流计算 Oceanus

SQL 开发指南

产品文档

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2020 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

SQL 开发指南
- 概述
- 术语和数据类型
  - 术语表
  - 数据类型
- 上下游开发指南
  - Kafka
  - JDBC
  - Elasticsearch
  - 自定义 Connector
- DDL 数据定义语句
  - CREATE TABLE
  - CREATE VIEW
  - CREATE FUNCTION
- SET 控制语句
  - Flink 配置项
- DML 数据操作语句
  - 查询语句
  - INSERT 语句
- 运算符和内置函数
  - 概览
  - 比较函数
  - 逻辑和位运算函数
  - 算术函数
  - 字符串操作函数
  - 条件函数
  - 类型转换函数
  - 时间相关函数
  - 聚合函数
  - 时间窗口函数
  - 其他函数
- 标识符与保留字
  - 命名规则
  - 保留字

SQL 开发指南

概述

最近更新时间：2020-09-16 17:03:25

本章节主要提供在独享集群上开发 SQL 作业的指南，包含了下面的内容：

- 术语和数据类型
- 上下游开发指南
- DDL 数据定义语句
- SET 控制语句
- DML 数据操作语句
- 运算符与内置函数
- 标识符与保留字

版权所有：腾讯云计算（北京）有限责任公司

术语和数据类型

术语表

最近更新时间：2020-12-01 14:53:04

流计算常用术语如下：

术语	详细说明
流计算	流计算是面向流式数据的计算，它从一个或多个流式数据源读取持续不断产生的数据，经过引擎中多个算子的组合进行高效计算，再根据实际需要，将结果输出至下游的多种数据目的，例如消息队列、数据库、数据仓库、存储服务等。
数据源（Source）	为流计算系统持续提供输入数据，例如腾讯云 CKafka 等。
数据目的（Sink）	流计算系统处理结果输出的地方，例如腾讯云 CKafka、云数据库 MySQL、PostgreSQL 等。
Schema	表示一个表的结构信息，例如各个列名、列类型等。对于 PostgreSQL 而言，Schema 是介于 Database 和 Table 之间的一个层级，可以理解成数据库内部的命名空间。
时间模式	指导系统处理数据时如何获取时间戳，目前支持 Event Time、Processing Time、Source Time 三种时间模式。
Event Time	Event Time 时间模式下，时间戳由输入数据的某个字段提供，可以用 WATERMARK FOR 语句指定该字段并启用 Event Time 时间模式，适用于数据源包含精确时间戳的场合。
Watermark	表示一个特定的时间点，在该时间点之前的所有数据已经得到妥善处理。 Watermark 由系统自动生成，用户可通过 WATERMARK FOR BOUNDED 语句指定时间戳的最大容差。
Processing Time	Processing Time 时间模式下，时间戳由系统自动生成并添加到数据源中（以`PROCTIME`命名，`SELECT *`时不可见，使用时必须显式指定）。它以每条数据被系统处理的时间作为时间戳，因而有一定的不可控性，适用于对时间精度要求不是很高的场合。
Source Time	在 Source Time 时间模式下，可使用 Kafka 每条记录所含元数据的时间戳作为流计算处理所使用的时间戳（以`SOURCETIME` 命名，`SELECT *`时不可见，使用时必须显式指定），避免了输入数据没有时间戳字段时，使用 Processing Time 模式带来的不可控性。
时间窗口	定义了多个时间段以及各个时间段之间的关系（例如是否可重叠、是否固定大小）。目前系统支持 TUMBLE、HOP、SESSION 三种时间窗口。具体见 时间窗口函数。
TencentDB for MySQL	TencentDB 是腾讯云提供的一种高性能、高可靠、可灵活伸缩的数据库托管服务。它让用户可以轻松在云端部署、使用 MySQL、PostgreSQL 等数据库。
CKafka	CKafka 是腾讯云提供的一个分布式的、高吞吐量、高可扩展性的消息系统，完全兼容0.10版本的 Kafka API。流计算目前支持 CSV 和 JSON 两种输入输出格式。
Tuple 与 Append 流	Tuple（又称为 Append）为数据流类型的一种，可以存放不含主键的流数据。用户可以不断追加新数据到这种数据流中，它不涉及到对之前已发出数据的更新操作。目前各种数据源和数据目的均支持 Append 流的输入输出。

版权所有：腾讯云计算（北京）有限责任公司

术语	详细说明
Upsert 流	Upsert 是 Update OR Insert 的简写，由 DISTINCT、不含时间窗口的 GROUP BY 语句、不含时间范围的 JOIN 语句等查询产生，它具有主键定义，如果后续发出的数据与之前的某条数据具有相同主键，则更新该条记录为新值；反之则新增一行数据。它可以确保之前发出的数据被更新以反映最新的值。目前只有使用云数据库 MySQL 和 PostgreSQL 作为数据目的（Sink）时，支持 Upsert 流的写入。
DDL 语句	DDL 即数据定义语言（Data Definition Language），为 SQL 语言的一个子集，由 CREATE 语句组成。它可以用来定义表、视图、以及用户自定义函数（UDF）等。
DML 语句	DML 即数据操作语言（Data Manipulation Language），为 SQL 语言的一个子集，包括 INSERT 和 SELECT 语句，可以用来对数据表、视图等进行选择、变换、筛选、插入等操作。

数据类型

最近更新时间：2020-09-16 17:03:44

流计算 Oceanus 采用符合 ANSI SQL 规范的定义，支持丰富的数据类型。用户在使用 `CREATE TABLE` 语句定义一个数据表时，可以用这些数据类型来定义每个字段的类型。

支持的类型列表

类型名称	使用说明
• CHAR • CHAR(n)	定长字符串。n 表示容纳的字符数，默认为1，即 CHAR 等价于 CHAR(1)。
• VARCHAR • VARCHAR(n) • STRING	可变长度字符串。n 表示最多容纳的字符数，默认为1，即 VARCHAR 等价于 VARCHAR(1)。**STRING 等价于 VARCHAR(2147483647)。**
• BINARY • BINARY(n)	固定长度的二进制字符串。n 表示容纳的字节数量，默认为1，即 BINARY 等价于 BINARY(1)。
• VARBINARY • VARBINARY(n) • BYTES	可变长度的二进制字符串。n 表示容纳的字节数量，默认为1，即 VARBINARY 等价于 VARBINARY(1)。**BYTES 等价于 VARBINARY(2147483647)。**
• DECIMAL • DECIMAL(p) • DECIMAL(p, s) • DEC • DEC(p) • DEC(p, s) • NUMERIC • NUMERIC(p) • NUMERIC(p, s)	固定精度的实数（定点数）。 • p 表示数字的总位数（精度），取值区间为[1，38]，默认值是10。 • s 表示小数点右边的位数（尾数），取值区间为[0，p]，默认值是0。 **DEC 和 NUMERIC 是 DECIMAL 的别名，可以任意互换使用，即 DECIMAL(p, s) 等价于 DEC(p, s) 也等价于 NUMERIC(p, s)。**
TINYINT	1个字节的整数。等价于 Java 的 Byte 类型，取值范围是[-128，127]。
SMALLINT	2个字节的整数。等价于 Java 的 Short 类型，取值范围是[-32768，32767]。
INT	4个字节的整数。等价于 Java 的 Integer 类型，取值范围是[-2147483648，2147483647]。
BIGINT	8个字节的整数。等价于 Java 的 Long 类型，取值范围是[-9223372036854775808，9223372036854775807]。
FLOAT	4个字节的单精度浮点数。等价于 Java 的 Float 类型。
DOUBLE	8个字节的双精度浮点数。等价于 Java 的 Double 类型。
DATE	日期类型，包含年-月-日。取值范围是[0000-01-01，9999-12-31]。

<table>
<tr><th>类型名称</th><th>使用说明</th></tr>
<tr><td>• TIME
• TIME(p)</td><td>不含时区信息的时间类型，包含时-分-秒及纳秒信息。
• 取值范围是[00:00:00.000000000，23:59:59.999999999]。
• p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为0。
• 此类型不支持闰秒。类似于 Java 的 LocalTime 类型。</td></tr>
<tr><td>• TIMESTAMP
• TIMESTAMP(p)
• TIMESTAMP WITHOUT TIME ZONE
• TIMESTAMP(p) WITHOUT TIME ZONE</td><td>不含时区信息的时间戳类型，精度可以达到纳秒级别。
• 取值范围是[0000-01-01 00:00:00.000000000，9999-12-31 23:59:59.999999999]。
• p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为6。
• 此类型不支持与 BIGINT（Java 的 Long 类型）之间相互转换。 TIMESTAMP WITHOUT TIMEZONE 类型等价于 TIMESTAMP 类型。
• 此类型不支持闰秒。类似于 Java 的 Timestamp 类型。</td></tr>
<tr><td>• TIMESTAMP WITH TIME ZONE
• TIMESTAMP(p) WITH TIME ZONE</td><td>含时区信息的时间戳类型。
• 取值范围是[0000-01-01 00:00:00.000000000 +14:59，9999-12-31 23:59:59.999999999 -14:59]。
• p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为6。
• 每条该类型的数据，都含有各自的时区信息。
• 此类型不支持闰秒。类似于 Java 的 OffsetDateTime 类型。</td></tr>
<tr><td>• TIMESTAMP WITH LOCAL TIME ZONE
• TIMESTAMP(p) WITH LOCAL TIME ZONE</td><td>含本地时区信息的时间戳类型。
• 取值范围是[0000-01-01 00:00:00.000000000 +14:59，9999-12-31 23:59:59.999999999 -14:59]。
• p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为6。
• 时区数据不存储在每条数据中，而是遵循全局的时区设置。
• 此类型不支持闰秒。类似于 Java 的 OffsetDateTime 类型。</td></tr>
<tr><td>• INTERVAL YEAR
• INTERVAL YEAR(p)
• INTERVAL YEAR(p) TO MONTH
• INTERVAL MONTH</td><td>表示以年和月表示的一段粗粒度的时间间隔，精度为月份。
• 语法为+年数-月数，例如+04-02。
• 取值范围是[-9999-11，+9999-11]。
• p 表示年的精度位数，取值范围是[1，4]，默认为2。</td></tr>
<tr><td>• INTERVAL DAY
• INTERVAL DAY(p1)
• INTERVAL DAY(p1) TO HOUR
• INTERVAL DAY(p1) TO MINUTE
• INTERVAL DAY(p1) TO SECOND(p2)
• INTERVAL HOUR
• INTERVAL HOUR TO MINUTE
• INTERVAL HOUR TO SECOND(p2)
• INTERVAL MINUTE
• INTERVAL MINUTE TO SECOND(p2)
• INTERVAL SECOND
• INTERVAL SECOND(p2)</td><td>表示以天、时、分、秒、纳秒表示的细粒度时间间隔，最高精度为纳秒。
• 取值范围是[-999999 23:59:59.999999999，+999999 23:59:59.999999999]。
• p1 表示天数精度的位数，p1 的取值范围是[1，6]，默认为2。
• p2 表示秒的小数位数，p2 的取值范围是[0，9]，默认为6。</td></tr>
</table>

版权所有：腾讯云计算（北京）有限责任公司

类型名称	使用说明
• ARRAY<t> • t ARRAY	数组类型，大小固定为2147483647。 • t 表示数组中元素的类型。 • **ARRAY<t>等价于 t ARRAY，例如ARRAY<INT>与 INT ARRAY 含义一致。**
MAP<kt, vt>	键值对映射类型，其中 kt 是键（key）的类型，vt 是值（value）的类型。
• MULTISET<t> • t MULTISET	允许重复元素的集合类型，别称为 Bag。同样地，MULTISET<t>等价于 t MULTISET。
• ROW<n0 t0, n1 t1, ...> • ROW<n0 t0 'd0', n1 t1 'd1', ...> • ROW(n0 t0, n1 t1, ...) • ROW(n0 t0 'd0', n1 t1 'd1', ...)	允许包含多个字段的复合类型，每个字段有自己的类型，类似于其他语言的 Struct 及 Tuple 类型。 • n 表示字段名。 • t 是字段的逻辑类型。 • d 是字段的描述。 • **尖括号和圆括号的两种写法是等价的，例如 ROW(field1 INT, field2 BOOLEAN) 等同于 ROW<field1 INT, field2 BOOLEAN>。**
BOOLEAN	三值布尔型，可选值为 TRUE、FALSE 和 UNKNOWN。如果不允许出现 UNKNOWN，可以定义为 BOOLEAN NOT NULL 类型。
RAW('class', 'snapshot')	可表示任意类型，例如 Flink 无法识别或者不需要识别的类型。 • class 表示原始类型。 • snapshot 表示 Base64 编码的序列化后的 TypeSerializerSnapshot 定义。
NULL	空值，类似 Java 等语言中的 null 值。

版权所有：腾讯云计算（北京）有限责任公司

上下游开发指南
Kafka

最近更新时间：2020-09-24 15:31:44

介绍

Kafka 数据管道是流计算系统中最常用的数据源（Source）和数据目的（Sink）。用户可以把流数据导入到 Kafka 的某个 Topic 中，通过 Flink 算子进行处理后，输出到相同或不同 Kafka 示例的另一个 Topic。

Kafka 支持同一个 Topic 多分区读写，数据可以从多个分区读入，也可以写入到多个分区，以提供更高的吞吐量，减少数据倾斜和热点。

使用范围

Kafka 支持用作数据源表（Source），也可以作为 Tuple 数据流的目的表（Sink），暂不支持 Upsert 数据流。

Kafka 还可以与 Debezium、Canal 等联用，对 MySQL、PostgreSQL 等传统数据库的变更进行捕获和订阅，然后 Flink 即可对这些变更事件进行进一步的处理。

示例：用作数据源（Source）

JSON 格式输入

```
CREATE TABLE `Data-Input` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 定义 Kafka 参数
'connector' = 'kafka',
'topic' = 'Data-Input', -- 替换为您要消费的 Topic
'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offset
s / group-offsets / timestamp 的任何一种
'properties.bootstrap.servers' = '172.28.28.13:9092', -- 替换为您的 Kafka 连接地址
'properties.group.id' = 'testGroup', -- 必选参数, 一定要指定 Group ID

-- 定义数据格式 (JSON 格式)
'format' = 'json',
'json.fail-on-missing-field' = 'false', -- 如果设置为 false, 则遇到缺失字段不会报错。
'json.ignore-parse-errors' = 'true' -- 如果设置为 true, 则忽略任何解析报错。
);
```

CSV 格式输入

版权所有：腾讯云计算（北京）有限责任公司

```
CREATE TABLE `Data-Input` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 定义 Kafka 参数
'connector' = 'kafka',
'topic' = 'Data-Input', -- 替换为您要消费的 Topic
'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offset
s / group-offsets / timestamp 的任何一种
'properties.bootstrap.servers' = '172.28.28.13:9092', -- 替换为您的 Kafka 连接地址
'properties.group.id' = 'testGroup', -- 必选参数, 一定要指定 Group ID

-- 定义数据格式 (CSV 格式)
'format' = 'csv'
);
```

Debezium 格式输入

```
CREATE TABLE `Data-Input` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 定义 Kafka 参数
'connector' = 'kafka',
'topic' = 'Data-Input', -- 替换为您要消费的 Topic
'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offset
s / group-offsets / timestamp 的任何一种
'properties.bootstrap.servers' = '172.28.28.13:9092', -- 替换为您的 Kafka 连接地址
'properties.group.id' = 'testGroup', -- 必选参数, 一定要指定 Group ID

-- 定义数据格式 (Debezium 输出的 JSON 格式)
'format' = 'debezium-json'
);
```

示例：用作数据目的（Sink）

JSON 格式输出

```
CREATE TABLE `Data-Output` (
`time` VARCHAR,
```

```
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 定义 Kafka 参数
'connector' = 'kafka',
'topic' = 'Data-Output', -- 替换为您要写入的 Topic
'properties.bootstrap.servers' = '172.28.28.13:9092', -- 替换为您的 Kafka 连接地址

-- 定义数据格式 (JSON 格式)
'format' = 'json',
'json.fail-on-missing-field' = 'false', -- 如果设置为 false, 则遇到缺失字段不会报错。
'json.ignore-parse-errors' = 'true' -- 如果设置为 true，则忽略任何解析报错。
);
```

CSV 格式输出

```
CREATE TABLE `Data-Output` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 定义 Kafka 参数
'connector' = 'kafka',
'topic' = 'Data-Output', -- 替换为您要写入的 Topic
'properties.bootstrap.servers' = '172.28.28.13:9092', -- 替换为您的 Kafka 连接地址

-- 定义数据格式 (CSV 格式)
'format' = 'csv'
);
```

通用 WITH 参数

参数值	必填	默认值	描述
connector	是	无	建议输入 `'kafka'`，并在内置 Connector 选框中选择 `flink-connector-kafka`。如果确实有读写旧版 Kafka 的需求，可以输入 `'kafka-0.11'`，并选择 `flink-connector-kafka-0.11`。
topic	是	无	要读写的 Kafka Topic 名。
properties.bootstrap.servers	是	无	逗号分隔的 Kafka Bootstrap 地址。

参数值	必填	默认值	描述
properties.group.id	作为数据源时必选	无	Kafka 消费时的 Group ID。
format	是	无	Kafka 消息的输入输出格式。目前支持'csv'、'json'、'avro'、'debezium-json'以及'canal-json'。
scan.startup.mode	否	group-offsets	Kafka 消费时的起始坐标，目前支持'earliest-offset'、'latest-offset'、'group-offsets'、'timestamp'以及'specific-offsets'。
scan.startup.specific-offsets	否	无	如果 scan.startup.mode 的值为'specific-offsets'，则必须使用本参数指定具体起始读取的偏移量。例如'partition:0,offset:42;partition:1,offset:300'。
scan.startup.timestamp-millis	否	无	如果scan.startup.mode 的值为'timestamp'，则必须使用本参数来指定开始读取的时间点（毫秒为单位的 Unix 时间戳）。
sink.partitioner	否	无	Kafka 输出时所用的分区器。目前支持的分区器如下：fixed：一个 Flink 分区对应不多于一个 Kafka 分区。round-robin：一个Flink 分区依次被分配到不同的 Kafka 分区。也可以通过继承 FlinkKafkaPartitioner 类，实现自定义分区逻辑。

JSON 格式 WITH 参数

参数值	必填	默认值	描述
json.fail-on-missing-field	否	false	如果为 true，则遇到缺失字段时，会让作业失败。如果为 false（默认值），则只会把缺失字段设置为 null 并继续处理。
json.ignore-parse-errors	否	false	如果为 true，则遇到解析异常时，会把这个字段设置为 null 并继续处理。如果为 false，则会让作业失败。
json.timestamp-format.standard	否	SQL	指定 JSON 时间戳字段的格式，默认是 SQL（格式是yyyy-MM-dd HH:mm:ss.s{可选精度}）。也可以选择 ISO-8601，格式是 yyyy-MM-ddTHH:mm:ss.s{可选精度}。

CSV 格式 WITH 参数

参数值	必填	默认值	描述
csv.field-delimiter	否	,	指定 CSV 字段分隔符，默认是半角逗号。

参数值	必填	默认值	描述
csv.line-delimiter	否	U&'\000A'	指定 CSV 的行分隔符，默认是换行符\n，SQL 中必须用U&'\000A'表示。如果需要使用回车符\r，SQL 中必须使用U&'\000D'表示。
csv.disable-quote-character	否	false	禁止字段包围引号。如果为 true，则 'csv.quote-character' 选项不可用。
csv.quote-character	否	"	字段包围引号，引号内部的作为整体看待。默认是''。
csv.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。
csv.allow-comments	否	false	忽略 # 开头的注释行，并输出为空行（请务必将 csv.ignore-parse-errors 设为 true）。
csv.array-element-delimiter	否	;	数组元素的分隔符，默认是;。
csv.escape-character	否	无	指定转义符，默认禁用转义。
csv.null-literal	否	无	将指定的字符串看作 null 值。

Debezium 格式 WITH 参数

参数值	必填	默认值	描述
debezium-json.schema-include	否	false	设置 Debezium Kafka Connect 时，如果指定了'value.converter.schemas.enable'参数，那么 Debezium 发来的 JSON 数据里会包含 Schema 信息，该选项需要设置为 true。
debezium-json.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。
debezium-json.timestamp-format.standard	否	SQL	指定 JSON 时间戳字段的格式，默认是 SQL（格式是 yyyy-MM-dd HH:mm:ss.s{可选精度}）。也可以选择 ISO-8601，格式是yyyy-MM-ddTHH:mm:ss.s{可选精度}。

版权所有：腾讯云计算（北京）有限责任公司

JDBC

最近更新时间：2020-09-16 17:03:58

介绍

JDBC Connector 提供了对 MySQL、PostgreSQL、Oracle 等常见 JDBC 数据库的读写支持。目前 Oceanus 提供的 `flink-connector-jdbc` Connector 组件已经内置了 MySQL 和 PostgreSQL 的客户端。

如果您希望连接其他的数据库，请通过附加【自定义程序包】的方式，上传相应的 JDBC Driver 的 JAR 包。

使用范围

JDBC 支持用作数据源表（Source，仅限于普通和维表 JOIN 的右表），也可以作为 Tuple 数据流的目的表（Sink），还可以作为 Upsert 数据流的目的表（Sink，需要包含主键）。

如果希望将 JDBC 数据库的变动记录，将其作为流式源表消费，可以使用 Debezium、Canal 等，对 JDBC 数据库的变更进行捕获和订阅，然后 Flink 即可对这些变更事件进行进一步的处理。可参见 Kafka。

示例：用作数据源（Source）

```
CREATE TABLE `Data-Input` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 指定数据库连接参数
'connector' = 'jdbc',
'url' = 'jdbc:mysql://10.1.28.93:3306/CDB', -- 请替换为您的实际 MySQL 连接参数
'table-name' = 'my-table', -- 需要写入的数据表
'username' = 'admin', -- 数据库访问的用户名（需要提供 INSERT 权限）
'password' = 'MyPa$$w0rd', -- 数据库访问的密码
'lookup.cache.max-rows' = '100', -- 读缓存大小
'lookup.cache.ttl' = '5000' -- 读缓存的 TTL
);
```

示例：用作数据目的（Tuple Sink）

```
CREATE TABLE `Data-Output` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 指定数据库连接参数
```

```
'connector' = 'jdbc',
'url' = 'jdbc:mysql://10.1.28.93:3306/CDB', -- 请替换为您的实际 MySQL 连接参数
'table-name' = 'my-table', -- 需要写入的数据表
'username' = 'admin', -- 数据库访问的用户名（需要提供 INSERT 权限）
'password' = 'MyPa$$w0rd', -- 数据库访问的密码
'sink.buffer-flush.max-rows' = '200', -- 批量输出的条数
'sink.buffer-flush.interval' = '2s' -- 批量输出的间隔
);
```

示例：用作数据目的（Upsert Sink）

```
CREATE TABLE `Data-Output` (
`id` BIGINT PRIMARY KEY NOT ENFORCED,
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
-- 指定数据库连接参数
'connector' = 'jdbc',
'url' = 'jdbc:mysql://10.1.28.93:3306/CDB', -- 请替换为您的实际 MySQL 连接参数
'table-name' = 'my-upsert-table', -- 需要写入的数据表
'username' = 'admin', -- 数据库访问的用户名（需要提供 INSERT 权限）
'password' = 'MyPa$$w0rd', -- 数据库访问的密码
'sink.buffer-flush.max-rows' = '200', -- 批量输出的条数
'sink.buffer-flush.interval' = '2s' -- 批量输出的间隔
);
```

通用 WITH 参数

参数值	必填	默认值	描述
connector	是	无	连接数据库时，需要填写 'jdbc'。
url	是	无	JDBC 数据库的连接 URL。
table-name	是	无	数据库表名。
driver	否	无	JDBC Driver 的类名。如果不输入，则自动从 url 中推断。
username	否	无	数据库用户名。'username' 和'password' 必须同时使用。
password	否	无	数据库密码。

版权所有：腾讯云计算（北京）有限责任公司

参数值	必填	默认值	描述
scan.partition.column	否	无	指定对输入分区扫描（Partitioned Scan）的列名，该列必须是数值类型、日期类型、时间戳类型等。关于分区扫描的细节，请参见本文后续部分。
scan.partition.num	否	无	分区扫描启用后，指定分区数。
scan.partition.lower-bound	否	无	分区扫描启用后，指定首个分区的最小值。
scan.partition.upper-bound	否	无	分区扫描启用后，指定最后一个分区的最大值。
scan.fetch-size	否	0	每次从数据库读取时，批量获取的行数。默认为0，表示一行一行读取，效率较低（吞吐量不高）。
lookup.cache.max-rows	否	无	查询缓存（Lookup Cache）中最多缓存的数据条数。
lookup.cache.ttl	否	无	查询缓存中每条记录最长的缓存时间。
lookup.max-retries	否	3	数据库查询失败时，最多重试的次数。
sink.buffer-flush.max-rows	否	100	批量输出时，缓存中最多缓存多少数据。如果设置为0，表示禁止输出缓存。
sink.buffer-flush.interval	否	1s	批量输出时，每批次最大的间隔（毫秒）。**如果 `'sink.buffer-flush.max-rows'` 设为 `'0'`，而这个选项不为零，则说明启用纯异步输出功能，即数据输出到算子、从算子最终写入数据库这两部分线程完全解耦。**
sink.max-retries	否	3	数据库写入失败时，最多重试的次数。

主键说明

对于 Append（Tuple）数据，JDBC 数据库的表不需要定义主键，也不建议定义主键（否则可能因为重复数据而造成写入失败）。

对于 Upsert 数据，JDBC 数据库的表**必须**定义主键，并需要在 DDL 语句的 CREATE TABLE 中，将相应列名也加上 `PRIMARY KEY NOT ENFORCED` 约束。

> 注意：
> - 对于 MySQL 表，Upsert 功能的实现依赖于 `INSERT .. ON DUPLICATE KEY UPDATE ..` 语法，常见版本的 MySQL 均支持该语法。
> - 对于 PostgreSQL 表，Upsert 功能实现依赖于 `INSERT .. ON CONFLICT .. DO UPDATE SET ..` 语法，该语法需要 PostgreSQL 9.5 及以上版本才可支持。

分区扫描

分区扫描（Partitioned Scan）可以加速多个并行度的 Source 算子读取 JDBC 数据表，每个子任务可以读取自己的专属分区。

版权所有：腾讯云计算（北京）有限责任公司

使用该功能时，所有四个 `scan.partition` 开头的参数都必须指定，否则会报错。

> **注意：**
> 这里的 `scan.partition.upper-bound` 指定的最大值和 `scan.partition.lower-bound` 指定的最小值，指的是每个分区的步长，不会影响最终读取的数据条数和精确性。

读取缓存

Flink 提供了读取缓存（Lookup Cache）功能，可以提升维表读取的性能。目前该缓存的实现是同步的，默认未启用（每次请求都会读取数据库，吞吐量很低），必须手动设置 `lookup.cache.max-rows` 和 `lookup.cache.ttl` 两个参数来启用该功能。

> **注意：**
> 如果缓存的 TTL 太长，或者缓存的条数太多，可能会造成数据库中数据更新后，Flink 作业仍然读取的是缓存中的旧数据。因此对于数据库变动敏感的作业，请谨慎使用缓存功能。

批量写入优化

通过设置 sink.buffer-flush 开头的两个参数，可以实现批量写入数据库。建议配合相应底层数据库的参数，以达到更好的批量写入效果，否则底层仍然会一条一条写入，效率不高。

- 对于 MySQL，建议在 url 连接参数后加入 rewriteBatchedStatements=true 参数。

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

- 对于 PostgreSQL，建议在 url 连接参数后加入 reWriteBatchedInserts=true 参数。

```
jdbc:postgresql://10.1.28.93:3306/PG?reWriteBatchedInserts=true&?currentSchema=数据库的Schema
```

Elasticsearch

最近更新时间：2020-09-16 17:03:53

介绍

Elasticsearch Connector 提供了对 Elasticsearch 写入支持。目前 Oceanus 提供了 `flink-connector-elasticsearch6` 和 `flink-connector-elasticsearch7` 两个版本的内置 Connector 包，分别为 Elasticsearch 6.x 和 7.x 版本提供支持。

如果您希望连接其他版本的 Elasticsearch，请通过附加【自定义程序包】的方式，上传相应的 Elasticsearch Sink 的 JAR 包。

使用范围

Elasticsearch 只支持写入，可以作为 Tuple 数据流的目的表（Sink），也可以作为 Upsert 数据流的目的表（Sink，自动以文档 `_id` 字段生成主键，并更新之前的文档版本）。

如果希望将 JDBC 数据库的变动记录，将其作为流式源表消费，可以使用 Debezium、Canal 等，对 JDBC 据库的变更进行捕获和订阅，然后 Flink 即可对这些变更事件进行进一步的处理。可参见 Kafka。

示例：用作 Elasticsearch 6 数据目的（Sink）

当写入 Elasticsearch 6.x 版本时，请在作业内置 Connector 中选择 `flink-connector-elasticsearch6` 。

> **注意：**
> 目前该语法不支持用户名和密码鉴权。如果需要此功能，请使用 Flink 1.10 的旧语法。

```
CREATE TABLE `Data-Input` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
'connector' = 'elasticsearch-6', -- 输出到 Elasticsearch 6
'hosts' = 'http://10.28.28.94:9200', -- Elasticsearch 的连接地址
'index' = 'my-index', -- Elasticsearch 的 Index 名
'document-type' = '_doc', -- Elasticsearch 的 Document 类型
'sink.bulk-flush.max-actions' = '1', -- 每条数据都刷新
'format' = 'json' -- 输出数据格式，目前只支持 'json'
);
```

示例：用作 Elasticsearch 7 数据目的（Sink）

当写入 Elasticsearch 7.x 版本时，请在作业内置 Connector 中选择 `flink-connector-elasticsearch7` 。

> ⚠ 注意：
> 目前该语法不支持用户名和密码鉴权。如果需要此功能，请使用 Flink 1.10 旧语法。

```
CREATE TABLE `Data-Output` (
`time` VARCHAR,
`client_ip` VARCHAR,
`method` VARCHAR
) WITH (
'connector' = 'elasticsearch-7', -- 输出到 Elasticsearch 7
'hosts' = 'http://10.28.28.94:9200', -- Elasticsearch 的连接地址
'index' = 'my-index', -- Elasticsearch 的 Index 名
'sink.bulk-flush.max-actions' = '1', -- 每条数据都刷新
'format' = 'json' -- 输出数据格式，目前只支持 'json'
);
```

通用 WITH 参数

参数值	必填	默认值	描述
connector	是	无	当写入 Elasticsearch 6.x 版本时，取值`elasticsearch-6`。当写入 Elasticsearch 7.x 及以上版本时，取值`elasticsearch-7`。
hosts	是	无	Elasticsearch 的连接地址。
index	是	无	数据要写入的 Index。支持固定 Index（例如 `'myIndex'`），也支持动态 Index（例如`'index-{log_ts|yyyy-MM-dd}'`）。
document-type	6.x 版本：必填 7.x 版本：不需要	无	Elasticsearch 文档的 Type 信息。当选择 `elasticsearch-7` 时，不能填写这个字段，否则会报错。
document-id.key-delimiter	否	_	为复合键生成 _id 时的分隔符 (默认是 "_")。例如有 a、b、c 三个主键，某条数据的 a 字段为 "1"，b 字段为 "2"，c 字段为 "3"，使用默认分隔符，则最终写入 Elasticsearch 的 _id 是 "1_2_3"。

参数值	必填	默认值	描述
failure-handler	否	fail	指定请求 Elasticsearch 失败时，错误处理策略。选项为： • fail：抛出一个异常。 • ignore：忽略错误，直接继续。 • retry_rejected：重试写入该条记录。 另外也支持自定义错误处理器，这里可以填写用户自己编写的 Handler 的类全名（需要上传自定义程序包）。
sink.flush-on-checkpoint	否	true	Flink 进行快照时，是否等待现有记录完全写入 Elasticsearch 。如果设置为 false，则可能造成恢复时部分数据丢失或者重复等异常情况，但快照速度会提升。
sink.bulk-flush.max-actions	否	1000	批量写入的最大条数。设置为 0 则禁用批量功能。
sink.bulk-flush.max-size	否	2mb	批量写入缓存的最大容量，必须以 mb 为单位。设置为 0 则禁用批量功能。
sink.bulk-flush.interval	否	1s	批量写入的刷新周期。设置为0则禁用批量功能。
sink.bulk-flush.backoff.strategy	否	DISABLED	批量写入时，失败重试的策略。 • DISABLED：不重试。 • CONSTANT：等待 sink.bulk-flush.backoff.delay 选项设置的毫秒后重试。 • EXPONENTIAL：一开始等待 sink.bulk-flush.backoff.delay 选项设置的毫秒后重试，每次失败后将指数增加下次的等待时间。
sink.bulk-flush.backoff.max-retries	否	8	批量写入时，最多失败重试的次数。
sink.bulk-flush.backoff.delay	否	50ms	批量写入失败时，每次重试之间的等待间隔（对于 CONSTANT 策略而言）或间隔的初始基数（对于 EXPONENTIAL 策略而言）。
connection.max-retry-timeout	否	无	重试请求的最大超时时间。
connection.path-prefix	否	无	指定每个 REST 请求的前缀，例如 '/v1'。通常不需要设置该选项。
format	否	json	指定输出的格式，默认是内置的 json 格式，可以使用 前文（Kafka）描述过的 JSON 格式选项，例如 json.fail-on-missing-field、json.ignore-parse-errors、json.timestamp-format.standard 等。

版权所有：腾讯云计算（北京）有限责任公司

自定义 Connector

最近更新时间：2020-12-11 15:24:37

介绍

如果内置的 Connector 不能满足您的需求，那么可以考虑**自定义 Connector** 功能，即用户可以自行上传实现了相应 Source 和 Sink 接口的类实现，然后作业在运行时会动态加载并调用。

选择合适的 Connector

用户可以选择第三方提供的 Connector 实现包（例如下面介绍的 Bahir），或者自行通过编程的方式实现。

Apache Bahir 第三方包

Apache Bahir 为 Flink 提供了常见的数据源和数据目的的扩展包。

目前 Bahir 支持如下的第三方组件：

- ActiveMQ
- Akka
- Flume
- InfluxDB
- Kudu
- Redis
- Netty

自行编程实现

使用新版 Connector API

Flink 1.11 引入了 Source 和 Sink 接口的 新版 API，简化了接口定义逻辑（合并 Upsert、Retract、Append Sink 接口），数据源也支持 Upsert 流了，同时也解耦了对 DataStream 和 Planner 等包的依赖关系。

使用旧版 Connector API

新版 API 仍然有一些无法支持的特性，例如过滤条件或者分区下推等。如果发现新版 API 无法实现功能，请参考 旧版 API。

构建并上传 Connector 包

步骤一：源码构建

建议参考现有的 Connector 的项目，修改其 `pom.xml` 配置文件，引入相关的依赖包，然后通过 Maven 构建一个 JAR 包。

> **注意：**
> 尽量使用 `maven-shade-plugin` 将常见的依赖（例如 Apache Commons、Guava 等相关的包）进行 shade 化，以避免引入的库与流计算平台本身的类发生冲突。

步骤二：上传程序包

可以在流计算的 程序包管理 界面，上传 Connector 的程序包。首次上传是 V1 版本，以此类推。

步骤三：分析开发页引用程序包

在 SQL 作业的【分析开发】页，可以选择引用之前上传的程序包和版本。

> **注意：**
> 请务必确认程序包的版本是否符合预期，避免出现各种不可预知的错误。

步骤四：保存并发布

选择程序包后，可以单击【保存】，也可以选择直接发布并启动运行。

DDL 数据定义语句
CREATE TABLE

最近更新时间：2020-09-18 10:08:53

CREATE TABLE 语句用来描述数据源（Source）或者数据目的（Sink）表，并将其定义为一张表，以供后续语句引用。

表定义语法

语法结构

```
CREATE TABLE 表名
(
{ <列定义> | <计算列定义> }[ , ...n]
[ <Watermark 定义> ]
[ <表约束定义, 例如 Primary Key 等> ][ , ...n]
)
[COMMENT 表的注释]
[PARTITIONED BY (分区列名1, 分区列名2, ...)]
WITH (键1=值1, 键2=值2, ...)
[ LIKE 其他的某个表 [( <LIKE 子句选项> )] ]
```

符号含义

CREATE TABLE 语句创建的表，既可以作为数据源表，也可以作为数据目的表。但是如果没有对应的 Connector，则会在运行时报错。

```
<列定义>:
列名 列类型 [ <列的约束定义> ] [COMMENT 列的注释]

<列的约束定义>:
[CONSTRAINT 约束名] PRIMARY KEY NOT ENFORCED

<表的约束定义>:
[CONSTRAINT 约束名] PRIMARY KEY (列名1, 列名2, ...) NOT ENFORCED

<计算列定义>:
列名 AS 计算列表达式 [COMMENT 列的注释]

<Watermark 定义>:
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Watermark策略表达式

<LIKE 子句选项>:
```

```
{
{ INCLUDING | EXCLUDING } { ALL | CONSTRAINTS | PARTITIONS }
| { INCLUDING | EXCLUDING | OVERWRITING } { GENERATED | OPTIONS | WATERMARKS }
}[, ...]
```

子句功能说明

计算列

计算列是一种虚拟列，它是逻辑上的定义而非数据源中实际存在的列，通常由同一个表的其他列、常量、变量、函数等计算而来。例如，如果数据源中定义了 price（商品单价）和 quantity（采购量），那么就可以新定义一个 cost（总成本）字段，即 `cost AS price * quantity`，这样就可以在后续查询里直接使用 cost 字段。

更常见的用法是使用计算列来实现非标准时间戳的标准化。例如在一个数据源中，时间戳字段 mytime 为 Unix 格式（例如以毫秒为单位的1599469771494），则可以通过计算列的方式 `ts AS TO_TIMESTAMP(FROM_UNIXTIME(mytime / 1000, 'yyyy-MM-dd HH:mm:ss'))`，将时间戳处理为 Flink 可识别的 Timestamp(3) 类型。另外，如果数据源时间戳字段虽然是 Timestamp(3) 格式，但是嵌套在 JSON 的其他字段中，也可以用计算列的方式将其解析出来。

> 注意：
> - 计算列只允许在 SELECT 语句中使用。
> - 对于 INSERT 语句，目的表中的计算列会被自动忽略。

WATERMARK

Watermark 定义

Watermark 决定着 Flink 作业的时间模式（详见下文的 Event Time/Processing Time 介绍小节），定义方式：

```
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Watermark策略表达式
```

例如 `WATERMARK FOR my_time_field AS my_time_field - INTERVAL '3' SECOND` 表示定义一个容差为 3 秒的 Watermark 策略。

- 某个 Rowtime 类型的列名：必须是 Flink 可识别的 `Timestamp(3)` 类型，且不是嵌套列。如果类型不对或者属于嵌套字段，则需要使用上文提到的“计算列”功能，创建一个转换后的虚拟列，作为 Rowtime 类型的列。
- 某个 Watermark 策略表达式：用于定义 Watermark 的生成策略，可以用各种表达式来描述一个 `Timestamp(3)` 类型的值，以作为每次生成 Watermark 时的依据。

定义示例如下：

```
CREATE TABLE StudentRecord (
Id BIGINT,
StudentName STRING,
RegistrationTime TIMESTAMP(3),
```

```
WATERMARK FOR RegistrationTime AS RegistrationTime - INTERVAL '3' MINUTE
) WITH ( ... ... );
```

Watermark 生成策略

单调递增时间戳的 Watermark 策略

如果时间戳可以确保是单调递增的，不存在乱序的情况，则可以用如下语法，以期得到最低的数据处理延迟。

```
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Rowtime类型的列名
```

下面语句的含义是将每个输入数据中最大时间戳作为 Watermark 的取值。因此如果存在乱序，就会造成晚到的数据未达到 Watermark 的界限而被丢弃。例如：

```
WATERMARK FOR my_time AS my_time
```

有限容忍乱序的 Watermark 策略

与上述的策略不同，本策略允许数据中存在一定范围的乱序。这个乱序范围由用户自行控制，如果设置的较大，则会带来较长的延迟（数据积压、等待）；如果设置的较小，则超过阈值的数据则可能被丢弃（造成结果不准确）。

```
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Rowtime类型的列名 - INTERVAL '时间长度' 时间单位
```

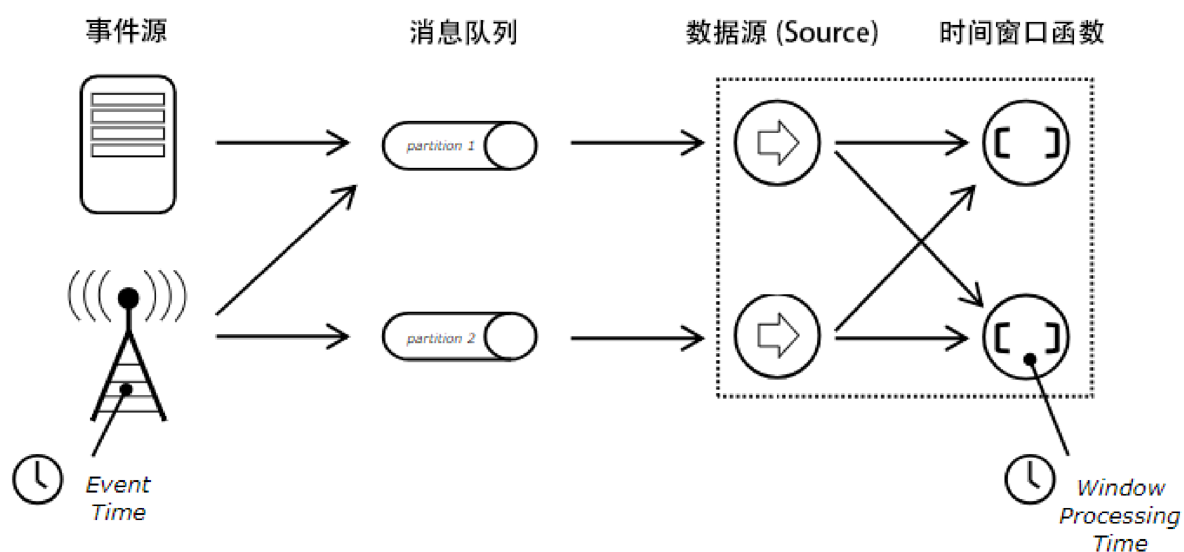
下面语句的含义是每个输入数据中最大的时间戳减去3秒的容差作为 Watermark 的取值。因此如果存在乱序，但是后来的数据比之前最大值相差不到3秒，也会被允许加入计算。

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

Event Time/Processing Time 介绍

对于基于窗口的操作（例如 GROUP BY、OVER、JOIN 条件中时间段的指定），流计算 Oceanus 支持 Event Time 和 Processing Time 两种时间处理模式。

版权所有：腾讯云计算（北京）有限责任公司



- Event Time 模式使用输入数据自带的时间戳，容忍一定程度的乱序数据输入（例如，更早的数据由于各节点处理能力、网络波动等不可预知的原因，来的却更晚），这个参数可以通过 BOUNDED 的第二个参数指定，单位是毫秒。该处理模式最精确，但要求输入数据自带时间戳。目前只支持数据源中以 timestamp 类型定义的字段，未来将会支持虚拟列，可将其他类型的列应用处理函数转换为系统接受的时间戳。
- Processing Time 处理模式不要求输入数据有时间戳，而是将该条数据被处理的时间戳自动加入数据，并以 PROCTIME（必须全为大写）字段命名。该列是隐藏的，`SELECT *`时不会出现，只有用户手动使用时才会被读取。

> **注意：**
> 对于同一个任务的所有数据源，只允许采用一种时间模式。若某个使用 Event Time 模式，则必须要求所有定义的 Table Source 都定义时间戳并声明 WATERMARK 时间戳字段。

主键 PRIMARY KEY

定义表或视图时，可以声明某些字段为主键（PRIMARY KEY），表示这些字段的值不会重复且不会为 NULL（即 SQL 的 NOT NULL + UNIQUE）。

主键的定义可以在列上，也可以单独使用 CONSTRAINT 语句定义。**不能对同一个表多次定义不同的主键**。Flink 因为无法保证数据源的每条数据主键不重复，目前只支持 PRIMARY KEY NOT ENFORCED 语法，即提醒用户需自行保证主键语义。

分区 PARTITIONED BY

如果在某个列上定义了 PARTITIONED BY 子句，则表明允许 Flink 对该列进行分区。主要影响 FileSystem Sink，它会根据分区的不同，为每个数据分区创建一个单独的目录。

流计算 Oceanus 不建议用户使用 FileSystem Sink，因为所有 TaskManager 运行结束后，文件系统的数据会被自动清理。

WITH 参数

WITH 参数通常用于指定数据源和数据目的的 Connector 所需参数，语法为 `'key1'='value1', 'key2' = 'value2'` 的键值对。

例如要写入 Kafka（腾讯云 CKafka 或自建 Kafka）时，需要指定服务器地址、消费的 Topic、消费的起始时间点等信息。

对于常见的上下游 Connector 的具体的使用方法，可参考 上下游开发指南。

LIKE 子句

LIKE 子句允许用于创建表（下文称为 B 表）时，引用其他表（下文称为 A 表）的结构，这样可以大幅节省 CREATE TABLE 语句的代码量，做到代码复用。例如，把同样的数据一份写入 Kafka Sink，另一份写入 Elasticsearch Sink，还有一部分写入 MySQL，那么就可以通过 LIKE 语句来实现定义三张表，同时复用列定义的效果。

定义一张 A 表：

```
CREATE TABLE A (
Id BIGINT,
StudentName STRING,
RegistrationTime TIMESTAMP(3)
) WITH ( ... 某些参数 ... );
```

再定义一个含 Watermark 的 B 表，则可以直接基于上面的表，创建一个新的表：

```
CREATE TABLE B (
WATERMARK FOR RegistrationTime AS RegistrationTime - INTERVAL '3' MINUTE
) WITH ( ... 另一些参数 ... ) LIKE `A`;
```

默认情况下 LIKE 语句与 WITH 参数无关，所以两个表允许使用完全不同的 WITH 参数集。如果希望继承原表的 WITH 参数信息，则需要通过 **LIKE 子句选项**来实现。

LIKE 子句选项

目前 LIKE 子句提供了如下的选项，可以控制引用（继承）某个 A 表的内容：

- CONSTRAINTS：主键（PRIMARY KEY）等约束
- GENERATED：计算列
- OPTIONS：WITH 参数
- PARTITIONS：PARTITIONED BY 定义
- WATERMARKS：WATERMARK FOR 定义
- ALL：以上所有

同时，Flink 提供了三种不同的合并策略：

- INCLUDING：继承 A 表的所有指定属性，但如果 A 表和 B 表的某些定义有冲突（例如含有相同字段定义）则报错。
- EXCLUDING：B 表中**不会**包含任何 A 表中已有的指定属性。
- OVERWRITING：继承 A 表的所有指定属性，如果 A 表和 B 表定义有冲突，则 B 表的定义会覆盖 A 表的定义。

如果未提供 LIKE 子句选项，默认行为是 `INCLUDING ALL OVERWRITING OPTIONS`，即 B 表会继承 A 表的所有定义和设置，但是会覆盖掉 WITH 参数。

版权所有：腾讯云计算（北京）有限责任公司

CREATE VIEW

最近更新时间：2020-12-23 11:54:47

用户可以使用 CREATE VIEW 语句创建视图。视图是一个虚拟表，基于某条 SELECT 语句。视图可以用在定义新的虚拟数据源（类型转换、列变换和虚拟列等），拆分过长代码等场景。

语法

```
CREATE VIEW 视图名 AS
SELECT 子句
```

示例一

创建一个名为 MyView 的视图：

```
CREATE VIEW MyView AS
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date_
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

示例二

在计算中由于数据量较大、函数方法类型匹配要求等原因，必须使用 TINYINT、SMALLINT 等类型。当 Kafka 等输入类型不符合需求时，可通过 CREATE VIEW 语句配合 CAST() 类型转换函数（参见 类型转换函数），实现定义虚拟视图作为新的数据源。

通过定义一个名为 KafkaSource2 的视图，实现将 KafkaSource1 数据源中的 BIGINT 类型的 status_code 列转为 VARCHAR 类型的列，命令如下：

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

注意：

- 不当的数据转换 CAST() 可能会导致精度损失，例如由 BIGINT 转为 INTEGER 或 TINYINT 等，请谨慎使用。
- 如果需要进行字符串（VARCHAR）和时间戳（TIMESTAMP）之间的类型转换，请参见 时间函数 中 TO_TIMESTAMP、DATE_FORMAT 等函数。

版权所有：腾讯云计算（北京）有限责任公司

CREATE FUNCTION

最近更新时间：2020-09-16 17:04:11

对于 SQL 作业，用户可以上传 自定义程序包，然后在作业分析开发页的参数设置中，引用该程序包。目前支持从本地上传，也可以引用账户下现有 COS 存储中的资源（仅限相同地域）。

这里的程序包既可以用来扩展 Connector 的功能（参见 自定义 Connector），也可以创建自定义函数（UDF）。

语法

目前流计算 Oceanus 支持 Java 和 Scala 两种语言编写的程序包。当用户上传了自定义程序包后，在界面上关联后即可用下面的 CREATE FUNCTION 语句来声明：

```
CREATE TEMPORARY SYSTEM FUNCTION 函数名
AS 函数类全名 [LANGUAGE JAVA|SCALA]
```

其中的**函数名**可以自行定义，但不要与现有的冲突。**函数类全名**为 Java 或 Scala 类的类全名（例如 `com.example.flink.MyCustomFunction`）。

命名覆盖

如果存在系统内置的同名函数时，用户使用上述语法创建的 UDF 会覆盖系统内置的函数。因此除非有意改变系统函数的功能，请不要创建与系统内置函数同名的自定义函数。

函数类型

目前 Flink 支持下面多种函数定义。

标量函数（Scalar Function）

标量函数简称 UDF，作用是将一个值转换为另一个值（一对一），例如系统内置的 `SUBSTRING`、`REPLACE` 等字符串操作函数，都属于标量函数。

表函数（Table Function）

表函数简称 UDTF，作用是将一个值转为表中的一行数据（一变多），这样可以在后续 JOIN 操作中作为右表。

聚合函数（Aggregate Function）

聚合函数简称 UDAGG，作用是将多行数据的一组值，聚合为一个最终值（多变一），例如系统内置的 `MAX`、`MIN`、`AVG` 等都属于聚合函数。

表聚合函数（Table Aggregate Function）

表聚合函数的作用是将多行数据的一组值，聚合为新的多行数据（多对多）。

异步表函数（Async Table Function）

异步表函数可作为一种特殊的数据源，例如可以通过它来对接外部的数据库、数据存储。

UDF 开发指南

由于 Flink 不同版本间 API 和文档迭代频繁，可参考 Flink 官方文档中的 UDF 开发指南。目前流计算 Oceanus 兼容开源版的 Flink 1.11 版本 API。

SET 控制语句
Flink 配置项

最近更新时间：2020-09-16 17:04:20

简介

SET 语句用来设置作业的 运行时参数。可以用 SET 语句来实现作业行为的微调，例如设置作业的重启策略、调整 SQL 的 Mini-Batch 配置、关闭异步快照、设置快照最小间隔、调整 RocksDB StateBackend 的缓存大小等。

> **注意：**
> - SET 命令属于高级用法，请谨慎使用，避免参数配置不当而引起的运行时异常。
> - 不是所有 Flink 运行时参数都支持设置。具体支持情况可以自行验证或通过工单咨询。

语法

```
SET 配置项 = '参数值';
```

其中字符串类型的参数值必须用半角单引号括起来，布尔值和数值型的可以不加引号。同时，语句行尾需加上分号。

示例

配置 At Least Once 快照策略

默认情况下，流计算 Oceanus 使用 Exactly-Once 作为默认的快照策略，该策略可以确保作业崩溃恢复后，有最精确的状态一致性，但是少数情况下可能会造成较大延迟。

如果允许作业崩溃恢复时，一部分重复数据再次参与计算（造成短期的结果不精确），可以通过调整 Flink 的快照策略为 At Least Once，这样会取得更好的快照性能，尤其是对于状态超大且多个流之间的速度不一致时效果明显。

```
SET execution.checkpointing.mode='AT_LEAST_ONCE';
```

关闭 Operator Chaining 功能

默认情况下，Flink 会将运行图中相同并行度的算子尽可能的绑在一起，避免数据上下游传输的序列化、反序列化额外开销。

如果出于定位问题的角度，希望看到每个算子的数据流入流出情况，则可以关闭这个 Operator Chaining 功能。

> **注意：**
> 关闭此功能后，作业的运行效率可能会大幅下降，请谨慎使用。

```
SET pipeline.operator-chaining=false;
```

版权所有：腾讯云计算（北京）有限责任公司

启用 Table 的 Mini-Batch 支持

Flink SQL 对聚合提供了 Mini-Batch 支持，可以显著提升吞吐量。默认没有开启，因为会增加处理时延。如果希望使用 Mini-Batch，可以通过的下面的设置项启用此功能（批次大小和延迟参数可以自行设置，但不可省略）：

```
SET table.exec.mini-batch.enabled=true;
SET table.exec.mini-batch.size=5000;
SET table.exec.mini-batch.allow-latency='200 ms';
```

DML 数据操作语句

查询语句

最近更新时间：2020-09-16 17:04:27

SELECT FROM

SELECT 不能单独使用，必须配合 `CREATE VIEW … AS` 或 INSERT INTO 使用，否则系统会提示没有合适的 Operator。

语法

```
SELECT 以逗号分隔的需要选中的字段
FROM 数据源或视图
WHERE 过滤条件
其他子查询
```

示例

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

WHERE

WHERE 用来过滤查询条件（谓词），多个并列的条件可以用 AND、OR 来连接。在与外部数据库 TencentDB 的表 JOIN 时，条件的连接只支持 AND。如需使用 OR 的功能，请参见 UNION ALL。

HAVING

HAVING 用于过滤 GROUP BY 之后的结果。**WHERE 在 GROUP BY 之前过滤，而 HAVING 在 GROUP BY 分组之后过滤。**

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

GROUP BY

在流计算 Oceanus 中，GROUP BY 用于对结果进行分组聚合。目前有含时间窗口（Window）类型的 GROUP BY 和不含窗口的 GROUP BY（即持续查询）。

- 含时间窗口（Window）类型的 GROUP BY 不会更新之前的结果，因而会产生 Append（Tuple）类型的数据流，只允许写入不带主键的 MySQL、PostgreSQL、Kafka、Elasticsearch 等数据目的。
- 不含窗口的 GROUP BY 会更新之前发出的记录，因而会产生 Upsert 类型的数据流，只允许写入含主键的云数据库 MySQL、PostgreSQL 数据目的表（Sink）、Elasticsearch 等，且主键必须与 GROUP BY 语句里 Upsert 字段一致。

含时间窗口的 GROUP BY

本示例定义了一个包含时间窗口的 GROUP BY 查询语句。关于时间窗口函数的使用方法，请参见 时间窗口函数。

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

- 在 Event Time 时间模式下，使用 `WATERMARK FOR` 定义时间戳字段，那么 TUMBLE 窗口函数的第一个参数必须为该字段。HOP 和 SESSION 窗口同理。
- 在 Processing Time 时间模式下，TUMBLE 窗口函数的第一个参数必须为表`proctime()`声明的字段。HOP 和 SESSION 窗口同理。

不含时间窗口的 GROUP BY（持续查询）

本示例定义了一个不包含时间窗口的 GROUP BY 查询语句，这种查询叫做持续查询，因为它会根据每条新到的数据来计算并决定是否更新之前发出的结果，因而会产生一个 Upsert 流。

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

> **注意：**
> 这种方式可能会因为 key 的数量过大或数据过多而发生内存溢出。因而请谨慎设置对象超时时间，不要过长。

JOIN

目前流计算 Oceanus 系统只支持等值连接 Equi-JOIN，即 JOIN 条件内包含至少一条令左右表某字段相等的过滤条件。

流和流的 Inner Equi-JOIN

目前流和流的连接也分为两种：含时间范围的和不含时间范围的。前者会生成 Append（Tuple）类型的流，而后者会生成 Upsert 类型的流。

含时间范围的 Inner JOIN

含时间范围的 JOIN 也称为 **Interval Join**，它的 WHERE 条件中需要至少一个等值连接的 JOIN 条件和一个指定的时间范围。这个时间范围可以用 <、<=、>=、> 或 BETWEEN … AND 等来表示。

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

示例：

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

不含时间范围的 Inner JOIN

不含时间范围的流-流 JOIN 的特点是只要求有至少一个等值连接，而不要求指定时间范围。也就是说，它会将历史以来所有的活跃数据参与计算（可以通过指定超时时间来去除不活跃的元素）。

> **注意：**
>
> - 可能会导致非常大的内存占用，需要谨慎使用。通常需要设置合适的对象超时时间，以及时清除失活的对象。
> - 这种查询会产生一个 Upsert 流，只能使用含主键的 MySQL、PostgreSQL、Elasticsearch 等数据目的（Sink）来接收数据。

示例：

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

Outer Equi-JOIN

Outer Equi-JOIN 会产生 Upsert 流，因此需要注意数据目的（Sink）必须可以接受 Upsert 数据流，例如 MySQL、PostgreSQL、Elasticsearch 等。

> **注意：**
>
> 由于不对 JOIN 的顺序做优化，JOIN 操作会依次按照 FROM 语句后定义的各表进行，因此可能会产生非常大的状态压力，甚至可能执行失败。

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

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

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

维表 JOIN

流计算 Oceanus 也支持流与 MySQL、PostgreSQL 数据库中维表（Temporal table，即随时间不断变化的表）的 JOIN，语法同上面介绍的完全一致，只是要求维表必须放在 JOIN 条件的右表。

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

> **注意：**
> 一定要加入 `FOR SYSTEM_TIME AS OF` 语句，否则虽然仍然可以执行 JOIN，但是只会全量读取一次数据库，结果可能不符合预期。

与 UDTF 的 JOIN

如果用户自定义了表函数（UDTF，即 User-Defined Table Function），则可以将表函数作为 JOIN 的右表，语法与普通 JOIN 类似，只是需要加上 `LATERAL TABLE( )` 关键字，将 UDTF 包围起来。

Inner UDTF JOIN

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

Left Outer UDTF JOIN

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

> **注意：**
> 目前 Left Outer UDTF JOIN 只支持 `ON TRUE` 语法，类似于 CROSS JOIN。

与数组进行 JOIN

流计算 Oceanus 系统支持和一个已定义的数组对象（可通过 值构造函数 构造数组对象 ARRAY）做 JOIN 操作。

示例： 假设 tags 是一个已定义的数组。

版权所有：腾讯云计算（北京）有限责任公司

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

UNION ALL

UNION ALL 用来合并两个查询的结果。

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

目前流计算 Oceanus 只支持 UNION ALL 而暂不支持 UNION，即不会对相同的行进行去重操作。

如果需要实现去重以达到 UNION 的效果，请配合 DISTINCT 使用。**DISTINCT 会让结果从 Append（Tuple）流变为 Upsert 流，因而只能使用含主键的 MySQL、PostgreSQL、Elasticsearch 数据目的（Sink）来接收数据。**

OVER Window 聚合

如果需要对数据流做基于滑动窗口的聚合（不使用 GROUP BY 的聚合），那么可以使用 OVER 来进行滑动窗口的聚合操作。在 OVER 中可以指定 PARTITION、ORDER、窗口范围等。

下面的示例定义了一个滑动窗口聚合查询，统计一个大小为3的滑动窗口的交易总额（amount）。其中对之前的行使用 PRECEDING，而目前还不支持 FOLLOWING。

另外 ORDER BY 后面只允许一个时间戳字段，本例使用数据源中声明的 `proctime` 字段。

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

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

ORDER BY

ORDER BY 用来对查询的结果做排序，默认是 ASC（升序排列），也可以显式指定 DESC（降序排列）。

> 注意：
> 要求第一个排序项必须是升序的时间列（Event Time 时间戳或 Processing Time 时间戳，即 PROCTIME），之后的排序项可以自由指定。

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

DISTINCT

DISTINCT 用来对查询结果进行去重，它必须放在 SELECT 后面。

```
SELECT DISTINCT users FROM Orders
```

- DISTINCT 会产生一个 Upsert 流，因而只有 Upsert 类型的数据目的（Sink）才可以接收其结果。而且长时间查询可能会导致内存占用过大，请谨慎使用。
- 通过设置合适的对象过期时间，可以及时清除失活对象来节省内存。

IN

可以使用 IN 关键字对判断指定集合（例如子查询）中是否存在某元素。

> 注意：
> 该操作的内存压力可能较大，请谨慎使用。

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

EXISTS

如果 EXISTS 后面子查询的结果大于或等于一行（存在数据），则返回 `true`。

> 注意：
> 该操作的内存压力可能较大，请谨慎使用。

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

ORDER BY

ORDER BY 可以对某个字段进行排序后输出。

> **注意：**
> 该操作的内存压力可能较大，请谨慎使用。

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

Grouping Sets、Rollup、Cube

对于 Grouping Sets、Rollup、Cube 操作，产生的是一个 Upsert 流，因此只有 Upsert 类型的数据目的才可以接收其结果。

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

模式匹配

目前流计算 Oceanus 支持 `MATCH_RECOGNIZE` 语句对一条输入流进行模式匹配，用户可以用 SQL 语句来描述 CEP（复杂事件处理）的逻辑。

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

```
C.id AS cid
PATTERN (A B C)
DEFINE
A AS name = 'a',
B AS name = 'b',
C AS name = 'c'
) AS T
```

上述例子定义了 A、B、C 三个事件，分别表示 name 字段等于 a、b、c 时的事件。PATTERN 表示事件触发的规则，即 A、B、C 三个事件连续出现时触发。MEASURES 指定了事件触发后的输出格式。

更多内容，请参见 Flink 官方文档的 表中的模式检测。

Top-N

Top-N 查询可以在一批流数据中，不断输出当前最新的前 N 大或者前 N 小的记录，因此输出类型为 Upsert 流，需要写入支持 Upsert 数据流的数据目的（Sink）。

Top-N 语法详情请参见 Flink 官方文档的 Top-N 查询。

相邻数据去重

有时候上游输入的数据可能会包含连续的重复值，该查询语句可以将重复的数据删除，只保留一个。去重的语法详情请参见 Flink 官方文档的 去重。

INSERT 语句

最近更新时间：2020-09-16 17:04:31

INSERT INTO

INSERT INTO 语句必须和 SELECT 子查询联用，SELECT 的数据会写入到指定的数据目的表（Table Sink）中。

语法

```
INSERT INTO 数据目的表
SELECT 子句
```

示例

将 SELECT 查询的结果插入名为 KafkaSink1 的数据目的表（Sink）。

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

Table Sink 注意事项

选择合适的 Connector 程序包

- 如果在 WITH 参数里指定了某个 Sink，那么请务必勾选相应的【内置 Connector】，或自行上传相应的 Connector 程序包。
- 如果缺少符合条件的 Connector 程序包，作业启动时会抛出`org.apache.flink.table.api.ValidationException: Could not find any factory`异常信息。
- 对于读写 Kafka 的场景，推荐使用不带版本号的 `flink-connector-kafka`程序包，并将 `connector.version`参数设置为 `universal`，以获得最新的功能适配。**不建议**选择 `flink-connector-kafka-0.11` 等带版本号的旧版程序包。

排除计算列

对于 INSERT INTO 的数据目的表，**计算列**是不考虑在内的。例如，某个 Sink 的定义如下，那么在 INSERT INTO MySink 后的 SELECT 语句，必须包含 a (VARCHAR) 和 b (BIGINT) 两个字段，且不允许加入 c 字段，因为 c 是虚拟的计算列。

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

Tuple 和 Upsert 数据流的区别

确保 Sink Table 定义了合适的 WITH 参数。例如有些 Connector 只支持作为数据源，不支持作为数据目的；还有的只支持 Tuple 类型的数据流，不支持 Upsert 数据流等。

版权所有：腾讯云计算（北京）有限责任公司

运算符和内置函数
概览

最近更新时间：2020-12-03 15:24:39

本节介绍流计算 Oceanus 系统内置的运算符和函数。

表格中函数名的符号含义如下：

- { }表示可选且任选一项，例如 {BOTH|LEADING|TRAILING} 表示可以不填（使用默认行为），也可以使用 BOTH、LEADING 或 TRAILING 的任意一项。不同项的功能不同。
- [ ]表示可选项，而*表示重复0或多次，例如 [, value]* 表示后续有0个或多个 value，这种用法常用来表示不定数目的参数，例如 value, value, value … value。
  - 小写字母构成的字符串表示某个变量，例如 value1、value、boolean、numeric 等。
  - 大写字母（例如 IS NULL）或符号（例如 =、<）表示运算符或内置函数。

内置函数分类：

- 比较函数
- 逻辑函数
- 算术函数
- 字符串操作函数
- 条件函数
- 类型转换函数
- 时间相关函数
- 聚合函数
- 时间窗口函数
- 其他函数

比较函数

最近更新时间：2020-09-16 17:05:36

比较函数的函数名和功能描述如下：

函数名	功能描述
value1 = value2	• 比较 value1 和 value2 是否相等，如果相等则返回 TRUE，如果不相等则返回 FALSE。 • NULL 与任何值比较的结果均为 NULL，在 WHERE 条件中会被当作 FALSE。因此请使用 IS NULL 而不是 = NULL 来与 NULL 进行比较。 • = 和 IS NOT DISTINCT FROM 的区别主要在于对 NULL 值的处理的方式不同。
value1 <> value2	比较 value1 和 value2 是否不相等。如果不等则返回 TRUE，否则返回 FALSE。
value1 > value2	比较 value1 是否大于 value2。如果大于返回 TRUE，否则返回 FALSE。
value1 >= value2	比较 value1 是否大于等于 value2。如果大于等于则返回 TRUE，否则返回 FALSE。
value1 < value2	比较 value1 是否小于 value2。如果小于则返回 TRUE，否则返回 FALSE。
value1 <= value2	比较 value1 是否小于等于 value2。如果小于等于则返回 TRUE，否则返回 FALSE。
value IS NULL	如果 value 为 NULL 则返回 TRUE，否则返回 FALSE。
value IS NOT NULL	如果 value 不为 NULL 则返回 TRUE，否则返回 FALSE。
value1 IS DISTINCT FROM value2	如果两个值不等（所有 NULL 值视为彼此相等），则返回 TRUE，否则返回 FALSE。
value1 IS NOT DISTINCT FROM value2	如果两个值相等（所有 NULL 值视为彼此相等），则返回 TRUE，否则返回 FALSE。
value1 BETWEEN [ASYMMETRIC | SYMMETRIC] value2 AND value3	• 默认行为是 ASYMMETRIC，表示若 value1 大于等于 value2 且小于等于 value3，则返回 TRUE，否则返回 FALSE。 • 如果显式指明 SYMMETRIC，则 value2 和 value3 的顺序可以相互交换而不影响结果。
value1 NOT BETWEEN value2 AND value3	如果 value1 小于 value2 或者大于 value3，则返回 TRUE，否则返回 FALSE。
string1 LIKE string2	如果 string1 符合 string2 表示的 pattern 则返回 TRUE，否则返回 FALSE。
string1 NOT LIKE string2	如果 string1 不符合 string2 表示的 pattern 则返回 TRUE，否则返回 FALSE。

函数名	功能描述
string1 SIMILAR TO string2	如果 string1 符合 string2 表示的正则表达式，则返回 TRUE，否则返回 FALSE。
string1 NOT SIMILAR TO string2	如果 string1 不符合 string2 表示的正则表达式，则返回 TRUE，否则返回 FALSE。
value IN (listItem [, listItem]*)	• 如果 value 处于 IN 后面的值列表中，则返回 TRUE。该语句等价于多个 OR 表达式的连接。 • 如果值列表中包含 NULL，则找不到时会返回 NULL，否则找不到时返回 FALSE。 • 如果 value 是 NULL，则结果永远为 NULL。
value NOT IN (listItem, [, listItem]*)	如果 value 不在 IN 后的值列表中，则返回 TRUE，否则返回 FALSE。
EXISTS (某个子查询)	• 如果子查询返回至少一行，则返回 TRUE，否则返回 FALSE。 • **该查询对内存压力较大，请谨慎使用。**
value IN (某个子查询)	• 如果子查询返回的多行结果中，有一条等于 value 的值，则返回 TRUE，否则返回 FALSE。 • **该查询对内存压力较大，请谨慎使用。**
value NOT IN (某个子查询)	• 如果子查询返回的多个结果中，没有一条等于 value，则返回 TRUE，否则返回 FALSE。 • **该查询对内存压力较大，请谨慎使用。**

逻辑和位运算函数

最近更新时间：2020-09-16 17:05:41

逻辑函数

逻辑函数用来执行逻辑运算，结果是布尔值（Boolean）。

逻辑状态有 TRUE、FALSE、UNKNOWN 三种（NULL 值的逻辑状态是 UNKNOWN），因而 NOT TRUE 不一定是 FALSE，还可能是 UNKNOWN。

函数名	功能描述
boolean1 OR boolean2	如果 boolean1 或者 boolean2 任意一个为 TRUE，则返回 TRUE，否则返回 FALSE。
boolean1 AND boolean2	当且仅当 boolean1 和 boolean2 均为 TRUE 时才返回 TRUE，否则返回 FALSE。
NOT boolean	如果 boolean 为 TRUE 则返回 FALSE；如果为 FALSE 则返回 TRUE；如果为 UNKNOWN 则返回 UNKNOWN。
boolean IS FALSE	如果 boolean 为 FALSE 则返回 TRUE；如果 boolean 为 UNKNOWN 则返回 FALSE
boolean IS NOT FALSE	如果 boolean 不为 FALSE 则返回 TRUE；如果为 UNKNOWN 则返回 TRUE。
boolean IS UNKNOWN	如果 boolean 为 UNKNOWN 则返回 TRUE，否则返回 FALSE。
boolean IS NOT UNKNOWN	如果 boolean 不为 UNKNOWN 则返回 TRUE，否则返回 FALSE。

算术函数

最近更新时间：2020-09-16 17:05:46

算术函数的函数名和功能描述如下：

函数名	功能描述
+numeric	返回 numeric 本身。
-numeric	返回 0-numeric 的值，即反转符号。
numeric1 + numeric2	计算 numeric1 加 numeric2 的结果。
numeric1 - numeric2	计算 numeric1 减 numeric2 的结果。
numeric1 * numeric2	计算 numeric1 乘 numeric2 的结果。
numeric1 / numeric2	计算 numeric1 除以 numeric2 的结果。
POWER(numeric1, numeric2)	计算 numeric1 的 numeric2 次方。
ABS(numeric)	返回 numeric 的绝对值。
MOD(numeric1, numeric2)	返回 numeric1 除以 numeric2 的余数。如果 numeric1 是负数，那么余数也为负数。
SQRT(numeric)	计算 numeric 的平方根。
LN(numeric)	计算 numeric 的自然对数（以 *e* 为底）。
LOG10(numeric)	计算 numeric 以10为底的对数。
LOG2(numeric)	计算 numeric 以2为底的对数。
LOG(numeric2) LOG(numeric1, numeric2)	如果提供一个参数，计算 numeric2 的自然对数（等价于 LN）。 如果提供两个参数，计算 numeric2 以 numeric1 为底的对数。
EXP(numeric)	计算 *e* 的 numeric 次方。
CEIL(numeric) CEILING(numeric)	返回 numeric 向上取整的值。
FLOOR(numeric)	返回 numeric 向下取整的值。
TRUNCATE(numeric1, numeric2)	对 numeric1 的小数部分以截断的方式取整，取整的位数由 numeric2 决定。 • 如果参数为 NULL，则结果也是 NULL。 • 如果 numeric2 为0或不填，则结果没有小数部分。numeric2 可为负数，此时对整数部分取整。 例如 TRUNCATE(42.345, 2) 返回42.34；TRUNCATE(42.345) 返回42.0；TRUNCATE(42.345, -1) 返回40.0。
SIN(numeric)	计算 numeric 的正弦值。

函数名	功能描述
SINH(numeric)	计算 numeric 的双曲正弦值（返回值为 DOUBLE 类型）。
COS(numeric)	计算 numeric 的余弦值。
COSH(numeric)	计算 numeric 的双曲余弦值（返回值为 DOUBLE 类型）。
TAN(numeric)	计算 numeric 的正切值。
TANH(numeric)	计算 numeric 的双曲正切值（返回值为 DOUBLE 类型）。
COT(numeric)	计算 numeric 的余切值。
ASIN(numeric)	计算 numeric 的反正弦值。
ACOS(numeric)	计算 numeric 的反余弦值。
ATAN(numeric)	计算 numeric 的反正切值。
ATAN2(numeric1, numeric2)	计算 (numeric1、numeric2) 坐标点的四象限反正切值。
DEGREES(numeric)	将 numeric 从弧度转为角度。
RADIANS(numeric)	将 numeric 从角度转为弧度。
SIGN(numeric)	得到 numeric 的符号，负数是-1，0返回0，正数是1。
ROUND(numeric, int)	对 numeric 取整，位数由 int 值给定，可正也可负。
PI()	返回一个可以代表 π 的值。
E()	返回一个可以代表自然对数的底数 e 的值。
RAND() RAND(种子值)	返回一个0.0 – 1.0（不包含）的伪随机数，可以指定一个整数作为种子值。
RAND_INTEGER(上限值) RAND_INTEGER(种子值, 上限值)	返回一个0.0 – 指定上限值（不包含）的伪随机数。
UUID()	返回一个随机生成的 Type-4 UUID 字符串。
LOG(numeric) LOG(base, numeric)	返回 numeric 的自然对数，或者返回以指定 base 为底的对数。
BIN(numeric)	获取 numeric 的二进制表示的字符串，例如输入4，则返回"100"。
HEX(numeric) HEX(string)	获取 numeric 或 string 的十六进制表示的字符串，例如输入15或"15" 则返回"F"。

版权所有：腾讯云计算（北京）有限责任公司

字符串操作函数

最近更新时间：2020-09-16 17:05:51

字符串操作函数的函数名和功能如下：

函数名	功能描述
string1 || string2	连接两个字符串，返回两个字符串拼接后的结果，等同于 CONCAT(string1, string2)。
CHAR_LENGTH(string)	返回字符串的长度。
CHARACTER_LENGTH(string)	与 CHAR_LENGTH(string) 相同。
UPPER(string)	返回 string 的全大写字母形式。
LOWER(string)	返回 string 的全小写字母形式。
POSITION(string1 IN string2)	获取 string1 在 string2 中第一次出现的位置（**位置从1开始计数**）。当 string1 在 string2 中找不到时，返回0。
TRIM({ BOTH | LEADING | TRAILING } string1 FROM string2)	去掉 string2 中以 string1 两端/开头/结尾的 string1 部分。默认会移除结果字符串两端的空格。
LTRIM(string)	去掉 string 字符串最左边的所有空格。例如 LTRIM(' Hello') 会返回 'Hello'。
RTRIM(string)	去掉 string 字符串最右边的所有空格。例如 RTRIM(' World ') 会返回 ' World'。
REPEAT(string, integer)	将 string 字符串重复 integer 次。例如 REPEAT('Meow', 3) 会返回 'MeowMeowMeow'。
REGEXP_REPLACE(string1, string2, string3)	对 string1 字符串以 string2 表示的正则表达式进行替换，替换内容是 string3。例如 REGEXP_REPLACE('banana', 'a|n', 'A') 返回 'bAAAAA'。
REPLACE(string1, string2, string3)	将 string1 字符串中所有的 string2 替换为 string3。例如 REPLACE('banana', 'a', 'A') 返回 'bAnAnA'。
OVERLAY(string1 PLACING string2 FROM start_pos [FOR length])	将 string1 从第 start_pos 位（**start_pos 从1开始计数**）开始的子串替换为 string2。可以指定替换的长度。
SUBSTRING(string from pos) [FOR length]	获取从 pos 位开始的子串，默认行为是直到源字符串的最后，可以使用 FOR 来指定子串的长度。其中字符串**起始 pos 从1开始计数**，而不是0。
REGEXP_EXTRACT(string1, string2[, integer])	从 string1 中提取正则分组，正则表达式为 string2，第一个括号为第一组，以此类推。可通过第三个参数 integer 来指定所需的分组号（从1开始）。如果不指定分组号或者分组号为0，则表示返回整个正则表达式匹配到的字符串。例如，REGEXP_EXTRACT('foothebar', 'foo(.*?)(bar)', 2) 返回 'bar'。
INITCAP(string)	将 string 中的单词，转为以大写字母开头，其他是小写字母（首字母大写）的形式。例如 INITCAP('i have a dream') 返回 'I Have A Dream'。

版权所有：腾讯云计算（北京）有限责任公司

函数名	功能描述
CONCAT(string1, string2 …)	连接多个字符串。若任意字符串为 NULL，则结果为 NULL。
CONCAT_WS(separator, string1, string2, …)	使用指定的分隔符 separator 连接多个字符串。如果 separator 为 NULL，则结果为 NULL。如果某个字符串为 NULL，则跳过它；但是不会跳过空字符串。例如 CONCAT_WS("~", "AA","BB", "", "CC") 会返回 AA~BB~~CC。
LPAD(text, length, padding)	使用 padding 指定的字符串从左侧填充 text 字符串到指定长度 length。如果 text 比 length 更长，则会截断到 length 的长度。
RPAD(text, length, padding)	使用 padding 指定的字符串从右侧填充 text 字符串到指定长度 length。如果 text 比 length 更长，则会截断到 length 的长度。
FROM_BASE64(string)	将 Base64 编码的 string 字符串解码为字符串。如果 string 为 NULL，则返回 NULL。
TO_BASE64(string)	将 string 表示的字符串编码为 Base64 字符串。
ASCII(string)	返回 string 字符串中第一个字符的 ASCII 码。如果 string 为 NULL，则返回 NULL。例如 ASCII('an apple') 返回97，因为首字母 'a' 的 ASCII 编码是97。
CHR(integer)	返回编码为 integer 的 ASCII 字符。例如 CHR(97) 返回 'a'。
ENCODE(string, charset)	将 string 字符串转码为 charset 指定的字符集编码的 BINARY 类型，例如 ENCODE(hello, 'GBK')。
DECODE(binary, charset)	将 binary 表示的 BINARY 类型以 charset 指定的字符集解码，例如 DECODE(binary_field, 'UTF-16LE')。
INSTR(string1, string2)	返回 string2 在 string1 字符串中首次出现的位置。如果任意参数为 NULL，结果为 NULL。
LEFT(string, n)	返回 string 从左起前 n 个字符。如果 n 为负数，则返回空字符串。如果任意参数为 NULL，结果为 NULL。
RIGHT(string, n)	返回 string 从右起后 n 个字符。如果 n 为负数，则返回空字符串。如果任意参数为 NULL，结果为 NULL。
LOCATE(string1, string2[, integer])	返回跳过 integer 个字符后，string1 在 string2 中首次出现的位置（**参数顺序与 INSTR 函数相反**）。如果未找到，则返回0。如果任意参数为 NULL，结果为 NULL。
PARSE_URL(string1, string2[, string3])	获取 URL 中的指定元素。string2 可选的值为 'HOST'、'PATH'、'QUERY'、'REF'、'PROTOCOL'、'AUTHORITY'、'FILE' 和 'USERINFO'。例如 `PARSE_URL('http://example.com/custom-path/file.php?q1=v1&q2=v2#custom-ref', 'HOST')`，返回网址中的主机名 'example.com'。例如 `PARSE_URL('http://example.com/custom-path/file.php?q1=v1&q2=v2#custom-ref', 'QUERY', 'q1')`，返回请求参数的值 'v1'。
REGEXP(string, regex)	如果 regex 表示的正则表达式可以匹配 string 中的字符串的任意子串，那么返回 TRUE，否则返回 FALSE。如果任意参数为 NULL，结果为 NULL。

版权所有：腾讯云计算（北京）有限责任公司

函数名	功能描述
REVERSE(string)	反转 string 字符串。如果任意参数为 NULL，结果为 NULL。
SPLIT_INDEX(string, separator, index)	将 string 表示的字符串以 separator 指定的分隔符拆分，并获取第 index 项，返回值为字符串 VARCHAR 类型。其中 **index 从0开始计数**。
SPLIT(string, separator)	将 string 表示的字符串以 separator 指定的分隔符拆分，并返回一个 Row 类型的对象。
STR_TO_MAP(string1[, string2, string3])	将 string1 字符串用 string2 提供的数据分隔符（默认为半角逗号，）和 string3 提供的键值间分隔符（默认为半角等号 =）进行拆分，结果为键值对 `MAP<STRING, STRING>` 类型。例如 STR_TO_MAP('k1=v1,k2=v2,k3=v3') 返回键值对（非字符串）{"k1": "v1", "k2": "v2", "k3": "v3"}。
SUBSTR(string[, pos[, length]])	返回 string 字符串从 pos 位置开始，长度为 length 的子串。如果不提供 length，则默认到该字符串尾部。
EXPLODE(inputStr, separator)	将某个字符串分割为一张有多行的临时表。这个函数属于 Table Function，需要使用 LATERAL TABLE ( ) 关键字来引用此动态生成的临时表并作为 JOIN 条件的右表。
GET_ROW_ARITY(row)	获取某个 Row 类型对象 row 的列数。
GET_ROW_FIELD_STR(row, index)	获取某个 Row 类型对象 row 的第 index 列的值，**index 从0开始计数**。返回值为字符串 VARCHAR 类型。
GET_JSON_OBJECT(json_str, path_str)	按 path_str 指定的 JSONPath 路径，获取某个 JSON 字符串 json_str 中的元素，可以任意嵌套。支持的 JSONPath 语法：`$`表示根对象，`.`表示子元素，`[]`表示数组索引，*为数组索引 [] 的通配符。

条件函数

最近更新时间：2020-09-16 17:05:55

条件函数的函数名和功能描述如下：

函数名	功能描述
CASE value WHEN value1 [, value11]* THEN result1 [WHEN valueN [, valueN1]* THEN resultN]* [ELSE resultZ] END	• 当满足 value1 ~ value11 的任意值时，返回 result1。 • 当满足 valueN ~ valueN1 的任意值时，返回 resultN。 • 否则返回 resultZ。
CASE WHEN condition1 THEN result1 [WHEN conditionN THEN resultN] * [ELSE resultZ] END	• 当满足 condition1 时返回 result1。 • 当满足 condition 时返回 resultN。 • 否则返回 resultZ。
NULLIF(value1, value2)	如果 value1 与 value2 相同则返回 NULL；否则返回第一个值。例如 NULLIF(5, 5) 返回 NULL，而 NULLIF(5, 0) 返回5。
COALESCE(value, value [, value]*)	如果前值是 NULL 则提供一个后续的值，例如 COLAESCE(NULL, 5) 则返回5。
IF(condition, true_value, false_value)	如果 condition 的条件满足，返回 true_value，否则返回 false_value。例如，IF(2 > 1, 2, 1) 返回2，而 IF (1 > 2, 99, 100) 返回100。
IS_ALPHA(string)	判断字符串是不是仅由纯字母组成。如果是则返回 true，否则返回 false。
IS_DECIMAL(string)	判断字符串是不是一个合法的数字（整数、小数、负数均可）。如果是则返回 true，否则返回 false。
IS_DIGIT(string)	判断字符串是不是仅由纯数字组成（即无符号整数）。如果是则返回 true，否则返回 false。
IF_NULL_STR(str, defaultValue)	如果 str 不为 NULL，则返回 str 本身；如果 str 为 NULL，则返回第二项参数 defaultValue。

类型转换函数

最近更新时间：2020-09-17 16:27:17

类型转换函数的函数名和功能描述如下：

> **注意：**
> 如果使用 CAST() 函数，将时间段 INTERVAL 转为数字，则结果会是字面值（可能不符合预期）。例如 `CAST(INTERVAL '1234' MINUTE AS BIGINT)`，则结果会是字面值1234，而非时间段表示的毫秒值。

函数名	功能描述
CAST(value AS type)	将某个值转为 type 类型，例如 CAST(\`hello\` AS VARCHAR) 会将 \`hello\` 字段转为 VARCHAR 类型。
CAN_CAST_TO(str, type)	判断 str 字符串是否可以被转换为 type 指定的类型，返回值为布尔型。返回值可以在 CASE 语句中作为条件使用。例如 CAN_CAST_TO('123456', 'INTEGER') 则返回 True，而 CAN_CAST_TO('a145', 'DOUBLE') 则返回 False。
TO_TIMESTAMP(string, simple_format)	以 Java 的 SimpleDateFormat 支持的时间格式化模板 simple_format，将 string 字符串格式化为 Timestamp 类型的时间戳。默认以东八区为准。 例如 TO_TIMESTAMP(ts, 'yyyy-MM-dd HH:mm:ss')。
DATE_FORMAT_SIMPLE(timestamp, simple_format)	将 BIGINT（Long）类型的字段以 Java 的 SimpleDateFormat 支持的时间格式化模板转为字符串形式。例如 DATE_FORMAT_SIMPLE(ts, 'yyyy-MM-dd HH:mm:ss')。
DATE_FORMAT(timestamp, format)	（**不建议使用**） 使用与 MySQL 兼容的 format 格式化 Timestamp 类型的时间戳为字符串。与 DATE_FORMAT_SIMPLE 的区别在于： • 采用 MySQL 的格式化语法而非 Java SimpleDateFormat 的格式化语法。使用方式与 MySQL 的 `date_parse()` 函数相同。 • timestamp 参数为 Timestamp 类型，而非 BIGINT（Long）类型。 • 该函数有 bug，不建议使用。 例如 `DATE_FORMAT(ts, '%Y, %d %M')` 返回字符串 '2017, 05 May'。
TIMESTAMP_TO_LONG(timestamp) 或 TIMESTAMP_TO_LONG(timestamp, mode)	将某个 TIMESTAMP 类型的参数转为 BIGINT (Long) 类型的值。 • 若 mode 为 'SECOND'，则转为以秒来计数的 Unix 时间戳，例如 1548403425。 • 若 mode 为其他值或者省略，则转为以毫秒计数的 Unix 时间戳，例如 1548403425512。

时间相关函数

最近更新时间：2020-09-17 16:27:21

时间相关函数的函数名和功能描述如下：

函数名	功能描述
DATE string	将 "yyyy-MM-dd" 形式表示的字符串转为 SQL 日期（DATE）类型。
TIME string	将 "HH:mm:ss" 形式表示的字符串转为 SQL 时间（TIME）类型。
TIMESTAMP string	将 "yyyy-MM-dd HH:mm:ss[.SSS]" 形式的字符串转为 SQL 时间戳（TIMESTAMP）类型。
INTERVAL string range	接受 "dd hh:mm:ss.fff" 形式的字符串（毫秒），或者 "yyyy-MM" 形式的字符串（月）。 • 对于毫秒，可以接受 DAY、MINUTE、DAY TO HOUR、DAY TO SECOND 作为 range。 • 对于月，接受 YEAR 或 YEAR TO MONTH 作为 range。例如 `INTERVAL '10 00:00:00.004' DAY TO SECOND`、`INTERVAL '10' DAY`、`INTERVAL '2-10' YEAR TO MONTH`。
CURRENT_DATE	返回当前的 SQL DATE 格式表示的日期（UTC）。
CURRENT_TIME	返回当前的 SQL TIME 格式表示的时间（UTC）。
CURRENT_TIMESTAMP	返回当前 SQL TIMESTAMP 格式表示的时间戳（UTC）。
LOCALTIME	返回本地时区表示的 SQL 时间。
LOCALTIMESTAMP	返回本地时区表示的 SQL 时间戳。
EXTRACT(timeintervalunit FROM temporal)	获取时间点或时间段字符串中的某项。例如 `EXTRACT(DAY FROM DATE '2006-06-05')` 返回5，而 `EXTRACT(YEAR FROM DATE '2018-06-12')` 则返回2018。
YEAR(date)	返回指定日期中的年份，等价于 EXTRACT(YEAR FROM date)。例如 `YEAR(DATE '2020-08-12')` 返回2020。
QUARTER(date)	返回指定日期的季度，等价于 EXTRACT(QUARTER FROM date)。例如 `QUARTER(DATE '2012-09-10')`返回3。
MONTH(date)	返回指定日期的月份，等价于 EXTRACT(MONTH FROM date)。例如 `MONTH(DATE '2012-09-10')` 返回9。
WEEK(date)	返回指定日期的周数，即当年的第几周，等价于 EXTRACT(WEEK FROM date)。例如 `WEEK(DATE '1994-09-27')` 返回39。
DAYOFYEAR(date)	返回指定日期在当年的天数，即当年的第几天（范围是[1，366]），等价于 EXTRACT(DOY FROM date)。例如 `DAYOFYEAR(DATE '1994-09-27')` 返回270。

版权所有：腾讯云计算（北京）有限责任公司

函数名	功能描述
DAYOFMONTH(date)	返回指定日期在当月的天数，即当月的第几天（范围是[1，31]），等价于 EXTRACT(DAY FROM date)。例如 DAYOFMONTH(DATE '1994-09-27') 返回27。
DAYOFWEEK(date)	返回指定日期在本周的天数，即本周的第几天（范围是[1，7]），等价于 EXTRACT(DOW FROM date)。例如 DAYOFWEEK(DATE '1994-09-27') 返回3。
HOUR(timestamp)	返回指定时间戳的小时部分（范围是[0，23]）。等价于 EXTRACT(HOUR FROM timestamp)。例如 HOUR('2017-10-02 12:25:44') 返回12。
MINUTE(timestamp)	返回指定时间戳的分钟部分（范围是[0，59]）。等价于 EXTRACT(MINUTE FROM timestamp)。例如 MINUTE('2017-10-02 12:25:44') 返回25。
SECOND(timestamp)	返回指定时间戳的秒部分（范围是[0，59]）。等价于 EXTRACT(SECOND FROM timestamp)。例如 SECOND('2017-10-02 12:25:44') 返回44。
FLOOR(timepoint TO timeintervalunit)	将一个时间点向下取整，例如 FLOOR(TIME '12:44:31' TO MINUTE) 返回12:44:00。
CEIL(timepoint TO timeintervalunit)	将一个时间点向上取整，例如 CEIL(TIME '12:44:31' TO MINUTE) 返回 12:45:00。
(timepoint, temporal) OVERLAPS (timepoint, temporal)	判断两个时间段是否重叠。例如 (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) 返回 TRUE；而 (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) 返回 FALSE。
TO_TIMESTAMP(string, simple_format)	将字符串格式的时间戳转为 Timestamp 类型。详见 类型转换函数。
DATE_FORMAT_SIMPLE (timestamp, simple_format)	将 BIGINT 类型以毫秒为单位的 Unix 时间戳格式化为字符串。详见 类型转换函数。
DATE_FORMAT(timestamp, format)	将 Timestamp 类型的时间戳格式化为字符串。详见 类型转换函数。
TIMESTAMP_TO_LONG(timestamp)	将 Timestamp 类型的时间戳转为 BIGINT 类型以毫秒或秒（可选）为单位的 Unix 时间戳。详见 类型转换函数。
TIMESTAMPADD(unit, interval, timestamp)	对指定 timestamp 增加一个时间段（允许为负数），单位区间必须为 SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER、YEAR 中的一种。例如 TIMESTAMPADD(WEEK, 1, '2013-01-02')返回'2013-01-09'。

函数名	功能描述
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	计算指定单位区间下，timepoint1 和 timepoint2 之间的时间差（允许为负数）。timepointunit 区间必须为 SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER、YEAR 中的一种。例如 `TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00')` 返回1，即相差正1天。
CONVERT_TZ(string1, string2, string3)	将 string1 表示的时间戳（必须是 'yyyy-MM-dd HH:mm:ss' 格式）从时区（允许缩写例如 "PST"，全称例如 "Asia/Shanghai"，或者任意时区例如 "GMT-8:00"）string2 转换到时区 string3。例如 CONVERT('1970-01-01 00:00:00', 'UTC', 'America/Los_Angeles') 返回 '1969-12-31 16:00:00'。
FROM_UNIXTIME(numeric[, string])	返回 numeric 代表的 Unix 时间戳（从 1970-01-01 00:00:00 UTC 到现今的秒数）转为 'YYYY-MM-DD hh:mm:ss' 格式的字符串。**默认使用 UTC+8 时区，即北京时间（Asia/Shanghai）**。
UNIX_TIMESTAMP()	返回以秒为单位的 Unix 时间戳（从 1970-01-01 00:00:00 UTC 到现今的秒数），类型为 BIGINT。**默认使用 UTC+8 时区，即北京时间（Asia/Shanghai）**。
UNIX_TIMESTAMP(string1[, string2])	将 string1 字符串以 string2 的格式（可选，默认是 'yyyy-MM-dd HH:mm:ss'）转为 Unix 时间戳，类型为 BIGINT。
TO_DATE(string1[, string2])	将 string1 字符串以 string2 的格式（可选，默认是 'yyyy-MM-dd HH:mm:ss'）转为 DATE 格式。
TO_TIMESTAMP(string1[, string2])	将 string1 字符串以 string2 的格式（可选，默认是 'yyyy-MM-dd HH:mm:ss'）转为 TIMESTAMP 格式。**默认使用 UTC+8 时区，即北京时间（Asia/Shanghai）**。
NOW()	返回当前的 SQL TIMESTAMP 时间戳，时区为 UTC。

聚合函数

最近更新时间：2020-09-16 17:06:09

聚合函数的函数名和功能描述如下：

函数名	功能描述
COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)	默认情况和 ALL 时，返回 expression 表达式筛选后，非 NULL 值的输入行数。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计总行数。
COUNT(*) COUNT(1)	返回输入的总行数，含 NULL 值。
AVG([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的算术平均值。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求平均。
SUM([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入和。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求和。
MAX([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的最大值（不可用于 TIMESTAMP 类型）。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求最大值。
MIN([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的最小值（不可用于 TIMESTAMP 类型）。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求最小值。
STDDEV_POP([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的总体标准差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求总体标准差。
STDDEV_SAMP([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的样本标准差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求样本标准差。
VAR_POP([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的总体方差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求总体方差。
VAR_SAMP([ALL | DISTINCT] expression) VARIANCE([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的样本方差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求样本方差。两种写法等价。
COLLECT([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的非 NULL 输入的 MULTISET 集合（允许重复值）。如果所有值都是 NULL，则返回一个空集。
RANK()	返回某个数据在一组数据中的排名，前后调用的结果可能不连续。例如有五个数据，其中两个并列第二，那么 RANK() 的结果是1、2、2、4、5。
DENSE_RANK()	返回某个数据在一组数据中的排名，前后调用的结果保证连续。例如有五个数据，其中两个并列第二，那么 RANK() 的结果是1、2、2、3、4。
ROW_NUMBER()	为一组数据的每行分配一个递增且连续的值，从1开始，不会重复。 例如有五个数据，其中两个并列第二，那么 RANK() 的结果是1、2、3、4、5。

函数名	功能描述
`LEAD(expression [, offset] [, default] )`	在窗口计算中，访问当前行之后 offset 行的数据，默认 offset 为1，即访问下一行的数据。default 表示无数据时的默认值，如果不提供，默认为 NULL。
`LAG(expression [, offset] [, default])`	在窗口计算中，访问当前行之前 offset 行的数据，默认 offset 为1，即访问上一行的数据。default 表示无数据时的默认值，如果不提供，默认为 NULL。
FIRST_VALUE(expression)	返回一系列数据中，第一个数据。
LAST_VALUE(expression)	返回一系列数据中，最后一个数据。
LISTAGG(expression [, separator])	将一组数据使用给定的分隔符进行连接，最终返回一个连接后的字符串。默认分隔符是半角逗号 ,。类似于其他语言的 String.join() 方法。

时间窗口函数

最近更新时间：2020-09-16 17:06:13

窗口函数是一种特殊的函数，它并不在 SELECT 的投影列表中使用，而是在 GROUP BY 子句中使用。流计算 Oceanus 支持三种类型的窗口函数：TUMBLE、HOP、SESSION。

TUMBLE WINDOW

TUMBLE 窗口是一个个相连但不重叠的固定周期的窗口。

语法

```
TUMBLE(time_attr, interval)
```

- 其中 interval 的用法可参见 时间相关函数。
- 第一个参数表示时间戳字段，表示每条记录被处理时的时间戳。

> 注意：
> - 如果在 Event Time 时间模式下（使用 WATERMARK FOR 语句定义了时间戳字段），那么 TUMBLE、HOP、SESSION 窗口函数的第一个参数必须为该字段。
> - 如果在 Processing Time 时间模式下，则 TUMBLE、HOP、SESSION 窗口函数的第一个参数必须为 proctime() 函数生成的计算列，下文用 PROCTIME 举例，请在实际作业中替换为实际的列名。

示例

```
TUMBLE(rowtime, INTERVAL '1' DAY)
```

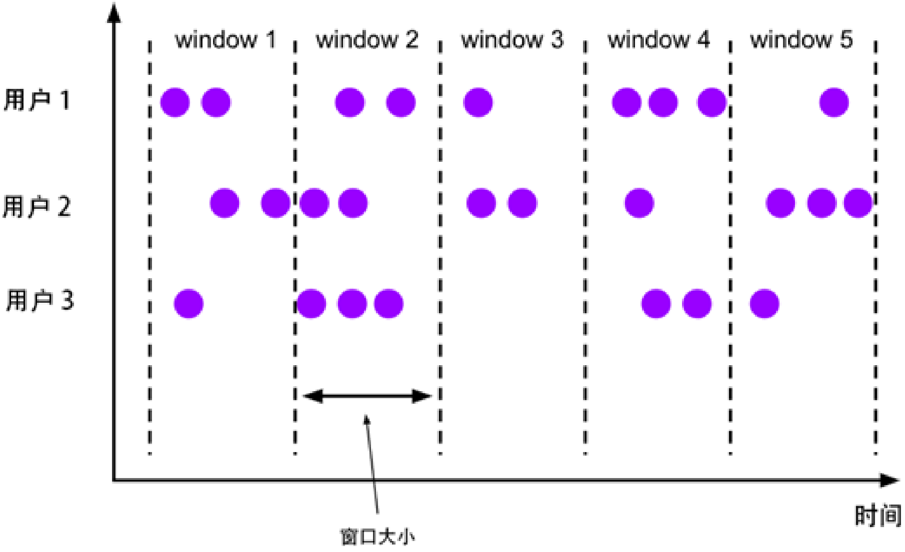
其中 rowtime 是数据源中的时间戳字段。

```
TUMBLE(PROCTIME, INTERVAL '2' HOUR)
```

其中 PROCTIME 是自动生成的记录被处理时的时间戳。

版权所有：腾讯云计算（北京）有限责任公司

TUMBLE 窗口的示意图：



HOP WINDOW

HOP WINDOW 是一种滑动窗口，它保持窗口大小不变，每次滑动指定的时间周期，因而允许窗口之间的相互重叠。

语法

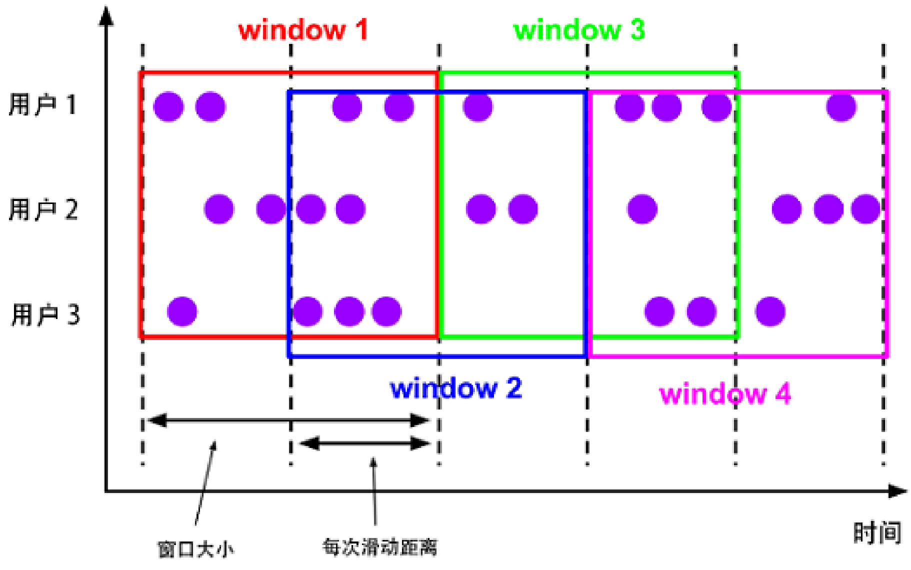
```
HOP(time_attr, sliding_interval, window_size_interval)
```

示例

```
HOP(rowtime, INTERVAL '1' HOUR, INTERVAL '1' DAY)
```

表示一个窗口大小为1天，滑动周期为1小时的 HOP 窗口。

HOP 窗口的示意图：



SESSION WINDOW

Session Window 并非以长度来划分窗口，而是以非活跃时间来划分。例如超过30分钟不活跃（没有新数据），则之前的窗口结束，下一个来到的数据将会形成一个新窗口。

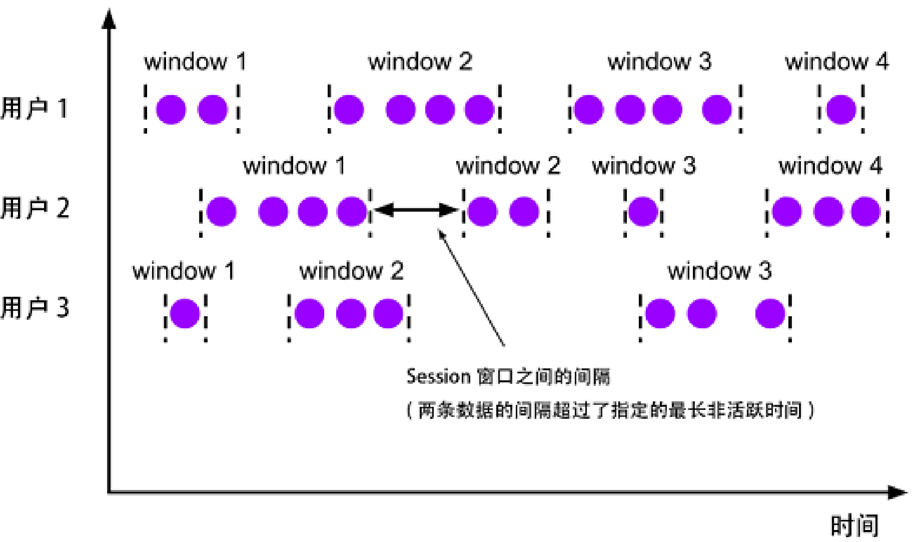
语法

```
SESSION(time_attr, interval)
```

示例

```
SESSION(rowtime, INTERVAL '12' HOUR)
```

SESSION 窗口的示意图：



另外，这三种窗口都有对应的辅助函数。以 TUMBLE 窗口为例（HOP、SESSION 也一样，只是前缀不同），有如下辅助函数：

- **TUMBLE_START**：表示 TUMBLE 窗口的起始值（包含）。示例如下：

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

- **TUMBLE_END**：表示 TUMBLE 窗口的末端界限（不包含，只能用于 SELECT 后的列，不能用作 JOIN 或 GROUP 以及 OVER 条件。若需要，请根据当前的时间模式（Event Time 或 Processing Time）分别使用下面的 TUMBLE_ROWTIME 或 TUMBLE_PROCTIME）。
  - **TUMBLE_ROWTIME**：表示 TUMBLE 窗口的末端界限（包含，可用作 JOIN 或 GROUP 以及 OVER 条件，Event Time 时间模式下使用）。示例如下：

```
SELECT user,
TUMBLE_START(rowtime, INTERVAL '12' HOUR) AS sStart,
TUMBLE_ROWTIME(rowtime, INTERVAL '12' HOUR) AS snd,
SUM(amount)
FROM Orders
GROUP BY TUMBLE(rowtime, INTERVAL '12' HOUR), user
```

- **TUMBLE_PROCTIME**：表示 TUMBLE 窗口的末端界限（包含，可用作 JOIN 或 GROUP 以及 OVER 条件，Processing Time 时间模式下使用）。示例如下：

```
SELECT user,
TUMBLE_START(PROCTIME, INTERVAL '12' HOUR) AS sStart,
TUMBLE_PROCTIME(PROCTIME, INTERVAL '12' HOUR) AS snd,
SUM(amount)
FROM Orders
GROUP BY TUMBLE(PROCTIME, INTERVAL '12' HOUR), user
```

其他函数

最近更新时间：2020-09-16 17:06:17

散列函数

散列函数的函数名和功能描述如下：

函数名	功能描述
MD5(string)	返回字符串的 MD5 值（32位十六进制数字组成的字符串）。如果输入为 NULL 则返回 NULL。
SHA1(string)	返回字符串的 SHA1 值（40位十六进制数字组成的字符串）。如果输入为 NULL 则返回 NULL。
SHA256(string)	返回字符串的 SHA256 值（64位十六进制数字组成的字符串）。如果输入为 NULL 则返回 NULL。
SHA224(string)	返回字符串的 SHA224 值（56位十六进制数字组成的字符串）。如果输入为 NULL 则返回 NULL。
SHA384(string)	返回字符串的 SHA384 值（96位十六进制数字组成的字符串）。如果输入为 NULL 则返回 NULL。
SHA512(string)	返回字符串的 SHA512 值（128位十六进制数字组成的字符串）。如果输入为 NULL 则返回 NULL。
SHA2(string, hashLength)	通用 SHA-2 系列函数，返回 string 字符串的哈希值，hashLength 为256时，等价于 SHA256(string)，其他的以此类推。

值访问函数

值访问函数的函数名和功能描述如下：

函数名	功能描述
tableName.compositeType.field	访问复合类型（Tuple、POJO）等的字段。
tableName.compositeType.*	访问 Tuple 或 POJO 的所有字段。

值构造函数

值构造函数的函数名和功能描述如下：

函数名	功能描述
(value, [, value]*) ROW(value, [, value]*)	创建一个包含若干值的行。两种写法完全等价。
ARRAY '[' value [, value]* ']'	创建一个包含若干值的数组。
MAP '[' key, value [, key, value]* ']'	创建一个包含若干键值对的映射。

集合函数

集合函数包括数组（Array）和键值对映射（Map）的操作，函数名和功能描述如下：

函数名	功能描述
CARDINALITY(array)	返回某个数组的长度。
array '[' index ']'	返回某个数组的指定位置的项（下标从1开始）。
ELEMENT(array)	返回单元素数组的内容（如果数组为空则返回 NULL；如果数组存放的元素大于一个，则抛出异常）。
CARDINALITY(map)	返回 map 中键值对的总个数。
map '[' key ']'	返回 map 中指定 key 所对应的值。

分组函数

分组函数用来做 GROUP BY 分组后的表示，函数名和功能描述如下：

函数名	功能描述
GROUP_ID()	返回一个唯一标识该 GROUP 所有键的整数。
GROUPING(expression1 [, expression2]*) GROUPING_ID(expression1 [, expression2]*)	返回指定分组表达式的分组 ID（二进制矢量转为十进制数）。

版权所有：腾讯云计算（北京）有限责任公司

标识符与保留字

命名规则

最近更新时间：2020-12-23 11:50:50

标识符用来唯一地表示表名或列名。当前流计算 Oceanus 中标识符的命名规则如下：

- 不能使用 数据类型 中所指定的保留字。若必须使用这些保留字作为表名或列名，请使用反引号（``）将其包括，例如 `time`。
- 请不要使用 PROCTIME 和 SOURCETIME 作为列名，以避免与系统自动生成的时间戳发生冲突。
- 不能以 _DataStreamTable_ 开头。
- 如果表名或列名含有空格或者特殊字符，请使用反引号将其括起来，例如 `HELLO WORLD`。
- 长度必须小于等于128个英文（半角）字符（一个汉字或全角字符等价于两个英文字符）。

版权所有：腾讯云计算（北京）有限责任公司

保留字

最近更新时间：2020-12-23 11:52:17

流计算 Oceanus 内部的保留字如下所示。在表名或列名中使用这些保留字时，必须使用反引号（` `）括起来，否则语法检查时会报错。

> **注意：**
> 在 Processing Time 时间模式下（即未使用 WATERMARK FOR 来定义数据源的时间戳字段），请不要使用 PROCTIME 作为列名，以免与系统自动生成的时间戳发生命名冲突。

A

```
A, ABS, ABSOLUTE, ACTION, ADA, ADD, ADMIN, AFTER, ALL, ALLOCATE, ALLOW,
ALTER, ALWAYS, AND, ANY, ARE, ARRAY, AS, ASC, ASENSITIVE, ASSERTION,
ASSIGNMENT, ASYMMETRIC, AT, ATOMIC, ATTRIBUTE, ATTRIBUTES, AUTHORIZATION, AVG
```

B

```
BEFORE, BEGIN, BERNOULLI, BETWEEN, BIGINT, BINARY, BIT, BLOB, BOOLEAN, BOTH BREADTH, BY
```

C

```
C, CALL, CALLED, CARDINALITY, CASCADE, CASCADED, CASE, CAST, CATALOG, CATALOG_NAME,
CEIL, CEILING, CENTURY, CHAIN, CHAR, CHARACTER, CHARACTERISTICS, CHARACTERS,
CHARACTER_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_NAME, CHARACTER_SET_SCHEMA,
CHAR_LENGTH, CHECK, CLASS_ORIGIN, CLOB, CLOSE, COALESCE, COBOL, COLLATE, COLLATION,
COLLATION_CATALOG, COLLATION_NAME, COLLATION_SCHEMA, COLLECT, COLUMN, COLUMN_NAME,
COMMAND_FUNCTION, COMMAND_FUNCTION_CODE, COMMIT, COMMITTED,CONDITION, CONDITION_NUMBER,
CONNECT, CONNECTION, CONNECTION_NAME, CONSTRAINT, CONSTRAINTS, CONSTRAINT_CATALOG,
CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORR,
CORRESPONDING, COUNT, COVAR_POP, COVAR_SAMP, CREATE, CROSS, CUBE, CUME_DIST, CURRENT,
CURRENT_CATALOG, CURRENT_DATE, CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRENT_ROLE,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_TRANSFORM_GROUP_FOR_TYPE,
CURRENT_USER, CURSOR, CURSOR_NAME, CYCLE, DATA, DATABASE, DATE, DATETIME_INTERVAL_CODE
```

D

```
DATETIME_INTERVAL_PRECISION, DAY, DEALLOCATE, DEC, DECADE, DECIMAL, DECLARE, DEFAULT,
DEFAULTS, DEFERRABLE, DEFERRED, DEFINED, DEFINER, DEGREE, DELETE, DENSE_RANK, DEPTH, DEREF,
DERIVED, DESC, DESCRIBE, DESCRIPTION, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISALLOW, DISCONNEC
T,
```

版权所有：腾讯云计算（北京）有限责任公司

```
DISPATCH, DISTINCT, DOMAIN, DOUBLE, DOW, DOY, DROP, DYNAMIC, DYNAMIC_FUNCTION, DYNAMIC_FUNCTION_C
ODE
```

E

```
EACH, ELEMENT, ELSE, END, END-EXEC, EPOCH, EQUALS, ESCAPE, EVERY, EXCEPT, EXCEPTION,
EXCLUDE, EXCLUDING, EXEC, EXECUTE, EXISTS, EXP, EXPLAIN, EXTEND, EXTERNAL, EXTRACT
```

F

```
FALSE, FETCH, FILTER, FINAL, FIRST, FIRST_VALUE, FLOAT, FLOOR, FOLLOWING, FOR, FOREIGN,
FORTRAN, FOUND, FRAC_SECOND, FREE, FROM, FULL, FUNCTION, FUSION
```

G

```
G, GENERAL, GENERATED, GET, GLOBAL, GO, GOTO, GRANT, GRANTED, GROUP, GROUPING
```

H

```
HAVING, HIERARCHY, HOLD, HOUR
```

I

```
IDENTITY, IMMEDIATE, IMPLEMENTATION, IMPORT, IN, INCLUDING, INCREMENT, INDICATOR,
INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTANCE, INSTANTIABLE, INT,
INTEGER, INTERSECT, INTERSECTION, INTERVAL, INTO, INVOKER, IS, ISOLATION
```

J

```
JAVA, JOIN
```

K

```
K, KEY, KEY_MEMBER, KEY_TYPE
```

L

```
LABEL, LANGUAGE, LARGE, LAST, LAST_VALUE, LATERAL, LEADING, LEFT, LENGTH, LEVEL,
LIBRARY, LIKE, LIMIT, LN, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, LOWER
```

版权所有：腾讯云计算（北京）有限责任公司

M

```
M, MAP, MATCH, MATCHED, MAX, MAXVALUE, MEMBER, MERGE, MESSAGE_LENGTH, MESSAGE_OCTET_LENGTH,
MESSAGE_TEXT, METHOD, MICROSECOND, MILLENNIUM, MIN, MINUTE, MINVALUE, MOD, MODIFIES, MODULE,
MONTH, MORE, 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, OCTETS, OCTET_LENGTH, OF, OFFSET, OLD, ON, ONLY, OPEN, OPTION, OPTIONS, OR,
ORDER, ORDERING, ORDINALITY, OTHERS, OUT, OUTER, OUTPUT, OVER, OVERLAPS, OVERLAY, OVERRIDING
```

P

```
PAD, PARAMETER, PARAMETER_MODE, PARAMETER_NAME, PARAMETER_ORDINAL_POSITION,
PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_NAME, PARAMETER_SPECIFIC_SCHEMA, PARTIAL,
PARTITION, PASCAL, PASSTHROUGH, PATH, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK, PLACING,
PLAN, PLI, POSITION, POWER, PRECEDING, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES,
PROCEDURE, PUBLIC
```

Q

```
QUARTER
```

R

```
RANGE, RANK, READ, READS, REAL, RECURSIVE, REF, REFERENCES, REFERENCING, REGR_AVGX,
REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY,
RELATIVE, RELEASE, REPEATABLE, RESET, RESTART, RESTRICT, RESULT, RETURN, RETURNED_CARDINALITY,
RETURNED_LENGTH, RETURNED_OCTET_LENGTH, RETURNED_SQLSTATE, RETURNS, REVOKE, RIGHT, ROLE, ROLLBACK
,
ROLLUP, ROUTINE, ROUTINE_CATALOG, ROUTINE_NAME, ROUTINE_SCHEMA, ROW, ROWS, ROW_COUNT, ROW_NUMBER
```

S

```
SAVEPOINT, SCALE, SCHEMA, SCHEMA_NAME, SCOPE, SCOPE_CATALOGS, SCOPE_NAME, SCOPE_SCHEMA,
SCROLL, SEARCH, SECOND, SECTION, SECURITY, SELECT, SELF, SENSITIVE, SEQUENCE, SERIALIZABLE,
```

版权所有：腾讯云计算（北京）有限责任公司

```
SERVER, SERVER_NAME, SESSION, SESSION_USER, SET, SETS, SIMILAR, SIMPLE, SIZE, 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,
STATE, STATEMENT, STATIC, STDDEV_POP, STDDEV_SAMP, STREAM, STRUCTURE, STYLE, SUBCLASS_ORIGIN,
SUBMULTISET, SUBSTITUTE, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER
```

T

```
TABLE, TABLESAMPLE, TABLE_NAME, TEMPORARY, THEN, TIES, TIME, TIMESTAMP, TIMESTAMPADD,
TIMESTAMPDIFF, TIMEZONE_HOUR, TIMEZONE_MINUTE, TINYINT, TO, TOP_LEVEL_COUNT, TRAILING,
TRANSACTION, TRANSACTIONS_ACTIVE, TRANSACTIONS_COMMITTED, TRANSACTIONS_ROLLED_BACK,
TRANSFORM, TRANSFORMS, TRANSLATE, TRANSLATION, TREAT, TRIGGER, TRIGGER_CATALOG,
TRIGGER_NAME, TRIGGER_SCHEMA, TRIM, TRUE, TYPE
```

U

```
UESCAPE, UNBOUNDED, UNCOMMITTED, UNDER, UNION, UNIQUE, UNKNOWN, UNNAMED, UNNEST,
UPDATE, UPPER, UPSERT, USAGE, USER, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_CODE,
USER_DEFINED_TYPE_NAME, USER_DEFINED_TYPE_SCHEMA, USING
```

V

```
VALUE, VALUES, VARBINARY, VARCHAR, VARYING, VAR_POP, VAR_SAMP, VERSION, VIEW
```

W

```
WATERMARK, WEEK, WHEN, WHENEVER, WHERE, WIDTH_BUCKET, WINDOW, WITH, WITHIN, WITHOUT,
WORK, WRAPPER, WRITE
```

X

```
XML
```

Y

```
YEAR
```

Z

版权所有：腾讯云计算（北京）有限责任公司

```
ZONE
```

版权所有：腾讯云计算（北京）有限责任公司