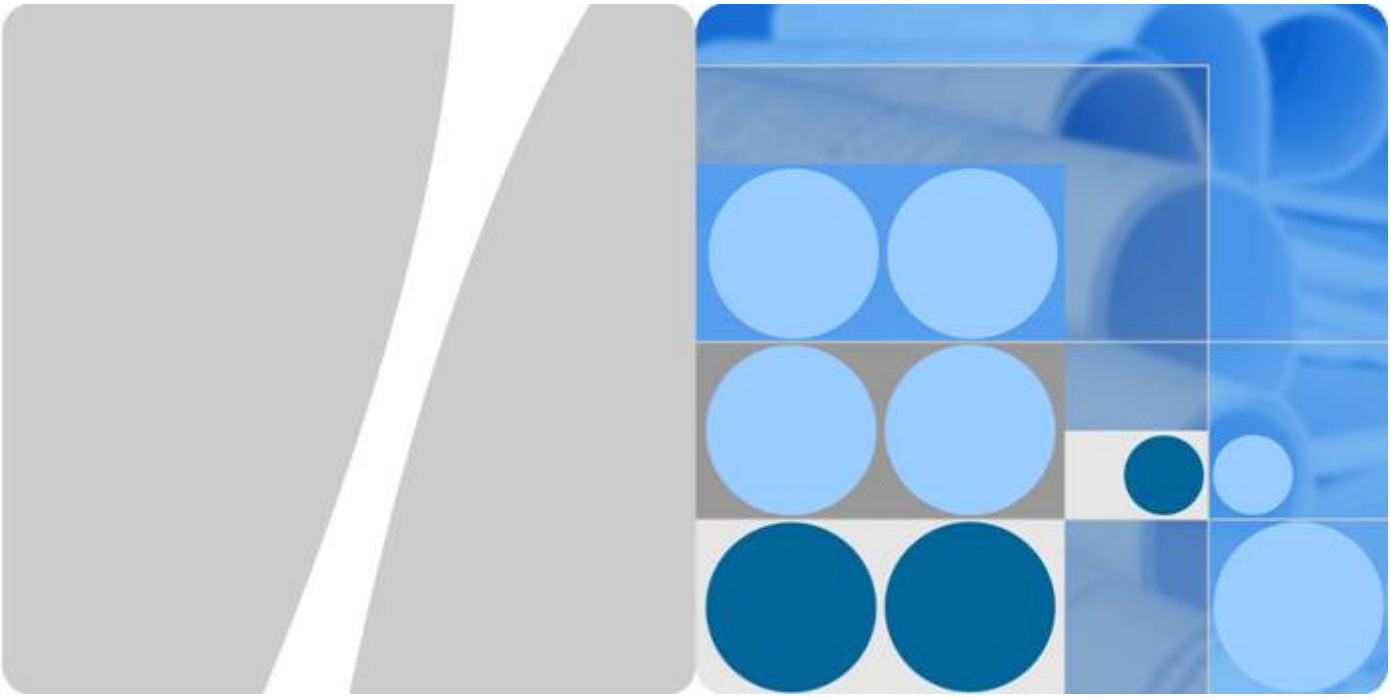

实时流计算服务

SQL 语法参考

文档版本　11

发布日期　2020-09-27

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 语法约束

- 当前Stream SQL只支持SELECT，FROM，WHERE，UNION，聚合，窗口，流表JOIN以及流流JOIN。
- 数据不能对Source流做insert into操作。
- Sink流不能用来做查询操作。

语法支持范围

- 基础类型： VARCHAR，STRING，BOOLEAN，TINYINT，SMALLINT，INTEGER/INT，BIGINT，REAL/FLOAT，DOUBLE，DECIMAL，DATE，TIME，TIMESTAMP
- Array：使用[]进行引用。例如：
  ```
  insert into temp select CARDINALITY(ARRAY[1,2,3]) FROM OrderA;
  ```

2 数据类型

概述

数据类型是数据的一个基本属性，用于区分不同类别的数据。不同的数据类型所占的存储空间不同，能够进行的操作也不相同。数据库中的数据存储在数据表中。数据表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。

华为大数据平台的Stream SQL与开源社区相同，支持原生数据类型和复杂数据类型。

原生数据类型

Stream SQL支持原生数据类型，请参见**表2-1**。

表 2-1 原生数据类型

数据类型	描述	存储空间	范围
VARCHAR	可变长度的字符	-	-
BOOLEAN	布尔类型	-	TRUE/FALSE
TINYINT	有符号整数	1字节	-128-127
SMALLINT	有符号整数	2字节	-32768-32767
INT	有符号整数	4字节	-2147483648～2147483647
INTEGER	有符号整数	4字节	-2147483648～2147483647
BIGINT	有符号整数	8字节	-9223372036854775808～9223372036854775807
REAL	单精度浮点型	4字节	-
FLOAT	单精度浮点型	4字节	-
DOUBLE	双精度浮点型	8字节	-

数据类型	描述	存储空间	范围
DECIMAL	固定有效位数和小数位数的数据类型	-	-
DATE	日期类型，描述了特定的年月日，以yyyy-MM-dd格式表示，例如2014-05-29	-	DATE类型不包含时间，所表示日期的范围为0000-01-01 to 9999-12-31
TIME	时间类型，以HH:mm:ss表示。 例如20:17:40	-	-
TIMESTAMP(3)	完整日期，包括日期和时间。 例如：1969-07-20 20:17:40	-	-
INTERVAL timeUnit [TO timeUnit]	时间间隔 例如：INTERVAL '1:5' YEAR TO MONTH, INTERVAL '45' DAY	-	-

复杂数据类型

Stream SQL支持复杂数据类型，如表2-2所示。

表 2-2 复杂数据类型

数据类型	描述
ARRAY	一组有序字段，所有字段的数据类型必须相同。
MAP	一组无序的键/值对。键的类型必须是原生数据类型，值的类型可以是原生数据类型或复杂数据类型。同一个MAP键的类型必须相同，值的类型也必须相同。

3 内置函数

3.1 数学运算函数

关系运算符

所有数据类型都可用关系运算符进行比较，并返回一个BOOLEAN类型的值。

关系运算符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

Stream SQL提供的关系运算符，请参见表3-1。

表 3-1 关系运算符

运算符	返回类型	描述
A = B	BOOLEAN	若A与B相等，返回TRUE，否则返回FALSE。用于做赋值操作。
A <> B	BOOLEAN	若A与B不相等，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL，该种运算符为标准SQL语法。
A < B	BOOLEAN	若A小于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A <= B	BOOLEAN	若A小于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A > B	BOOLEAN	若A大于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A >= B	BOOLEAN	若A大于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A IS NULL	BOOLEAN	若A为NULL则返回TRUE，否则返回FALSE。
A IS NOT NULL	BOOLEAN	若A不为NULL，则返回TRUE，否则返回FALSE。

运算符	返回类型	描述
A IS DISTINCT FROM B	BOOLEAN	若A与B不相等，则返回TRUE，将空值视为相同。
A IS NOT DISTINCT FROM B	BOOLEAN	若A与B相等，则返回TRUE，将空值视为相同。
A BETWEEN [ASYMMETRIC | SYMMETRIC] B AND C	BOOLEAN	若A大于或等于B且小于或等于C，则返回TRUE。 ● ASYMMETRIC：表示B和C位置相关。 例如：A BETWEEN ASYMMETRIC B AND C 等价于 (A BETWEEN B AND C)。 ● SYMMETRIC：表示B和C位置不相关。 例如：A BETWEEN SYMMETRIC B AND C 等价于 (A BETWEEN B AND C) OR (A BETWEEN C AND B)。
A NOT BETWEEN B AND C	BOOLEAN	若A小于B或大于C，则返回TRUE。
A LIKE B [ESCAPE C]	BOOLEAN	若A与模式B匹配，则返回TRUE。 必要时可以定义转义字符C。
A NOT LIKE B [ESCAPE C]	BOOLEAN	若A与模式B不匹配，则返回TRUE。 必要时可以定义转义字符C。
A SIMILAR TO B [ESCAPE C]	BOOLEAN	若A与正则表达式B匹配，则返回TRUE。 必要时可以定义转义字符C。
A NOT SIMILAR TO B [ESCAPE C]	BOOLEAN	若A与正则表达式B不匹配，则返回TRUE。 必要时可以定义转义字符C。
value IN (value [, value]*)	BOOLEAN	若值等于列表中的值，则返回TRUE。
value NOT IN (value [, value]*)	BOOLEAN	若值不等于列表中的每个值，则返回TRUE。

注意事项

- double、real和float值存在一定的精度差。且我们不建议直接使用等号“=”对两个double类型数据进行比较。用户可以使用两个double类型相减，而后取绝对值的方式判断。当绝对值足够小时，认为两个double数值相等，例如:


```
abs(0.9999999999 - 1.0000000000) < 0.000000001 //0.9999999999和1.0000000000为10位精度，而0.000000001为9位精度，此时可以认为0.9999999999和1.0000000000相等。
```

- 数值类型可与字符串类型进行比较。做大小(>,<,>=,<=)比较时，会默认将字符串转换为数值类型，因此不支持字符串内有除数字字符之外的字符。
- 字符串之间可以进行比较。

逻辑运算符

常用的逻辑操作符有AND、OR和NOT，优先级顺序为：NOT>AND>OR。

运算规则请参见**表3-2**，表中的A和B代表逻辑表达式。

表 3-2 逻辑运算符

运算符	返回类型	描述
A OR B	BOOLEAN	若A或B为TRUE，则返回TRUE，且支持三值逻辑。
A AND B	BOOLEAN	若A和B为TRUE，则返回TRUE，且支持三值逻辑。
NOT A	BOOLEAN	若A不为TRUE则返回TRUE；若A为UNKNOWN，返回UNKNOWN。
A IS FALSE	BOOLEAN	若A为FALSE则返回TRUE； 若A为UNKNOWN，则返回FALSE。
A IS NOT FALSE	BOOLEAN	若A不为FALSE则返回TRUE；若A为UNKNOWN，则返回TRUE。
A IS TRUE	BOOLEAN	若A为TRUE，则返回TRUE；若A为UNKNOWN，则返回FALSE。
A IS NOT TRUE	BOOLEAN	若A不为TRUE则返回TRUE；若A为UNKNOWN，则返回TRUE。
A IS UNKNOWN	BOOLEAN	若A为UNKNOWN，则返回TRUE。
A IS NOT UNKNOWN	BOOLEAN	若A不为UNKNOWN，则返回TRUE。

注意事项

逻辑操作符只允许boolean类型参与运算，不支持隐式类型转换。

算术运算符

算术运算符包括双目运算符与单目运算符，这些运算符都将返回数字类型。Stream SQL所支持的算术运算符如**表3-3**所示。

表 3-3 算术运算符

运算符	返回类型	描述
+ numeric	所有数字类型	返回数字。
- numeric	所有数字类型	返回负数。
A + B	所有数字类型	A和B相加。结果数据类型与操作数据类型相关，例如一个整数类型数据加上一个浮点类型数据，结果数值为浮点类型数据。
A - B	所有数字类型	A和B相减。结果数据类型与操作数据类型相关。
A * B	所有数字类型	A和B相乘。结果数据类型与操作数据类型相关。
A / B	所有数字类型	A和B相除。结果是一个double（双精度）类型的数值。
POWER(A, B)	所有数字类型	返回A数的B次方乘幂。
ABS(numeric)	所有数字类型	返回数值的绝对值。
MOD(A, B)	所有数字类型	返回A除以B的余数（模数）。 返回值只有在A为负数时才为负数。
SQRT(A)	所有数字类型	返回A的平方根。
LN(A)	所有数字类型	返回A的自然对数（基数e）。
LOG10(A)	所有数字类型	返回A的基数10对数。
EXP(A)	所有数字类型	返回e的a次方。
CEIL(A) CEILING(A)	所有数字类型	将参数向上舍入为最接近的整数。例如ceil(21.2)，返回22。
FLOOR(A)	所有数字类型	对给定数据进行向下舍入最接近的整数。例如floor(21.2)，返回21。
SIN(A)	所有数字类型	计算给定A的正弦值。
COS(A)	所有数字类型	计算给定A的余弦值。

运算符	返回类型	描述
TAN(A)	所有数字类型	计算给定A的正切值。
COT(A)	所有数字类型	计算给定A的余切值。
ASIN(A)	所有数字类型	计算给定A的反正弦值。
ACOS(A)	所有数字类型	计算给定A的反余弦值。
ATAN(A)	所有数字类型	计算给定A的反正切值。
DEGREES(A)	所有数字类型	返回弧度所对应的角度。
RADIANS(A)	所有数字类型	返回角度所对应的弧度。
SIGN(A)	所有数字类型	返回a所对应的正负号，a为正返回1，a为负，返回-1，否则则返回0。
ROUND(A, d)	所有数字类型	返回小数部分d位之后数字的四舍五入，d为int型。例如round(21.263,2)，返回21.26。
PI()	所有数字类型	返回pi的值。

注意事项

字符串类型不能参与算术运算。

3.2 字符串函数

字符串运算符

常用的字符串运算符运算规则请参见表3-4，表中的A和B代表字符串表达式。

表 3-4 字符串运算符

运算符	返回类型	描述
A || B	STRING	A字符串和B字符串的拼接。
CHAR_LENGTH(A)	INT	返回A字符串中的字符数。

运算符	返回类型	描述
CHARACTER_LENGTH(A)	INT	返回A字符串中的字符数。
UPPER(A)	STRING	返回大写字母A。
LOWER(A)	STRING	返回小写字母a。
POSITION(A IN B)	INT	返回B中第一次出现A的位置。
TRIM({ BOTH | LEADING | TRAILING } A FROM B)	STRING	从B中除去字符串首尾/首位/末尾的A。 默认情况下，首尾的A都被删除 。
OVERLAY(A PLACING B FROM integer [FOR B])	STRING	将B替换A的子字符串。
SUBSTRING(A FROM integer)	STRING	返回从给定位置开始的A的子字符串。起始位置从1开始。
SUBSTRING(A FROM integer FOR integer)	STRING	返回从给定位置开始，给定长度的A的子字符串。起始位置从1开始。
INITCAP(A)	STRING	返回字符串，将单词首字母转换为大写，其余为小写。 单词是由非字母、数字、字符分隔的字母、数字、字符序列。
MD5(String expr)	STRING	返回字符串的md5值。
SHA1(String expr)	STRING	返回字符串的SHA1值。
SHA256(String expr)	STRING	返回字符串的SHA256值。
replace(String expr, String toreplace, String replace)	STRING	字符串替换函数，将字符串expr中的所有toreplace替换成replace。
hash_code(String expr)	INT	获取哈希值，参数除string外，也支持int/bigint/float/double。
string_to_array(value, delimiter)	Array[String]	将字符串value按delimiter分隔为字符串数组。
CONCAT(String A, String B, …)	STRING	返回两个或多个字符串的拼接。
CONCAT_WS(String separator, String A, String B, ...)	STRING	返回两个或多个字符串的拼接，并使用separator作为分隔符连接各个字符串。

运算符	返回类型	描述
RPAD(String str, INT len, String pad)	STRING	将pad字符串拼接到str字符串的的右端，直到新的字符串达到指定长度len为止。 ● len为负数时返回为null； ● len小于str长度，返回str裁剪为len长度的字符串。
LPAD(String str, INT len, String pad)	STRING	将pad字符串拼接到str字符串的的左端，直到新的字符串达到指定长度len为止。 ● len为负数时返回为null； ● len小于str长度，返回str裁剪为len长度的字符串

3.3 时间函数

Stream SQL所支持的时间函数如表3-5所示。

表 3-5 时间函数

函数	返回值	描述
DATE string	DATE	将日期字符串以“yyyy-MM-dd”的形式解析为SQL日期。
TIME string	TIME	将时间字符串以“HH:mm:ss”形式解析为SQL时间。
TIMESTAMP string	TIMESTAMP	将时间字符串转换为时间戳，时间字符串格式为：“yyyy-MM-dd HH:mm:ss.fff”。
INTERVAL string range	INTERVAL	interval表示时间间隔，有两种类型，一种为“yyyy-MM”，即保存年份和月份，精度到月份，它的Range可以为YEAR或者YEAR To Month；一种为天 时间(“dd HH:mm:sss.fff'”)，用来保存天数、小时、分钟、秒和毫秒， 精度最低到毫秒， 他的range可以为DAY TO HOUR，DAY TO MINUTE，DAY TO SECOND或DAY TO milliseconds，比如range为DAY TO SECOND就表示天数、小时、分钟、秒的位置都有效，精度到秒，DAY TO MINUTE就表示精度到分钟。 例如： INTERVAL '10 00:00:00.004' DAY TO milliseconds表示间隔10天4毫秒， INTERVAL '10' DAY表示间隔10天, INTERVAL '2-10' YEAR TO MONTH表示间隔2年10个月。
CURRENT_DATE	DATE	以UTC时区返回当前SQL日期。

函数	返回值	描述
CURRENT_TIME	TIME	以UTC时区返回当前SQL时间。
CURRENT_TIMESTAMP	TIMESTAMP	以UTC时区返回当前SQL时间戳。
LOCALTIME	TIME	返回当前时区的当前SQL时间。
LOCALTIMESTAMP	TIMESTAMP	返回当前时区的当前SQL时间戳。
EXTRACT(timeintervalunit FROM temporal)	INT	提取时间点的一部分或者时间间隔。以int类型返回该部分。 例如：提取日期“2006-06-05”中的日为5。
FLOOR(timepoint TO timeintervalunit)	TIME	向下对齐时间。 例如：FLOOR(TIME '12:44:31' TO MINUTE)按分钟对齐到12:44:00。
CEIL(timepoint TO timeintervalunit)	TIME	向上对齐时间。 例如：CEIL(TIME '12:44:31' TO MINUTE)按分钟对齐到12:45:00。
QUARTER(date)	INT	从SQL日期返回一年的四分之一。
(timepoint, temporal) OVERLAPS (timepoint, temporal)	BOOLEAN	确定两个时间间隔是否重叠。时间点和时间被转换成在两个时间点（开始，结束）定义的范围之内，该计算函数是***leftEnd >= rightStart && rightEnd >= leftStart***。当左边结束时间点大于等于右边开始时间点，且右边结束时间点大于等于左边开始时间点，则函数返回true值，否则返回false。 例如： ● 左边的结束时间点是“3:55:00”（2:55:00+1:00:00）大于右边的开始点是“3:30:00”；且右边的结束时间点是“5:30:00”（3:30:00+2:00:00）大于左边开始时间点“2:55:00”，则返回值为true。(TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) 返回值是true。 ● 左边的结束时间点是“10:00:00”小于右边的开始点是“10:15:00”；且右边的结束时间点是“13:15:00”（10:15:00+3:00:00）大于左边开始时间点“9:00:00”，则返回值为false。(TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) 返回值是false。
TO_LOCALTIMESTAMP(long expr)	TIMESTAMP	转换为本地时间。

函数	返回值	描述
UNIX_TIMESTAMP	BIGINT	返回指定参数的时间戳，时间戳类型为BIGINT类型，单位为秒。 支持如下几种使用方法： ● UNIX_TIMESTAMP()：没有参数时，返回当前时间的时间戳。 ● UNIX_TIMESTAMP(STRING datestr)：包含一个参数时，返回参数所表示的时间戳，datestr格式必须为yyyy-MM-dd HH:mm:ss。 ● UNIX_TIMESTAMP(STRING datestr, STRING format)：包含两个参数时，第二个参数可以指定datestr的格式，返回第一个参数所表示的时间戳。
UNIX_TIMESTAMP_MS	BIGINT	返回指定参数的时间戳，时间戳类型为BIGINT类型，单位为毫秒。 支持如下几种使用方法： ● UNIX_TIMESTAMP_MS()：没有参数时，返回当前时间的时间戳。 ● UNIX_TIMESTAMP_MS(STRING datestr)：包含一个参数时，返回参数所表示的时间戳，datestr格式必须为yyyy-MM-dd HH:mm:ss.SSS。 ● UNIX_TIMESTAMP_MS(STRING datestr, STRING format)：包含两个参数时，第二个参数可以指定datestr的格式，返回第一个参数所表示的时间戳。

注意事项

无。

示例

```
insert into temp SELECT Date '2015-10-11'  FROM  OrderA;//返回日期
insert into temp1 SELECT Time '12:14:50'  FROM  OrderA;//返回时间
insert into temp2 SELECT Timestamp '2015-10-11 12:14:50'  FROM  OrderA;//返回时间戳
```

3.4 类型转换函数

语法格式

CAST(value AS type)

语法说明

类型强制转换。

注意事项

若输入为NULL，则返回NULL。

示例

将amount值转换成字符串，长度为转换后的实际长度，配置的长度无效。

```
insert into temp select cast(amount as VARCHAR(10)) from source_stream;
```

表 3-6 类型转换函数示例

示例	说明
cast(v1 as varchar)	将v1转换为字符串类型，v1可以是数值类型，TIMESTAMP/DATE/TIME
cast (v1 as int)	将v1转换为int, v1可以是数值类型或字符类
cast(v1 as timestamp)	将v1转换为timestamp类型，v1可以是字符串或者bigint类型或DATE/TIME
cast(v1 as date)	将v1转换为date类型， v1可以是字符串或者TIMESTAMP

3.5 聚合函数

聚合函数是从一组输入值计算一个结果。例如使用COUNT函数计算SQL查询语句返回的记录行数。聚合函数如表3-7所示。

表 3-7 聚合函数表

函数	返回值类型	描述
COUNT(value [, value]*)	DOUBLE	返回值不为空的个数。
COUNT(*)	BIGINT	返回元组个数。
AVG(numeric)	DOUBLE	返回所有输入值的数字的平均值（算术平均值）。
SUM(numeric)	DOUBLE	返回所有输入值之间的数值之和。
MAX(value)	DOUBLE	返回所有输入值的值的最大值。
MIN(value)	DOUBLE	返回所有输入值的值的最小值。
STDDEV_POP(value)	DOUBLE	返回所有输入值之间的数字字段的总体标准偏差。
STDDEV_SAMP(value)	DOUBLE	返回所有输入值之间的数字字段的样本标准偏差。
VAR_POP(value)	DOUBLE	返回所有输入值之间的数字字段的总体方差（总体标准偏差的平方）。
VAR_SAMP(value)	DOUBLE	返回所有输入值之间的数字字段的样本方差（样本标准偏差的平方）。

注意事项

无。

示例

无。

3.6 表值函数

表值函数可以将一行转多行，一列转为多列，仅支持在JOIN LATERAL TABLE中使用。

表 3-8 表值函数表

函数	返回值类型	描述
split_cursor(value, delimiter)	cursor	将字符串value按delimiter分隔为多行字符串。

示例

输入一条记录("student1", "student2, student3")，输出两条记录("student1", "student2") 和 ("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 其他函数

数组函数

表 3-9 数组函数表

函数	返回值类型	描述
CARDINALITY(ARRAY)	INT	返回数组的元素个数。
ELEMENT(ARRAY)	-	使用单个元素返回数组的唯一元素。 如果数组为空，则返回null。如果数组有多个元素，则抛出异常。

注意事项

无。

示例

返回数组的元素个数为3。

```
insert into temp select CARDINALITY(ARRAY[TRUE, TRUE, FALSE]) from source_stream;
```


返回'HELLO WORLD'。

```
insert into temp select ELEMENT(ARRAY['HELLO WORLD']) from source_stream;
```

属性访问函数

表 3-10 属性访问函数表

函数	返回值类型	描述
tableName.compositeType.field	-	选择单个字段，通过名称访问Apache Flink复合类型（如Tuple，POJO等）的字段并返回其值。
tableName.compositeType.*	-	选择所有字段，将Apache Flink复合类型（如Tuple，POJO等）和其所有直接子类型转换为简单表示，其中每个子类型都是单独的字段。

注意事项

无。

示例

无。

4 地理函数

函数说明

基本地理空间几何元素介绍说明如表4-1所示。

表 4-1 基本地理空间几何元素表

地理空间几何元素（统称geometry）	说明	举例
ST_POINT(latitude, longitude)	地理点，包含经度和维度两个信息。	ST_POINT(1.12012, 1.23401)
ST_LINE(array[point1...pointN])	地理线，由多个地理点（ST_POINT）按顺序连接成的折线或直线。	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])	地理多边形，由首尾相同的多个地理点（ST_POINT）按顺序连线围成的封闭多边形区域。	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)	地理圆形，由圆心地理点（ST_POINT）和半径构成的地理圆形区域。	ST_CIRCLE(ST_POINT(1.0, 1.0), 1.234)

用户可以以基本地理空间几何元素为基础，构造复杂的地理空间几何元素，具体的变换方法见表4-2。

表 4-2 基于基本地理空间几何元素构造复杂几何元素的变换表

变换方法	说明	举例
ST_BUFFER(geometry, distance)	创建一个环绕包含给定地理空间几何元素的多边形，并以给定距离作为环绕距离，通常使用该函数构造一定宽度的公路范围用于偏航检测。	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)	创建一个多边形，其范围为给定的两个地理空间几何元素的交叠区域。	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)	创建一个包含给定的地理空间几何元素的最小矩形。	ST_ENVELOPE(ST_CIRCLE(ST_POINT(1.0, 1.0), 2.0))

CS提供丰富的对地理空间几何元素的操作和位置判断函数，具体的SQL标量函数介绍说明见**表4-3**。

表 4-3 SQL 标量函数表

函数	返回值	说明
ST_DISTANCE(point_1, point_2)	DOUBLE	计算两个地理点之间的欧几里得距离。 示例如下： Select ST_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input
ST_GEODESIC_DISTANCE(point_1, point_2)	DOUBLE	计算两个地理点之间的测地距离，即两个地理点之间地表最短路径距离。 示例如下： Select ST_GEODESIC_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input
ST_PERIMETER(polygon)	DOUBLE	计算多边形的周长。 示例如下： Select ST_PERIMETER(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]) FROM input

函数	返回值	说明
ST_AREA(polygon)	DOUBLE	计算多边形区域的面积。 示例如下： 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	判断一个多边形是否与另一个多边形有重叠区域。 示例如下： 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	检查两条线段是否相互交叉，而非线条所在的直线是否交叉。 示例如下： 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	一个点是否包含在几何体（多边形或圆形）内。 示例如下： SELECT ST_WITHIN(ST_POINT(x11, y11), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_CONTAINS(polygon_1, polygon_2)	BOOLEAN	判断第一个几何体是否包含第二个几何体。 示例如下： 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

函数	返回值	说明
ST_COVERS(polygon_1, polygon_2)	BOOLEAN	第一个几何体是否覆盖第二个几何体。与ST_CONTAINS相似，但在边界重叠情况下ST_COVER判断为TRUE，ST_CONTAINS判断为FALSE。 示例如下： 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	判断一个多边形是否与另一个多边形不相交（不重叠）。 示例如下： 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

地理函数的基准坐标系标准为全球通用的GPS坐标系标准WGS84，GPS坐标不能直接在百度地图（BD09标准）或者google地图（GCJ02标准）上使用，会有偏移现象，为了在不同地理坐标系之间切换，CS提供了坐标系转换的一系列函数，并且还提供地理距离与米之间的转换函数。详见**表4-4**。

表 4-4 地理坐标系转换函数与距离单位转换函数表

函数	返回值	说明
WGS84_TO_BD09(geometry)	对应的百度地图坐标系地理空间几何元素	将GPS坐标系下的地理空间几何元素转换成百度地图坐标系下对应的地理空间几何元素。示例如下： WGS84_TO_BD09(ST_CIRCLE(ST_POINT(x, y), r))
WGS84_TO_CJ02(geometry)	对应的Google地图坐标系地理空间几何元素	将GPS坐标系下的地理空间几何元素转换成Google地图坐标系下对应的地理空间几何元素。示例如下： WGS84_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))

函数	返回值	说明
BD09_TO_WGS84(geometry)	对应的GPS坐标系地理空间几何元素	将百度地图坐标系下的地理空间几何元素转换成GPS坐标系下对应的地理空间几何元素。示例如下： BD09_TO_WGS84(ST_CIRCLE(ST_POINT(x, y), r))
BD09_TO_CJ02(geometry)	对应的Google地图坐标系地理空间几何元素	将百度地图坐标系下的地理空间几何元素转换成Google地图坐标系下对应的地理空间几何元素。示例如下： BD09_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))
CJ02_TO_WGS84(geometry)	对应的GPS坐标系地理空间几何元素	将Google地图坐标系下的地理空间几何元素转换成GPS坐标系下对应的地理空间几何元素。示例如下： CJ02_TO_WGS84(ST_CIRCLE(ST_POINT(x, y), r))
CJ02_TO_BD09(geometry)	对应的百度地图坐标系地理空间几何元素	将Google地图坐标系下的地理空间几何元素转换成百度地图坐标系下对应的地理空间几何元素。示例如下： CJ02_TO_BD09(ST_CIRCLE(ST_POINT(x, y), r))
DEGREE_TO_METER(distance)	DOUBLE	将地理函数的距离数值转换成以“米”为单位的数值。示例如下（以米为单位计算地理三边形周长）： DEGREE_TO_METER(ST_PERIMETER(ST_POLYGON(ARRAY[ST_POINT(x1,y1), ST_POINT(x2,y2), ST_POINT(x3,y3), ST_POINT(x1,y1)])))

函数	返回值	说明
METER_TO_DEGREE(numerical_value)	DOUBLE	将以“米”为单位的数值转换成地理函数可计算的距离单位数值。示例如下（画出以指定地理点为圆心，半径1公里的圆）： ST_CIRCLE(ST_POINT(x,y), METER_TO_DEGREE(1000))

CS还提供了基于窗口的SQL地理聚合函数用于SQL逻辑涉及窗口和聚合的场景。详见表4-5的介绍说明。

表 4-5 时间相关 SQL 地理聚合函数表

函数	说明	举例
AGG_DISTANCE(point)	距离聚合函数，用于计算窗口内所有相邻地理点的距离总和。	SELECT AGG_DISTANCE(ST_POINT(x,y)) FROM input GROUP BY HOP(rowtime, INTERVAL '1' HOUR, INTERVAL '1' DAY)
AVG_SPEED(point)	平均速度聚合函数，用于计算窗口内所有地理点组成的移动轨迹的平均速度，单位为“米/秒”。	SELECT AVG_SPEED(ST_POINT(x,y)) FROM input GROUP BY TUMBLE(proctime, INTERVAL '1' DAY)

注意事项

无。

示例

偏航检测样例：

```
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(34.588691,105.74997)]),0.001));
```

IP 地理函数

表 4-6 IP 地理函数表

函数	返回值	说明
IP_TO_COUNTRY	STRING	获取IP地址所在的国家名称。以中文形式返回。
IP_TO_PROVINCE	STRING	获取IP地址所在的省份。 用法说明： ● IP_TO_PROVINCE(STRING ip)：以中文形式，返回IP地址所在的省份。 ● IP_TO_PROVINCE(STRING ip, STRING lang)：以指定语言返回IP地址所在的省份。如果lang为EN，则返回英文名称；如果为CN或者其他，则返回中文名称。 **说明** ● 当IP无法被解析到省份时，返回该IP所属的国家。当IP无法被解析时，返回“未知”。 ● 函数返回的国内省份中文或者英文名称均为简称，和中国政府官网一致。 中文参考如下链接：http://www.gov.cn/guoqing/2005-09/13/content_5043917.htm 英文参考如下链接：http://english.gov.cn/archive/
IP_TO_CITY	STRING	获取IP地址所在的城市名称。以中文形式返回。 **说明** 当IP无法被解析到城市时，返回该IP所属的省份或者国家。当IP无法被解析时，返回“未知”。
IP_TO_CITY_GEO	STRING	获取IP地址所在城市的经纬度，格式为“纬度,经度”。 用法说明： IP_TO_CITY_GEO(STRING ip)：返回IP所在城市的经纬度。

5 数据定义语句

5.1 创建输入流

5.1.1 DIS 输入流

概述

创建source流从数据接入服务（DIS）获取数据。用户数据从DIS接入，CS从DIS的通道读取数据，作为作业的输入数据。CS可通过DIS的source源将数据从生产者快速移出，进行持续处理，适用于将云服务外数据导入云服务后进行过滤、实时分析、监控报告和转储等场景。

数据接入服务（Data Ingestion Service，简称DIS）为处理或分析流数据的自定义应用程序构建数据流管道，主要解决云服务外的数据实时传输到云服务内的问题。数据接入服务每小时可从数十万种数据源（如IoT数据采集、日志和定位追踪事件、网站点击流、社交媒体源等）中连续捕获、传送和存储数TB数据。DIS的更多信息，请参见《数据接入服务用户指南》。

语法

语法格式

```
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
```

语法说明

表 5-1 语法说明

参数	是否必选	说明
type	是	数据源类型，“dis”表示数据源为数据接入服务。

参数	是否必选	说明
region	是	数据所在的DIS区域。
channel	是	数据所在的DIS通道名称。
partition_count	否	数据所在的DIS通道分区数。该参数和partition_range参数不能同时配置。当该参数没有配置的时候默认读取所有partition。
partition_range	否	指定作业从DIS通道读取的分区范围。该参数和partition_count参数不能同时配置。当该参数没有配置的时候默认读取所有partition。 partition_range = "[2:5]"时，表示读取的分区范围是2-5，包括分区2和分区5。
encode	是	数据编码格式，可选为“csv”、“json”、“xml”、“email”、“blob”和“user_defined”。 **说明** • 若编码格式为“csv”，则需配置“field_delimiter”属性。 • 若编码格式为“json”，则需配置“json_config”属性。 • 若编码格式为“xml”，则需配置“xml_config”属性。 • 若编码格式为“email”，则需配置“email_key”属性。 • 若编码格式为“blob”，表示不对接收的数据进行解析，流属性仅能有一个且数据格式为ARRAY[TINYINT]。 • 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
field_delimiter	否	属性分隔符，仅当编码格式为csv时，用户可以自定义属性分隔符，默认为“，”。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 • 当引用符号为双引号时，请设置quote = "\u005c \u0022"进行转义。 • 当引用符号为单引号时，则设置quote = "'"。 **说明** 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。
json_config	否	当编码格式为json时，用户需要通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1; field2=data_json.field2; field3=$”，其中field3=$表示field3的内容为整个json串。
xml_config	否	当编码格式为xml时，用户需要通过该参数来指定xml字段和流定义字段的映射关系，格式为“field1=data_xml.field1; field2=data_xml.field2”。

参数	是否必选	说明
email_key	否	当编码格式为email时，用户需要通过该参数来指定需要提取的信息，需要列出信息的key值，需要与流定义字段一一对应，多个key值时以逗号分隔，例如“Message-ID, Date, Subject, body”，其中由于邮件正文没有关键字，CS规定其关键字为“body”。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现解码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现解码类的入参，仅支持一个string类型的参数。
offset	否	• 当启动作业后再获取数据，则该参数无效。 • 当获取数据后再启动作业，用户可以根据需求设置该参数的数值。 例如当offset= "100"时，则表示CS服务从DIS服务中的第100条数据开始处理。
start_time	否	DIS数据读取起始时间。 • 当该参数配置时则从配置的时间开始读取数据，有效格式为yyyy-MM-dd HH:mm:ss。 • 当没有配置start_time也没配置offset的时候，读取最新数据。 • 当没有配置start_time但配置了offset的时候，则从offset开始读取数据。
timeindicator	否	在流中增加时间戳，可增加“processing time”时间戳或者“event time”时间戳。 **说明** • 若设置“processing time”，则为proctime.proctime。当设置了proctime.proctime时，会在原有属性字段基础上多增加一个proctime系统时间戳属性，假设原有字段为3个，设置了proctime.proctime后会变成4个，设置rowtime属性字段不会发生变化。 • 若设置“event time”，可选择流中的某个属性来作为时间戳，格式为attr_name.rowtime。 • 以上两者可以同时设置。
enable_checkpoint	否	是否启用checkpoint功能，可配置为true（启用）或者false（停用）， 默认为false。
checkpoint_app_name	否	DIS服务的消费者标识，当不同作业消费相同通道时，需要区分不同的消费者标识，以免checkpoint混淆。
checkpoint_interval	否	DIS源算子做checkpoint的时间间隔，单位秒，默认为60。

注意事项

用来做时间戳的属性类型必须为long或者timestamp。

示例

- CSV编码格式：从DIS通道读取数据，记录为csv编码，并且以逗号为分隔符。

```
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;
```

- JSON编码格式：从DIS通道读取数据，记录为json编码。数据示例：{"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;
```

- XML编码格式：从DIS通道读取数据，记录为xml编码。

```
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"
);
```

xml数据示例如下：

```
<?xml version="1.0" encodeing="utf-8"?>
```



```
<root>
 <person>
  <pid>362305199010025042</pid>
  <pname>xiaoming</pname>
  <page>28</page>
  <plocation>内蒙古,乌兰察布市,集宁区,商都县</plocation>
  <pbir>1990-10-02</pbir>
  <phealthy>true</phealthy>
  <pgrade>[A,B,C]</pgrade>
 </person>
</root>
```

- EMAIL编码格式：从DIS通道读取数据，每条记录为一封完整邮件。

```
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"
);
```

email数据示例如下：

```
Message-ID: <20090629183903250 4254@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 输入流

概述

创建source流从对象存储服务（OBS）获取数据。CS从OBS上读取用户存储的数据，作为作业的输入数据。适用于大数据分析、原生云应用程序数据、静态网站托管、备份/活跃归档、深度/冷归档等场景。

对象存储服务（Object Storage Service，简称OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。OBS的更多信息，请参见《对象存储服务控制台指南》。

语法

语法格式

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

语法说明

表 5-2 语法说明

参数	是否必选	说明
type	是	数据源类型，“obs”表示数据源为对象存储服务。
region	是	对象存储服务所在区域。
bucket	是	数据所在的OBS桶名。
object_name	是	数据所在的OBS桶中的对象名。
row_delimiter	是	行间的分隔符。
field_delimiter	是	属性分隔符。 当编码格式为csv时，用户可以自定义属性分隔符，默认为“，”。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 • 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 • 当引用符号为单引号时，则设置quote = """。 **说明** 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。

参数	是否必选	说明
version_id	否	版本号，当obs里的桶或对象有设置版本的时候需填写，否则不用配置该项。
timeindicator	否	在流中增加时间戳，可增加“processing time”时间戳或者“event time”时间戳。 **说明** • 若设置“processing time”，则为proctime.proctime。 当设置了proctime.proctime时，会在原有属性字段基础上多增加一个proctime系统时间戳属性，假设原有字段为3个，设置了proctime.proctime后会变成4个，设置rowtime属性字段不会发生变化。 • 若设置“event time”，可选择流中的某个属性来作为时间戳，格式为attr_name.rowtime。 • 以上两者可以同时设置。

注意事项

用来做时间戳的属性类型必须为long或者timestamp。

示例

从OBS的桶读取对象为input.csv的文件，文件以'\n'划行， 以','划列。

```
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 输入流

概述

创建source流从表格存储服务CloudTable的HBase中获取数据，作为作业的输入数据。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。CS可以从HBase中读取数据，用于过滤分析、数据转储等场景。

表格存储服务（CloudTable），是公有云基于Apache HBase提供的分布式、可伸缩、全托管的KeyValue数据存储服务，为CS提供了高性能的随机读写能力，适用于海量结

构化数据、半结构化数据以及时序数据的存储和查询应用，适用于物联网IOT应用和通用海量KeyValue数据存储与查询等场景。CloudTable的更多信息，请参见《表格存储服务用户指南》。

语法

语法格式

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

语法说明

表 5-3 语法说明

参数	是否必选	说明
type	是	数据源类型，“CloudTable”表示数据源为表格存储服务。
region	是	表格存储服务所在区域。
cluster_id	是	待读取数据表所属集群id。 如何查看CloudTable的集群id，请参见《表格存储服务用户指南》中“查看集群基本信息”章节。
table_name	是	待读取数据的表名，如需指定namespace，可表示为：namespace_name:table_name 。
table_columns	是	待读取的列，具体形式如："rowKey,f1:c1,f1:c2,f2:c1"，并且保证与source相同的列数。
timeindicator	否	在流中增加时间戳，可增加Processing Time时间戳或者Event Time时间戳。 **说明** • 若设置Processing Time时间戳，则timeindicator取值为proctime.proctime。 当设置了proctime.proctime时，会在原有属性字段基础上多增加一个proctime系统时间戳属性，假设原有字段为3个，设置了proctime.proctime后会变成4个。 • 若设置Event Time时间戳，可选择流中的某个属性来作为时间戳，格式为attr_name.rowtime，这里attr_name表示流中的某个属性。 • 以上两者可以同时设置。

注意事项

用来做时间戳的属性类型必须为long或者timestamp。

示例

从CloudTable的HBase中读取对象为car_infos的表。

```
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 输入流

概述

创建source流从Kafka获取数据，作为作业的输入数据。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。MRS基于Apache Kafka在公有云平台部署并托管了Kafka集群。

前提条件

- Kafka服务端的端口如果监听在hostname上，则需要将Kafka Broker节点的hostname和IP的对应关系添加到CS集群中。Kafka Broker节点的hostname和IP请联系Kafka服务的部署人员。如何添加IP域名映射，请参见《 实时流计算服务用户指南 》中**集群管理**章节中的“添加IP域名映射”部分。
- Kafka是线下集群，需要通过VPC服务的对等连接功能将CS服务与Kafka进行对接。

  如何建立对等连接，请参见《 实时流计算服务用户指南 》中**对等连接**章节。

语法

语法格式

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

语法说明

表 5-4 语法说明

参数	是否必选	说明
type	是	数据源类型，“Kafka”表示数据源。

参数	是否必选	说明
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过对等连接的方式开通CS集群和Kafka集群的连接）。
kafka_group_id	是	group id。
kafka_topic	是	读取的Kafka的topic。
encode	是	数据编码格式，可选为“csv”、“json”和“blob”。 ● 若编码格式为“csv”，则需配置“field_delimiter”属性。 ● 若编码格式为“json”，则需配置“json_config”属性。 ● 当编码格式为"blob"时，表示不对接收的数据进行解析，流属性仅能有一个且为Array[TINYINT]类型。
json_config	否	当encode为json时，用户可以通过该参数指定json字段和流属性字段的映射关系，格式为"field1=json_field1;field2=json_field2"。
field_delimiter	否	当encode为csv时，用于指定csv字段分隔符，默认为逗号。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 ● 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 ● 当引用符号为单引号时，则设置quote = "'"。 **说明** 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。
timeindicator	否	在流中增加时间戳，可增加“processing time”时间戳或者“event time”时间戳。 **说明** ● 若设置“processing time”，则为proctime.proctime。 当设置了proctime.proctime时，会在原有属性字段基础上多增加一个proctime系统时间戳属性，假设原有字段为3个，设置了proctime.proctime后会变成4个，设置rowtime属性字段不会发生变化。 ● 若设置“event time”，可选择流中的某个属性来作为时间戳，格式为attr_name.rowtime。 ● 以上两者可以同时设置。

参数	是否必选	说明
start_time	否	kafka数据读取起始时间。 当该参数配置时则从配置的时间开始读取数据，有效格式为yyyy-MM-dd HH:mm:ss。start_time要不大于当前时间，若大于当前时间，则不会有数据读取出。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"

注意事项

用来做时间戳的属性类型必须为long或者timestamp。

示例

从Kafka名称为test的topic中读取数据。

```
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 开源 Kafka 输入流

概述

创建source流从Kafka获取数据，作为作业的输入数据。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

前提条件

- Kafka服务端的端口如果监听在hostname上，则需要将Kafka Broker节点的hostname和IP的对应关系添加到CS集群中。Kafka Broker节点的hostname和IP请联系Kafka服务的部署人员。如何添加IP域名映射，请参见《 实时流计算服务用户指南 》中**集群管理**章节中的“添加IP域名映射”部分。
- Kafka是线下集群，需要通过VPC服务的对等连接功能将CS服务与Kafka进行对接。

  如何建立对等连接，请参见《 实时流计算服务用户指南 》中**对等连接**章节。

语法

语法格式

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

语法说明

表 5-5 语法说明

参数	是否必选	说明
type	是	数据源类型，“Kafka”表示数据源。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过对等连接的方式开通CS集群和Kafka集群的连接）。
kafka_group_id	是	group id。
kafka_topic	是	读取的Kafka的topic。
encode	是	数据编码格式，支持“json”，“csv”、“blob”和“user_defined”。 ● 若编码格式为“csv”，则需配置“field_delimiter”属性。 ● 若编码格式为“json”，则需配置“json_config”属性。 ● 当编码格式为“blob”时，表示不对接收的数据进行解析，流属性仅能有一个且为Array[TINYINT]类型。 ● 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
json_config	否	当encode为json时，用户可以通过该参数指定json字段和流属性字段的映射关系，格式为"field1=json_field1;field2=json_field2"。
field_delimiter	否	当encode为csv时，用于指定csv字段分隔符，默认为逗号。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现解码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现解码类的入参，仅支持一个string类型的参数。

参数	是否必选	说明
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 ● 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 ● 当引用符号为单引号时，则设置quote = "'"。 **说明** 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。
timeindicator	否	在流中增加时间戳，可增加“processing time”时间戳或者“event time”时间戳。 **说明** ● 若设置“processing time”，则为proctime.proctime。当设置了proctime.proctime时，会在原有属性字段基础上多增加一个proctime系统时间戳属性，假设原有字段为3个，设置了proctime.proctime后会变成4个，设置rowtime属性字段不会发生变化。 ● 若设置“event time”，可选择流中的某个属性来作为时间戳，格式为attr_name.rowtime。 ● 以上两者可以同时设置。
start_time	否	kafka数据读取起始时间。 当该参数配置时则从配置的时间开始读取数据，有效格式为yyyy-MM-dd HH:mm:ss。start_time要不大于当前时间，若大于当前时间，则不会有数据读取出。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"

注意事项

用来做时间戳的属性类型必须为long或者timestamp。

示例

从Kafka读取对象为test的topic。数据实例:{"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 创建输出流

5.2.1 DIS 输出流

概述

CS将作业的输出数据写入数据接入服务（DIS）中。适用于将数据过滤后导入DIS通道，进行后续处理的场景。

数据接入服务（Data Ingestion Service，简称DIS）为处理或分析流数据的自定义应用程序构建数据流管道，主要解决云服务外的数据实时传输到云服务内的问题。数据接入服务每小时可从数十万种数据源（如IoT数据采集、日志和定位追踪事件、网站点击流、社交媒体源等）中连续捕获、传送和存储数TB数据。DIS的更多信息，请参见《数据接入服务用户指南》。

语法

语法格式

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

语法说明

表 5-6 语法说明

参数	是否必选	说明
type	是	输出通道类型，dis表示输出到数据接入服务。
region	是	数据所在的DIS所在区域。
channel	是	DIS通道。
partition_key	否	数据输出分组主键，多个主键用逗号分隔。当该参数没有配置的时候则随机派发。
encode	是	数据编码格式，可选为“csv”、“json”和“user_defined”。 **说明** • 若编码格式为“csv”，则需配置“field_delimiter”属性。 • 若编码格式为“json”，则需使用“enable_output_null”来配置是否输出空字段，具体见示例。 • 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
field_delimiter	是	属性分隔符。 • 当编码格式为csv时，需要设置属性分隔符，用户可以自定义，如：“，”。 • 当编码格式为json时，则不需要设置属性之间的分隔符。

参数	是否必选	说明
json_config	否	当编码格式为json时，用户可以通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1;field2=data_json.field2”。
enable_output_null	否	当编码格式为json时，需使用该参数来配置是否输出空字段。 当该参数为true表示输出空字段（值为null），若为false表示不输出空字段。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现编码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现编码类的入参，仅支持一个string类型的参数。

注意事项

无。

示例

- CSV编码格式：数据输出到DIS通道，使用csv编码，并且以逗号为分隔符，多个分区用car_owner做为key进行分发。数据输出示例："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编码格式：数据输出到DIS通道，使用json编码，多个分区用car_owner，car_brand 做为key进行分发,“enableOutputNull”为“true”表示输出空字段（值为null），若为“false”表示不输出空字段。数据示例："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 输出流

概述

创建sink流将CS数据输出到对象存储服务（OBS）。CS可以将作业分析结果输出到OBS上。适用于大数据分析、原生云应用程序数据、静态网站托管、备份/活跃归档、深度/冷归档等场景。

对象存储服务（Object Storage Service，简称OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。OBS的更多信息，请参见《对象存储服务控制台指南》。

前提条件

OBS输出流功能仅支持输出数据到3.0版本以上的桶，请先查看桶信息确认桶的版本。

语法

语法格式

```
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 = ""
)
```

语法说明

表 5-7 语法说明

参数	是否必选	说明
type	是	输出通道类型，“obs”表示输出到对象存储服务。
region	是	对象存储服务所在区域。
encode	是	编码方式。当前支持csv/json/orc/carbondata/avro/avro_merge/parquet七种格式。

参数	是否必选	说明
field_delimiter	否	属性分隔符。 仅当编码方式为csv时需要配置，若不配置，默认分隔符为逗号。
row_delimiter	否	行分隔符。当编码格式为csv、json时需要设置。
json_config	否	当编码格式为json时，用户可以通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1;field2=data_json.field2”。
obs_dir	是	文件存储目录。格式为{桶名}/{目录名}， 如obs-a1/dir1/subdir。当编码格式为csv（append为false）、json（append为false）、avro_merge、parquet时，支持参数化。
file_prefix	否	输出文件名前缀。生成的文件会以file_prefix.*x的方式命名，*如file_prefix.1、 file_prefix.2，若没有设置，默认文件前缀为temp。此配置项不适用于carbondata文件。
rolling_size	否	单个文件最大允许大小。 **说明** • rolling_size和rolling_interval必须至少配一样或者都配置。 • 当文件大小超过设置size后，会生成新文件。 • 支持的单位包括KB/MB/GB，若没写单位，表示单位为字节数。 • 当编码格式为orc和carbondata时不需要设置。
rolling_interval	否	数据保存到对应目录的时间模式。 **说明** • rolling_size和rolling_interval必须至少配一样或者都配置。 • 设置后数据会按照输出时间输出到相应时间目录下。 • 支持的格式为yyyy/MM/dd/HH/mm， 最小单位只到分钟，大小写敏感。例如配置为yyyy/MM/dd/HH， 则数据会写入对应小时这个时间点所产生的目录下，比如2018-09-10 16:40产生的数据就会写到{obs_dir}/2018-09-10_16目录下。 • 当rolling_size和rolling_interval都配置时，表示每个时间所对应的目录下，单个文件超过设置大小时，另起新文件。
quote	否	修饰符，仅当编码格式为csv时可配置，配置后会在每个属性前后各加上修饰符，建议使用不可见字符配置，如"\u0007"。
array_bracket	否	数组括号，仅当编码格式为csv时可配置， 可选值为"()", "{}", "[]"， 例如配置了"{}", 则数组输出格式为{a1,a2}。

参数	是否必选	说明
append	否	值为true或者false，默认为true。 当OBS不支持append模式，且编码格式为csv和json时，可将该参数设置为false。Append为false时需要设置max_record_num_per_file和dump_interval。
max_record_num_per_file	否	文件最大记录数，当编码格式为csv（append为false）、json（append为false）、orc、carbondata、avro、avro_merge和parquet时需配置，表明一个文件最多存储记录数，当达到最大值，则另起新文件。
dump_interval	否	触发周期， 当编码格式为orc或者配置了DIS通知提醒时需进行配置。 ● 在orc编码方式中，该配置表示周期到达时，即使文件记录数未达到最大个数配置，也将文件上传到OBS上。 ● 在DIS通知提醒功能中，该配置表示每周期往DIS发送一个通知提醒，表明该目录已写完。
dis_notice_channel	否	OBS目录完成通知通道。表示每周期往DIS通道中发送一条记录，该记录内容为OBS目录路径，表明该目录已书写完毕。
max_record_num_cache	否	记录最大缓存数，仅当编码格式为carbondata时，可以配置该字段，且最小值不能小于max_record_num_per_file，默认值为max_record_num_per_file。
carbon_properties	否	carbon属性，仅当编码格式为carbondata时，可以配置该字段，值为"k1=v1, k2=v2"的形式，支持carbon-sdk中函数withTableProperties所支持的所有配置项，另外还支持IN_MEMORY_FOR_SORT_DATA_IN_MB和UNSAFE_WORKING_MEMORY_IN_MB两个配置项。
encoded_data	否	当编码格式为json（append为false）、avro_merge和parquet时，可通过配置该参数指定真正需要编码的数据，格式为${field_name}，表示直接将该流字段的内容作为一个完整的记录进行编码。

注意事项

当配置项支持参数化时，表示将记录中的一列或者多列作为该配置项的一部分。例如当配置项设置为car_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。

示例

- 将car_infos数据输出到OBS的obs-sink桶下，输出目录为car_infos， 输出文件以greater_30作为文件名前缀，当单个文件超过100M时新起一个文件，同时数据输出用csv编码，使用逗号作为属性分隔符，换行符作为行分隔符。

```
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"
);
```

- carbondata编码格式示例

```
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"
);
```

- orc编码格式示例

```
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 输出流

概述

CS将作业的输出数据输出到CloudTable的HBase中。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。HBase支持消息数据、报表数据、推荐类数据、风控类数据、日志数据、订单数据等结构化、半结构化的KeyValue数据存储。 利用CS，用户可方便地将海量数据高速、低时延写入HBase。

表格存储服务（CloudTable），是公有云基于Apache HBase提供的分布式、可伸缩、全托管的KeyValue数据存储服务，为CS提供了高性能的随机读写能力，适用于海量结构化数据、半结构化数据以及时序数据的存储和查询应用，适用于物联网IOT应用和通用海量KeyValue数据存储与查询等场景。CloudTable的更多信息，请参见**《表格存储服务用户指南》**。

语法

语法格式

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 = "")

语法说明

表 5-8 语法说明

参数	是否必选	说明
type	是	输出通道类型，“cloudtable”表示输出到CloudTable（HBase）。
region	是	表格存储服务所在区域。
cluster_id	是	待读取数据表所属集群id。
table_name	是	待插入数据的表名，支持参数化，例如当需要某一列或者几列作为表名的一部分时，可表示为”car_pass_inspect_with_age_${car_age}“，其中car_age为列名。
table_columns	是	待插入的列，具体形式如："rowKey,f1:c1,f1:c2,f2:c1"，其中必须指定rowKey，当某列不需要加入数据库时，以第三列为例，可表示为"rowKey,f1:c1,,f2:c1"。
illegal_data_table	否	如果指定该参数，异常数据（比如：rowKey不存在）会写入该表（rowKey为时间戳加六位随机数字，schema为info:data, info:reason），否则会丢弃。
create_if_not_exist	否	当待写入的表或者列族不存在时，是否创建，值为true或者false，默认值为false。

参数	是否必选	说明
batch_insert_data_num	否	表示一次性批量写入的数据条数，值必须为正整数，上限为100，默认值为10。

注意事项

- 当配置项支持参数化时，表示将记录中的一列或者多列作为该配置项的一部分。例如当配置项设置为car_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。
- 通过这种方式将数据写入到CloudTable的Hbase，速度受限，推荐使用独享集群直连方式。

示例

将流qualified_cars的数据输出到表格存储服务CloudTable的HBase中。

```
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 输出流

概述

CS将作业的输出数据输出到CloudTable的OpenTSDB中。OpenTSDB是基于HBase的分布式的，可伸缩的时间序列数据库。它存储的是时间序列数据，时间序列数据是指在不同时间点上收集到的数据，这类数据反映了一个对象随时间的变化状态或程度。支持秒级别数据的采集监控，进行永久存储，索引和查询，可用于系统监控和测量、物联网数据、金融数据和科学实验结果数据的收集监控。

表格存储服务（CloudTable），是公有云基于Apache HBase提供的分布式、可伸缩、全托管的KeyValue数据存储服务，为CS提供了高性能的随机读写能力，适用于海量结构化数据、半结构化数据以及时序数据的存储和查询应用，适用于物联网IOT应用和通用海量KeyValue数据存储与查询等场景。CloudTable的更多信息，请参见**《表格存储服务用户指南》**。

前提条件

确保CloudTable的集群已经开启了OpenTSDB服务。如何开启，请参见《表格存储服务用户指南》中**开启OpenTSDB**章节。

语法

语法格式

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 = "")

语法说明

表 5-9 语法说明

参数	是否必选	说明
type	是	输出通道类型，“opentsdb”表示输出到表格存储服务CloudTable（OpenTSDB）。
region	是	表格存储服务所在区域。
cluster_id	否	待插入数据所属集群的id，该参数与tsdb_link_address必须指定其中一个。
tsdb_metrics	是	数据点的metric，支持参数化。
tsdb_timestamps	是	数据点的timestamp，数据类型支持LONG、INT、SHORT和STRING，仅支持指定动态列。
tsdb_values	是	数据点的value，数据类型支持SHORT、INT、LONG、FLOAT、DOUBLE和STRING，支持指定动态列或者常数值。
tsdb_tags	是	数据点的tags，每个tags里面至少一个标签值，最多8个标签值，支持参数化。
batch_insert_data_num	否	表示一次性批量写入的数据量（即数据条数），值必须为正整数，上限为100，默认值为8。
tsdb_link_address	否	待插入数据所属集群的OpenTsdb链接地址，使用该参数时，作业需要运行在独享CS集群，且CS集群需要与CloudTable集群建立对等连接，该参数与cluster_id必须指定其中一个。 **说明** 如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

注意事项

- 当配置项支持参数化时，表示将记录中的一列或者多列作为该配置项的一部分。例如当配置项设置为car_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。

示例

将流weather_out的数据输出到表格存储服务CloudTable的OpenTSDB中。

```
CREATE SINK STREAM weather_out (
  timestamp_value LONG, /* 时间 */
  temperature FLOAT, /* 温度值 */
  humidity FLOAT, /* 湿度值 */
  location STRING /* 地点 */
)
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 输出流

概述

CS将作业的输出数据输出到关系型数据库（RDS）中。目前支持PostgreSQL和MySQL两种数据库。PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。MySQL数据库适用于各种WEB应用、电子商务应用、企业应用、移动应用等场景，减少IT部署和维护成本。

关系型数据库（Relational Database Service，简称RDS）是一种基于云计算平台的在线关系型数据库服务。RDS包括如下几种类型的数据库：MySQL、HWSQL、PostgreSQL和Microsoft SQL Server。RDS的更多信息，请参见**《关系型数据库用户指南》**。

前提条件

- 请务必确保您的账户下已在关系型数据库（RDS）里创建了PostgreSQL或MySQL类型的RDS实例。

  如何创建RDS实例，请参见**《关系型数据库快速入门》**中“购买实例”章节。
- 该场景作业需要运行在CS的独享集群上，因此要与RDS实例建立VPC对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

  如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法

语法格式

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

语法说明

表 5-10 语法说明

参数	是否必选	说明
type	是	输出通道类型，rds表示输出到关系型数据库中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址，格式为："{database_type}://ip:port/database" 目前支持两种数据库连接：MySQL和PostgreSQL ● MySQL: 'mysql://ip:port/database' ● PostgreSQL: 'postgresql://ip:port/database'
table_name	是	要插入数据的数据库表名。

db_columns	否	支持配置输出流属性和数据库表属性的对应关系，需严格按照输出流的属性顺序配置。
		示例：
		`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"`
		`);`
		student_name对应数据库里的name属性，student_age对应数据库里的age属性。
		说明
		当不配置db_columns时，若输出流属性个数小于数据库表属性个数，并且数据库多出的属性都是nullable或者有默认值时，这种情况也允许。

primary_key	否	如果想通过主键实时更新表中的数据，需要在创建数据表的时候增加primary_key配置项，如下面例子中的c_timeminute。配置primary_key后，在进行数据写入操作时，如果primary_key存在，则进行更新操作，否则进行插入操作。
		示例：
		`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");`

参数	是否必选	说明
operation_field	否	该配置项用于指定数据的处理方式，需要配置为${field_name}的形式，field_name的类型必读为string，field_name所代表的真正内容表示为D或者DELETE时，表示删除数据库中该条记录，其余默认插入数据。

注意事项

stream_id所定义的流格式需和数据库中的表格式一致。

示例

将流audi_cheaper_than_30w的数据输出到数据库test的audi_cheaper_than_30w表下。

```
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 RDS 和 DWS 数据同步输出流

概述

CS将支持根据binlog格式实时同步数据输出到关系型数据库（RDS）及数据仓库（DWS）中。目前支持PostgreSQL和MySQL两种数据库。PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。MySQL数据库适用于各种WEB应用、电子商务应用、企业应用、移动应用等场景，减少IT部署和维护成本。

关系型数据库（Relational Database Service，简称RDS）是一种基于云计算平台的在线关系型数据库服务。RDS包括如下几种类型的数据库：MySQL、HWSQL、PostgreSQL和Microsoft SQL Server。

RDS的更多信息，请参见**《关系型数据库用户指南》**。

数据仓库服务（Data Warehouse Service）是一种高性能且完全托管的大规模并行处理的数据库服务：支持行、列存储，提供一键部署、数据高速入库等特性。旨在满足云上用户的安全、可靠、快速，基于海量数据进行数据挖掘和数据存储、分析的需求。

DWS的更多信息，请参见**《数据仓库服务管理指南》**。

前提条件

- 请务必确保您的账户下已在关系型数据库（RDS）里创建了PostgreSQL或MySQL类型的RDS实例。

  如何创建RDS实例，请参见《关系型数据库快速入门》中“购买实例”章节。
- 该场景作业需要运行在CS的独享集群上，因此要与RDS实例建立VPC对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

  如何设置安全组规则，请参见《虚拟私有云用户指南》中“安全组”章节。

语法

语法格式

```
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",
);
```

语法说明

该特性支持对mysql、postgresql、dws进行数据库同步操作，支持对同一数据库多张表进行同步。

表 5-11 参数说明

参数	是否必选	说明
db_url	是	数据库连接地址，格式为："{database_type}://ip:port/database" 目前支持以下数据库连接： ● MySQL: 'mysql://ip:port/database' ● PostgreSQL和DWS: 'postgresql://ip:port/database'
username	是	数据库连接用户名。
password	是	数据库连接密码。
table_name	是	表名列，即说明sink流中的哪个字段用来表示表名，格式为${sink中的属性名}，该属性一般是从binlog日志中的表字段解析而来。
operation_field	是	操作类型列，即说明sink流中的哪个字段用来表示操作类型，格式为${sink中的属性名}，操作类型值支持I/U/D/INSERT/UPDATE/DELETE。该属性一般是从binlog日志中的操作类型字段解析而来。
before	是	修改前内容列，即说明sink流中的哪个字段用来表示更新前的记录内容，格式为${sink中的属性名}。
after	是	即说明sink流中的哪个字段用来表示更新后的记录内容，格式为${sink中的属性名} 。
tranx_id	否	即说明sink流中的哪个字段用来表示事务Id，格式为${sink中的属性名}。
commit	否	即说明sink流中的哪个字段用来表示事务是否结束，格式为${sink中的属性名}。
sql	否	即说明sink流中的哪个字段用来表示DDL操作SQL，格式为${sink中的属性名}。
table_name_map	否	当原始数据库里的表名和最终目的数据库里的表名不一致时需要进行配置，可以配置为常量，也可以配置为变量。 ● 常量配置方式： table_name_map="distdbtable", 表示所有记录最终都输出到数据库的distdbtable这张表。 ● 变量配置方式： table_name_map="${sink中的属性名}"，例如，table_name_map="${dbTableName}，表示当前记录的目的表由dbTableName这个属性决定。 当不配置的时候，默认目的表和原始表一样。

参数	是否必选	说明
column_name_map	否	当原始数据库里的列名和最终目的数据库里的列名不一致时需要进行配置，可以配置为常量，也可以配置为变量。 ● 常量配置方式：column_name_map="originAttr1=distAttr1,originAttr2=distAttr2", 表示所有记录里涉及列名为originAttr1的最终都映射到数据库里的distAttr1字段。 ● 变量配置方式：column_name_map="${sink中的属性名}"，例如，column_name_map="${dbColumnMap}，表示当前记录的目的列名由dbColumnMap这个属性决定。 当不配置的时候，默认目的列名和原始列名一样。
schema_case_sensitive	否	说明目标表中schema是否大小写敏感，默认为“false”，即大小写不敏感。
db_type	否	说明目标数据库的类型，默认为DWS。

注意事项

1. 对于待插入及更新后数据，需要映射到after字段，对于更新前及待删除数据，需要映射到before字段。

2. 该输出流按顺序同步binlog到目标库，所以需要确保进入输出流的binlog是符合预期时序及事务划分的。建议在源端，根据binlog的事务关联性，进行表级别的划分，将事务相关联的表的binlog，划分到同一个源分区。

示例

该示例为从DIS接收由maxwell生成的binlog数据，并将数据同步到数据库中。

假设有如下三条数据：第一条为插入操作，第二条为更新操作，第三条为删除操作。

```
{
  "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表对应数据库中Table1表，T2对应Table2表，T3对应Table3，则相应的SQL描述如下：

```
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 输出流（通过 JDBC 方式）

概述

CS将作业的输出数据输出到数据仓库服务（DWS）中。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于公有云基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见**《数据仓库服务管理指南》**。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。
  如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在CS的独享集群上，因此要与DWS集群建立VPC对等连接，且用户可以根据实际所需设置相应安全组规则。
  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。
  如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法

语法格式

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

语法说明

表 5-12 语法说明

参数	是否必选	说明
type	是	输出通道类型，rds表示输出到关系型数据库或者数据仓库服务中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址格式为： postgresql://ip:port/database
table_name	是	要插入数据的数据库表名。数据库表需事先创建好。

参数	是否必选	说明

db_columns	否	支持配置输出流属性和数据库表属性的对应关系，需严格按照输出流的属性顺序配置。
		示例：
		`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"`
		`);`
		student_name对应数据库里的name属性，student_age对应数据库里的age属性。
		说明
		当不配置db_columns时，若输出流属性个数小于数据库表属性个数，并且数据库多出的属性都是nullable或者有默认值时，这种情况也允许。

primary_key	否	如果想通过主键实时更新表中的数据，需要在创建数据表的时候增加primary_key配置项，如下面例子中的c_timeminute。配置primary_key后，在进行数据写入操作时，如果primary_key存在，则进行更新操作，否则进行插入操作。
		示例：
		`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");`

注意事项

stream_id所定义的流格式需和数据库中的表格式一致。

示例

将流audi_cheaper_than_30w的数据输出到数据库test的audi_cheaper_than_30w表下。

```
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 输出流（通过 OBS 转储方式）

概述

创建sink流将CS数据通过OBS转储方式输出到数据仓库服务(DWS)，即CS数据先输出到OBS，然后再从OBS导入到DWS。如何导入OBS数据到DWS具体可参考《数据仓库服务数据库开发指南》中“从OBS并行导入数据到集群”章节。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于公有云基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见《数据仓库服务管理指南》。

注意事项

- 通过OBS转储支持两种中间文件方式：
  - ORC： ORC格式不支持Array数据类型，如果使用ORC格式，需先在DWS中创建外部服务器，具体可参考《数据仓库服务数据库开发指南》中“创建外部服务器”章节。
  - CSV： CSV格式默认记录分隔符为换行符，若属性内容中有换行符，建议配置quote，具体参见表5-13。
- 如果要写入的表不存在，则会自动创建表。由于CS SQL类型不支持text，如果存在长文本，建议先在数据库中创建表。

前提条件

- 确保已创建OBS桶和文件夹。

  如何创建OBS桶，具体请参见《对象存储服务控制台指南》中的“创建桶”章节。

  如何新建文件夹，具体请参见《对象存储服务控制台指南》中的“新建文件夹”章节。
- 该场景作业需要运行在CS的独享集群上，因此要与DWS集群建立VPC对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中对等连接章节。

  如何设置安全组规则，请参见《虚拟私有云用户指南》中“安全组”章节。

语法

语法格式

```
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 = ""
);
```

语法说明

表 5-13 语法说明

参数	是否必选	说明
type	是	输出通道类型，dws表示输出到数据仓库服务中。
region	是	数据仓库服务所在区域。
encode	是	编码方式。当前支持csv和orc两种方式。
field_delimiter	否	属性分隔符。当编码方式为csv时需要配置，建议尽量用不可见字符作为分隔符，如\u0006\u0002。
quote	否	单字节，建议使用不可见字符，如\u0007。
db_obs_server	否	已在数据库中创建的外部服务器，如obs_server。 如何创建外部服务器，具体操作步骤可参考《数据仓库服务数据库开发指南》中**创建外部服务器**章节。 如果编码方式为orc格式时需指定该参数。
obs_dir	是	中间文件存储目录。格式为{桶名}/{目录名}， 如obs-a1/dir1/subdir。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址。格式为/ip:port/database，如“192.168.1.21:8000/test1”。
table_name	是	数据表名，若表不存在，则自动创建。
max_record_num_per_file	是	每个文件最多存储多少条记录。当文件记录数少于最大值时，该文件会延迟一个转储周期输出。
max_dump_file_num	是	执行一次转储操作时最多转储多少文件。 当本次转储操作发现文件数小于最大值，则会延迟一个转储周期输出。
dump_interval	是	转储周期，单位为秒。
delete_obs_temp_file	否	是否要删除obs上的临时文件，默认为“true”，若设置为“false”，则不会删除obs上的文件，需用户自己清理。

示例

- CSV格式转储。

```
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"
);
```

- ORC格式转储。

```
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 输出流

概述

CS将作业的输出数据输出到文档数据库服务（DDS）中。

文档数据库服务（Document Database Service，简称DDS）完全兼容MongoDB协议，提供安全、高可用、高可靠、弹性伸缩和易用的数据库服务，同时提供一键部署、弹性扩容、容灾、备份、恢复、监控和告警等功能。DDS的更多信息，请参见**《文档数据库服务用户指南》**。

前提条件

- 请务必确保您的账户下已在文档数据库服务（DDS）里创建了DDS实例。

  如何创建DDS实例，请参考**《文档数据库服务快速入门》**中“快速购买文档数据库实例”章节。
- 目前仅支持未开启SSL认证的集群实例，不支持副本集与单节点的类型实例。
- 该场景作业需要运行在CS的独享集群上，请确保已创建CS独享集群。

如何创建CS独享集群，具体操作请参见《 实时流计算服务用户指南 》中**集群管理**章节。

- 确保CS独享集群与DDS集群建立对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《 实时流计算服务用户指南 》中**对等连接**章节。

  如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

语法

语法格式

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

语法说明

表 5-14 语法说明

参数	是否必选	说明
type	是	输出通道类型，dds表示输出到文档数据库服务中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	DDS实例的访问地址，形如：ip1:port,ip2:port/database/collection。
field_names	是	待插入数据字段的key，具体形式如："f1,f2,f3"，并且保证与sink中数据列一一对应。
batch_insert_data_num	否	表示一次性批量写入的数据量，值必须为正整数，默认值为10。

示例

将流qualified_cars 的数据输出到文档数据库collectionTest。

```
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 输出流

概述

CS将作业的输出数据输出到消息通知服务（SMN）中。

消息通知服务（Simple Message Notification，简称SMN）为CS提供可靠的、可扩展的、海量的消息处理服务，它大大简化系统耦合，能够根据用户的需求，向订阅终端主动推送消息。可用于连接云服务、向多个协议推送消息以及集成在产生或使用通知的任何其他应用程序等场景。SMN的更多信息，请参见**《消息通知服务用户指南》**。

语法

语法格式

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

语法说明

表 5-15 语法说明

参数	是否必选	说明
type	是	输出通道类型，smn表示输出到消息通知服务中。
region	是	SMN所在区域。
topic_urn	否	SMN服务的主题URN，用于静态主题URN配置。作为消息通知的目标主题，需要提前在SMN服务中创建。 与urn_column配置两者至少存在一个，同时配置时，topic_urn优先级更高。
urn_column	否	包含主题URN内容的字段名，用于动态主题URN配置。 与topic_urn配置两者至少存在一个，同时配置时，topic_urn优先级更高。
message_subject	是	发往SMN服务的消息标题，用户自定义。
message_column	是	输出流的字段名，其内容作为消息的内容，用户自定义。目前只支持默认的文本消息。

注意事项

无。

示例

将流over_speed_warning的数据输出到消息通知服务SMN中。

```
//静态主题配置
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"
);
//动态主题配置
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 输出流

概述

CS将作业的输出数据输出到云搜索服务的Elasticsearch中。Elasticsearch是基于Lucene的当前流行的企业级搜索服务器，具备分布式多用户的能力。其主要功能包括全文检索、结构化搜索、分析、聚合、高亮显示等。能为用户提供实时搜索、稳定可靠的服务。适用于日志分析、站内搜索等场景。

云搜索服务（Cloud Search Service，简称CSS）为CS提供托管的分布式搜索引擎服务，完全兼容开源Elasticsearch搜索引擎，支持结构化、非结构化文本的多条件检索、统计、报表。云搜索服务的更多信息，请参见**《云搜索服务用户指南》**。

前提条件

- 请务必确保您的账户下已在云搜索服务里创建了集群。如何创建集群请参考《云搜索服务用户指南》中**创建集群**章节。
- 该场景作业需要运行在CS的独享集群上，因此要与云搜索服务建立VPC对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

  如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法

语法格式

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= "");

语法说明

表 5-16 语法说明

参数	是否必选	说明
type	是	输出通道类型，es表示输出到云搜索服务中。
region	是	数据所在的云搜索服务所在区域。例如"cn-north-1" 。
cluster_address	是	云搜索服务集群的内网访问地址，例如：x.x.x.x:x，多个地址时以逗号分隔。
es_index	是	待插入数据的索引，支持参数化。
es_type	是	待插入数据的文档类型，支持参数化。
es_fields	是	待插入数据字段的key，具体形式如："id,f1,f2,f3,f4"，并且保证与sink中数据列一一对应；如果不使用key，而是采用随机的属性字段，则无需使用id关键字，具体形式如："f1,f2,f3,f4,f5"。
batch_insert_data_num	是	表示一次性批量写入的数据量，值必须为正整数，上限为100，默认值为10。
action	否	当值为add时，表示遇到相同id时，数据被强制覆盖，当值为upsert时，表示遇到相同id时，更新数据（选择upsert时，es_fileds字段中必须指定id），默认值为add。
enable_output_null	否	使用该参数来配置是否输出空字段。当该参数为true表示输出空字段（值为null），若为false表示不输出空字段。默认为false。
max_record_num_cache	否	记录最大缓存数。

注意事项

当配置项支持参数化时，表示将记录中的一列或者多列作为该配置项的一部分。例如当配置项设置为car_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。

示例

将流qualified_cars的数据输出到云搜索服务的集群。

```
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 输出流

概述

CS将作业的输出数据输出到分布式缓存服务（DCS）的Redis中。Redis是一种支持Key-Value等多种数据结构的存储系统。可用于缓存、事件发布或订阅、高速队列等场景，提供字符串、哈希、列表、队列、集合结构直接存取，基于内存，可持久化。有关Redis的详细信息，请访问Redis官方网站**https://redis.io/**。

分布式缓存服务（DCS）为CS提供兼容Redis的即开即用、安全可靠、弹性扩容、便捷管理的在线分布式缓存能力，满足用户高并发及快速数据访问的业务诉求。DCS的更多信息，请参见**《分布式缓存服务用户指南》**。

前提条件

- 请务必确保您的账户下已在分布式缓存服务（DCS）里创建了Redis类型的缓存实例。

  如何创建Redis类型的缓存实例，请参考**《分布式缓存服务用户指南》**中“申请Redis缓存实例”章节。
- 该场景作业需要运行在CS的独享集群上，因此要与DCS集群建立对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

  如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- 用户通过VPC对等访问DCS实例时，除了满足VPC对等网跨VPC访问的约束之外，还存在如下约束：
  - 当创建DCS实例时使用了172.16.0.0/12~24网段时，CS集群不能在192.168.1.0/24、192.168.2.0/24、192.168.3.0/24网段。
  - 当创建DCS实例时使用了192.168.0.0/16~24网段时，CS集群不能在172.31.1.0/24、172.31.2.0/24、172.31.3.0/24网段。
  - 当创建DCS实例时使用了10.0.0.0/8~24网段时，CS集群不能在172.31.1.0/24、172.31.2.0/24、172.31.3.0/24网段。

语法

语法格式

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= "");

语法说明

表 5-17 语法说明

参数	是否必选	说明
type	是	输出通道类型，dcs_redis表示输出到分布式缓存服务的Redis存储系统中。

参数	是否必选	说明
region	是	数据所在的DCS所在区域。
cluster_address	是	Redis实例连接地址。
password	否	Redis实例连接密码，当设置为免密访问时，省略该配置项。
value_type	是	该参数可配置为如下选项或选项的组合： ● 支持指定插入数据类型，包括：string, list, hash, set, zset； ● 支持设置key的过期时间，包括expire, pexpire, expireAt, pexpireAt； ● 支持删除key命令，包括del, hdel； 当需要使用多个命令时，用“;”分隔。
key_value	是	设置具体的key和value，key_value对必须与value_type所指定的类型数相对应，用“;”分隔，且key和value均支持参数化，动态列名采用${列名}表示。

注意事项

- 当配置项支持参数化时，表示将记录中的一列或者多列作为该配置项的一部分。例如当配置项设置为car_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。
- 字符":", ",", ";", "$", "{", "}"已被征用为特殊分隔符，暂时没有提供转义功能，禁止在key和value中作为普通字符使用，否则会影响解析，导致程序异常。

示例

将流qualified_cars的数据输出到DCS服务的Redis类型的缓存实例中。

```
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 输出流

概述

CS将作业的输出数据输出到Kafka中。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。MRS基于Apache Kafka在公有云平台部署并托管了Kafka集群。

前提条件

- Kafka服务端的端口如果监听在hostname上，则需要将Kafka Broker节点的hostname和IP的对应关系添加到CS集群中。Kafka Broker节点的hostname和IP请联系Kafka服务的部署人员。如何添加IP域名映射，请参见《实时流计算服务用户指南》中**集群管理**章节中的“添加IP域名映射”部分。
- Kafka是线下集群，需要通过VPC服务的对等连接功能将CS服务与Kafka进行对接。

  如何建立对等连接，请参见《实时流计算服务用户指南》中**对等连接**章节。

语法

语法格式

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

语法说明

表 5-18 语法说明

参数	是否必选	说明
type	是	输出通道类型，"kafka"表示输出到Kafka中。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过对等连接的方式开通CS集群和Kafka集群的连接）。
kafka_topic	是	写入的topic。
encode	是	编码格式，当前暂时只支持json。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"

注意事项

无

示例

将数据输出到Kafka中。

```
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 输出流

概述

CS将作业的输出数据输出到MRS的HBase中。

前提条件

- 确保您的账户下已在MapReduce服务（MRS）里创建了您配置的集群。当前CS仅支持与未开启Kerberos认证的集群对接。
- 该场景作业需要运行在CS的独享集群上，请确保已创建CS独享集群。

  如何创建CS独享集群，具体操作请参见《实时流计算服务用户指南》中**集群管理**章节。
- 确保CS独享集群与MRS集群建立对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

  如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- 确保在CS集群hosts文件中添加MRS集群zookeeper地址的ip和域名映射。

  如何添加IP域名映射，请参见《实时流计算服务用户指南》中**集群管理**章节中的“添加IP域名映射”部分。

语法

语法格式

```
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 = "")
```

语法说明

表 5-19 语法说明

参数	是否必选	说明
type	是	输出通道类型，"mrs_hbase"表示输出到MRS的HBase中。
region	是	MRS服务所在区域。
cluster_address	是	待插入数据表所属集群zookeeper地址，形如：ip1,ip2:port。
table_name	是	待插入数据的表名。 支持参数化，例如当需要某一列或者几列作为表名的一部分时，可表示为”car_pass_inspect_with_age_${car_age}“，其中car_age为列名。

参数	是否必选	说明
table_columns	是	待插入的列，具体形式如："rowKey,f1:c1,f1:c2,f2:c1"，其中必须指定rowKey，当某列不需要加入数据库时，以第三列为例，可表示为"rowKey,f1:c1,,f2:c1"。
illegal_data_table	否	如果指定该参数，异常数据（比如：rowKey不存在）会写入该表（rowKey为taskNo加下划线加时间戳加六位随机数字，schema为info:data, info:reason），否则会丢弃。
batch_insert_data_num	否	表示一次性批量写入的数据条数，值必须为正整数，上限为1000，默认值为10。
action	否	表示数据是插入还是删除，可选值为add和delete，默认值为add。

注意事项

无

示例

将数据输出到MRS的HBase中。

```
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 输出流

概述

CS将作业的输出数据输出到API网关服务的开放API中，用户通过API网关服务的调用API方式访问数据。API网关（API Gateway）是为开发者、合作伙伴提供的高性能、高可用、高安全的API托管服务，帮助用户轻松构建、管理和部署任意规模的API。借助API网关，可以简单、快速、低成本、低风险地实现系统集成、微服务聚合、Serverless架构。API网关服务的更多信息，请参见**《API网关服务用户指南（调用API）》**。

前提条件

- 请务必确保您的账户下已在API网关服务创建了应用-。如何创建调用API的应用请参考《API网关服务用户指南（调用API）》中“创建应用”章节。

语法

语法格式

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

语法说明

表 5-20 语法说明

参数	是否必选	说明
type	是	输出通道类型，apig表示输出到API网关服务。
region	是	数据所在的API网关所在区域。
app_id	是	调用API的应用ID。
encode	是	数据编码格式，可选为“csv”和“json”。 **说明** • 若编码格式为“csv”，则需配置字段分隔符。 • 若编码格式为“json”，则需配置是否输出空字段，具体见示例。
field_delimiter	是	属性分隔符。 • 当编码格式为csv时，需要设置属性分隔符，用户可以自定义，如：“，”。 • 当编码格式为json时，则不需要设置属性之间的分隔符。
json_config	否	当编码格式为json时，用户可以通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1;field2=data_json.field2”。

注意事项

无。

示例

- CSV编码格式：数据输出到API网关，使用csv编码，并且以逗号为分隔符，多个分区用car_owner做为key进行分发。数据输出示例："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编码格式：数据输出到API网关，使用json编码，多个分区用car_owner，car_brand 做为key进行分发。“enableOutputNull”为“true”表示输出空字段（值为null），若为“false”表示不输出空字段。数据示例："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 开源 Kafka 输出流

概述

CS将作业的输出数据输出到Kafka中。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

前提条件

- Kafka服务端的端口如果监听在hostname上，则需要将Kafka Broker节点的hostname和IP的对应关系添加到CS集群中。Kafka Broker节点的hostname和IP请联系Kafka服务的部署人员。如何添加IP域名映射，请参见《 实时流计算服务用户指南 》中**集群管理**章节中的“添加IP域名映射”部分。
- Kafka是线下集群，需要通过VPC服务的对等连接功能将CS服务与Kafka进行对接。

  如何建立对等连接，请参见《 实时流计算服务用户指南 》中**对等连接**章节。

语法

语法格式

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

语法说明

表 5-21 语法说明

参数	是否必选	说明
type	是	输出通道类型，"kafka"表示输出到Kafka中。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过对等连接的方式开通CS集群和Kafka集群的连接）。
kafka_topic	是	写入的topic
encode	是	数据编码格式，可选为“csv”、“json”和“user_defined”。 **说明** • 若编码格式为“csv”，则需配置“field_delimiter”属性。 • 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
filed_delimiter	否	当encode为csv时，用于指定各字段分隔符，默认为逗号。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现编码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现编码类的入参，仅支持一个string类型的参数。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"

注意事项

无

示例

将流kafka_sink的数据输出到Kafka中。

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

5.3 创建中间流

中间流用来简化sql逻辑，若sql逻辑比较复杂，可以写多个sql语句，用中间流进行串接。中间流仅为逻辑意义上的流，不会产生数据存储。

语法格式

CREATE TEMP STREAM stream_id (attr_name attr_type (',' attr_name attr_type)*)

语法说明

无。

注意事项

无。

示例

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

5.4 创建表

5.4.1 创建 Redis 表

创建Redis表用于与输入流连接，数据输出到分布式缓存服务（DCS）的Redis中。

DCS的详细信息请参见《分布式缓存服务用户指南》。

流表JOIN语法请参见流表JOIN。

语法

语法格式

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="")?);

语法说明

表 5-22 语法说明

参数	是否必选	说明
type	是	输出通道类型，dcs_redis表示输出到分布式缓存服务的Redis存储系统中。
cluster_address	是	Redis实例连接地址。
password	否	Redis实例连接密码，当设置为免密访问时，省略该配置项。
value_type	是	指定数据类型。支持的数据类型包括：string, list, hash, set, zset。
key_column	是	指定代表Redis key属性的列名。

参数	是否必选	说明
hash_key_column	否	当value_type设置为hash时，需要指定本字段作为第二级key属性的列名。
cache_max_num	否	表示最大缓存的查询结果数，默认值为32768。
cache_time	否	表示数据库查询结果在内存中缓存的最大时间。单位为毫秒，默认值为10000，当值为0时表示不缓存。

注意事项

1. 请务必确保您的账户下已在分布式缓存服务（DCS）里创建了Redis类型的缓存实例。

   如何创建Redis类型的缓存实例请参考《分布式缓存服务用户指南》。
2. 该场景作业需要运行在CS的独享集群上，因此要与DCS集群建立VPC对等连接；且用户可以根据实际所需设置相应安全组规则。

   如何建立VPC对等连接请参考《实时流计算服务用户指南》中**对等连接**章节。

   如何设置安全组规则请参见《虚拟私有云用户指南》中“安全组”章节。

示例

Redis表用于与输入流连接。

```
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 创建 RDS 表

创建RDS表用于与输入流连接，数据输出到关系型数据库服务（RDS）中。RDS的更多信息，请参见《关系型数据库用户指南》。

流表JOIN语法请参见**流表JOIN**。

前提条件

- 请务必确保您的账户下已在关系型数据库（RDS）里创建了PostgreSQL或MySQL类型的RDS实例。

  如何创建RDS实例，请参见《关系型数据库快速入门》中“购买实例”章节。
- 该场景作业需要运行在CS的独享集群上，因此要与RDS实例建立VPC对等连接，且用户可以根据实际所需设置相应安全组规则。

  如何建立VPC对等连接，请参考《实时流计算服务用户指南》中**对等连接**章节。

  如何设置安全组规则，请参见《虚拟私有云用户指南》中“安全组”章节。

语法

语法格式

```
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 = ""
);
```

语法说明

表 5-23 语法说明

参数	是否必选	说明
type	是	输出通道类型，rds表示输出到关系型数据库中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址，格式为："{database_type}://ip:port/database" 目前支持两种数据库连接：MySQL和PostgreSQL ● MySQL: 'mysql://ip:port/database' ● PostgreSQL: 'postgresql://ip:port/database'
table_name	是	用于查询数据的数据库表名。
db_columns	否	流属性和数据库表的字段对应关系。当sink流中流属性字段和数据库表中的流属性字段不完全匹配时，该参数必配。格式为“dbtable_attr1,dbtable_attr2,dbtable_attr3“。
cache_max_num	否	表示最大缓存的查询结果数，默认值为32768。
cache_time	否	表示数据库查询结果在内存中缓存的最大时间。单位为毫秒，默认值为10000，当值为0时表示不缓存。

示例

RDS表用于与输入流连接。

```
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 数据操作语句

6.1 SQL 语法定义

```
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

语法格式

SELECT [ALL | DISTINCT] { * | projectItem [, projectItem]* } FROM tableExpression [WHERE booleanExpression] [GROUP BY { groupItem [, groupItem]* }] [HAVING booleanExpression]

语法说明

SELECT语句用于从表中选取数据或者插入常量数据。

注意事项

- 所查询的表必须是已经存在的表，否则会出错。
- WHERE关键字指定查询的过滤条件，过滤条件中支持算术运算符，关系运算符，逻辑运算符。
- GROUP BY指定分组的字段，可以单字段分组，也可以多字段分组。

示例

找出数量超过3的订单。

```
insert into temp SELECT  * FROM Orders WHERE units > 3;
```

插入一组常量数据。

```
insert into temp select 'Lily’, ‘male’, ‘student’, 17;
```

WHERE 过滤子句

语法格式

SELECT { * | projectItem [, projectItem]* } FROM tableExpression [WHERE booleanExpression]

语法说明

利用WHERE子句过滤查询结果。

注意事项

- 所查询的表必须是已经存在的，否则会出错。
- WHERE条件过滤，将不满足条件的记录过滤掉，返回满足要求的记录。

示例

找出数量超过3并且小于10的订单。

```
insert into temp SELECT  * FROM Orders WHERE units > 3 and units < 10;
```


HAVING 过滤子句

功能描述

利用HAVING子句过滤查询结果。

语法格式

SELECT [ALL | DISTINCT] { * | projectItem [, projectItem]* } FROM tableExpression [WHERE booleanExpression] [GROUP BY { groupItem [, groupItem]* }] [HAVING booleanExpression]

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤，HAVING子句支持算术运算，聚合函数等。

注意事项

如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。

示例

根据字段name对表student进行分组，再按组将score最大值大于95的记录筛选出来。

```
insert into temp SELECT name, max(score) FROM student GROUP BY name HAVING max(score) >95
```

按列 GROUP BY

功能描述

按列进行分组操作。

语法格式

SELECT [ALL | DISTINCT] { * | projectItem [, projectItem]* } FROM tableExpression [WHERE booleanExpression] [GROUP BY { groupItem [, groupItem]* }]

语法说明

GROUP BY：按列可分为单列GROUP BY与多列GROUP BY。

- 单列GROUP BY：指GROUP BY子句中仅包含一列。
- 多列GROUP BY：指GROUP BY子句中不止一列，查询语句将按照GROUP BY的所有字段分组，所有字段都相同的记录将被放在同一组中。

注意事项

无。

示例

根据score及name两个字段对表student进行分组，并返回分组结果。

```
insert into temp SELECT name,score, max(score) FROM student GROUP BY name,score;
```

用表达式 GROUP BY

功能描述

按表达式对流进行分组操作。

语法格式

SELECT [ALL | DISTINCT] { * | projectItem [, projectItem]* } FROM tableExpression [WHERE booleanExpression] [GROUP BY { groupItem [, groupItem]* }]

语法说明

groupItem：可以是单字段，多字段，也可以是字符串函数等调用，不能是聚合函数。

注意事项

无。

示例

先利用substring函数取字段name的子字符串，并按照该子字符串进行分组，返回每个子字符串及对应的记录数。

```
insert into temp SELECT substring(name,6),count(name) FROM student GROUP BY substring(name,6);
```

GROUP BY 中使用 HAVING 过滤

功能描述

利用HAVING子句在表分组后实现过滤。

语法格式

SELECT [ALL | DISTINCT] { * | projectItem [, projectItem]* } FROM tableExpression [WHERE booleanExpression] [GROUP BY { groupItem [, groupItem]* }] [HAVING booleanExpression]

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。

注意事项

- 如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。HAVING与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。
- HAVING中除聚合函数外所使用的字段必须是GROUP BY中出现的字段。
- HAVING子句支持算术运算，聚合函数等。

示例

先依据num对表transactions进行分组，再利用HAVING子句对查询结果进行过滤，price与amount乘积的最大值大于5000的记录将被筛选出来，返回对应的num及price与amount乘积的最大值。

```
insert into temp SELECT num, max(price*amount) FROM transactions WHERE time > '2016-06-01' GROUP BY num HAVING max(price*amount)>5000;
```

UNION

语法格式

query UNION [ALL] query

语法说明

UNION返回多个查询结果的并集。

注意事项

- 集合运算是以一定条件将表首尾相接，所以其中每一个SELECT语句返回的列数必须相同，列的类型一定要相同，列名不一定要相同。
- UNION默认是去重的，UNION ALL是不去重的。

示例

输出Orders1和Orders2的并集，不包含重复记录。

```
insert into temp SELECT  * FROM Orders1 UNION SELECT  * FROM Orders2;
```

6.3 条件表达式

CASE 表达式

语法格式

CASE value WHEN value1 [, value11]* THEN result1 [WHEN valueN [, valueN1]* THEN resultN]* [ELSE resultZ] END

或

CASE WHEN condition1 THEN result1 [WHEN conditionN THEN resultN]* [ELSE resultZ] END

语法说明

- 当value值为value1则返回result1，都不满足则返回resultZ，若没有else语句，则返回null。
- 当condition1为true时返回result1，都不满足则返回resultZ，若没有else语句，则返回null。

注意事项

- 所有result的类型都必须一致。
- 所有condition类型都必须是布尔类型。
- 当没有满足的分支时，若指定else语句，则返回else的值，若没有else语句，则返回null。

示例

当units等于5时返回1，否则返回0。

示例1：

```
insert into temp SELECT  CASE units WHEN 5 THEN 1 ELSE 0 END FROM Orders;
```

示例2：

```
insert into temp SELECT CASE WHEN units = 5 THEN 1 ELSE 0 END FROM Orders;
```


NULLIF 表达式

语法格式

NULLIF(value, value)

语法说明

如果值相同，则返回NULL。 例如，NULLIF（5，5）返回NULL; NULLIF（5，0）返回5。

注意事项

无。

示例

当units等于3时返回null，否则返回units。

```
insert into temp SELECT  NULLIF(units, 3) FROM Orders;
```

COALESCE 表达式

语法格式

COALESCE(value, value [, value]*)

语法说明

返回从左到右第一个不为NULL的参数的值。

注意事项

所有value的类型都必须一致。

示例

返回5。

```
insert into temp SELECT  COALESCE(NULL, 5) FROM Orders;
```

6.4 窗口

GROUP WINDOW

语法说明

Group Window定义在GROUP BY里，每个分组只输出一条记录，包括以下几种：

- 分组函数

表 6-1 分组函数表

函数名	说明
TUMBLE(time_attr, interval)	跳跃窗口。 time_attr可以设置processing-time或者event-time。 interval设置窗口周期。
HOP(time_attr, interval, interval)	拓展的跳跃窗口(等价于datastream的滑动窗口)，可以分别设置输出触发周期和窗口周期。
SESSION(time_attr, interval)	会话窗口，interval表示多长时间没有记录则关闭窗口。

- 窗口函数

表 6-2 窗口函数表

函数名	说明
TUMBLE_START(time_attr, interval)	返回跳跃窗口开始时间。
TUMBLE_END(time_attr, interval)	返回跳跃窗口结束时间。
HOP_START(time_attr, interval, interval)	返回拓展的跳跃窗口开始时间。
HOP_END(time_attr, interval, interval)	返回拓展的跳跃窗口结束时间。
SESSION_START(time_attr, interval)	返回会话窗口开始时间。
SESSION_END(time_attr, interval)	返回会话窗口结束时间。

示例

```
// 每天计算SUM（金额）（事件时间）。
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;

// 每天计算SUM（金额）（处理时间）。
insert into temp SELECT name,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;

// 每个小时计算事件时间中最近24小时的SUM（数量）。
insert into temp SELECT product,
    SUM(amount)
    FROM Orders
    GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

// 计算每个会话的SUM（数量），间隔12小时的不活动间隙（事件时间）。
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

Over Window与Group Window区别在于Over window每一行都会输出一条记录。

语法格式

OVER ([PARTITION BY partition_name]ORDER BY proctime|rowtime(ROWS number PRECEDING) |(RANGE (BETWEEN INTERVAL '1' SECOND PRECEDING AND CURRENT ROW | UNBOUNDED preceding)))

语法说明

表 6-3

参数	参数说明
PARTITION BY	指定分组的主键，每个分组各自进行计算。
ORDER BY	指定数据按processing time或event time作为时间戳。
ROWS	个数窗口。
RANGE	时间窗口。

注意事项

- 同一select里所有聚合函数定义的窗口都必须保持一致。
- 当前Over窗口只支持前向计算(preceding)，不支持following计算。
- 必须指定ORDER BY 按processing time或event time。
- 不支持对常量做聚合操作，如sum(2)。

示例

```
// 计算从规则启动到目前为止的计数及总和(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;

// 计算最近四条记录的计数及总和(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;

// 计算最近60s的计数及总和(in eventtime),基于事件时间处理，事件时间为Orders中的timeattr字段。
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 流表 JOIN

流与表进行连接操作，从表中查询并补全流字段。目前支持连接RDS表和DCS服务的Redis表。通过ON条件描述查询的Key，并补全表结构的Value字段。

RDS表的数据定义语句请参见**创建RDS表**。

Redis表的数据定义语句请参见**创建Redis表**。

语法

语法格式

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

语法说明

ON条件中只支持表属性等值查询，当存在二级Key时（Redis值类型为HASH情况下），需要AND表达Key和Hash Key等值查询。

注意事项

无。

示例

将车辆信息输入流与车辆价格表做等值连接后，获取车辆价格信息并填入车辆信息输出流后输出。

```
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 = ","
);

/** 创建数据维表，用于和输入流连接，实现字段回填
  *
  * 根据实际情况修改以下选项：
  * value_type：redis的键值对应值类型，支持STRING、HASH、SET、ZSET、LIST，其中HASH类型需要指定
hash_key_column作为二层主键，集合类型将用逗号拼接所有查询出来的值
  * key_column：维表主键对应的列名
  * hash_key_column：当redis的键值对应值类型为HASH时，HASHMAP的KEY对应的列名，当值类型非HASH
时，无需指定改配置
  * cluster_address：DCS服务redis集群地址
  * password：DCS服务redis集群密码
  **/
```



```
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 自定义函数

概述

CS支持三种自定义函数：

- UDF：自定义函数，支持一个或多个输入参数，返回一个结果值。
- UDTF：自定义表值函数，支持一个或多个输入参数，可返回多行多列。
- UDAF：自定义聚合函数，将多条记录聚合成一个值。

说明

自定义函数仅能在独享集群中使用，不支持在共享集群中使用。

POM 依赖

```
<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>
```


示例程序

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

使用方式

1. 将写好的自定义函数打成JAR包，并上传到OBS上。
2. 在CS管理控制台的左侧导航栏中，单击“作业管理”，在需要编辑作业对应的“操作”列中，单击“编辑”，进入作业编辑页面。
3. 在“运行参数设置”页签，“自定义函数Jar包”选择存放在OBS上的JAR文件，单击“保存”。

   选定JAR包以后，SQL里添加UDF声明语句，就可以像普通函数一样使用了。

   ```
   CREATE FUNCTION udf_test AS 'com.huawei.udf.UdfScalarFunction';
   ```

UDF

UDF函数需继承ScalarFunction函数，并实现eval方法。open函数及close函数可选。

编写代码示例

```
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;
  }
  /**
   * 初始化操作，可选
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  /**
   * 自定义逻辑
   * @param s
   * @return
   */
   public int eval(String s) {
     return s.hashCode() * factor;
   }
   /**
    * 可选
    */
   @Override
   public void close() {}
}
```

使用示例

```
CREATE FUNCTION udf_test AS 'com.huawei.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

UDTF函数需继承TableFunction函数，并实现eval方法。open函数及close函数可选。如果需要UDTF返回多列，只需要将返回值声明成Tuple或Row即可。若使用Row，需要重载getResultType声明返回的字段类型。

编写代码示例


```
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);
  /**
   * 初始化操作，可选
   * @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);
    }
  }
  /**
   * 函数返回类型声明
   * @return
   */
  @Override
  public TypeInformation<Row> getResultType() {
  return Types.ROW(Types.STRING, Types.INT);
  }
  /**
   * 可选
   */
  @Override
  public void close() {}
}
```

使用示例

UDTF支持CROSS JOIN和LEFT JOIN，在使用UDTF时需要带上 LATERAL 和TABLE 两个关键字。

- CROSS JOIN：对于左表的每一行数据，假设UDTF不产生输出，则这一行不进行输出。
- LEFT JOIN：对于左表的每一行数据，假设UDTF不产生输出，这一行仍会输出，UDTF相关字段用null填充。

```
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

UDAF函数需继承AggregateFunction函数。首先需要创建一个用来存储计算结果的Accumulator，如示例里的WeightedAvgAccum。

编写代码示例

```
public class WeightedAvgAccum {
public long sum = 0;
public int count = 0;
}
```



```
import org.apache.flink.table.functions.AggregateFunction;
import java.util.Iterator;
/**
* 第一个类型变量为聚合函数返回的类型，第二个类型变量为Accumulator类型
* Weighted Average user-defined aggregate function.
*/
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {
  // 初始化Accumulator
  @Override
  public WeightedAvgAccum createAccumulator() {
    return new WeightedAvgAccum();
  }
  // 返回Accumulator存储的中间计算值
  @Override
  public Long getValue(WeightedAvgAccum acc) {
    if (acc.count == 0) {
      return null;
    } else {
      return acc.sum / acc.count;
 }
}
// 根据输入更新中间计算值
public void accumulate(WeightedAvgAccum acc, long iValue) {
acc.sum += iValue;
acc.count += 1;
}
// Restract撤回操作，和accumulate操作相反
public void retract(WeightedAvgAccum acc, long iValue) {
acc.sum -= iValue;
acc.count -= 1;
}
// 合并多个accumulator值
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;
}
}
// 重置中间计算值
public void resetAccumulator(WeightedAvgAccum acc) {
acc.count = 0;
acc.sum = 0L;
}
}
```

使用示例

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


7 配置时间模型

Flink中主要提供两种时间模型：Processing Time和Event Time。

CS允许在创建Source Stream和Temp Stream的时候指定时间模型以便在后续计算中使用。

配置 Processing Time

Processing Time是指系统时间，与数据本身的时间戳无关，即在Flink算子内计算完成的时间。

语法格式

CREATE SOURCE STREAM stream_name(...) WITH (...)

TIMESTAMP BY proctime.proctime;

CREATE TEMP STREAM stream_name(...)

TIMESTAMP BY proctime.proctime;

语法说明

设置Processing Time只需在timestamp by后配置proctime.proctime即可，后续可以直接使用proctime字段。

注意事项

无。

示例

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* 学号 */
  student_name STRING, /* 姓名 */
  subject STRING, /* 学科 */
  score INT /* 成绩 */
)
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;
```

配置 Event Time

Event Time是指事件产生的时间，即数据产生时自带时间戳。

语法格式

CREATE SOURCE STREAM stream_name(...) WITH (...)

TIMESTAMP BY {attr_name}.rowtime

SET WATERMARK (RANGE {time_interval} | ROWS {literal}, {time_interval});

语法说明

设置Event Time需要选定流中的某一个属性来作为时间戳，同时需要设置Watermark策略。

由于网络等原因，有时会导致乱序的产生；对于迟来的数据，需要Watermark来保证一个特定的时间后去触发Window进行计算。Watermark主要是用来处理乱序数据，流处理从事件产生，到发送到CS服务，中间有一个过程。

Watermark有两种设置策略：

- 按时间周期
  SET WATERMARK(range interval {time_unit}, interval {time_unit})
- 按事件个数
  SET WATERMARK(rows literal, interval {time_unit})

说明

一个逗号表示一个参数，第一个参数表示Watermark发送周期，第二个参数表示允许最大延迟时间。

注意事项

无。

示例

- 每10s发送一次watermark，事件最大允许延迟时间为20s。

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* 学号 */
  student_name STRING, /* 姓名 */
  subject STRING, /* 学科 */
  score INT, /* 成绩 */
  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;
```

- 每收到10个数据发送一次watermark，事件最大允许延迟时间为20s。

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* 学号 */
  student_name STRING, /* 姓名 */
  subject STRING, /* 学科 */
  score INT, /* 成绩 */
  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 CEP 模式匹配

复杂事件处理（Complex Event Process，简称CEP）用来检测无尽数据流中的复杂模式，拥有从不同的数据行中辨识查找模式的能力。模式匹配是复杂事件处理的一个强大援助。

例子包括受一系列事件驱动的各种业务流程，例如在安全应用中侦测异常行为；在金融应用中查找价格、交易量和其他行为的模式。其他常见的用途如欺诈检测应用和传感器数据的分析等。

语法格式

```
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
```

说明

SQL中的模式匹配是用MATCH_RECOGNIZE子句执行。MATCH_RECOGNIZE子句执行如下任务：

- 使用PARTITION BY 和ORDER BY子句对MATCH_RECOGNIZE子句中的数据进行逻辑分区和排序。
- 使用PATTERN子句来定义要查找的数据行的模式。这些模式使用规则表达式语法。
- 使用DEFINE子句指定PATTERN模式变量所需的逻辑条件。
- 使用MEASURES子句定义度量，这是一些可在SQL查询的其他部分所使用的表达式。

语法说明

表 8-1 语法说明

参数	是否必选	说明
PARTITION BY	否	将数据行进行逻辑上的分组。
ORDER BY	否	在分区中对数据行进行逻辑排序。
[ONE ROW | ALL ROWS] PER MATCH	否	为每个匹配选择输出汇总或者明细。 ● ONE ROW PER MATCH：每次检测到完整的匹配后进行汇总输出。 ● ALL ROWS PER MATCH：检测到完整的匹配后会把匹配过程中每条具体记录进行输出。 示例如下： `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` 示例说明： 假设MyTable数据格式为(id,name), 有三条数据(1,a) (2,b), (3,c)。 那么ONE ROW PER MATCH会输出B的平均值2。 ALL ROWS PER MATCH会将每条记录及B的平均值输出，也就是输出(1,a, null), (2,b,2), (3,c,2)。
MEASURES	否	定义要输出的度量值。

参数	是否必选	说明
PATTERN	是	定义要匹配的模式。 ● 连续事件 PATTERN (A B C)即表示检测连续的ABC事件。 ● 逻辑事件PATTERN (A | B)即表示检测A或者B。 ● 输出排除- A -，只有用在ALL ROWS PER MATCH里才有意义，通过指定排除某一模式变量，可以不输出该变量的匹配行。如： `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` 示例说明： 假设MyTable数据格式为(id,name), 有三条数据(1,a) (2,b), (3,c)。 模式B被排除，模式还是按照ABC进行检测，但是在输出时只会输出(1,a, null), (3,c,2), B的记录不会输出。 ● 修饰符 – *： 0次或多次迭代，如A* 匹配0次或多次A – +： 1次或多次迭代，如A+ 匹配1次或多次A – ?： 0次或1次迭代，如A? 匹配0次或多次A – {n}： n 次迭代 (n > 0)，如A{5} 匹配5次A – {n,}： n 次或更多次迭代 (n >= 0)，如A{5,} 匹配>=5次A – {n,m}： n次至m次 (包括n和m) 迭代 (0 <= n <= m, 0 < m)，如A{3,6} 匹配3至6次A – {,m}： 0次至m次 (包括0和m) 迭代 (m > 0)，如A{,4} 匹配0至4次A

参数	是否必选	说明
SUBSET	否	定义组合变量。 示例如下：E为BC的组合，求E.id平均值即求BC合集的平均值。 `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	是	定义主要的模式变量条件。
AFTER MATCH SKIP	否	定义在一个匹配找到之后从哪里开始下一轮匹配。 ● SKIP TO NEXT ROW ： 在当前匹配第一行之后的下一行开始下一轮模式匹配 ● SKIP PAST LAST ROW ： 在当前匹配的最后一行之后的下一行开始下一轮匹配 ● SKIP TO FIRST variable：从当前匹配的第一个variable开始下一轮匹配 ● SKIP TO LAST variable：从当前匹配的最后一个variable开始下一轮匹配 ● SKIP TO variable：同SKIP TO LAST variable

CEP 支持的函数

表 8-2 函数说明

函数	说明
MATCH_NUMBER()	说明本次匹配属于第几次匹配。可用在MEASURES和DEFINE子句中。
CLASSIFIER()	说明当前记录被匹配到PATTERN里的哪个模式变量里。可用在MEASURES和DEFINE子句中。
FIRST()/LAST()	每次匹配里的第一个/最后一个记录。比如PATTERN (A B+ C), FIRST(B.id)代表匹配里的第一个B的id，LAST(B.id)代表匹配里的最后一个B的id。
NEXT()/PREV()	相对偏移，可用在DEFINE里。比如PATTERN (A B+) DEFINE B AS B.price > PREV(B.price)。

函数	说明
RUNNING/FINAL	RUNNING 表示匹配过程中间值，FINAL表示最终结果值，RUNNING/FINAL一般只在ALL ROWS PER MATCH里才有意义。比如有三条记录(a, 2), (b, 6), (c, 12), 那么RUNNING AVG(A.price)和FINAL AVG(A.price)的值就是(2, 6), (4, 6), (6, 6)。
聚合函数(COUNT, SUM, AVG, MAX, MIN)	聚合操作，可用在MEASURES和DEFINE子句中。关于聚合函数的详细信息，请参见**聚合函数**。

示例

- 套牌车检测

5分钟内在不同区域的城市道路或者高速道路的摄像头采集到相同牌照的车辆数据，通过对号牌切换特征的模式匹配，实现套牌车检测。

```
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;
```

该规则表示5分钟内在两个不同摄像区域内检测到同一车牌号车辆，为了防止出现误判，即车辆确实从A区域行驶到B区域，检查到B区域后A区域又检测到了该车牌，这种情况则认为是真正的套牌车。

输入数据：

浙B88888,zone_A

浙AZ626M,zone_A

浙B88888,zone_A

浙AZ626M,zone_A

浙AZ626M,zone_A

浙B88888,zone_B

浙B88888,zone_B

浙AZ626M,zone_B

浙AZ626M,zone_B

浙AZ626M,zone_C

浙B88888,zone_A

浙B88888,zone_A

则会输出：

浙B88888,zone_A,zone_B

- 告警抑制，如果连续出现多次A事件只输出一次A事件

```
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;
```

该语句表示一检测到与之前不一样的事件即进行输出，如果与之前一样则不再进行输出。

输入数据：

1,A

2,A

3,A

4,B

5,B

6,C

7,D

8,D

输出数据：

A

B

C

D

9 StreamingML

9.1 异常检测

异常检测应用场景相当广泛，包括了入侵检测，金融诈骗检测，传感器数据监控，医疗诊断和自然数据检测等。异常检测经典算法包括统计建模方法，基于距离计算方法，线性模型和非线性模型等。

我们采用一种基于随机森林的异常检测方法：

- One-pass算法，O(1)均摊时空复杂度。
- 随机森林结构仅构造一次，模型更新仅仅是节点数据分布值的更新。
- 节点存储多个窗口的数据分布信息，能够检测数据分布变化。
- 异常检测和模型更新在同一个代码框架中完成。

语法格式

```
SRF_UNSUP(ARRAY[字段1, 字段2, ...], '可选参数列表')
```

说明

- 函数输出为[0, 1]区间的DOUBLE值，表示数据的异常打分。
- 字段名必须为一致的数值类型，若字段类型不同，可通过CAST函数转义，例如[a, CAST(b as DOUBLE)]。
- 可选参数列表语法为"key1=value,key2=value2,..."。

参数说明

表 9-1 参数说明

参数	是否必选	说明	默认值
transientThreshold	否	连续transientThreshold个窗口发生数据改变表示发生数据概念迁移。	5
numTrees	否	随机森林中Tree的数量。	15

参数	是否必选	说明	默认值
maxLeafCount	否	Tree最大叶子节点数量。	15
maxTreeHeight	否	Tree最大高度。	12
seed	否	算法使用的随机种子值。	4010
numClusters	否	分类数，默认包含异常和非异常两类。	2
dataViewMode	否	算法学习模式。 ● history：学习所有历史数据。 ● horizon：仅考虑最近一段时间历史数据，默认为4个窗口。	history

示例

对于数据流MyTable中的c字段运行异常检测算法，当异常分大于0.8时输出异常。

```
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 时间序列预测

流数据处理中经常需要对于时间序列数据进行建模和预测，建模是指提取数据中有用的统计信息和数据特征，预测是指使用模型对未来的数据进行推测。CS服务提供了一系列随机线性模型，帮助用户在线实时进行模型的建模和预测。

ARIMA (Non-Seasonal)

ARIMA（Auto-Regressive Integrated Moving Average）是时间序列预测中的经典模型，和AR/MA/ARMA模型之间联系紧密。

- AR/MA/ARMA适用于**平稳**序列 (stationary)
  - AR(p)：自回归模型，当前值可以描述为p个之前值的线性组合。利用线性组合的权值即可预测下一个值。
  - MA(q)：移动平均模型，当前值可以描述为序列均值加上q个之前值的白噪声的线性组合。利用线性组合的权值也可预测下一个值。
  - ARMA(p, q)：自回归移动平均模型，综合了AR和MA两个模型的优势，在ARMA模型中，自回归过程负责量化当前数据与前期数据之间的关系，移动平均过程负责解决随机变动项的求解问题，因此，该模型比AR/MA更为有效和常用。
- ARIMA适用于**非平稳**序列 (non-stationary)。ARIMA(p, q, d)中p为自回归项数，q为滑动平均项数，d为使之成为平稳序列所做的差分次数（阶数）。

语法格式


```
AR_PRED(field, degree)：使用AR模型预测新数据。
AR_COEF(field, degree)：返回AR模型的权值。
ARMA_PRED(field, degree)：使用ARMA模型预测新数据。
ARMA_COEF(field, degree)：返回ARMA模型的权值。
ARIMA_PRED(field, degree, derivativeOrder)：使用ARIMA预测新数据。
```

表 9-2 参数说明

参数	是否必选	说明	默认值
field	是	数据在数据流中的字段名。	-
degree	否	指定使用之前数据项的个数，当前实现中限定p = q = degree。	5
derivativeOrder	否	指定差分次数，通常设置为1或者2。	1

示例

分别使用AR，ARMA，ARIMA结合窗口进行时间序列预测。

```
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

Holt Winters算法是Exponential smoothing方法中的一种，主要特点是可以捕捉时间序列中的**季节性**趋势。

语法格式

```
HOLT_WINTERS(field, seasonality, forecastOrder)
```

表 9-3 参数说明

参数	是否必选	说明
field	是	数据在数据流中的字段名。
seasonality	是	季节性变化的周期。例如数据点是按天采集，季节性周期是一周，则seasonality为7。
forecastOrder	否	指定需要预测的元素。 当forecastOrder为1，预测下一个元素。 当forecastOrder为2，预测下下一个元素。默认值为1。 使用此参数时需要保证over窗口的大小大于设置的forecastOrder。

示例

使用HOLT WINTERS函数结合窗口进行时间序列预测。

```
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 实时聚类

聚类算法是非监督算法中非常典型的一类算法，经典的K-Means算法通过提前确定类别数目，计算数据点之间的距离来分类。对于离线静态数据集，我们可以依赖领域中知识来确定类别数目，运行K-Means算法可以取得比较好的聚类效果。但是对于在线实时流数据，数据是在不断变化和演进，类别数目极有可能发生变化，CS服务提供一种能够应对此类场景，无需提前设定聚类数目，并且低延时的在线聚类算法。

算法大致思想为：定义一种距离函数，两两数据点之间如果距离小于某个阈值，则他们属于同一个类别。若某数据点和多个类别中心点的距离都小于这个阈值，则多个类别会发生合并操作。当数据流中的数据到达，算法会分别计算与所有类别的距离，从而决定此数据作为一个新类别或者归属于某类别。

语法格式

```
CENTROID(ARRAY[field_names], distance_threshold)：加入当前数据点后，该数据点所属分类中心。
CLUSTER_CENTROIDS(ARRAY[field_names], distance_threshold)：加入当前数据点后，所有分类中心。
ALL_POINTS_OF_CLUSTER(ARRAY[field_names], distance_threshold)：加入当前数据点后，该分类所有数据点。
ALL_CLUSTERS_POINTS(ARRAY[field_names], distance_threshold)：加入当前数据点后，所有分类对应的所有数据点。
```

说明

- 聚类算法可以应用在**无界流**中。

参数说明

表 9-4 参数说明

参数	是否必选	说明
field_names	是	数据在数据流中的字段名，多字段以逗号隔开。例如ARRAY[a, b, c]。
distance_threshold	是	距离阈值，当两数据点距离小于阈值时，它们将属于同一个类别。

示例

分别使用四种函数结合窗口来实时计算聚类的相关信息。

```
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 深度学习模型预测

深度学习已经广泛应用于图像分类、图像识别和语音识别等不同领域，CS服务中提供了若干函数实现加载深度学习模型并进行预测的能力。

目前可支持的模型包括DeepLearning4j 模型和Keras模型。由于Keras它能够以 TensorFlow、CNTK或者 Theano 作为后端运行，导入来自Keras的神经网络模型，可以借此导入Theano、Tensorflow、Caffe、CNTK等主流学习框架的模型。

语法格式

```
-- 图像分类， 返回预测图像分类的类别id
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) -- 适用于
Keras模型

-- 文本分类，返回预测文本分类的类别id
DL_TEXT_MAX_PREDICTION_INDEX(field_name, model_path, is_dl4j_model) -- 采用默认word2vec模型
DL_TEXT_MAX_PREDICTION_INDEX(field_name, word2vec_path, model_path, is_dl4j_model)
```

说明

模型及配置文件等需存储在用户的OBS中，路径格式为"obs://**your_ak**:**your_sk**@obs.**your_obs_region**.myhuaweicloud.com:443/**your_model_path**"。例如你的模型存放在华北区的OBS上，桶名为dl_model，文件名为model.h5，则路径填写为"obs://**your_ak**:**your_sk**@obs.cn-north-1.myhuaweicloud.com:443/dl_model/model.h5"。

参数说明

表 9-5 参数说明

参数	是否必选	说明
field_name	是	数据在数据流中的字段名。 图像分类中field_name类型需声明为ARRAY[TINYINT]。 文本分类中field_name类型需声明为String。
model_path	是	模型存放在OBS上的完整路径，包括模型结构和模型权值。
is_dl4j_model	是	是否是deeplearning4j的模型。 true代表是deeplearning4j，false代表是keras模型。

参数	是否必选	说明
keras_model_config_path	是	模型结构存放在OBS上的完整路径。在keras中通过model.to_json()可得到模型结构。
keras_weights_path	是	模型权值存放在OBS上的完整路径。在keras中通过model.save_weights(filepath)可得到模型权值。
word2vec_path	是	word2vec模型存放在OBS上的完整路径。

示例

图片分类预测我们采用Mnist数据集作为流的输入，通过加载预训练的deeplearning4j模型或者keras模型，可以实时预测每张图片代表的数字。

```
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
```

文本分类预测我们采用一组新闻标题数据作为流的输入，通过加载预训练的deeplearning4j模型或者keras模型，可以实时预测每个新闻标题所属的类别，比如经济，体育，娱乐等。

```
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 保留关键字

Stream SQL将一些字符串组合保留为关键字以备将来使用。 如果要使用以下字符串作为字段名称，请确保使用反引号（例如`value`，`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

