372036854775808 to 9223372036854775807

Data Type	Description	Storage Space	Value Range
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

Table 2-2 lists complex data types supported by Stream SQL.

Table 2-2 Complex data types

Data Type	Description
ARRAY	A group of ordered fields that are must be of the same data type
MAP	A group of unordered key/value pairs. Keys must be of the native data type, whereas values can be of the native data type or complex data type. All keys or values in a map must be of the same data type.

3 Built-In Functions

3.1 Mathematical Operation Functions

Relational Operators

All data types can be compared by using relational operators and the result is returned as a Boolean value.

Relational operators are binary operators. Types of the compared data must be the same or the types must support implicit conversion.

Table 3-1 lists all relational operators supported by Stream SQL.

Table 3-1 Relational operators

Operator	Returned Data Type	Description
A = B	BOOLEAN	If A equals B, **TRUE** is returned. Otherwise, **FALSE** is returned. This operator is used for value assignment.
A <> B	BOOLEAN	If A is not equal to B, **TRUE** is returned. Otherwise, **FALSE** is returned. If A or B is NULL, **NULL** is returned. This operator follows the standard SQL syntax.
A < B	BOOLEAN	If A is less than B, **TRUE** is returned. Otherwise, **FALSE** is returned. If A or B is NULL, **NULL** is returned.
A <= B	BOOLEAN	If A is less than or equal to B, **TRUE** is returned. Otherwise, **FALSE** is returned. If A or B is NULL, **NULL** is returned.
A > B	BOOLEAN	If A is greater than B, **TRUE** is returned. Otherwise, **FALSE** is returned. If A or B is NULL, **NULL** is returned.

Operator	Returned Data Type	Description
A >= B	BOOLEAN	If A is greater than or equal to B, **TRUE** is returned. Otherwise, **FALSE** is returned. If A or B is NULL, **NULL** is returned.
A IS NULL	BOOLEAN	If A is NULL, **TRUE** is returned. Otherwise, **FALSE** is returned.
A IS NOT NULL	BOOLEAN	If A is not NULL, **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. ● 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.
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.

Operator	Returned Data Type	Description
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.

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 3-2 lists the calculation rules. A and B indicate logical expressions.

Table 3-2 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.

Operator	Returned Data Type	Description
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.

Arithmetical Operators

Arithmetic operators include binary operators and unary operators, for all of which, the returned results are of the numeric type. **Table 3-3** lists arithmetic operators supported by Stream SQL.

Table 3-3 Arithmetical 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 data type depends on the operation data types. For example, if a floating-point number is added to an integer, a floating-point number will be returned.
A - B	All numeric types	A minus B. The result data type depends on the operation data types.

Operator	Returned Data Type	Description
A * B	All numeric types	Multiplies A and B. The result data type depends on the operation data types.
A / B	All numeric types	Divides A by B. The result is a number of the double data type (double-precision).
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	Returns the value of e raised to the power of A.
CEIL(A) CEILING(A)	All numeric types	Returns the smallest integer that is greater than or equal to A. For example: ceil(21.2) = 22.
FLOOR(A)	All numeric types	Returns 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.
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.
DEGREES(A)	All numeric types	Converts the value A from radians to degrees.
RADIANS(A)	All numeric types	Converts the value 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	Returns the value of pi.

Precautions

Data of the string type is not allowed in arithmetic operations.

3.2 String Functions

String Operators

Table 3-4 lists common operation rules for string operators. A and B indicate string expressions.

Table 3-4 String operators

Operator	Returned Data Type	Description
A || B	STRING	Returns the string concatenated from A and B.
CHAR_LENGTH(A)	INT	Returns the number of characters in string A.
CHARACTER_LENGTH(A)	INT	Returns the number of characters in string A.
UPPER(A)	STRING	Returns the uppercase letter A.
LOWER(A)	STRING	Returns the lowercase letter a.

Operator	Returned Data Type	Description
POSITION(A IN B)	INT	Returns the position where A first appears in B.
TRIM({ BOTH | LEADING | TRAILING } A FROM B)	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.
OVERLAY(A PLACING B FROM integer [FOR B])	STRING	Replaces A with B.
SUBSTRING(A FROM integer)	STRING	Returns the substring that starts from a fixed position of A. The start position starts from 1.
SUBSTRING(A FROM integer FOR integer)	STRING	Returns the substring starting from a fixed position and with the given length A. The start position starts from 1.
INITCAP(A)	STRING	Returns the string whose first letter is in uppercase and the other letters in lowercase. Words are sequences of alphanumeric characters separated by non-alphanumeric characters.
MD5(String expr)	STRING	Returns the md5 value of a string.
SHA1(String expr)	STRING	Returns the SHA1 value of a string.
SHA256(String expr)	STRING	Returns the SHA256 value of a string.
replace(String expr, String toreplace, String replace)	STRING	Replaces all "toreplace" in the expr string with "replace".
hash_code(String expr)	INT	Obtains the hash value. In addition to string, the parameter supports int, bigint, float, and double.
string_to_array(value, delimiter)	Array[String]	Separates the "value" string as character string arrays by using the delimiter.
CONCAT(String A, String B, ...)	STRING	Returns the concatenation result of two or more strings.
CONCAT_WS(String separator, String A, String B, ...)	STRING	Returns the concatenation result of two or more strings and uses separator as the delimiter to connect each string.

Operator	Returned Data Type	Description
RPAD(String str, INT len, String pad)	STRING	Concatenates the pad string to the right of the str string until the length of the new string reaches the specified length len. • 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 string obtained from cutting the str string to the length of len is returned.
LPAD(String str, INT len, String pad)	STRING	Concatenates the pad string to the left of the str string until the length of the new string reaches the specified length len. • 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 string obtained from cutting the str string to the length of len is returned.

3.3 Time Functions

Table 3-5 lists the time functions supported by Stream SQL.

Table 3-5 Time functions

Function	Returned Data Type	Description
DATE string	DATE	Parses the date string (**yyyy-MM-dd**) to a SQL date.
TIME string	TIME	Parses the time string (**HH:mm:ss**) to the SQL time.
TIMESTAMP string	TIMESTAMP	Converts the time string into timestamp. The time string format is **yyyy-MM-dd HH:mm:ss.fff**.

Function	Returned Data Type	Description
INTERVAL string range	INTERVAL	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. For example: **INTERVAL '10 00:00:00.004' DAY TO milliseconds** indicates that the interval is 10 days and 4 milliseconds. **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.
CURRENT_DATE	DATE	Returns the SQL date of UTC time zone.
CURRENT_TIME	TIME	Returns the SQL time of UTC time zone.
CURRENT_TIMESTAMP	TIMESTAMP	Returns the SQL timestamp of UTC time zone.
LOCALTIME	TIME	Returns the SQL time of the current time zone.
LOCALTIMESTAMP	TIMESTAMP	Returns the SQL timestamp of the current time zone.
EXTRACT(timeintervalunit FROM temporal)	INT	Extracts part of the time point or interval. Returns the part in the int type. For example, extract the date **2006-06-05** and return 5.
FLOOR(timepoint TO timeintervalunit)	TIME	Rounds 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	Rounds a time point up to the given unit. For example, **12:45:00** is returned from CEIL(TIME '12:44:31' TO MINUTE).
QUARTER(date)	INT	Returns the quarter from the SQL date.

Function	Returned Data Type	Description
(timepoint, temporal) OVERLAPS (timepoint, temporal)	BOOLEAN	Checks 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 ***leftEnd >= rightStart && rightEnd >= leftStart***. 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. For example: ● 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_LOCALTIMESTAMP(long expr)	TIMESTAMP	Converts the time to the local time.
UNIX_TIMESTAMP	BIGINT	Returns the timestamp of a specified parameter. The timestamp type is BIGINT and the unit is second. The following methods are supported: ● 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**.

Function	Returned Data Type	Description
UNIX_TIMESTAMP_MS	BIGINT	Returns the timestamp of a specified parameter. The timestamp type is BIGINT and the unit is millisecond. The following methods are supported: • 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.
```

3.4 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

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;
```


Table 3-6 Examples of type conversion functions

Example	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, bigint, 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.

3.5 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 3-7** lists aggregate functions.

Table 3-7 Aggregate functions

Function	Returned Data Type	Description
COUNT(value [, value]*)	DOUBLE	Returns count of values that are not null.
COUNT(*)	BIGINT	Returns count of tuples.
AVG(numeric)	DOUBLE	Returns the average (arithmetic mean) of all input values.
SUM(numeric)	DOUBLE	Returns the sum of all input numerical values.
MAX(value)	DOUBLE	Returns the maximum value among all input values.
MIN(value)	DOUBLE	Returns the minimum value among all input values.
STDDEV_POP(value)	DOUBLE	Returns the population standard deviation of all numeric fields of all input values.
STDDEV_SAMP(value)	DOUBLE	Returns the sample standard deviation of all numeric fields of all input values.
VAR_POP(value)	DOUBLE	Returns the population variance (square of population standard deviation) of numeral fields of all input values.

Function	Returned Data Type	Description
VAR_SAMP(value)	DOUBLE	Returns the sample variance (square of the sample standard deviation) of numeric fields of all input values.

Precautions

None

Example

None

3.6 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 3-8 Table-valued functions

Function	Returned 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;
```


3.7 Other Functions

Array Functions

Table 3-9 Array functions

Function	Returned Data Type	Description
CARDINALITY(ARRAY)	INT	Returns the number of elements in an array.
ELEMENT(ARRAY)	-	Returns 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.

Precautions

None

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 3-10 Attribute access functions

Function	Returned Data Type	Description
tableName.compositeType.field	-	Selects a single field, uses the name to access the field of Apache Flink composite types, such as Tuple and POJO, and returns the value.
tableName.compositeType.*	-	Selects all fields, and converts Apache Flink composite types, such as Tuple and POJO, and all their direct subtypes into a simple table. Each subtype is a separate field.

Precautions

None

Example

None

4 Geographical Functions

Description

Table 4-1 describes the basic geospatial geometric elements.

Table 4-1 Basic geospatial geometric element table

Geospatial geometric elements	Description	Example
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)])
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 4-2 describes the related transformation methods.

Table 4-2 Transformation methods for building complex geometric elements based on basic geospatial geometric elements

Transformation Method	Description	Example
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))

CS provides multiple functions used for performing operations on and determining locations of geospatial geometric elements. **Table 4-3** describes the SQL scalar functions.

Table 4-3 SQL scalar function table

Function	Returned Data Type	Description
ST_DISTANCE(point_1, point_2)	DOUBLE	Calculates the Euclidean distance between the two geographical points. An example is provided as follows: Select ST_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input
ST_GEODESIC_DISTANCE(point_1, point_2)	DOUBLE	Calculates the shortest distance along the surface between two geographical points. An example is provided as follows: Select ST_GEODESIC_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input

Function	Returned Data Type	Description
ST_PERIMETER(polygon)	DOUBLE	Calculates the circumference of a polygon. An example is provided as follows: 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. An example is provided as follows: 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. An example is provided as follows: 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. An example is provided as follows: 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
ST_WITHIN(point, polygon)	BOOLEAN	Checks whether one point is contained inside a geometry (polygon or circle). An example is provided as follows: 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

Function	Returned Data Type	Description
ST_CONTAINS(polygon_1, polygon_2)	BOOLEAN	Checks whether the first geometry contains the second geometry. An example is provided as follows: 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. An example is provided as follows: 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. An example is provided as follows: 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, CS 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 4-4**.

Table 4-4 Functions for geographical coordinate system conversion and distance-unit conversion

Function	Returned Data 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. An example is provided as follows: 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. An example is provided as follows: 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. An example is provided as follows: 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. An example is provided as follows: BD09_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))

Function	Returned Data 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. An example is provided as follows: 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. An example is provided as follows: 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	Returned Data 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))

CS 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 4-5**.

Table 4-5 Time-related SQL geographical aggregation function table

Function	Description	Example
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

Table 4-6 IP functions

Function	Returned Data Type	Description
IP_TO_COUNTRY	STRING	Obtains the name of the country where the IP address is located. The value is returned in Chinese.
IP_TO_PROVINCE	STRING	Obtains the province where the IP address is located. Usage: • IP_TO_PROVINCE(STRING ip): Determines the province where the IP address is located and returns the English 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. If **lang** is **EN**, the English province name is returned. If **lang** is not **EN**, the Chinese province name is returned. **NOTE** • 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 Chinese or English name returned by the function for the Chinese province is the short name, which is the same as that provided on the Chinese government official website. For details about the Chinese province names, visit the following website: http://www.gov.cn/guoqing/2005-09/13/content_5043917.htm For details about the English province names, visit the following website: http://english.gov.cn/archive/
IP_TO_CITY	STRING	Obtains the name of the city where the IP address is located. The value is returned in Chinese. **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	Returned Data 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: *Latitude*, *Longitude*. Usage: IP_TO_CITY_GEO(STRING ip): Returns the longitude and latitude of the city where the IP address is located.

5 DDL Statement

5.1 Creating a Source Stream

5.1.1 DIS Source Stream

Overview

Create a source stream to read data from DIS. DIS accesses user data and CS reads data from the DIS stream as input data for jobs. This cloud ecosystem is applicable to scenarios where data out of cloud services is imported into cloud services for filtering, real-time analysis, monitoring and reporting, 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

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= "")(TIMESTAMP BY timeindicator (',' timeindicator)?);timeindicator:PROCTIME '.' PROCTIME| ID '.' ROWTIME

Description

Table 5-1 Syntax description

Parameter	Mandatory	Description
type	Yes	Data source type. Value **dis** indicates that the data source is DIS.
region	Yes	Region where DIS for storing the data is located.
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 CS 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 this parameter is set to [2:5], the CS job will ingest data in partitions 2 and 5.
encode	Yes	Data encoding format. Available options include **csv**, **json**, **xml**, **email**, and **blob**. **NOTE** • **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].
field_delimiter	No	Attribute delimiter. You can set this parameter only when **encode** is set to **csv**. The default value is a comma (**,**).

Parameter	Mandatory	Description
quote	No	Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters. ● If double quotation marks are used as the quoted symbol, set this parameter to "\u005c\u0022" for character conversion. ● If a single quotation mark is used as the quoted symbol, set this parameter to a comma (,). **NOTE** 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	If **encode** is set to **json**, you need to 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.
xml_config	No	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.
email_key	No	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 CS specifies "body" as the keyword.
offset	No	● 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**, CS starts from the 100th data record in DIS.

Parameter	Mandatory	Description
start_time	No	Start time for reading DIS data. • If this parameter is specified, CS reads data read from the specified time. The parameter value is in the format of yyyy-MM-dd HH:mm:ss. • If neither **start_time** nor **offset** is specified, CS reads the latest data. • If **start_time** is not specified but **offset** is specified, CS reads data from the data record specified by **offset**.
timeindicator	No	Timestamp added in the source stream. The value can be **processing time** or **event time**. **NOTE** • If this parameter is set to **processing time**, the format is **proctime.proctime**. In this case, an attribute proctime will be added to the original attribute field. If there are three attributes in the original attribute field, four attributes will be exported after this parameter is set to **processing time**. However, the attribute length remains unchanged if the rowtime attribute is specified. • If this parameter is set to **event time**, you can select an attribute in the stream as the timestamp. The format is **attr_name.rowtime**. • This parameter can be simultaneously set to **processing time** and **event time**.
enable_checkpoint	No	Whether to enable the checkpoint function. Value **true** indicates to enable the checkpoint function, and value **false** indicates to disable the checkpoint function. 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

The attribute type used as the timestamp must be long or timestamp.

Example

- In CSV encoding format, CS 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 = "cn-north-1" ,
  channel = "csinput",

  encode = "csv",
  field_delimiter = ","
)TIMESTAMP BY car_timestamp.rowtime;
```

- In JSON encoding format, CS 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 = "cn-north-1" ,
  channel = "csinput",

  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;"
)TIMESTAMP BY car_timestamp.rowtime;
```

- In XML encoding format, CS 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 = "cn-north-1" ,
  channel = "dis-cs-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" encodeing="utf-8"?>

<root>
```



```
 <person>
  <pid>362305199010025042</pid>
  <pname>xiaoming</pname>
  <page>28</page>
  <plocation>Shangdu County, Jining District, Ulanchap, Inner Mongolia</plocation>
  <pbir>1990-10-02</pbir>
  <phealthy>true</phealthy>
  <pgrade>[A,B,C]</pgrade>
 </person>
</root>
```

- In EMAIL encoding format, CS 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 = "cn-north-1" ,
channel = "csinput",

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: <20090629183903250425 4@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
```


5.1.2 OBS Source Stream

Overview

Create a source stream to obtain data from OBS. CS 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, see the **Object Storage Service Console Operation Guide**.

Syntax

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 = "")(TIMESTAMP BY timeindicator (',' timeindicator)?);timeindicator:PROCTIME '.' PROCTIME| ID '.' ROWTIME

Description

Table 5-2 Syntax description

Parameter	Mandatory	Description
type	Yes	Data source type. Value **obs** indicates that the data source is OBS.
region	Yes	Region to which OBS belongs.
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.
row_delimiter	Yes	Separator used to separate every two rows.
field_delimiter	Yes	Separator used to separate every two attributes. • You can set this parameter only when **encode** is set to **csv**. The default value is a comma (**,**). • This parameter is not required if the JSON encoding format is adopted.

Parameter	Mandatory	Description
quote	No	Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters. ● If double quotation marks are used as the quoted symbol, set this parameter to "\u005c \u0022" for character conversion. ● If a single quotation mark is used as the quoted symbol, set this parameter to a comma (,). **NOTE** 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.
timeindicator	No	Timestamp added in the source stream. The value can be **processing time** or **event time**. **NOTE** ● If this parameter is set to **processing time**, the format is **proctime.proctime**. In this case, an attribute proctime will be added to the original attribute field. If there are three attributes in the original attribute field, four attributes will be exported after this parameter is set to **processing time**. However, the attribute length remains unchanged if the rowtime attribute is specified. ● If this parameter is set to **event time**, you can select an attribute in the stream as the timestamp. The format is **attr_name.rowtime**. ● This parameter can be simultaneously set to **processing time** and **event time**.

Precautions

The attribute type used as the timestamp must be long or timestamp.

Example

The **input.csv** file is read from the OBS bucket. Rows are separated by **'\n'** and columns are separated by **','**.

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
WITH (
  type = "obs",
  bucket = "obssource",
  region = "cn-north-1" ,
  object_name = "input.csv",
  row_delimiter = "\n",
  field_delimiter = ",",

)TIMESTAMP BY car_timestamp.rowtime;
```

5.1.3 HBase Source Stream

Overview

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. CS 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 CS 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**.

Syntax

Syntax

CREATE SOURCE STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)WITH (type = "cloudtable",region = "",cluster_id = "",table_name = "",table_columns = "")(TIMESTAMP BY timeindicator (',' timeindicator)?);timeindicator:PROCTIME '.' PROCTIME| ID '.' ROWTIME

Description

Table 5-3 Syntax description

Parameter	Mandatory	Description
type	Yes	Data source type. Value **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 **CloudTable Service User Guide**.
table_name	Yes	Name of the table where data is to be read. If namespace needs to be specified, the value can be 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.
timeindicator	No	Timestamp added in the source stream. The value can be **Processing Time** or **Event Time**. **NOTE** • If this parameter is set to **Processing Time**, set **timeindicator** to **proctime.proctime**. In this case, an attribute proctime will be added to the original attribute field. If there are three attributes in the original attribute field, four attributes will be exported after this parameter is set to **processing time**. • If this parameter is set to **Event Time**, you can select an attribute in the stream as the timestamp that is in the format of attr_name.rowtime, where attr_name indicates the attribute in the stream. • This parameter can be simultaneously set to **processing time** and **event time**.

Precautions

The attribute type used as the timestamp must be long or timestamp.

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 = "cn-north-1" ,
  cluster_id = "209ab1b6-de25-4c48-8e1e-29e09d02de28",
  table_name = "carinfo",
  table_columns = "rowKey,info:owner,info:age,car:speed,car:miles"
);
```

5.1.4 MRS Kafka Source Stream

Overview

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 CS cluster. 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 the description of **Adding an IP-Domain Mapping** in **Cluster Management** in the *Cloud Stream Service User Guide*.
- When using offline Kafka clusters, use VPC peering connections to connect CS to Kafka.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

Syntax

Syntax

CREATE SOURCE STREAM kafka_source (name STRING, age int)WITH (type = "kafka",kafka_bootstrap_servers = "",kafka_group_id = "",kafka_topic = "",encode = "json")(TIMESTAMP BY timeindicator (',' timeindicator)?);timeindicator:PROCTIME '.' PROCTIME| ID '.' ROWTIME

Description

Table 5-4 Syntax description

Parameter	Mandatory	Description
type	Yes	Data source type. Value **Kafka** indicates that the data source is Kafka.
kafka_bootstrap_servers	Yes	Port that connects CS to Kafka. Use VPC peering connections to connect CS clusters with Kafka clusters.

Parameter	Mandatory	Description
kafka_group_id	Yes	Group ID.
kafka_topic	Yes	Kafka topic to be read.
encode	Yes	Data encoding format. The value can be **csv**, **json**, or **blob**. ● **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.
json_config	No	If **encode** is set to **json**, you can use this parameter to specify the mapping between JSON fields and stream attribute fields. The format is field1=json_field1;field2=json_field2.
field_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.
quote	No	Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters. ● If double quotation marks are used as the quoted symbol, set this parameter to "\u005c\u0022" for character conversion. ● If a single quotation mark is used as the quoted symbol, set this parameter to a comma (,). **NOTE** 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.

Parameter	Mandatory	Description
timeindicator	No	Timestamp added in the source stream. The value can be **processing time** or **event time**. **NOTE** • If this parameter is set to **processing time**, the format is **proctime.proctime**. In this case, an attribute proctime will be added to the original attribute field. If there are three attributes in the original attribute field, four attributes will be exported after this parameter is set to **processing time**. However, the attribute length remains unchanged if the rowtime attribute is specified. • If this parameter is set to **event time**, you can select an attribute in the stream as the timestamp. The format is **attr_name.rowtime**. • This parameter can be simultaneously set to **processing time** and **event time**.
start_time	No	Start time when Kafka data is ingested. If this parameter is specified, CS reads data read from the specified time. The parameter value is in the format of 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.

Precautions

The attribute type used as the timestamp must be long or timestamp.

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"
);
```

5.1.5 Open-Source Kafka Source Stream

Overview

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 CS cluster. 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 the description of **Adding an IP-Domain Mapping** in **Cluster Management** in the *Cloud Stream Service User Guide*.
- When using offline Kafka clusters, use VPC peering connections to connect CS to Kafka.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

Syntax

Syntax

CREATE SOURCE STREAM kafka_source (name STRING, age int)WITH (type = "kafka",kafka_bootstrap_servers = "",kafka_group_id = "",kafka_topic = "",encode = "json",json_config="")(TIMESTAMP BY timeindicator (',' timeindicator)?);timeindicator:PROCTIME '.' PROCTIME| ID '.' ROWTIME

Description

Table 5-5 Syntax description

Parameter	Mandatory	Description
type	Yes	Data source type. Value **Kafka** indicates that the data source is Kafka.
kafka_bootstrap_servers	Yes	Port that connects CS to Kafka. Use VPC peering connections to connect CS clusters with Kafka clusters.
kafka_group_id	Yes	Group ID.
kafka_topic	Yes	Kafka topic to be read.

Parameter	Mandatory	Description
encode	Yes	Data encoding format. The value can be **json**, **csv**, or **blob**. ● **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.
json_config	No	If **encode** is set to **json**, you can use this parameter to specify the mapping between JSON fields and stream attribute fields. The format is field1=json_field1;field2=json_field2.
field_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.
quote	No	Quoted symbol in a data format. The attribute delimiters between two quoted symbols are treated as common characters. ● If double quotation marks are used as the quoted symbol, set this parameter to "\u005c \u0022" for character conversion. ● If a single quotation mark is used as the quoted symbol, set this parameter to a comma (,). **NOTE** 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.

Parameter	Mandatory	Description
timeindicator	No	Timestamp added in the source stream. The value can be **processing time** or **event time**. **NOTE** • If this parameter is set to **processing time**, the format is **proctime.proctime**. In this case, an attribute proctime will be added to the original attribute field. If there are three attributes in the original attribute field, four attributes will be exported after this parameter is set to **processing time**. However, the attribute length remains unchanged if the rowtime attribute is specified. • If this parameter is set to **event time**, you can select an attribute in the stream as the timestamp. The format is **attr_name.rowtime**. • This parameter can be simultaneously set to **processing time** and **event time**.
start_time	No	Start time when Kafka data is ingested. If this parameter is specified, CS reads data read from the specified time. The parameter value is in the format of 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.

Precautions

The attribute type used as the timestamp must be long or timestamp.

Example

Read Kafka topic **test**. Data instance: {"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"
);
```


5.2 Creating a Sink Stream

5.2.1 DIS Sink Stream

Overview

CS writes the 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

Syntax

CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)WITH (type = "dis",region = "",channel = "",partition_key = "",encode= "",field_delimiter= "");

Description

Table 5-6 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **dis** indicates that data is stored to DIS.
region	Yes	Region where DIS for storing the data is located.
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.

Parameter	Mandatory	Description
encode	Yes	Data encoding format. The value can be **csv** and **json**. **NOTE** • If the encoding format is **csv**, you need to configure attribute separators. • If the encoding format is **json**, you need to configure **enableOutputNull** to control whether to generate an empty field. For details, see the examples.
field_delimiter	Yes	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 (**,**). • 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.
enableOutputNull	No	If **encode** is set to **json**, you need to specify this parameter to control 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.

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 = "cn-north-1" ,
  channel = "csoutput",
  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 = "csoutput",
  region = "cn-north-1" ,
  partition_key = "car_owner,car_brand",
  encode = "json",
  enable_output_null = "false"
);
```

5.2.2 OBS Sink Stream

Overview

Create a sink stream to export CS data to OBS. CS 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, see the **Object Storage Service Console Operation Guide**.

Precautions

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

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 = "",
        max_record_num_cache = "",
        carbon_properties = ""
)
```

Description

Table 5-7 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **obs** indicates that data is stored to OBS.
region	Yes	Region to which OBS belongs.
encode	Yes	Encoding format. Currently, formats CSV, JSON, ORC, and CarbonData 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. ● This parameter does not need to be configured if the CarbonData, JSON, or ORC encoding format is adopted.
row_delimiter	No	Row delimiter. This parameter does not need to be configured if the CarbonData or ORC 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.
obs_dir	Yes	Directory for storing files. The directory is in the format of {Bucket name}/{Directory name}, for example, obs-a1/dir1/subdir.

Parameter	Mandatory	Description
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. This parameter is not applicable to CarbonData files.
rolling_size	No	Maximum size of a file. **NOTE** • 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 or CarbonData encoding format is adopted.
rolling_interval	No	Time mode, in which data is saved to the corresponding directory. **NOTE** • 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 16:40 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.
array_bracket	No	Array bracket, which can be configured only when the CSV encoding format is adopted. The available options are **()**, **{}**, and **[]**. For example, if you set this parameter to **{}**, the array output format is {a1, a2}.

Parameter	Mandatory	Description
append	No	The value can be **true** or **false**. The default value is **true**. If OBS does not support the append mode, 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 configured only when the ORC or CarbonData encoding format is adopted. After this parameter is specified, a new file is generated to store extra data records after the number of records stored in a file reaches the allowed quantity.
dump_interval	No	Triggering period. This parameter needs to be configured when the ORC encoding format is adopted or notification to DIS is enabled. ● 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 stream where CS sends the record that contains the OBS directory CS periodically sends the DIS stream a record, which contains the OBS directory, indicating that no more new files will be generated in the directory.
max_record_num_cache	No	Maximum number of cache records. This parameter can be set only when the CarbonData encoding format is adopted. The minimum value of this parameter cannot be less than that of **max_record_num_per_file**. The default value is **max_record_num_per_file**.
carbon_properties	No	Carbon attribute. This field can be configured only when the CarbonData encoding format is adopted. The value is in the format of k1=v1, k2=v2. All configuration items supported by the withTableProperties function in carbon-sdk are supported. In addition, the configuration items **IN_MEMORY_FOR_SORT_DATA_IN_MB** and **UNSAFE_WORKING_MEMORY_IN_MB** are supported.

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 = "cn-north-1" ,
  field_delimiter = ",",
  row_delimiter = "\n",
  obs_dir = "obs-sink/car_infos",
   file_prefix = "greater_30",
   rolling_size = "100m"
);
```

- Example of the CarbonData 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 = "cn-north-1" ,
  encode = "carbondata",
  obs_dir = "cs-append-2/carbondata",
  max_record_num_per_file = "1000",
  max_record_num_cache = "2000",
  dump_interval = "60",
  ROLLING_INTERVAL = "yyyy/MM/dd/HH/mm",
  dis_notice_channel = "dis-notice",
  carbon_properties = "long_string_columns=MessageBody,
IN_MEMORY_FOR_SORT_DATA_IN_MB=512"
);
```

- 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 = "cn-north-1" ,
  encode = "orc",
  obs_dir = "cs-append-2/obsorc",
  FILE_PREFIX = "es_info",
  max_record_num_per_file = "100000",
  dump_interval = "60"
);
```


5.2.3 HBase Sink Stream

Overview

CS 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 CS, 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 CS 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**.

Syntax

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 = "")
```

Description

Table 5-8 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **cloudtable** indicates that data is stored to CloudTable (HBase).
region	Yes	Region to which CloudTable belongs.
cluster_id	Yes	ID of the cluster to which the data table to be read 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.

Parameter	Mandatory	Description
table_columns	Yes	Columns to be inserted. The parameter value is the following format: **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 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**, and **false** is used by default.
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.

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 = "cn-north-1" ,
  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"
);
```


5.2.4 OpenTSDB Sink Stream

Overview

CS 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 CS 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

Ensure that OpenTSDB has been enabled on the CloudTable clusters. For details about how to enable OpenTSDB, see **Enabling OpenTSDB** in the *CloudTable Service User Guide*.

Syntax

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 = "")

Description

Table 5-9 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **opentsdb** indicates that data is stored to CloudTable (OpenTSDB).
region	Yes	Region to which CloudTable belongs.
cluster_id	Yes	ID of the cluster to which data is to be inserted.
tsdb_metrics	Yes	Metric of a data point, which can be specified through parameter configurations.

Parameter	Mandatory	Description
tsdb_timestamps	Yes	Timestamp of a data point. The data type can be TIMESTAMP, LONG, or INT. 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 **100**. 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 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 = "cn-north-1" ,
  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"
);
```

5.2.5 RDS Sink Stream

Overview

CS outputs the 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. RDS includes databases of the following types: MySQL, HWSQL, PostgreSQL, and Microsoft SQL Server. 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 Guide**.
- In this scenario, jobs must run on the exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with RDS instance. You can also set the security group rules as required.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

  For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Syntax

Syntax

CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)WITH (type = "rds",username = "",password = "",db_url = "",table_name = "");

Description

Table 5-10 Syntax 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.

Parameter	Mandatory	Description
db_url	Yes	Database connection address, for example, **{database_type}:// ip:port/database**. Currently, two types of database connections are supported: MySQL and PostgreSQL. ● MySQL: 'mysql://ip:port/ database' ● PostgreSQL: 'postgresql:// ip:port/database'
table_name	Yes	Name of the table where data will be inserted.
db_columns	No	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. Example: `create sink stream a3(student_name string, student_age int) with ( type = "rds", username = "root", password = "xxxxxxxx", db_url = "mysql://192.168.0.102:8635/test1", db_columns = "name,age", table_name = "t1" );` In the example, **student_name** corresponds to the name attribute in the database, and **student_age** corresponds to the age attribute in the database. **NOTE** 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.

Parameter	Mandatory	Description

primary_key	No	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 write operation, data is updated if the specified **primary_key** exists. Otherwise, data is inserted.
		Example:
		`CREATE SINK STREAM test(c_timeminute LONG, c_cnt LONG)`
		`WITH (`
		`type = "rds",`
		`username = "root",`
		`password = "xxxxxxxx",`
		`db_url = "mysql://192.168.0.12:8635/test",`
		`table_name = "test",`
		`primary_key = "c_timeminute");`

operation_field	No	Processing method of specified data in the format of ${field_name}. 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.

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"
);
```


5.2.6 Synchronizing Data to RDS and DWS

Overview

CS can synchronize data to Relational Database Service (RDS) and Data Warehouse Service (DWS) in real time based on the binlog format. 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. RDS includes databases of the following types: MySQL, HWSQL, PostgreSQL, and Microsoft SQL Server.

For more information about RDS, see the **Relational Database Service User Guide**.

DWS is a high-performance, fully hosted, and large-scale parallel processing database service. It supports row- and column-based storage and provides features such as one-click deployment and rapid data import to the database. It aims to meet the requirements of cloud users for secure, reliable, and fast data mining, storage, and analysis.

For more information about DWS, see the **Data Warehouse Service Management 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 Guide**.
- In this scenario, jobs must run on the exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with RDS instance. You can also set the security group rules as required.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

  For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Syntax

Syntax

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
WITH (
type = "db_sync",
region = "",
db_url = "",
```



```
username = "",
password = "",
table_name = "${attr_name}",
operation_field = "${attr_name}",
before = "${attr_name}",
after = "${attr_name}",
tranx_id = "${attr_name}",
commit = "${attr_name}",
sql = "${attr_name}",
table_name_map = "${attr_name}",
column_name_map ="${attr_name}",
schema_case_sensitive = "false",
db_type = "dws",
);
```

Description

This feature supports database synchronization for MySQL, PostgreSQL, and DWS. Multiple tables in the same database can be synchronized.

Table 5-11 Parameter description

Parameter	Mandatory	Description
db_url	Yes	Indicates the database connection address, for example, **{database_type}://ip:port/database**. Currently, the following database connections are supported: • MySQL: 'mysql://ip:port/database' • PostgreSQL and DWS: 'postgresql://ip:port/database'
username	Yes	Indicates the username of a database.
password	Yes	Indicates the password of a database.
table_name	Yes	Indicates the table name, that is the table name field in the sink stream. The format is **${attribute name}**. This attribute is parsed from the table field in the binlog.

Parameter	Mandatory	Description
operation_field	Yes	Indicates the operation type, that is the operation type field in the sink stream. The format is **${attribute name}**. The operation type can be **I**, **U**, **D**, **INSERT**, **UPDATE**, or **DELETE**. This attribute is parsed from the operation type field in the binlog.
before	Yes	Indicates the original content field in the sink stream. The format is **${attribute name}**.
after	Yes	Indicates the new content field in the sink stream. The format is **${attribute name}**.
tranx_id	No	Indicates the event ID field in the sink stream. The format is **${attribute name}**.
commit	No	Indicates the event completed field in the sink stream. The format is **${attribute name}**.
sql	No	Indicates the field of DDL SQL statements in the sink stream. The format is **${attribute name}**.
table_name_map	No	If the table name in the original database is different from that in the destination database, you need to set this parameter to a constant or variable. ● Constant configuration: table_name_map="distdbtable", indicating that all records are exported to the distdbtable table in the database. ● Variable configuration: table_name_map="${attribute name}". table_name_map="${dbTableName} indicates that the destination table name is determined by the dbTableName attribute. If this parameter is not set, the destination table is the same as the original table by default.
column_name_map	No	If the column name in the original database is different from that in the destination database, you need to set this parameter to a constant or variable. ● Constant configuration: column_name_map="originAttr1=distAttr1,originAttr2=distAttr2", indicating that all records whose column name is originAttr1 are mapped to the distAttr1 field in the database. ● Variable configuration: column_name_map="${attribute name}". column_name_map="${dbColumnMap} indicates that the destination column name is determined by the dbColumnMap attribute. If this parameter is not set, the destination table is the same as the original table by default.

Parameter	Mandatory	Description
schema_case_sensitive	No	Indicates whether the schema in the target table is case-insensitive. The default value is **false**, indicating that the schema is case-insensitive.
db_type	No	Indicates the type of the destination database. The default value is **DWS**.

Precautions

1. Data to be inserted and data updated must be mapped to the **after** field. Data to be deleted and updated must be mapped to the **before** field.

2. The binlog of the sink stream is synchronized to the target database in sequence. Therefore, ensure that the binlog of the source stream meets the expected time sequence and transaction division. You are advised to divide the binlogs of transaction-associated tables into the same source partition based on the transaction association of the binlogs at the source end.

Example

This example shows how to receive Maxwell binlogs from DIS and synchronize the data to the database.

Assume that there are three pieces of data. The first piece is an insert operation, the second piece is an update operation, and the third piece is a delete operation.

```
{
  "table":"TEST.T1",
  "op_type":"I",
  "current_ts":"2019-04-05T10:21:51.200000",
  "after":{
  "ID":111,
  "NAME":"karl",
  "AGE":21 }
}
{
  "table":"TEST.T2",
  "op_type":"U",
  "current_ts":"2019-04-05T10:21:51.200000",
  "before":{"ID":22},
  "after":{
  "ID":22,
  "NAME":"sherryUpdate",
  "AGE":23 }
}
{
  "table":"TEST.T3",
  "op_type":"D",
  "current_ts":"2019-04-05T10:21:51.200000",
  "before":{
  "ID":111,
  "NAME":"karl",
  "AGE":21 }
}
```


T1 corresponds to table1 in the database, T2 corresponds to table 2, and T3 corresponds to table 3. The SQL statements are as follows:

```
CREATE SOURCE STREAM mysqlBinLog (
  dbName String,
  tableName STRING,
  op_type STRING,
  beforeData STRING,
  afterData STRING,
  mysql STRING,
  xid LONG,
  tranxCommit BOOLEAN)
WITH (
      type = "dis",
      region = "cn-north-4",
      channel = "dis-input",
      encode = "json",
      json_config =
"dbName=database;tableName=table;op_type=type;beforeData=old;afterData=data;mysql=sql;xid=xid;tranxC
ommit=commit",
      enable_checkpoint = "true",
      checkpoint_app_name = "dis-sync-app",
      checkpoint_interval = "600",
      offset = "-1000"
     );

CREATE SINK STREAM dwsRepo (
  tableName STRING,
  op_type STRING,
  beforeData STRING,
  afterData STRING,
  xid LONG,
  tranxCommit BOOLEAN,
  mysql String)
WITH (
      type = "db_sync",
      region = "cn-north-4",
      db_url = "",
      username = "",
      password = "",
      cache_time = "86400000",
      table_name = "${tableName}",
      operation_field = "${op_type}",
      before = "${beforeData}",
      after = "${afterData}",
      tranx_id = "${xid}",
      commit = "${tranxCommit}",
      sql = "${mysql}",
      schema_case_sensitive = "false",
      db_type = "dws"
     );

INSERT INTO dwsRepo
SELECT
    "schema." || tableName,
    op_type,
    beforeData,
    afterData,
    xid,
    tranxCommit,
    mysql
FROM mysqlBinLog;
```


5.2.7 DWS Sink Stream (JDBC Mode)

Overview

CS outputs the 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 public 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 exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with DWS clusters. You can also set the security group rules as required.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

  For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Syntax

Syntax

CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)WITH (type = "rds",username = "",password = "",db_url = "",table_name = "");

Description

Table 5-12 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **rds** indicates that data is stored to RDS or DWS.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.

Parameter	Mandatory	Description
db_url	Yes	Format of the address used to connect to the database, which is as follows: 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.

db_columns	No	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.
		Example:
		`create sink stream a3(student_name  string, student_age int) with (`
		`type = "rds",`
		`username = "root",`
		`password = "xxxxxxxx",`
		`db_url = "postgresql://192.168.0.102:8635/test1",`
		`db_columns = "name,age",`
		`table_name = "t1"`
		`);`
		In the example, **student_name** corresponds to the name attribute in the database, and **student_age** corresponds to the age attribute in the database.
		NOTE
		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	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 write operation, data is updated if the specified **primary_key** exists. Otherwise, data is inserted.
		Example:
		`CREATE SINK STREAM test(c_timeminute LONG, c_cnt LONG)`
		`WITH (`
		`type = "rds",`
		`username = "root",`
		`password = "xxxxxxxx",`
		`db_url = "postgresql://192.168.0.12:8635/test",`
		`table_name = "test",`
		`primary_key = "c_timeminute");`

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://192.168.1.1:8635/test",
  table_name = "audi_cheaper_than_30w"
);
```

5.2.8 DWS Sink Stream (OBS-based Dumping)

Overview

Create a sink stream to export CS data to DWS through OBS-based dumping, specifically, output CS 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 public 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 the 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 Service Database 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 5-13**.
- If the target table does not exist, a table is automatically created. CS data of the SQL type does not support text. If a long text exists, you are advised to create a table in the database.
- 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**.

Syntax

Syntax

```
CREATE SINK STREAM stream_id (
   attr_name attr_type (',' attr_name attr_type)*
)WITH (
        type = "dws",
        region = "",
```



```
      encode = "",
      field_delimiter = "",
      quote = "",
      db_obs_server = "",
      obs_dir = "",
      username = "",
      password =  "",
      db_url = "",
      table_name = "",
      max_record_num_per_file = "",
      max_dump_file_num = "",
      dump_interval = ""
);
```

Description

Table 5-13 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **dws** indicates that data is stored to DWS.
region	Yes	Region where DWS is located.
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, **\u0006\u0002**.
quote	No	Single byte. It is recommended that invisible characters be used, for example, **u0007**.
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 *Data Warehouse Service Database Development Guide*. 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.

Parameter	Mandatory	Description
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_num_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.
max_dump_file_num	Yes	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.
dump_interval	Yes	Dumping period. The unit is second.

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 = "cn-north-1" ,
  encode = "csv",
  field_delimiter = "\u0006\u0006\u0002",
  quote = "\u0007",
  obs_dir = "cs-append-2/dws",
  username = "",
  password =  "",
  db_url = "192.168.1.12:8000/test1",
  table_name = "table1",
  max_record_num_per_file = "100",
  max_dump_file_num = "10",
  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 = "cn-north-1" ,
```



```
  encode = "orc",
  db_obs_server = "obs_server",
  obs_dir = "cs-append-2/dws",
  username = "",
  password =  "",
  db_url = "192.168.1.12:8000/test1",
  table_name = "table1",
  max_record_num_per_file = "100",
  max_dump_file_num = "10",
  dump_interval = "10"
);
```

5.2.9 DDS Sink Stream

Overview

CS 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. For more information about DDS, see the **Document Database Service User Guide**.

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 Quick Start**.
- Currently, only instances that are not enabled with SSL authentication are supported.

Syntax

Syntax

CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)WITH (type = "dds",username = "",password = "",db_url = "",field_names = "");

Description

Table 5-14 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **dds** indicates that data is stored to DDS.
username	Yes	Username for connecting to a database.
password	Yes	Password for connecting to a database.

Parameter	Mandatory	Description
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 specific format is as follows: "f1,f2,f3". Ensure that there is a one-to-one mapping between data in the parameter value and data columns 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 = "cn-north-1" ,
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"
);
```

5.2.10 SMN Sink Stream

Overview

CS exports job output data to SMN.

SMN provides reliable and flexible large-scale message notification services to CS. 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

Syntax

CREATE SINK STREAM stream_id xxx WITH(type = "smn",region = "",topic_urn = "",urn_column = "",message_subject = "",message_column = "")

Description

Table 5-15 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **smn** indicates that data is stored to SMN.
region	Yes	Region to which SMN belongs.
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 containing 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 = "cn-north-1" ,
  topic_Urn = "urn:smn:cn-north-1:38834633fd6f4bae813031b5985dbdea:ddd",
  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 = "cn-north-1" ,
urn_column = "over_speed_urn",
message_subject = "message title",
message_column = "over_speed_message"
);
```

5.2.11 Elasticsearch Sink Stream

Overview

CS exports 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 CS with structured and unstructured data search, statistics, and report capabilities. For more information about CSS, see the **Cloud Search Service User Guide**.

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 exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with CSS. You can also set the security group rules as required.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

  For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Syntax

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= "");

Description

Table 5-16 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **es** indicates that data is stored to CSS.
region	Yes	Region where CSS is located. For example, **cn-north-1**.
cluster_address	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 storing the data to be inserted.
es_type	Yes	Document type of the data to be inserted.
es_fields	Yes	Key of the data field to be inserted. The parameter is in the format of "Id, f1, f2, f3, f4". Ensure that the parameter value has a one-to-one mapping with data columns in the sink stream. If the key is not used, remove the **id** keyword. Specifically, the parameter is in the format of "F1, f2, f3, f4, f5".
batch_insert_data_num	Yes	Amount of data 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

None

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 = "cn-north-1" ,
  cluster_address = "192.168.0.212:9200",
  es_index = "china",
  es_type = "zhejiang",
```



```
  es_fields = "id,owner,age,speed,miles",
  batch_insert_data_num = "10"
);
```

5.2.12 DCS Sink Stream

Overview

CS exports 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 exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with DCS clusters. You can also set the security group rules as required.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service 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 CS cluster 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 CS cluster 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 CS cluster 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

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= "");

Description

Table 5-17 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **dcs_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: ● 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 ${column name}.

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 = "xxxxxxxx",
  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}}"
);
```

5.2.13 MRS Kafka Sink Stream

Overview

CS exports the job output data 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 CS cluster. 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 the description of **Adding an IP-Domain Mapping** in **Cluster Management** in the *Cloud Stream Service User Guide*.
- When using offline Kafka clusters, use VPC peering connections to connect CS to Kafka.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

Syntax

Syntax

CREATE SINK STREAM kafka_sink (name STRING) WITH(type = "kafka",kafka_bootstrap_servers = "",kafka_topic = "",encode = "json")

Description

Table 5-18 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **kafka** indicates that data is stored to Kafka.
kafka_bootstrap_servers	Yes	Port that connects CS to Kafka. Use VPC peering connections to connect CS clusters with Kafka clusters.
kafka_topic	Yes	Kafka topic into which CS writes data.
encode	Yes	Encoding format. Currently, only JSON is supported.

Precautions

None

Example

Output data to Kafka.

```
CREATE SINK STREAM kafka_sink (name STRING)
WITH (
  type="kafka",
  kafka_bootstrap_servers =  "ip1:port1,ip2:port2",
  kafka_topic = "testsink",
  encode = "json"
);
```

5.2.14 MRS HBase Sink Stream

Overview

CS exports the job output data to HBase of MRS.

Prerequisites

- An MRS cluster has been created by using your account. Currently, CS can only interconnect with the MRS cluster disabled with Kerberos authentication.
- In this scenario, the job needs to run in an exclusive cluster. Ensure that an exclusive cluster has been created.

  For details about how to create an exclusive cluster, see **Cluster Management** in the *Cloud Stream Service User Guide*.
- Ensure that a VPC peering connection has been set up between the exclusive cluster and the MRS cluster, and security group rules have been configured based on the site requirements.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

  For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Syntax

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 = "")

Description

Table 5-19 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **mrs_hbase** indicates that data is stored to HBase of MRS.
region	Yes	Region where MRS resides.
cluster_address	Yes	ZooKeeper address of the cluster to which the table where data is to be inserted belongs. The value is in the format of 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 parameter value is the following format: **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.
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**.

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 = "cn-north-1" ,
  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"
);
```

5.2.15 APIG Sink Stream

Overview

CS outputs the job result to open APIs of APIG. You can access data through API calling on APIG. APIG is an API hosting service with high performance, availability, and security. It allows users to create, manage, and deploy APIs at any scale. With APIG, you can implement system integration, micro-service aggregation, and serverless architectures easily and quickly with low costs and risks. For more information about API Gateway, see the **API Gateway User Guide (API Calling)**.

Prerequisites

- Ensure that you have created an application on APIG using your account. For details about how to create an application using API calling, see **Creating an Application** in the **API Gateway User Guide (API Calling)**.

Syntax

Syntax

CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)WITH (type = "apig",region = "",app_id= "",encode= "",field_delimiter= "");

Description

Table 5-20 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **apig** indicates that data is stored to APIG.
region	Yes	Region where APIG for storing the data is located.
app_id	Yes	ID of the application used for API calling.
encode	Yes	Data encoding format. The value can be **csv** and **json**. **NOTE** • If the encoding format is **csv**, you need to configure field separators. • If the encoding format is **json**, you need to configure whether to generate an empty field. For details, see the examples.
field_delimiter	Yes	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 (**,**). • 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.

Precautions

None

Example

- CSV: Data is written to APIG as codes in CSV format which 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 = "apig",
  app_id = "xxxxxxxxxx",
  region = "cn-north-1" ,
  encode = "csv",
  field_delimiter = ","
);
```

- JSON: Data is output to APIG 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 = "apig",
  app_id = "6ac32a6596614bf69fb5c5553f26963f",
  region = "cn-north-1" ,
  encode = "json",
  enable_output_null = "false"
);
```

5.2.16 Open-Source Kafka Sink Stream

Overview

CS exports the job output data 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 CS cluster. 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 the description of **Adding an IP-Domain Mapping** in **Cluster Management** in the *Cloud Stream Service User Guide*.
- When using offline Kafka clusters, use VPC peering connections to connect CS to Kafka.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

Syntax

Syntax

CREATE SINK STREAM kafka_sink (name STRING) WITH(type = "kafka",kafka_bootstrap_servers = "",kafka_topic = "",encode = "json")

Description

Table 5-21 Syntax description

Parameter	Mandatory	Description
type	Yes	Output channel type. Value **kafka** indicates that data is stored to Kafka.
kafka_bootstrap_servers	Yes	Port that connects CS to Kafka. Use VPC peering connections to connect CS clusters with Kafka clusters.
kafka_topic	Yes	Kafka topic into which CS writes data.
encode	Yes	This parameter can be set to **json** or **csv**.
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.

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"
);
```

5.3 Creating a Temporary Stream

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)*)

Description

None

Precautions

None

Example

```
create temp stream a2(attr1 int, attr2 string);
```

5.4 Creating a Table

5.4.1 Creating a Redis Table

Create a Redis table to interconnect with the source stream so that data can be output to the DCS Redis table.

For details about DCS, see the **Distributed Cache Service User Guide**.

For details about the JOIN syntax, see **JOIN Between Stream Data and Table Data**.

Syntax

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="")?);

Description

Table 5-22 Parameter description in the syntax

Parameter	Mandatory	Description
type	Yes	Indicates the output channel type. Value **dcs_redis** indicates that data is exported to DCS Redis.
cluster_address	Yes	Indicates the Redis instance connection address.
password	No	Indicates the 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.
key_column	Yes	Indicates the column name of the Redis key attribute.

Parameter	Mandatory	Description
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

1. 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**.

2. In this scenario, jobs must run on the exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with DCS clusters. You can also set the security group rules as required.

   For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service 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 = "xxxxxxxx"
);
```

5.4.2 Creating an RDS Table

Create an RDS table to interconnect with the source stream so that data can be output to RDS. For more information about RDS, see the **Relational Database Service User Guide**.

For details about the JOIN syntax, see **JOIN Between Stream Data and Table Data**.

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 Guide**.

- In this scenario, jobs must run on the exclusive cluster of CS. Therefore, CS must interconnect with the VPC that has been connected with RDS instance. You can also set the security group rules as required.

  For details about how to set up the VPC peering connection, see **VPC Peering Connection** in the *Cloud Stream Service User Guide*.

  For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.

Syntax

Syntax

```
CREATE TABLE  table_id (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "rds",
  region = "",
  username = "",
  password = "",
  db_url = "",
  table_name = ""
);
```

Description

Table 5-23 Parameter description in the syntax

Parameter	Mandatory	Description
type	Yes	Indicates the output channel type. Value **rds** indicates that data is stored to RDS.
username	Yes	Indicates the username of a database.
password	Yes	Indicates the password of a database.
db_url	Yes	Indicates the database connection address, for example, **{database_type}://ip:port/database**. Currently, two types of database connections are supported: MySQL and PostgreSQL. • MySQL: 'mysql://ip:port/database' • PostgreSQL: 'postgresql://ip:port/database'
table_name	Yes	Indicates the name of the database table for data query.

Parameter	Mandatory	Description
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 = "southchina",
  channel = "csinput",
  encode = "csv",
  field_delimiter = ","
);

CREATE TABLE  db_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
WITH (
  type = "rds",
  region = "southchina",
  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 = "cn-north-1",
  channel = "csoutput",
  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;
```


6 DML Statement

6.1 SQL 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 ]* ')'
```

6.2 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 to-be-queried table must exist.
- 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 { 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 statement 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

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;
```

6.3 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;
```


6.4 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 6-1 Array functions

Function Name	Description
TUMBLE(time_attr, interval)	Indicates the tumble window. **time_attr** can be set to **processing-time** or **event-time**. **interval** specifies the window period.
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 6-2 Window functions

Function Name	Description
TUMBLE_START(time_attr, interval)	Indicates the start time of returning to the tumble window.
TUMBLE_END(time_attr, interval)	Indicates the end time of returning to the tumble window.
HOP_START(time_attr, interval, interval)	Indicates the start time of returning to the extended tumble window.
HOP_END(time_attr, interval, interval)	Indicates the end time of returning to the extended tumble window.
SESSION_START(time_attr, interval)	Indicates the start time of returning to the session window.

Function Name	Description
SESSION_END(time_attr, interval)	Indicates the end time of returning to the session window.

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 6-3

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.

Parameter	Description
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
AND CURRENT ROW) as cnt2
   FROM Orders;
```

6.5 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 DDL of the RDS table, see **Creating an RDS Table**.

For details about the DDL of the Redis table, see **Creating a Redis Table**.

Syntax

Syntax

FROM tableExpression JOIN tableExpression ON value11 = value21 [AND value12 = value22]

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 = "cn-north-1",
  channel = "csinput",
  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 = "xxxxxxxx"
);

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 = "cn-north-1",
  channel = "csoutput",
  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";
```

6.6 User-Defined Functions

Overview

CS 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 exclusive clusters.

POM Dependency

```
<dependency>
   <groupId>org.apache.flink</groupId>
   <artifactId>flink-table_2.11</artifactId>
   <version>1.5.0</version>
   <scope>provided</scope>
</dependency>
<dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-streaming-java_2.11</artifactId>
      <version>1.5.0</version>
      <scope>provided</scope>
</dependency>
```

Example

https://github.com/huaweicloud/huaweicloud-cs-sdk/tree/master/huaweicloud-cs-udf-example

Using UDFs

1. Compress the prepared UDFs into a JAR package and upload the package to OBS.
2. In the left navigation pane of the CS management console, click **Job Management**. 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.
3. On the **Running Parameters** page, select the JAR file on OBS for **JAR File to Be Inserted** and click **Save**.

After the JAR package is selected, add the UDF statement to the SQL statement.

```
CREATE FUNCTION udf_test AS 'com.huawei.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.huawei.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.huawei.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.huawei.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```


7 Configuring Time Models

Flink provides two time models: processing time and event time.

CS 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 = "cn-north-1",
  channel = "csinput",
  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 CS 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 the watermark every 10s with the maximum event delay of 20s.

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* Student ID */
  student_name STRING, /* Name */
  subject STRING, /* Subject */
  score INT, /* Score */
  time2 BIGINT
)
WITH (
  type = "dis",
  region = "cn-north-1",
  channel = "csinput",
  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 each time 10 pieces of data are received and the maximum event delay is 20s.

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* Student ID */
  student_name STRING, /* Name */
  subject STRING, /* Subject */
  score INT, /* Score */
  time2 BIGINT
)
WITH (
  type = "dis",
  region = "cn-north-1",
  channel = "csinput",
  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;
```


8 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 ]
    [ SUBSET subsetItem [, subsetItem ]* ]
    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.

Description

Table 8-1 Parameter description in the syntax

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	Chooses summaries or details for each match. • 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. An example is provided as follows: `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` Example description Assume that the format of MyTable is (id, name) and there are three data records: (1, a), (2, b), and (3, c). ONE ROW PER MATCH outputs the average value 2 of B. ALL ROWS PER MATCH outputs each record and the average value of B, specifically, (1,a, null), (2,b,2), (3,c,2).
MEASURES	No	Defines calculations for export from the pattern matching.

Parameter	Mandatory	Description
PATTERN	Yes	Defines the row pattern that will be matched. ● PATTERN (A B C) indicates to detect concatenated events A, B, and C. ● PATTERN (A | B) indicates to detect A or B. ● - A - is valid only in the ALL ROWS PER MATCH. If you specify to exclude a pattern variable from the output, then the row pattern that is matched will not be output. For example: `SELECT * FROM MyTable MATCH_RECOGNIZE ( 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` Example description Assume that the format of MyTable is (id, name) and there are three data records: (1, a), (2, b), and (3, c). Pattern B is excluded. Patterns A, B, and C are detected, but only the following records are output: (1,a, null), (3,c,2), B. ● Modifiers – *: 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 ($n \geq 0$). For example, A{5,} indicates to match A for five or more times. – {n, m}: between *n* and *m* (inclusive) iterations ($0 \leq n \leq m$, $0 < m$). 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.

Parameter	Mandatory	Description
SUBSET	No	Defines union row pattern variables. In this following example, E is a combination of B and C. The average value of E.id is the average value of the subset BC. `SELECT * FROM MyTable MATCH_RECOGNIZE ( MEASURES AVG(E.id) as eid ONE ROW PER MATCH PATTERN (A B C A) SUBSET E = (B,C) DEFINE A AS A.name = 'a', B AS B.name = 'b', C as C.name = 'c' ) MR`
DEFINE	Yes	Defines primary pattern variables.
AFTER MATCH SKIP	No	Defines where to restart the matching process after a match is found. • 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 8-2 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.

Function	Description
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 about aggregation functions, 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

- Alarm suppression. If event A is consecutively reported for multiple times, only event A is reported only once.

```
INSERT INTO inhibition
SELECT * FROM event MATCH_RECOGNIZE
(
  MEASURES FIRST(B.event_name) as Bname
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  PATTERN (B+?)
  DEFINE
    B AS B.event_name <> PREV(B.event_name) or PREV(B.event_name) is null
) MR;
```

This statement indicates that an event that is different from the previous one is reported, while a detected duplicate event is not reported.

Input data:

1,A

2,A

3,A

4,B

5,B

6,C

7,D

8,D

Output data:

A

B

C

D

9 StreamingML

9.1 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.

CS uses an anomaly detection method based on the random forest, which has the following characteristics:

- The one-pass algorithm is used with O(1) amortised 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,..."

Parameters

Table 9-1 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. • 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 '300' SECOND PRECEDING AND CURRENT ROW) > 0.8
        THEN 'anomaly'
        ELSE 'not anomaly'
   END
FROM MyTable
```

9.2 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. CS 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 9-2 Parameter description

Parameter	Mandatory	Description	Default Value
field	Yes	Name of the field, where data is used for forecasting is located, 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
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 9-3 Parameter description

Parameter	Mandatory	Description
field	Yes	Name of the field, where data is used for forecasting is located, 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 hw1,
   HOLT_WINTERS(b, 5, 2) OVER (ORDER BY rowtime ROWS  BETWEEN 5 PRECEDING AND CURRENT
ROW) AS hw2,
FROM MyTable
```


9.3 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, CS 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**.

Parameters

Table 9-4 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
```

9.4 Deep Learning Model Prediction

Deep learning has a wide range of applications in many industries, such as image classification, image recognition, and speech recognition. CS 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**.myhuaweicloud.com:443/**your_model_path**. For example, if your model is stored on OBS deployed in the **cn-north-1** region, the bucket name is **dl_model**, and the file name is **model.h5**, set the path to obs://**your_ak**:**your_sk**@obs.cn-north-1.myhuaweicloud.com:443/dl_model/model.h5.

Parameters

Table 9-5 Parameter description

Parameter	Mandatory	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
```


10 Reserved Keywords

Stream 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
- CARBON_PROPERTIES
- CARDINALITY
- CASCADE
- CASCADED

- CASE
- CAST
- CATALOG
- CATALOG_NAME
- CEIL
- CEILING
- CENTURY
- CHAIN
- CHANNEL
- CHAR
- CHARACTER
- CHARACTERISTICTS
- 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
- MULTISET
- 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
- 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

