1303_Spark 语法参考

文档版本 01

发布日期 2026-07-17

华为云计算技术有限公司

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云计算技术有限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，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

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。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 Spark SQL 语法概览

数据库相关语法

语法	说明
2.1 创建数据库	创建新的数据库
2.2 删除数据库	删除已有数据库
2.3 查看指定数据库	查看数据库属性
2.4 查看所有数据库	列出所有数据库
2.5 修改数据库属性	修改数据库的属性或位置

创建管理表相关语法

语法	说明
3.2.1 使用DataSource语法创建Managed Table	使用DataSource语法创建管理表
3.2.2 使用Hive语法创建Managed Table	使用Hive语法创建管理表

创建 External 表相关语法

语法	说明
3.2.3 使用DataSource语法创建External Table	使用DataSource语法创建OBS外部表
3.2.4 使用Hive语法创建External Table	使用Hive语法创建OBS外部表

删除表相关语法

语法	说明
3.3 删除表	删除已有表

查看表相关语法

语法	说明
3.4.1 查看所有表	列出数据库下所有表
3.4.2 查看建表语句	查看表的建表DDL
3.4.3 查看表属性	查看表的TBLPROPERTIES
3.4.4 查看指定表所有列	查看表的列信息
3.4.5 查看指定表所有分区	查看表的分区信息
3.4.6 查看表统计信息	查看表的统计信息

修改表相关语法

语法	说明
3.5 修改表	修改表属性
3.5.1 添加列	添加新列
3.5.2 修改列注释	修改列的注释信息

分区表相关语法

语法	说明
3.7.1 添加分区	添加新分区
3.7.2 重命名分区	重命名分区
3.7.3 删除分区	删除分区
3.7.5 更新表分区信息	更新分区信息
指定筛选条件删除分区	按条件删除分区

导入导出数据相关语法

语法	说明
4.1 导入数据	导入数据到表
4.2 插入数据	插入数据到表
4.4 清空数据	清空表数据
5 导出查询结果	导出查询结果到OBS

视图相关语法

语法	说明
6.1 创建视图	创建新视图
6.2 删除视图	删除已有视图

数据权限相关语法

语法	说明
9.2 创建角色	创建新角色
9.4 删除角色	删除已有角色
9.5 绑定角色	绑定角色到用户
9.6 解绑角色	解绑用户角色
9.3 显示角色	显示所有角色
9.7 分配权限	分配权限给角色或用户
9.8 回收权限	回收已分配的权限
9.9 显示已授权限	显示已分配的权限
9.10 显示所有角色和用户的绑定关系	显示角色用户绑定
9.1 数据权限列表	列出所有权限类型

自定义函数相关语法

语法	说明
12.2 创建函数	创建自定义函数
12.3 删除函数	删除自定义函数

语法	说明
12.4 显示函数详情	查看函数详情
12.5 显示所有函数	列出所有函数

查看计划相关语法

语法	说明
8 查看计划	查看SQL执行计划

2 数据库相关

2.1 创建数据库

功能描述

创建数据库。

语法格式

```
CREATE [DATABASE | SCHEMA] [IF NOT EXISTS] db_name
  [COMMENT db_comment]
  LOCATION obs_path
  [WITH DBPROPERTIES (property_name=property_value, ...)];
```

关键字

- **IF NOT EXISTS**：所需创建的数据库已存在时不报错，静默跳过。
- **COMMENT**：对数据库的描述。
- **LOCATION**：指定数据库在文件系统中的存储路径。必须指定，且为OBS格式。例如obs://bucket-name/db-path/。不指定LOCATION或不满足格式要求将报错。
- **WITH DBPROPERTIES**：数据库的属性，属性名和属性值成对出现。

参数说明

表 2-1 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以数字和下划线开头。
db_comment	数据库描述。
obs_path	数据库存储路径，必须为obs://格式，指向有效的对象存储路径。
property_name	数据库属性名。
property_value	数据库属性值。

注意事项

- DATABASE与SCHEMA两者没有区别，可替换使用，建议使用DATABASE。
- "default"为内置数据库，不能创建名为"default"的数据库。
- 使用DESCRIBE DATABASE EXTENDED可查看数据库的属性信息。

示例 1：创建数据库

创建数据库testdb。如果testdb已存在则不报错。

```
CREATE DATABASE IF NOT EXISTS testdb
  LOCATION 'obs://my-bucket/testdb/';
```

示例 2：创建带注释和属性的数据库

创建数据库customer_db，添加注释和属性信息。

```
CREATE DATABASE IF NOT EXISTS customer_db
  COMMENT 'This is customer database'
  LOCATION 'obs://my-bucket/customer_db/'
  WITH DBPROPERTIES (ID=001, Name='John');
```

示例 3：验证数据库属性

使用DESCRIBE DATABASE EXTENDED查看数据库的详细信息，包括属性。

```
DESCRIBE DATABASE EXTENDED customer_db;
```

结果示例：

表 2-2 结果示例

database_description_item	database_description_value
Database Name	customer_db
Description	This is customer database

database_description_item	database_description_value
Location	obs://my-bucket/customer_db/
Properties	((ID,001), (Name,John))

示例 4：使用 SCHEMA 关键字创建数据库

SCHEMA与DATABASE等效，可替换使用。

```
CREATE SCHEMA IF NOT EXISTS payroll_db
  LOCATION 'obs://my-bucket/payroll_db/';
```

2.2 删除数据库

功能描述

删除数据库及其关联的文件系统目录。如果数据库不存在，系统将抛出异常。

语法格式

```
DROP [DATABASE | SCHEMA] [IF EXISTS] db_name [RESTRICT|CASCADE];
```

关键字

IF EXISTS：所需删除的数据库不存在时使用，可避免系统报错。

注意事项

- DATABASE与SCHEMA两者没有区别，可替换使用，建议使用DATABASE。
- RESTRICT表示如果该数据库不为空（有表存在），DROP操作会报错，执行失败，RESTRICT是默认逻辑。
- CASCADE表示即使该数据库不为空（有表存在），DROP也会级联删除下面的所有表，需要谨慎使用该功能。
- 删除数据库后，该数据库在OBS上的存储路径不会被自动删除，需要手动清理。

参数说明

表 2-3 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线组成。不能是纯数字，且不能以数字和下划线开头。

示例 1：删除空数据库

删除数据库inventory_db。如果数据库不为空，操作将失败。


```
DROP DATABASE inventory_db;
```

示例 2：删除非空数据库（级联删除）

删除数据库inventory_db及其中的所有表。

```
DROP DATABASE inventory_db CASCADE;
```

示例 3：使用 IF EXISTS 安全删除

删除数据库inventory_db。如果数据库不存在，不报错。

```
DROP DATABASE IF EXISTS inventory_db CASCADE;
```

示例 4：使用 SCHEMA 关键字删除数据库

SCHEMA与DATABASE等效，可替换使用。

```
DROP SCHEMA IF EXISTS inventory_db;
```

2.3 查看指定数据库

功能描述

返回指定数据库的元数据信息，包括数据库名称、数据库描述、文件系统上的存储路径等。如果指定EXTENDED选项，还会返回数据库的属性信息。

语法格式

```
DESCRIBE DATABASE [EXTENDED] db_name;
```

关键字

EXTENDED：除了显示上述信息外，还会额外显示数据库的属性（DBPROPERTIES）信息。

参数说明

表 2-4 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线组成。不能是纯数字，且不能以数字和下划线开头。

注意事项

- 如果所要查看的数据库不存在，则系统报错。
- EXTENDED选项会额外显示DBPROPERTIES中定义的属性信息。
- DESC与DESCRIBE等效，可替换使用。
- DATABASE与SCHEMA等效，可替换使用。

示例 1：查看数据库基本信息

查看employees数据库的名称、描述和存储路径。

```
DESCRIBE DATABASE employees;
```

结果示例：

database_description_item	database_description_value
Database Name	employees
Description	For software companies
Location	obs://my-bucket/employees/

示例 2：查看数据库详细信息（含属性）

使用EXTENDED选项查看数据库的属性信息。

```
DESCRIBE DATABASE EXTENDED employees;
```

结果示例：

database_description_item	database_description_value
Database Name	employees
Description	For software companies
Location	obs://my-bucket/employees/
Properties	((Create-by,kevin), (Create-date,09/01/2019))

示例 3：使用 DESC 简写

DESC是DESCRIBE的简写形式。

```
DESC DATABASE employees;
```

示例 4：查看使用 SCHEMA 关键字创建的数据库

DATABASE与SCHEMA等效。

```
DESC DATABASE deployment;
```

结果示例：

database_description_item	database_description_value
Database Name	deployment
Description	Deployment environment

database_description_item	database_description_value
Location	obs://my-bucket/deployment/

2.4 查看所有数据库

功能描述

查看当前工程下所有的数据库。可使用LIKE子句按正则表达式模式筛选数据库名称。如果不提供模式，则列出系统中所有数据库。

语法格式

```
SHOW [DATABASES | SCHEMAS] [LIKE regex_expression];
```

关键字

- LIKE：使用正则表达式模式筛选结果。
- DATABASES / SCHEMAS：两者等效，均可使用。

参数说明

表 2-5 参数说明

参数	描述
regex_expression	用于筛选结果的正则表达式。除*和|外，模式的工作方式与正则表达式相同。 *可用于匹配0个或多个字符，|用于分隔多个不同的正则表达式。模式匹配不区分大小写。

注意事项

DATABASES与SCHEMAS是等效的，都将返回所有的数据库名称。

示例 1：列出所有数据库

查看当前实例中的所有数据库。

```
SHOW DATABASES;
```

表 2-6 结果示例

databaseName
default
payments_db

payroll_db

示例 2：按模式筛选数据库

查看所有以"pay"开头的数据库名称。

```
SHOW DATABASES LIKE 'pay*';
```

表 2-7 结果示例

databaseName
payments_db
payroll_db

示例 3：使用 SCHEMAS 关键字

SCHEMAS与DATABASES等效，可替换使用。

```
SHOW SCHEMAS;
```

示例 4：使用多模式筛选

查看以"pay"开头或以"hr"开头的数据库。

```
SHOW DATABASES LIKE 'pay*|hr*';
```

2.5 修改数据库属性

功能描述

ALTER DATABASE语句用于修改数据库的属性或位置。DATABASE、SCHEMA和NAMESPACE关键字可互换使用。如果数据库不存在，会报错。

语法格式

设置属性

```
ALTER { DATABASE | SCHEMA | NAMESPACE } database_name
    SET { DBPROPERTIES | PROPERTIES } ( property_name = property_value [, ...] )
```

删除属性

```
ALTER { DATABASE | SCHEMA | NAMESPACE } database_name
    UNSET { DBPROPERTIES | PROPERTIES } ( property_name [, ...] )
```

修改位置

```
ALTER { DATABASE | SCHEMA | NAMESPACE } database_name
    SET LOCATION 'new_location'
```


关键字

- **DATABASE / SCHEMA / NAMESPACE**：三者等效，可互换使用，建议使用DATABASE。
- **SET DBPROPERTIES / SET PROPERTIES**：设置数据库属性，DBPROPERTIES和PROPERTIES等效。
- **UNSET DBPROPERTIES / UNSET PROPERTIES**：删除数据库属性。如果指定的属性不存在，命令会忽略并成功执行。
- **SET LOCATION**：修改数据库的默认存储位置。

参数说明

表 2-8 参数说明

参数	是否必选	描述
database_name	是	要修改的数据库名称。
SET DBPROPERTIES / SET PROPERTIES	是	设置数据库属性。指定的属性值会覆盖同名的已有属性值。主要用于记录数据库的元数据信息。
UNSET DBPROPERTIES / UNSET PROPERTIES	是	删除数据库属性。如果指定的属性不存在，命令会忽略并成功执行。
property_name	是	属性名称。
property_value	是（SET时）	属性值。
SET LOCATION	是	修改数据库的默认存储位置。不会将当前数据库目录中的数据从旧位置移动到新位置，也不影响数据库下已有表/分区的位置。

注意事项

- 修改数据库位置不会自动迁移已有数据，仅影响后续新建表的默认位置。
- SET LOCATION的路径需为OBS格式（如obs://bucket-name/path/）。
- DATABASE、SCHEMA和NAMESPACE关键字可互换使用，建议使用DATABASE。
- DBPROPERTIES和PROPERTIES关键字可互换使用。
- 使用DESCRIBE DATABASE EXTENDED可验证属性和位置的修改结果。

示例 1：设置数据库属性

创建数据库inventory，并设置属性。

```
CREATE DATABASE IF NOT EXISTS inventory
  LOCATION 'obs://my-bucket/inventory/';

ALTER DATABASE inventory SET DBPROPERTIES ('Edited-by' = 'John', 'Edit-date' = '01/01/2001');
```

验证属性：


```
DESCRIBE DATABASE EXTENDED inventory;
```

结果示例：

database_description_item	database_description_value
Database Name	inventory
Location	obs://my-bucket/inventory/
Properties	((Edit-date,01/01/2001), (Edited-by,John))

示例 2：修改数据库位置

```
ALTER DATABASE inventory SET LOCATION 'obs://my-bucket/new-inventory/';
```

验证新位置：

```
DESCRIBE DATABASE EXTENDED inventory;
```

示例 3：删除数据库属性

```
ALTER DATABASE inventory UNSET DBPROPERTIES ('Edited-by');
```

删除不存在的属性时命令会忽略并成功执行：

```
ALTER DATABASE inventory UNSET DBPROPERTIES ('non-existent');
```

示例 4：使用 SCHEMA/NAMESPACE 关键字

SCHEMA、NAMESPACE与DATABASE等效，可替换使用。

```
ALTER SCHEMA inventory SET DBPROPERTIES ('Owner' = 'Admin');
ALTER NAMESPACE inventory SET DBPROPERTIES ('Department' = 'Engineering');
```


3 表相关

3.1 表类型说明

External Table 与 Managed Table 对比

AI DataLake Spark SQL端点支持两种表类型：External Table（外部表）和Managed Table（管理表）。

特性	External Table	Managed Table
数据存储位置	OBS外部路径（通过LOCATION指定）	平台内部存储
删除表行为	仅删除元数据，数据保留在OBS中	删除元数据和数据
建表关键字	不使用EXTERNAL关键字，通过指定LOCATION标识	不指定LOCATION
数据生命周期	由用户管理OBS数据	由平台管理
适用场景	数据已存在于OBS、需跨服务共享数据	临时数据、中间结果、平台托管数据
与开源Spark差异	开源Spark使用CREATE EXTERNAL TABLE，平台不使用EXTERNAL关键字	与开源Spark一致

特性	External Table	Managed Table
支持的建表语法	DataSource语法、Hive语法	DataSource语法、Hive语法

说明

AI DataLake的External Table（OBS表）与开源Spark的External Table概念一致，但建表语法**不使用EXTERNAL关键字**，这是与开源的主要差异点。通过指定LOCATION 'obs://...'路径来创建外部表。

CREATE TABLE 默认存储引擎变更

Spark 3.x中，CREATE TABLE不指定USING子句时默认使用Hive存储格式。

Spark 4.0中，默认存储格式由spark.sql.sources.default配置决定，通常为parquet。

表 3-1 行为变更对比

场景	Spark 3.x	Spark 4.0
CREATE TABLE t (id INT)	默认Hive格式	默认parquet格式
CREATE TABLE t (id INT) USING hive	Hive格式	Hive格式（显式指定）
CREATE TABLE t (id INT) USING parquet	parquet格式	parquet格式（显式指定）

- **使用建议：**
  - 所有建表语句明确指定存储格式，避免依赖默认值。未指定USING的建表语句需确认预期格式
  - AI DataLake建议所有建表语句显式指定USING子句
  - AI DataLake的Managed Table推荐使用USING parquet或USING orc
  - 依赖Hive格式特性的表需显式指定USING hive
- **推荐写法：显式指定格式**

```
-- 创建Managed Table，使用parquet格式
CREATE TABLE t1 (id INT, name STRING) USING parquet;

-- 创建Managed Table，使用hive格式
CREATE TABLE t2 (id INT, name STRING) USING hive;

-- 创建OBS表，使用parquet格式
CREATE TABLE t3 (id INT, name STRING) USING parquet
LOCATION 'obs://bucket/path';
```

3.2 创建表

3.2.1 使用 DataSource 语法创建 Managed Table

功能描述

使用DataSource语法创建管理表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

注意事项

- CTAS建表语句不能指定表的属性。
- 若没有指定分隔符，则默认为逗号（,）。
- 关于分区表的使用说明：
  - 创建分区表时，PARTITIONED BY中指定分区列必须是表中的列，且必须在Column列表中指定类型。分区列只支持STRING, BOOLEAN, TINYINT, SMALLINT, SHORT, INT, BIGINT, LONG, DECIMAL, FLOAT, DOUBLE, DATE, TIMESTAMP类型。
  - 创建分区表时，分区字段必须是表字段的最后一个字段或几个字段，且多分区字段的顺序也必须对应。否则将出错。
  - 单表分区数最多允许200000个。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  USING file_format
  [OPTIONS (key1=val1, key2=val2, ...)]
  [PARTITIONED BY (col_name1, col_name2, ...)]
  [COMMENT table_comment]
  [AS select_statement];
```

关键字

- IF NOT EXISTS：指定该关键字以避免表已经存在时报错。
- USING：指定存储格式。
- OPTIONS：指定建表时的属性名与属性值。
- COMMENT：字段或表描述。
- PARTITIONED BY：指定分区字段。
- AS：使用CTAS创建表。

参数说明

表 3-2 基本参数

参数	是否必选	描述
db_name	否	Database名称。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。

参数	是否必选	描述
table_name	是	Database中的表名。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。 匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。 特殊字符需要使用单引号（'）包围起来。表名对大小写不敏感，即不区分大小写。
col_name	是	以逗号分隔的带数据类型的列名。列名由字母、数字和下划线（_）组成。不能是纯数字，且至少包含一个字母。列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。请参考原生数据类型。
col_comment	否	列字段描述。仅支持字符串常量。
file_format	是	管理表数据存储格式，支持：PARQUET和ORC格式。
table_comment	否	表描述。仅支持字符串常量。
select_statement	否	用于CTAS命令，将源表的SELECT查询结果或某条数据插入到新创建的管理表中。

表 3-3 OPTIONS 参数

参数	是否必选	描述	默认值
multiLevelDirEnable	否	是否迭代查询子目录中的数据。当配置为true时，查询该表时会迭代读取该表路径中所有文件，包含子目录中的文件。	false
compression	否	指定压缩格式。一般为parquet格式时指定该参数，推荐使用zstd压缩格式。	-

示例 1：创建基本管理表

创建名为table1的非分区管理表，使用USING关键字指定存储格式为ORC格式。

```
CREATE TABLE IF NOT EXISTS table1 (
  col_1 STRING,
  col_2 INT
)
USING orc;
```


示例 2：创建带默认值和注释的管理表

创建名为employee的管理表，附带列注释和表注释。

```
CREATE TABLE IF NOT EXISTS employee (
    id      INT COMMENT '员工编号',
    name    STRING COMMENT '员工姓名',
    salary  DECIMAL(10,2) COMMENT '员工薪资'
)
USING parquet
COMMENT '员工信息管理表';
```

示例 3：使用不同格式创建管理表（ORC、JSON）

创建使用不同数据格式的管理表。

```
-- 使用ORC格式
CREATE TABLE IF NOT EXISTS table_orc (
    id      INT,
    data    STRING
)
USING orc;

-- 使用JSON格式
CREATE TABLE IF NOT EXISTS table_json (
    id      INT,
    info    STRING
)
USING json;
```

示例 4：创建带分区的管理表

创建名为student的分区管理表，使用院系编号（facultyNo）和班级编号（classNo）进行分区，并配置OPTIONS属性。

```
CREATE TABLE IF NOT EXISTS student (
    Name        STRING,
    facultyNo   INT,
    classNo     INT
)
USING parquet
PARTITIONED BY (facultyNo, classNo)
OPTIONS (
    compression = 'zstd'
);
```

3.2.2 使用 Hive 语法创建 Managed Table

功能描述

使用Hive语法创建管理表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

注意事项

- CTAS建表语句不能指定表的属性。
- Hive管理表不支持在建表时指定多字符的分隔符。
- 关于分区表的使用说明：
  - 创建分区表时，PARTITIONED BY中指定分区列必须是不在表中的列，且需要指定数据类型。分区列支持STRING, BOOLEAN, TINYINT, SMALLINT,

SHORT, INT, BIGINT, LONG, DECIMAL, FLOAT, DOUBLE, DATE, TIMESTAMP等Hive开源支持的类型。

- 支持指定多个分区字段，分区字段只需在PARTITIONED BY关键字后指定，不能像普通字段一样在表名后指定，否则将出错。
- 单表分区数最多允许200000个。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name2 col_type2 [COMMENT col_comment2], ...)]
  [ROW FORMAT row_format]
  STORED AS file_format
  [TBLPROPERTIES (key1=val1, key2=val2, ...)]
  [AS select_statement];

row_format:
  : SERDE serde_cls [WITH SERDEPROPERTIES (key1=val1, key2=val2, ...)]
  | DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]]
      [COLLECTION ITEMS TERMINATED BY char]
      [MAP KEYS TERMINATED BY char]
      [LINES TERMINATED BY char]
      [NULL DEFINED AS char]
```

关键字

- **IF NOT EXISTS**：指定该关键字以避免表已经存在时报错。
- **COMMENT**：字段或表描述。
- **PARTITIONED BY**：指定分区字段。
- **ROW FORMAT**：行数据格式。
- **STORED AS**：指定所存储的文件格式，当前该关键字只支持指定TEXTFILE、AVRO、ORC、SEQUENCEFILE、RCFILE、PARQUET几种格式。创建管理表时必须指定此关键字。
- **TBLPROPERTIES**：用于为表添加key/value的属性。在表存储格式为PARQUET时，可以通过指定TBLPROPERTIES(parquet.compression = 'zstd')来指定表压缩格式为zstd。
- **AS**：使用CTAS创建表。

参数说明

表 3-4 基本参数

参数	是否必选	描述
db_name	否	Database名称。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	是	Database中的表名。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。特殊字符需要使用单引号（''）包围起来。表名对大小写不敏感，即不区分大小写。

参数	是否必选	描述
col_name	是	列字段名称。列字段由字母、数字和下划线（ _ ）组成。不能是纯数字，且至少包含一个字母。列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。请参考原生数据类型。
col_comment	否	列字段描述。仅支持字符串常量。
row_format	否	行数据格式。ROW FORMAT功能只支持TEXTFILE类型的表。
file_format	是	管理表数据存储格式：支持TEXTFILE、AVRO、ORC、SEQUENCEFILE、RCFILE、PARQUET。
table_comment	否	表描述。仅支持字符串常量。
key = value	否	设置TBLPROPERTIES具体属性和值。在表存储格式为PARQUET时，可以通过指定TBLPROPERTIES(parquet.compression = 'zstd')来指定表压缩格式为zstd。
select_statement	否	用于CTAS命令，将源表的SELECT查询结果或某条数据插入到新创建的管理表中。

示例 1：创建基本 Hive 格式管理表

创建名为table1的非分区管理表，使用STORED AS关键字指定存储格式为ORC。

```
CREATE TABLE IF NOT EXISTS table1 (
   col_1   STRING,
   col_2   INT
)
STORED AS orc;
```

示例 2：创建 STORED AS PARQUET 的管理表

创建名为table_parquet的PARQUET格式管理表，并通过TBLPROPERTIES指定压缩格式。

```
CREATE TABLE IF NOT EXISTS table_parquet (
   id      INT,
   name    STRING,
   score   DOUBLE
)
STORED AS parquet
TBLPROPERTIES (
   parquet.compression = 'zstd'
);
```

示例 3：创建带 ROW FORMAT DELIMITED 的管理表

创建名为table4的TEXTFILE类型管理表，并设置ROW FORMAT行格式。


```
CREATE TABLE IF NOT EXISTS table4 (
   col_1   STRING,
   col_2   INT
)
STORED AS textfile
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '/'
   COLLECTION ITEMS TERMINATED BY '$'
   MAP KEYS TERMINATED BY '#'
   LINES TERMINATED BY '\n'
   NULL DEFINED AS 'NULL';
```

示例 4：创建带 SERDE 的管理表

创建名为table_serde的管理表，使用自定义SERDE类。

```
CREATE TABLE IF NOT EXISTS table_serde (
   id      INT,
   data    STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
   'field.delim' = ',',
   'serialization.format' = ','
)
STORED AS textfile;
```

3.2.3 使用 DataSource 语法创建 External Table

功能描述

使用DataSource语法创建external表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

推荐使用OBS并行文件系统进行存储。并行文件系统是一种高性能文件系统，提供毫秒级别访问时延，TB/s级别带宽和百万级别的IOPS，适用于大数据交互式分析场景。

注意事项

- 创建表时不会统计大小。
- 添加数据时会修改大小为0。
- 如需查看表大小可以通过OBS查看。
- CTAS建表语句不能指定表的属性。
- OBS目录下包含子目录的场景：创建表时，若指定路径为OBS上的目录，且该目录下包含子目录（或嵌套子目录），则子目录下的所有文件类型及其内容也是表内容。您需要保证所指定的目录及其子目录下所有文件类型和建表语句中指定的存储格式一致，所有文件内容和表中的字段一致，否则查询将报错。您可以在建表语句OPTIONS中设置multiLevelDirEnable为true以查询子目录下的内容，此参数默认值为false（注意，此配置项为表属性，请谨慎配置。Hive表不支持此配置项）。
- 关于分区表的使用说明：
  - 创建分区表时，PARTITIONED BY中指定分区列必须是表中的列，且必须在Column列表中指定类型。分区列只支持STRING, BOOLEAN, TINYINT, SMALLINT, SHORT, INT, BIGINT, LONG, DECIMAL, FLOAT, DOUBLE, DATE, TIMESTAMP类型。

  - 创建分区表时，分区字段必须是表字段的最后一个字段或几个字段，且多分区字段的顺序也必须对应。否则将出错。
  - 单表分区数最多允许200000个。
- External Table与Managed Table建表差异：

差异项	External Table (external表)	Managed Table (管理表)
LOCATION	必填，指定OBS存储路径	不指定
数据删除	DROP TABLE仅删除元数据	DROP TABLE删除元数据和数据
EXTERNAL关键字	不使用（与开源不同）	不使用
USING / STORED AS	必须指定	必须指定

与开源Spark的差异：开源Spark使用CREATE EXTERNAL TABLE ... LOCATION创建外部表，平台不使用EXTERNAL关键字，而是通过指定LOCATION路径来区分外部表和管理表。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  USING file_format
  [OPTIONS (path 'obs_path', key1=val1, key2=val2, ...)]
  [PARTITIONED BY (col_name1, col_name2, ...)]
  [COMMENT table_comment]
  [AS select_statement]
```

关键字

- IF NOT EXISTS：指定该关键字以避免表已经存在时报错。
- USING：指定存储格式。
- OPTIONS：指定建表时的属性名与属性值。
- COMMENT：字段或表描述。
- PARTITIONED BY：指定分区字段。
- AS：使用CTAS创建表。

参数说明

表 3-5 基本参数

参数	是否必选	描述
db_name	否	Database名称。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。

参数	是否必选	描述
table_name	是	Database中的待创建的表名。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。特殊字符需要使用单引号（'）包围起来。表名对大小写不敏感，即不区分大小写。
col_name	是	以逗号分隔的带数据类型的列名。列名由字母、数字和下划线（_）组成。不能是纯数字，且至少包含一个字母。列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。请参考原生数据类型。
col_comment	否	列字段描述。仅支持字符串常量。
file_format	是	用于创建表的输入格式。支持ORC、PARQUET、JSON、CSV、AVRO类型。
path	是	数据文件所在的OBS存储路径，推荐使用OBS并行文件系统存储。格式：obs://bucketName/tblPath，bucketName即桶名称，tblPath是目录名称。目录后不需要指定文件名。当OBS的目录下文件夹与文件同名时，创建OBS表指向的路径会优先指向文件而非文件夹。
table_comment	否	表描述信息。仅支持字符串常量。
select_statement	否	用于CTAS命令，将源表的SELECT查询结果或某条数据插入到新创建的OBS表中。

表 3-6 OPTIONS 参数

参数	是否必选	描述	默认值
path	否	指定的表路径，即OBS存储路径。	-
multiLevelDirEnable	否	嵌套子目录场景下，是否迭代查询子目录中的数据。当配置为true时，查询该表时会迭代读取该表路径中所有文件，包含子目录中的文件。	false
dataDelegated	否	是否需要在删除表或分区时，清除path路径下的数据。	false
compression	否	指定压缩格式。一般为parquet格式时指定该参数，推荐使用zstd压缩格式。	-

表 3-7 CSV 数据格式 OPTIONS 参数

参数	是否必选	描述	默认值
delimiter	否	数据分隔符。	,
quote	否	引用字符。	"
escape	否	转义字符。	\
multiLine	否	列数据中是否包含回车符或转行符，true为包含，false为不包含。	false
dateFormat	否	指定CSV文件中DATE字段的日期格式。	yyyy-MM-dd
timestampFormat	否	指定CSV文件中TIMESTAMP字段的日期格式。	yyyy-MM-dd HH:mm:ss
mode	否	指定解析CSV时的模式。 PERMISSIVE：宽容模式，遇到错误的字段时设置该字段为NULL； DROPMALFORMED：遇到错误的字段时丢弃整行； FAILFAST：报错模式，遇到错误的字段时直接报错。	PERMISSIVE
header	否	CSV是否包含表头信息，true表示包含表头信息，false为不包含。	false
nullValue	否	设置代表NULL的字符，例如nullValue="nl"表示设置nl代表NULL。	-
comment	否	设置代表注释开头的字符，例如comment='#'表示以#开头的行为注释。	-
compression	否	设置数据的压缩格式。目前支持gzip、bzip2、deflate压缩格式，若不希望压缩则输入none。	none
encoding	否	数据的编码格式。支持utf-8、gb2312、gbk三种。	utf-8

示例 1：创建基本 OBS 非分区表

创建名为table1的OBS非分区表，使用USING关键字指定存储格式为ORC格式。

```
CREATE TABLE IF NOT EXISTS table1 (
  col_1   STRING,
```



```
    col_2   INT
)
USING orc
OPTIONS (path 'obs://bucketName/filePath');
```

示例 2：创建 OBS 分区表

创建名为student的分区表，使用院系编号（ facultyNo ）和班级编号（ classNo ）进行分区。

```
CREATE TABLE IF NOT EXISTS student (
    Name        STRING,
    facultyNo   INT,
    classNo     INT
)
USING csv
OPTIONS (path 'obs://bucketName/filePath')
PARTITIONED BY (facultyNo, classNo);
```

示例 3：创建带注释和属性的 OBS 分区表

创建名为table3的OBS分区表，附带列注释、表注释以及OPTIONS属性配置。

```
CREATE TABLE IF NOT EXISTS table3 (
    col_1   STRING COMMENT '用户名称',
    col_2   INT COMMENT '用户编号'
)
USING parquet
OPTIONS (
    path 'obs://bucketName/filePath',
    multiLevelDirEnable = true,
    dataDelegated = true,
    compression = 'zstd'
)
PARTITIONED BY (col_2)
COMMENT '用户信息表';
```

示例 4：使用不同数据格式创建 OBS 表

创建使用JSON、AVRO等不同数据格式的OBS表。

```
-- 使用JSON格式
CREATE TABLE IF NOT EXISTS table_json (
    id      INT,
    name    STRING
)
USING json
OPTIONS (path 'obs://bucketName/jsonPath');

-- 使用AVRO格式
CREATE TABLE IF NOT EXISTS table_avro (
    id      INT,
    data    STRING
)
USING avro
OPTIONS (path 'obs://bucketName/avroPath');
```

3.2.4 使用 Hive 语法创建 External Table

功能描述

使用Hive语法创建OBS表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

推荐使用OBS并行文件系统进行存储。并行文件系统是一种高性能文件系统，提供毫秒级别访问时延，TB/s级别带宽和百万级别的IOPS，适用于大数据交互式分析场景。

注意事项

- 创建表时会统计大小。
- 添加数据时不会修改大小。
- 如需查看表大小可以通过OBS查看。
- CTAS建表语句不能指定表的属性。
- **关于分区表的使用说明：**
  - 创建分区表时，PARTITIONED BY中指定分区列必须是不在表中的列，且需要指定数据类型。分区列支持STRING, BOOLEAN, TINYINT, SMALLINT, SHORT, INT, BIGINT, LONG, DECIMAL, FLOAT, DOUBLE, DATE, TIMESTAMP等Hive开源支持的类型。
  - 支持指定多个分区字段，分区字段只需在PARTITIONED BY关键字后指定，不能像普通字段一样在表名后指定，否则将出错。
  - 单表分区数最多允许200000个。
- **关于创建表时设置多字符的分隔符：**
  - 只有指定ROW FORMAT SERDE为 org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe时，字段分隔符才支持设置为多字符。
  - 只有Hive OBS表支持在建表时指定多字符的分隔符，Hive管理表不支持在建表时指定多字符的分隔符。
  - 指定了多字符分隔的表不支持INSERT、IMPORT等写数语句。如需添加数据，请将数据文件直接放到表对应的OBS路径下即可。
- **External Table与Managed Table建表差异：**

差异项	External Table	Managed Table
LOCATION	必填，指定OBS存储路径	不指定
数据删除	DROP TABLE仅删除元数据	DROP TABLE删除元数据和数据
EXTERNAL关键字	不使用（与开源不同）	不使用
STORED AS	必须指定	必须指定

与开源Spark的差异：开源Spark使用CREATE EXTERNAL TABLE ... LOCATION创建外部表，平台不使用EXTERNAL关键字，而是通过指定LOCATION路径来区分外部表和管理表。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name2 col_type2 [COMMENT col_comment2], ...)]
  [ROW FORMAT row_format]
```



```
  [STORED AS file_format]
  LOCATION 'obs_path'
  [TBLPROPERTIES (key1=val1, key2=val2, ...)]
  [AS select_statement]

row_format:
  : SERDE serde_cls [WITH SERDEPROPERTIES (key1=val1, key2=val2, ...)]
  | DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]]
      [COLLECTION ITEMS TERMINATED BY char]
      [MAP KEYS TERMINATED BY char]
      [LINES TERMINATED BY char]
      [NULL DEFINED AS char]
```

关键字

- **IF NOT EXISTS**：指定该关键字以避免表已经存在时报错。
- **COMMENT**：字段或表描述。
- **PARTITIONED BY**：指定分区字段。
- **ROW FORMAT**：行数据格式。
- **STORED AS**：指定所存储的文件格式，当前该关键字只支持指定TEXTFILE、AVRO、ORC、SEQUENCEFILE、RCFILE、PARQUET格式。
- **LOCATION**：指定OBS的路径。创建OBS表时必须指定此关键字。
- **TBLPROPERTIES**：允许用户给表添加key/value的属性。
- **AS**：使用CTAS创建表。

参数说明

表 3-8 基本参数

参数	是否必选	描述
db_name	否	Database名称。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	是	Database中的表名。由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。特殊字符需要使用单引号（''）包围起来。表名对大小写不敏感，即不区分大小写。
col_name	是	列字段名称。列字段由字母、数字和下划线（_）组成。不能是纯数字，且至少包含一个字母。列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。请参考原生数据类型。
col_comment	否	列字段描述。仅支持字符串常量。
row_format	否	行数据格式。ROW FORMAT功能只支持TEXTFILE类型的表。
file_format	是	OBS表存储格式，支持TEXTFILE、AVRO、ORC、SEQUENCEFILE、RCFILE、PARQUET。

参数	是否必选	描述
table_comment	否	表描述。仅支持字符串常量。
obs_path	是	数据文件所在的OBS存储路径，推荐使用OBS并行文件系统存储。格式：obs://bucketName/tblPath，bucketName即桶名称，tblPath是目录名称。目录后不需要指定文件名。当OBS的目录下文件夹与文件同名时，创建OBS表指向的路径会优先指向文件而非文件夹。
key = value	否	设置TBLPROPERTIES具体属性和值。
select_statement	否	用于CTAS命令，将源表的SELECT查询结果或某条数据插入到新创建的OBS表中。

表 3-9 TBLPROPERTIES 主要参数

参数	是否必选	描述
comment	否	表描述信息。
orc.compress	否	ORC存储格式表的一个属性，用来指定ORC存储的压缩方式。支持取值为：ZLIB、SNAPPY、NONE。
auto.purge	否	当设置为true时，删除或者覆盖的数据会不经过回收站，直接被删除。

示例 1：创建基本 Hive 格式 OBS 非分区表

创建名为table1的OBS非分区表，使用STORED AS关键字指定存储格式为ORC。

```
CREATE TABLE IF NOT EXISTS table1 (
  col_1   STRING,
  col_2   INT
)
STORED AS orc
LOCATION 'obs://bucketName/filePath';
```

示例 2：创建 STORED AS TEXTFILE 的 OBS 表

创建名为table_text的TEXTFILE格式OBS表。

```
CREATE TABLE IF NOT EXISTS table_text (
  id      INT,
  name    STRING
)
STORED AS textfile
LOCATION 'obs://bucketName/textFilePath';
```

示例 3：创建带 ROW FORMAT 的 OBS 表

创建名为table4的TEXTFILE类型OBS非分区表，并设置ROW FORMAT行格式。


```
CREATE TABLE IF NOT EXISTS table4 (
   col_1   STRING,
   col_2   INT
)
STORED AS textfile
LOCATION 'obs://bucketName/filePath'
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '/'
   COLLECTION ITEMS TERMINATED BY '$'
   MAP KEYS TERMINATED BY '#'
   LINES TERMINATED BY '\n'
   NULL DEFINED AS 'null';
```

示例 4：创建带 SERDE 的 OBS 表

创建名为table5的OBS表，使用自定义SERDE并设置多字符分隔符。

```
CREATE TABLE IF NOT EXISTS table5 (
   col_1   STRING,
   col_2   INT
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe'
WITH SERDEPROPERTIES (
   'field.delim' = '/#'
)
STORED AS textfile
LOCATION 'obs://bucketName/filePath';
```

3.3 删除表

功能描述

删除表。DROP TABLE 用于删除已有的表及其元数据。

- OBS表（外部表）：仅删除元数据信息，存放在OBS上的数据不会被删除。
- 管理表：同时删除数据及相应的元数据信息。

语法格式

```
DROP TABLE [ IF EXISTS ] [ db_name. ] table_name
```

关键字

IF EXISTS：若指定，当表不存在时不会报错；若未指定且表不存在，将抛出异常。

参数说明

参数	是否必选	描述
db_name	否	数据库名称。若未指定，则默认使用当前数据库。命名规则：由字母、数字和下划线（_）组成，不能是纯数字，且不能以数字和下划线开头。
table_name	是	要删除的表名称。

注意事项

- 所要删除的表必须是当前数据库下存在的，否则会出错，可以通过添加 IF EXISTS 来避免出错。
- 对于OBS表（外部表），DROP TABLE 仅删除元数据，OBS路径下的数据文件仍会保留；对于管理表，元数据和数据将同时被删除。

示例 1：基本删除表

删除当前数据库下名为 employee 的表。

```
DROP TABLE employee;
```

示例 2：使用 IF EXISTS 避免报错

删除名为 employee 的表，若该表不存在则不会报错。

```
DROP TABLE IF EXISTS employee;
```

示例 3：删除指定数据库的表

删除 company_db 数据库下名为 employee 的表。

```
DROP TABLE IF EXISTS company_db.employee;
```

示例 4：OBS 表与管理表的区别说明

以下示例演示OBS表与Managed Table在删除时的行为差异：

```
-- 删除管理表：元数据和数据同时被删除
DROP TABLE IF EXISTS managed_table;

-- 删除OBS表（外部表）：仅删除元数据，OBS上的数据文件保留
DROP TABLE IF EXISTS obs_table;
```

若需要同时清理OBS表对应的数据文件，需手动删除OBS路径下的文件，例如 obs://bucket-name/path/to/data/。

3.4 查看表

3.4.1 查看所有表

功能描述

查看当前数据库下所有的表及视图。可通过 IN / FROM 指定数据库，使用 LIKE 进行模式匹配筛选。

语法格式

```
SHOW TABLES [ { FROM | IN } db_name ] [ LIKE regex_expression ]
```


参数说明

参数	是否必选	描述
FROM / IN	否	指定数据库名称，用于查看特定数据库下的表及视图。FROM 和 IN 功能相同，可互换使用。
db_name	否	数据库名称。若未指定，则默认查看当前数据库。命名规则：由字母、数字和下划线（_）组成，不能是纯数字，且不能以数字和下划线开头。
LIKE regex_expression	否	使用正则表达式对表名进行模式匹配筛选。 支持 *（匹配任意字符序列）和 `（或）等通配符。

注意事项

无

示例 1：查看当前数据库中的所有表与视图

```
SHOW TABLES;
```

返回结果示例：

```
+---------+---------+-----------+
|database |tableName|isTemporary|
+---------+---------+-----------+
|default  |employee |false      |
|default  |student  |false      |
|         |temp_view|true       |
+---------+---------+-----------+
```

示例 2：查看指定数据库中的所有表

查看 company_db 数据库下的所有表。

```
SHOW TABLES FROM company_db;
```

示例 3：使用 LIKE 模式筛选表名

查看当前数据库下所有以 emp 开头的表。

```
SHOW TABLES LIKE "emp*";
```

示例 4：查看指定数据库中以特定前缀开头的表

查看 company_db 数据库下所有以 dept 开头的表。

```
SHOW TABLES IN company_db LIKE "dept*";
```


3.4.2 查看建表语句

功能描述

返回对应表的建表语句（CREATE TABLE statement）。SHOW CREATE TABLE用于查看表的完整定义语句。

- 默认返回DataSource格式的建表语句。
- 使用AS SERDE子句可返回Hive格式的建表语句，适用于Hive SerDe表。

语法格式

SHOW CREATE TABLE [db_name.] table_name [AS SERDE]

参数说明

参数	是否必选	描述
db_name	否	数据库名称。若未指定，则默认使用当前数据库。
table_name	是	表名称。所涉及的表必须存在，否则会出错。
AS SERDE	否	用于生成Hive SerDe格式的建表语句，适用于使用Hive存储格式的表。若未指定，则默认生成DataSource格式的建表语句。

注意事项

- 所涉及的表必须存在，否则会出错。
- 不使用AS SERDE时，返回DataSource格式的建表语句（推荐）。
- 使用AS SERDE时，返回Hive SerDe格式的建表语句，包含ROW FORMAT SERDE、INPUTFORMAT、OUTPUTFORMAT等Hive特有属性。

示例 1：基本查看建表语句

查看employee表的建表语句。

```
SHOW CREATE TABLE employee;
```

返回结果示例（DataSource 格式）：

```
+----------------------+
|createtab_stmt      |
+----------------------+
|CREATE TABLE employee (
  id INT,
  name STRING,
  salary DOUBLE)
USING parquet
TBLPROPERTIES (
  'transient_lastDdlTime' = '1718534400')
+----------------------+
```


示例 2：查看 DataSource 表的建表语句

创建一个 DataSource 格式的表并查看其建表语句：

```
-- 创建 DataSource 格式的表
CREATE TABLE sales (order_id INT, amount DOUBLE, order_date DATE)
  USING parquet
  LOCATION 'obs://my-bucket/warehouse/sales';

-- 查看建表语句
SHOW CREATE TABLE sales;
```

示例 3：查看 Hive 格式表的建表语句（AS SERDE）

使用 AS SERDE 查看 Hive 格式的建表语句：

```
-- 创建 Hive 格式的表
CREATE TABLE logs (id INT, message STRING)
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
  STORED AS TEXTFILE;

-- 以 Hive SerDe 格式查看建表语句
SHOW CREATE TABLE logs AS SERDE;
```

返回结果示例（Hive SerDe 格式）：

```
+----------------------+
|createtab_stmt      |
+----------------------+
|CREATE TABLE logs (
  id INT,
  message STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS
  INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
  OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
TBLPROPERTIES (
  'serialization.format' = '1',
  'transient_lastDdlTime' = '1718534400')
+----------------------+
```

示例 4：查看指定数据库的表建表语句

查看 company_db 数据库下 employee 表的建表语句。

```
SHOW CREATE TABLE company_db.employee;
```

3.4.3 查看表属性

功能描述

查看表的属性。SHOW TBLPROPERTIES 用于返回表的属性键值对信息，可查看全部属性或指定属性的值。

语法格式

```
SHOW TBLPROPERTIES [ db_name. ] table_name [ ( 'property_name' ) ]
```

关键字

SHOW TBLPROPERTIES：查看表属性的关键字，TBLPROPERTIES和PROPERTIES可互换使用。

参数说明

参数	是否必选	描述
db_name	否	数据库名称。若未指定，则默认使用当前数据库。
table_name	是	表名称。
property_name	否	属性名称，需用单引号包围。若未指定，则返回表的所有属性及其值；若指定，则仅返回该属性对应的值。注意：property_name 大小写敏感，且不能同时指定多个 property_name，否则会出错。

注意事项

- property_name 大小写敏感。例如 'owner' 和 'Owner' 是不同的属性。
- 不能同时指定多个 property_name，否则会出错。
- 若指定的 property_name 不存在，将返回空值。

示例 1：查看表的所有属性

查看 employee 表的所有属性及其值。

```
SHOW TBLPROPERTIES employee;
```

返回结果示例：

```
+--------------------+--------------------+
|key                 |value               |
+--------------------+--------------------+
|owner               |spark               |
|transient_lastDdlTime|1718534400          |
+--------------------+--------------------+
```

示例 2：查看指定属性的值

查看 employee 表中属性 owner 的值。

```
SHOW TBLPROPERTIES employee ('owner');
```

返回结果示例：

```
+-----+
|value|
+-----+
|spark|
+-----+
```

示例 3：查看带自定义属性的表的属性

先创建一个带有自定义TBLPROPERTIES的表，再查看其属性：

```
-- 创建带有自定义属性的表
CREATE TABLE employee (id INT, name STRING)
  TBLPROPERTIES ('owner' = 'admin', 'department' = 'hr', 'created_by' = 'spark_sql');

-- 查看所有属性
SHOW TBLPROPERTIES employee;
```


示例 4：property_name 大小写敏感说明

property_name 大小写敏感，以下查询将返回不同结果：

```
-- 查看小写属性名 'owner' 的值（可正确匹配）
SHOW TBLPROPERTIES employee ('owner');

-- 查看大写属性名 'Owner' 的值（与 'owner' 不同，可能返回空值）
SHOW TBLPROPERTIES employee ('Owner');
```

'owner' 和 'Owner' 被视为不同的属性名。使用时需确保大小写与定义时一致。

3.4.4 查看指定表所有列

功能描述

查看指定表中的所有列。可通过 FROM / IN 指定表所属的数据库。

语法格式

SHOW COLUMNS { FROM | IN } table_name [{ FROM | IN } db_name]

参数说明

参数	是否必选	描述
table_name	是	表名称。
db_name	否	数据库名称。用于限定表所属的数据库。
FROM / IN	是（第一组）	指定要查看列的表。FROM 和 IN 功能相同，可互换使用。
FROM / IN	否（第二组）	指定表所属的数据库。与第一组关键字功能相同。

注意事项

- 所指定的表必须是数据库中存在的表，否则会出错。
- 第一组 FROM / IN 后跟表名，第二组 FROM / IN 后跟数据库名，两者功能相同但位置含义不同。

示例 1：查看表的所有列

查看student表中的所有列。

```
SHOW COLUMNS FROM student;
```

返回结果示例：

```
+---------+
|col_name |
+---------+
|id       |
|name     |
|age      |
```



```
|score    |
+---------+
```

示例 2：指定数据库查看表的列

查看school_db数据库下student表的所有列。

```
SHOW COLUMNS FROM student FROM school_db;
```

示例 3：使用 IN 关键字查看表的列

使用IN关键字查看student表中的所有列。

```
SHOW COLUMNS IN student;
```

示例 4：使用 IN 关键字指定数据库

使用IN关键字指定数据库查看student表的所有列。

```
SHOW COLUMNS IN student IN school_db;
```

3.4.5 查看指定表所有分区

功能描述

查看指定表的所有分区。可通过 PARTITION 子句指定分区条件进行筛选，支持部分分区键匹配。

语法格式

```
SHOW PARTITIONS [ db_name. ] table_name [ PARTITION partition_specs ]
```

其中 partition_specs 的语法为：

```
partition_specs = ( partition_col_name = partition_col_value [ , partition_col_name =
partition_col_value , ... ] )
```

参数说明

参数	是否必选	描述
db_name	否	数据库名称。若未指定，则默认使用当前数据库。命名规则：由字母、数字和下划线（_）组成，不能是纯数字，且不能以下划线开头。
table_name	是	表名称。所查看的表必须是分区表，否则会出错。命名规则：由字母、数字和下划线（_）组成，不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果包含特殊字符需要使用反引号（``）包围起来。
PARTITION partition_specs	否	分区筛选条件，格式为 key=value 形式。若分区字段为多个字段，可以不包含所有字段，系统会返回匹配上的所有分区信息。

注意事项

- 所要查看分区的表必须存在且是分区表，否则会出错。
- partition_specs 中可以只指定部分分区键，系统将返回所有匹配的分区。例如，对于 (year, month, day) 三级分区表，只指定 year=2024 将返回该年份下的所有分区。

示例 1：查看表的所有分区

查看 student 表下的所有分区。

```
SHOW PARTITIONS student;
```

返回结果示例：

```
+-------------------+
|partition          |
+-------------------+
|year=2024/month=01 |
|year=2024/month=02 |
|year=2024/month=03 |
+-------------------+
```

示例 2：查看指定分区

查看 student 表中 year=2024 且 month=01 的分区。

```
SHOW PARTITIONS student PARTITION(year='2024', month='01');
```

示例 3：部分分区键匹配

查看 student 表中 year=2024 的所有分区（不指定 month，将返回该年份下的所有月份分区）。

```
SHOW PARTITIONS student PARTITION(year='2024');
```

返回结果示例：

```
+-------------------+
|partition          |
+-------------------+
|year=2024/month=01 |
|year=2024/month=02 |
|year=2024/month=03 |
+-------------------+
```

示例 4：查看指定数据库的分区表

查看 school_db 数据库下 student 表的所有分区。

```
SHOW PARTITIONS school_db.student;
```

3.4.6 查看表统计信息

功能描述

查看表的统计信息。默认返回所有列的列名和列数据类型，使用 EXTENDED 或 FORMATTED 可查看详细的表元数据信息。

DESCRIBE 与 DESC 简写等效。

语法格式

{ DESCRIBE | DESC } [EXTENDED | FORMATTED] [db_name.] table_name

参数说明

参数	是否必选	描述
DESCRIBE / DESC	是	查看表统计信息的关键字，两者等效。DESC 是 DESCRIBE 的简写形式。
EXTENDED	否	显示表的所有元数据信息，包括表的详细统计信息和配置信息，通常在调试时使用。
FORMATTED	否	以格式化的表格形式显示所有元数据信息，与 EXTENDED 类似但输出更易读。
db_name	否	数据库名称。若未指定，则默认使用当前数据库。命名规则：由字母、数字和下划线（_）组成，不能是纯数字，且不能以下划线开头。
table_name	是	表名称。命名规则：由字母、数字和下划线（_）组成，不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果包含特殊字符需要使用反引号（``）包围起来。

注意事项

- 若所查看的表不存在，将会出错。
- 默认情况下仅返回列名和数据类型；使用 EXTENDED 或 FORMATTED 可查看包含分区信息、存储格式、表属性等在内的完整元数据。

示例 1：基本 DESCRIBE 查看表信息

查看 student 表所有列的列名与列数据类型。

```
DESCRIBE student;
```

返回结果示例：

```
+---------+----------+-------+
|col_name |data_type |comment|
+---------+----------+-------+
|id       |int       |NULL   |
|name     |string    |NULL   |
|age      |int       |NULL   |
|score    |double    |NULL   |
+---------+----------+-------+
```

示例 2：使用 DESC 简写

使用 DESC 简写形式查看 student 表的列信息，与 DESCRIBE 等效。

```
DESC student;
```


示例 3：使用 EXTENDED 查看详细信息

查看 student 表的详细元数据信息，包括分区、存储格式、表属性等。

```
DESCRIBE EXTENDED student;
```

返回结果将包含：

- 基本列信息（列名、数据类型、注释）
- 表的详细元数据（数据库、所有者、创建时间、存储格式、表属性等）

示例 4：查看指定数据库的表

查看 school_db 数据库下 student 表的列信息。

```
DESCRIBE school_db.student;
```

3.5 修改表

3.5.1 添加列

功能描述

添加一个或多个新列到表中。

语法格式

```
ALTER TABLE [db_name.]table_name ADD COLUMNS (col_name1 col_type1 [COMMENT
col_comment1], ...);
```

关键字

- **ADD COLUMNS**：添加列。
- **COMMENT**：列描述。

参数说明

参数	是否必选	描述
db_name	否	Database名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。
table_name	是	表名称。
col_name	是	列字段名称。
col_type	是	列字段类型。
col_comment	否	列描述。仅支持字符串常量。

注意事项

该SQL不建议并发执行，并发情况下添加列结果可能互相覆盖。

示例 1：添加单列

为表t1添加一个INT类型的列age。

```
ALTER TABLE t1 ADD COLUMNS (age INT);
```

示例 2：添加多列

为表t1同时添加column2和column3两个列。

```
ALTER TABLE t1 ADD COLUMNS (column2 INT, column3 STRING);
```

示例 3：添加带注释的列

为表t1添加一个STRING类型的列address，并附带列注释。

```
ALTER TABLE t1 ADD COLUMNS (address STRING COMMENT '用户地址信息');
```

示例 4：指定数据库的表添加列

为数据库mydb中的表employee添加列salary。

```
ALTER TABLE mydb.employee ADD COLUMNS (salary DECIMAL(10,2) COMMENT '员工薪资');
```

3.5.2 修改列注释

功能描述

修改非分区表或分区表的列注释信息。Spark 4.0推荐使用ALTER COLUMN语法，同时兼容CHANGE COLUMN语法。

语法格式

- **语法一（推荐，Spark 4.0）：**
  ```
  ALTER TABLE [db_name.]table_name ALTER COLUMN col_name COMMENT 'col_comment';
  ```
- **语法二（兼容）：**
  ```
  ALTER TABLE [db_name.]table_name CHANGE COLUMN col_name col_name col_type COMMENT 'col_comment';
  ```

关键字

- **ALTER COLUMN**：修改列属性（Spark 4.0推荐语法）。
- **CHANGE COLUMN**：修改列（兼容语法，需同时指定列名和类型）。
- **COMMENT**：列描述。

参数说明

参数	是否必选	描述
db_name	否	Database名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。
table_name	是	表名称。

参数	是否必选	描述
col_name	是	列字段名称。必须是已存在的列。
col_type	仅CHANGE COLUMN语法必选	列数据类型。CHANGE COLUMN语法不支持修改列数据类型，此处指定的是创建表时的列数据类型。ALTER COLUMN语法无需指定。
col_comment	是	修改后的列注释信息。注释内容为长度不超过1024字节的有效字符串。

注意事项

- Spark 4.0推荐使用ALTER COLUMN语法修改列注释，该语法更简洁，无需重复指定列数据类型。
- CHANGE COLUMN语法为兼容语法，使用时必须指定与建表时一致的列数据类型。

示例 1：使用 ALTER COLUMN 修改注释（推荐）

使用Spark 4.0推荐的ALTER COLUMN语法修改表t1中c1列的注释。

```
ALTER TABLE t1 ALTER COLUMN c1 COMMENT 'the new comment';
```

示例 2：使用 CHANGE COLUMN 修改注释（兼容）

使用兼容语法修改表t1中c1列的注释，需同时指定列名和数据类型。

```
ALTER TABLE t1 CHANGE COLUMN c1 c1 STRING COMMENT 'the new comment';
```

示例 3：指定数据库修改列注释

修改数据库mydb中表employee的name列注释。

```
ALTER TABLE mydb.employee ALTER COLUMN name COMMENT '员工姓名';
```

示例 4：修改多列注释

分别修改表t1中多个列的注释。

```
ALTER TABLE t1 ALTER COLUMN c1 COMMENT '第一列说明';
ALTER TABLE t1 ALTER COLUMN c2 COMMENT '第二列说明';
```

3.6 表别名

功能描述

给表或者子查询结果起别名。

语法格式

```
SELECT attr_expr_list FROM table_reference [AS] alias;
```


关键字

- **table_reference**：可以是表、视图或者子查询。
- **AS**：可用于连接table_reference和alias，是否添加此关键字不会影响命令执行结果。

注意事项

- 所要查询的表必须是已经存在的，否则会出错。
- 别名的命名必须在别名的使用之前，否则会出错。此外，建议不要重名。

示例

- 给表simple_table起为n的别名，并利用n.name访问simple_table中的name字段。
  ```
  SELECT n.score FROM simple_table n WHERE n.name = "leilei";
  ```
- 将子查询的结果命名为m，并利用SELECT * FROM m返回子查询中的所有结果。
  ```
  SELECT * FROM (SELECT * FROM simple_table WHERE score > 90) AS m;
  ```

3.7 分区相关

3.7.1 添加分区

功能描述

为已存在的OBS分区表添加一个或多个分区。该命令将分区元数据信息注册到元数据库中，使后续查询可以按分区列进行数据检索。

OBS分区表生成分区信息主要有以下两种场景：

- 向OBS分区表插入对应的分区数据，数据插入成功后自动生成分区元数据信息。
- 手动拷贝分区目录和数据到OBS分区表路径下，再执行本命令添加分区元数据信息。

语法格式

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION partition_specs1 [LOCATION 'location_path1']
  PARTITION partition_specs2 [LOCATION 'location_path2']
  ...;
```

参数说明

参数	描述
table_name	表名称。
IF NOT EXISTS	可选关键字。指定该关键字后，若分区已存在则不会报错，静默跳过。
partition_specs	分区规格，格式为 (col1=val1, col2=val2, ...)。若表有多个分区列，添加分区时需指定所有分区列的值，顺序可任意。

参数	描述
location_path	可选。分区的存储路径。若指定，该路径必须已存在，否则会报错。

注意事项

- 向表中添加分区时，此表和分区列（建表时 PARTITIONED BY 指定的列）必须已存在，而所要添加的分区不能重复添加，否则将出错。已添加的分区可通过 IF NOT EXISTS 避免报错。
- 若分区表是按照多个字段进行分区的，添加分区时需要指定所有的分区字段，指定字段的顺序可任意。
- partition_specs 中的参数默认带有 ( )，例如：PARTITION (dt='2009-09-09', city='xxx')。
- 在添加分区时若指定路径，则该路径必须是已经存在的，否则会出错。
- 若添加多个分区，每组 PARTITION partition_specs LOCATION 'location_path' 之间用空格隔开。
- 若新增分区指定的路径包含子目录（或嵌套子目录），则子目录下面的所有文件类型及内容也将作为该分区的记录。需要保证该分区目录下所有文件类型和文件内容与表的字段一致，否则查询将报错。

示例 1：添加单个分区

为表 student 添加一个分区，分区列为 facultyNo=10, classNo=101。

```
ALTER TABLE student ADD
  PARTITION (facultyNo=10, classNo=101);
```

示例 2：添加多个分区

为表 student 同时添加两个分区。

```
ALTER TABLE student ADD
  PARTITION (facultyNo=10, classNo=101)
  PARTITION (facultyNo=20, classNo=102);
```

示例 3：使用 IF NOT EXISTS 添加分区

添加分区时使用 IF NOT EXISTS 避免分区已存在时报错。

```
ALTER TABLE student ADD IF NOT EXISTS
  PARTITION (facultyNo=10, classNo=101);
```

示例 4：添加分区并指定 LOCATION

为OBS表添加分区，并指定分区数据的存储路径。

```
ALTER TABLE testobstable ADD
  PARTITION (external_data='22')
  LOCATION 'obs://bucketName/datapath/external_data=22';
```


3.7.2 重命名分区

功能描述

修改OBS分区表中某个分区的分区值。该命令仅支持操作OBS表，不支持对管理表进行操作。

语法格式

```
ALTER TABLE table_name
  PARTITION partition_specs
  RENAME TO PARTITION partition_specs;
```

参数说明

参数	描述
table_name	表名称。
partition_specs	分区规格，格式为 (col1=val1, col2=val2, ...)。RENAME TO 前为原分区值，后为新分区值。若分区表有多个分区列，需指定所有分区列的值，顺序可任意。

注意事项

- 该命令仅支持操作OBS表，不支持对管理表进行操作。
- 所要重命名分区的表和分区必须已存在，否则会出错。新分区名不能与其他分区重名，否则将出错。
- 若分区表是按照多个字段进行分区的，重命名分区时需要指定所有的分区字段，指定字段的顺序可任意。
- partition_specs 中的参数默认带有 ( )，例如：PARTITION (dt='2009-09-09', city='xxx')。

示例 1：单分区列重命名

将表 log_data 中分区 year=2023 重命名为 year=2024。

```
ALTER TABLE log_data
  PARTITION (year='2023')
  RENAME TO PARTITION (year='2024');
```

示例 2：多分区列重命名

将表 student 中分区 city='city1', dt='2023-01-01' 重命名为 city='city2', dt='2023-06-01'。

```
ALTER TABLE student
  PARTITION (city='city1', dt='2023-01-01')
  RENAME TO PARTITION (city='city2', dt='2023-06-01');
```

示例 3：日期分区重命名

将表 sales 中日期分区 dt='2023-12-31' 重命名为 dt='2024-01-01'。


```
ALTER TABLE sales
  PARTITION (dt='2023-12-31')
  RENAME TO PARTITION (dt='2024-01-01');
```

示例 4：分区值变更（仅修改部分分区列值）

将表 event_log 中分区 region='east', type='click' 重命名为 region='west', type='click'，仅修改 region 列值。

```
ALTER TABLE event_log
  PARTITION (region='east', type='click')
  RENAME TO PARTITION (region='west', type='click');
```

3.7.3 删除分区

功能描述

删除分区表的一个或多个分区。管理表和OBS表均支持使用本命令删除分区。对于OBS表，还可以使用指定筛选条件删除分区按条件批量删除。

语法格式

```
ALTER TABLE [db_name.]table_name
  DROP [IF EXISTS]
  PARTITION partition_spec1 [, PARTITION partition_spec2, ...];
```

参数说明

参数	描述
db_name	可选。数据库名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。
table_name	表名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（''）包围起来。
IF EXISTS	可选关键字。指定该关键字后，若分区不存在则不会报错，静默跳过。
partition_spec	分区规格，格式为 (col1=val1, col2=val2, ...)。若分区字段为多个字段，可以不包含所有的字段，会删除匹配上的所有分区。

注意事项

- 所要删除分区的表必须是已经存在的表，否则会出错。
- 所要删除的分区必须是已经存在的，否则会出错，可通过语句中添加 IF EXISTS 避免该错误。
- 对于OBS表，删除分区仅删除元数据，OBS目录中的数据文件不会自动删除。

示例 1：删除指定分区

删除表 student 中 facultyNo=20, classNo=103 的分区。


```
ALTER TABLE student DROP
  PARTITION (facultyNo=20, classNo=103);
```

示例 2：使用 IF EXISTS 删除分区

使用IF EXISTS避免分区不存在时报错。

```
ALTER TABLE student DROP IF EXISTS   PARTITION (facultyNo=99, classNo=999);
```

示例 3：同时删除多个分区

在一条语句中删除多个分区。

```
ALTER TABLE student DROP IF EXISTS
  PARTITION (facultyNo=99, classNo=999);
```

示例 4：指定数据库的表删除分区

删除数据库 academy 中表 student 的分区。

```
ALTER TABLE academy.student DROP IF EXISTS
  PARTITION (facultyNo=20, classNo=103);
```

3.7.4 修改表分区位置

功能描述

修改OBS表分区的存储位置。该命令仅支持OBS表（外部表），不支持管理表。

语法格式

```
ALTER TABLE table_name
  PARTITION partition_specs
  SET LOCATION 'obs_path';
```

关键字

- **PARTITION**：指定要修改位置的分区。
- **SET LOCATION**：设置分区的新存储路径。
- **LOCATION**：分区路径，必须为OBS格式。

参数说明

参数	是否必选	描述
table_name	是	表名称。
partition_specs	是	分区规格，格式为(col1=val1, col2=val2, ...)。分区必须已存在。
obs_path	是	新的OBS存储路径，必须为已存在的绝对路径。

注意事项

- 该命令仅支持OBS表（外部表），不支持管理表。
- 所要修改位置的表分区必须是已经存在的，否则将报错。
- 指定的新OBS路径必须是已经存在的绝对路径，否则将报错。
- 若新路径包含子目录，则子目录下面的所有文件类型及内容也将作为该分区的记录，需保证与表字段一致。
- 修改分区位置不会自动迁移已有数据。

示例 1：修改单分区列的分区位置

```
ALTER TABLE student
  PARTITION (dt='2008-08-08')
  SET LOCATION 'obs://my-bucket/student/dt=2008-08-08/';
```

示例 2：修改多分区列的分区位置

```
ALTER TABLE student
  PARTITION (dt='2008-08-08', city='beijing')
  SET LOCATION 'obs://my-bucket/student/dt=2008-08-08/city=beijing/';
```

示例 3：修改后验证分区位置

```
ALTER TABLE student PARTITION (dt='2008-08-08') SET LOCATION 'obs://my-bucket/student/new_path/';

SHOW PARTITIONS student;
```

示例 4：指定数据库的表修改分区位置

```
ALTER TABLE myschool.student
  PARTITION (class='101')
  SET LOCATION 'obs://my-bucket/myschool/student/class=101/';
```

3.7.5 更新表分区信息

功能描述

恢复（恢复注册）分区表中所有尚未在元数据库中注册的分区信息。典型应用场景：当手动在OBS上添加分区目录后，通过本命令将新增的分区信息刷新到元数据库中，之后即可通过 SHOW PARTITIONS 查看到新增的分区。

Spark SQL提供两种等价语法：MSCK REPAIR TABLE 和 ALTER TABLE ... RECOVER PARTITIONS。

语法格式

- 方式一：MSCK REPAIR TABLE
  ```
  MSCK REPAIR TABLE table_name;
  ```
- 方式二：RECOVER PARTITIONS
  ```
  ALTER TABLE table_name RECOVER PARTITIONS;
  ```

参数说明

参数	描述
table_name	表名称。分区表的名称，该表必须已存在。

注意事项

- 分区目录名称必须按照指定的格式输入，即 tablepath/partition_column_name=partition_column_value。
- 两条语法功能等价，可任选其一使用。
- 本命令主要用于OBS分区表场景，手动在OBS上添加分区目录后刷新元数据。

示例 1：使用 MSCK REPAIR TABLE 更新分区信息

更新表 ptable 在元数据库中的分区信息。

MSCK REPAIR TABLE ptable;

示例 2：使用 RECOVER PARTITIONS 更新分区信息

使用等价语法更新表 ptable 的分区信息。

ALTER TABLE ptable RECOVER PARTITIONS;

示例 3：添加 OBS 目录后修复分区

假设已手动在OBS上创建了分区目录 obs://bucketName/datapath/dt=2024-01-01，执行以下命令将新分区注册到元数据中。

MSCK REPAIR TABLE sales_data;

示例 4：修复分区后查看分区列表

修复分区信息后，通过 SHOW PARTITIONS 验证新分区已注册。

```
-- 修复分区
ALTER TABLE sales_data RECOVER PARTITIONS;

-- 查看所有分区
SHOW PARTITIONS sales_data;
```


4 数据操作语法

4.1 导入数据

功能描述

LOAD DATA可用于导入CSV、Parquet、ORC、JSON、Avro格式的数据，内部将转换成Parquet数据格式进行存储。

使用限制

datasource表和iceberg表不支持LOAD DATA操作。

语法格式

```
LOAD DATA INPATH 'folder_path' INTO TABLE [db_name.]table_name
  OPTIONS(property_name=property_value, ...);
```

关键字

- INPATH：数据路径。
- OPTIONS：属性列表。

参数说明

参数	是否必选	描述
folder_path	是	原始数据文件夹或者文件的OBS路径。
db_name	否	数据库名称。若未指定，则使用当前数据库。

参数	是否必选	描述
table_name	是	需要导入数据的管理表的名称。

以下是可以在导入数据时使用的配置选项：

- DATA_TYPE: 指定导入的数据类型，当前支持CSV、Parquet、ORC、JSON、Avro类型，默认值为“CSV”。

  配置项为OPTIONS('DATA_TYPE'='CSV')

  导入CSV和JSON文件时，有三种模式可以选择：

  - PERMISSIVE：选择PERMISSIVE模式时，如果某一列数据类型与目标表列数据类型不匹配，则该行数据将被设置为null。
  - DROPMALFORMED：选择DROPMALFORMED模式时，如果某一列数据类型与目标表列数据类型不匹配，则不导入该行数据。
  - FAILFAST：选择FAILFAST模式时，如果某一列类型不匹配，则会抛出异常，导入失败。

  模式设置可通过在OPTIONS中添加 OPTIONS('MODE'='PERMISSIVE')进行设置。
- DELIMITER：可以在导入命令中指定分隔符，默认值为“,”。

  配置项为OPTIONS('DELIMITER'=',')。

  对于CSV数据，支持如下所述分隔符：

  - 制表符，例如：'DELIMITER'='\t'。
  - 任意的二进制字符，例如：'DELIMITER'='\u0001(^A)'。
  - 单引号（'），单引号必须在双引号（" "）内。例如：'DELIMITER'= "'"。
  - 管理表还支持\001（^A）和\017（^Q），例如：'DELIMITER'='\001(^A)'，'DELIMITER'='\017(^Q)'。
- QUOTECHAR：可以在导入命令中指定引号字符。默认值为**'**。

  配置项为OPTIONS('QUOTECHAR'='"')
- COMMENTCHAR：可以在导入命令中指定注释字符。在导入操作期间，如果在行的开头遇到注释字符，那么该行将被视为注释，并且不会被导入。默认值为#。

  配置项为_OPTIONS('COMMENTCHAR'='#')_
- HEADER：用来表示源文件是否有表头。取值范围为“true”和“false”。“true”表示有表头，“false”表示无表头。默认值为“false”。如果没有表头，可以在导入命令中指定FILEHEADER参数提供表头。

  配置项为_OPTIONS('HEADER'='true')_
- FILEHEADER：如果源文件中没有表头，可在LOAD DATA命令中提供表头。

  OPTIONS('FILEHEADER'='column1,column2')
- ESCAPECHAR：如果用户想在CSV上对Escape字符进行严格验证，可以提供Escape字符。默认值为“\\”。

  配置项为OPTIONS('ESCAPECHAR'='\\')

  说明

  如果在CSV数据中输入ESCAPECHAR，该ESCAPECHAR必须在双引号（" "）内。例如："a\b"。

- MAXCOLUMNS：该可选参数指定了在一行中，CSV解析器解析的最大列数。
  配置项为_OPTIONS('MAXCOLUMNS'='400')_

  表 4-1 MAXCOLUMNS 参数

可选参数名称	默认值	最大值
MAXCOLUMNS	2000	20000

  说明

  设置MAXCOLUMNS Option的值后，导入数据会对executor的内存有要求，所以导入数据可能会由于executor内存不足而失败。

- DATEFORMAT：指定列的日期格式。
  OPTIONS('DATEFORMAT'='dateFormat')

  说明

  - 默认值为：yyyy-MM-dd。
  - 日期格式由Java的日期模式字符串指定。在Java的日期和时间模式字符串中，未加单引号(')的字符'A' 到'Z' 和'a' 到'z' 被解释为模式字符，用来表示日期或时间字符串元素。若模式字符使用单引号 (') 引起来，则在解析时只进行文本匹配，而不进行解析。Java模式字符定义请参见**表4-2**。

  表 4-2 日期及时间模式字符定义

模式字符	日期或时间元素	示例
G	纪元标识符	AD
y	年份	1996; 96
M	月份	July; Jul; 07
w	年中的周数	27(该年的第27周)
W	月中的周数	2(该月的第2周)
D	年中的天数	189(该年的第189天)
d	月中的天数	10(该月的第10天)
u	星期中的天数	1 = 星期一, ..., 7 = 星期日
a	am/pm 标记	pm(下午时)
H	24小时数(0-23)	2
h	12小时数(1-12)	12
m	分钟数	30
s	秒数	55

模式字符	日期或时间元素	示例
S	毫秒数	978
z	时区	Pacific Standard Time; PST; GMT-08:00

- TIMESTAMPFORMAT：指定列的时间戳格式。

  OPTIONS('TIMESTAMPFORMAT'='timestampFormat')

  说明

  - 默认值为：yyyy-MM-dd HH:mm:ss。
  - 时间戳格式由Java的时间模式字符串指定。Java时间模式字符串定义详见**表4-2**。

- MODE：指定导入过程错误记录的处理模式，支持三种选项：PERMISSIVE、DROPMALFORMED和FAILFAST。

  OPTIONS('MODE'='permissive')

  说明

  - PERMISSIVE（默认）：尽可能地解析bad records，如果遇到不能转换的字段，则整行为null
  - DROPMALFORMED：忽略掉无法解析的bad records
  - FAILFAST：遇到无法解析的记录时，抛出异常并使Job失败

- BADRECORDSPATH：指定导入过程中错误记录的存储目录。

  OPTIONS('BADRECORDSPATH'='obs://bucket/path')

  说明

  配置该选项后，MODE不可配，固定为"DROPMALFORMED"，即将能够成功转换的记录导入到目标表，而将转换失败的记录存储到指定错误记录存储目录。

注意事项

- 导入OBS表时，创建OBS表时指定的路径必须是文件夹，若建表路径是文件将导致导入数据失败。
- 仅支持导入位于OBS路径上的原始数据。
- 不建议对同一张表并发导入数据，因为有一定概率发生并发冲突，导致导入失败。
- 导入数据时只能指定一个路径，路径中不能包含逗号。
- 当OBS桶目录下有文件夹和文件同名时，导入数据会优先指向该路径下的文件而非文件夹。
- 导入PARQUET、ORC及JSON类型数据时，必须指定_DATA_TYPE_这一OPTIONS，否则会以默认的“CSV”格式进行解析，从而导致导入的数据格式不正确。
- 导入CSV及JSON类型数据时，如果包含日期及时间列，需要指定_DATEFORMAT_及_TIMESTAMPFORMAT_选项，否则将以默认的日期及时间戳格式进行解析。

示例

- 可使用下列语句将CSV文件导入到管理表，“t”为表名。

```
LOAD DATA INPATH 'obs://aidatalake/data.csv' INTO TABLE t
  OPTIONS('DELIMITER'=',' , 'QUOTECHAR'='"','COMMENTCHAR'='#','HEADER'='false');
```

- 可使用下列语句将JSON文件导入到管理表，“jsontb”为表名。

```
LOAD DATA INPATH 'obs://aidatalake/alltype.json' into table jsontb
  OPTIONS('DATA_TYPE'='json','DATEFORMAT'='yyyy/MM/dd','TIMESTAMPFORMAT'='yyyy/MM/dd
HH:mm:ss');
```

4.2 插入数据

功能描述

将SELECT查询结果或某条数据插入到表中。

约束限制

- insert overwrite语法不适用于同一张表内部的“自读自写”场景（包括分区表和非分区表），直接在原表上执行insert overwrite可能会导致数据丢失或数据不一致的风险。

  如果要实现“自读自写”场景数据操作，建议使用一个临时表来处理数据。

  "自读自写"即目的表和数据源表都是同一张表。例如，假设将student表中class_no = 1的学生信息提取出来并覆盖原表，以下语句即为自读自写场景典型语句。

```
INSERT OVERWRITE TABLE student SELECT name FROM student WHERE class_no = 1;
```

- 使用Hive和Datasource（除Hudi外）表在执行数据修改类命令（例如insert into，load data）时由于数据源不支持事务性，在系统故障或队列资源重启后，可能会导致数据重复或数据不一致等问题。

  为了避免这种情况，建议优先选择支持事务性的数据源，如Hudi类型数据源，该类数据源具备ACID（Atomicity、Consistency、Isolation、Durability）能力，有助于确保数据的一致性和准确性。

语法格式

- 将SELECT查询结果插入到表中

```
INSERT INTO [TABLE] [db_name.]table_name
  [PARTITION part_spec] select_statement;
INSERT OVERWRITE TABLE [db_name.]table_name
  [PARTITION part_spec] select_statement;
part_spec:
   : (part_col_name1=val1 [, part_col_name2=val2, ...])
```

- 将某条数据插入到表中

```
INSERT INTO [TABLE] [db_name.]table_name
  [PARTITION part_spec] VALUES values_row [, values_row ...];
INSERT OVERWRITE TABLE [db_name.]table_name
  [PARTITION part_spec] VALUES values_row [, values_row ...];
values_row:
  : (val1 [, val2, ...])
```


关键字

表 4-3 INSERT 关键字说明

参数	描述
db_name	需要执行INSERT命令的表所在数据库的名称。
table_name	需要执行INSERT命令的表的名称。
part_spec	指定详细的分区信息。若分区字段为多个字段，需要包含所有的字段，但是可以不包含对应的值，系统会匹配上对应的分区。单表分区数最多允许100000个。
select_statement	源表上的SELECT查询（支持管理表、OBS表）。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 如果动态分区不需要指定分区，则将“part_spec”作为普通字段放置SELECT语句中。
- 被插入的OBS表在建表时只能指定文件夹路径。
- 源表和目标表的数据类型和列字段个数应该相同，否则插入失败。
- 不建议对同一张表并发插入数据，可能会由于并发冲突导致插入数据结果异常。
- INSERT INTO命令用于将查询的结果追加到目标表中。
- INSERT OVERWRITE命令用于覆盖目标表中已有的数据。
- INSERT INTO命令可以并行执行，INSERT OVERWRITE命令只有在分区表下不同的插入到不同静态分区才可以并行。
- INSERT INTO命令和INSERT OVERWRITE命令同时执行，其结果是未知的。
- 在从源表插入数据到目标表的过程中，无法在源表中导入或更新数据。
- 对于Hive分区表的动态INSERT OVERWRITE，支持覆盖涉及到的分区数据，不支持覆盖整表数据。
- 如果需要覆盖DataSource表指定分区数据，需要先配置参数：spark.sql.dynamicPartitionOverwrite.enabled=true，再通过”insert overwrite”语句实现，”spark.sql.dynamicPartitionOverwrite.enabled”默认值为”false”，表示覆盖整表数据。例如：
  ```
  insert overwrite table tb1 partition(part1='v1', part2='v2') select * from ...
  ```
- 通过配置”spark.sql.shuffle.partitions”参数可以设置非管理表在OBS桶中插入的文件个数，同时，为了避免数据倾斜，在INSERT语句后可加上”distribute by rand()”，可以增加处理作业的并发量。例如：
  ```
  insert into table table_target select * from table_source distribute by cast(rand() * N as int);
  ```


示例

说明

导入数据前已参考创建OBS表或者创建管理表中的示例描述创建对应的表。

- 示例1：将SELECT查询结果插入到表中
  - 使用DataSource语法创建一个parquet格式的分区表
    ```
    CREATE TABLE data_source_tab1 (col1 INT, p1 INT, p2 INT)
      USING PARQUET PARTITIONED BY (p1, p2);
    ```
  - 插入查询结果到分区 (p1 = 3, p2 = 4)中
    ```
    INSERT INTO data_source_tab1 PARTITION (p1 = 3, p2 = 4)
      SELECT id FROM RANGE(1, 3);
    ```
  - 插入新的查询结果到分区 (p1 = 3, p2 = 4) 中
    ```
    INSERT OVERWRITE TABLE data_source_tab1 PARTITION (p1 = 3, p2 = 4)
      SELECT id FROM RANGE(3, 5);
    ```
- 示例2：将某条数据插入表中
  - 使用Hive语法创建一个parquet格式的分区表
    ```
    CREATE TABLE hive_serde_tab1 (col1 INT, p1 INT, p2 INT)
      USING HIVE OPTIONS(fileFormat 'PARQUET') PARTITIONED BY (p1, p2);
    ```
  - 插入两条数据到分区 (p1 = 3, p2 = 4)中
    ```
    INSERT INTO hive_serde_tab1 PARTITION (p1 = 3, p2 = 4)
      VALUES (1), (2);
    ```
  - 插入新的数据到分区 (p1 = 3, p2 = 4) 中
    ```
    INSERT OVERWRITE TABLE hive_serde_tab1 PARTITION (p1 = 3, p2 = 4)
      VALUES (3), (4);
    ```

执行 Insert into 后数据重复怎么办?

- 问题现象

使用Hive和Datasource（除Hudi外）表在执行数据修改类命令（例如insert into，load data）时由于数据源不支持事务性，在系统故障或队列资源重启后，可能会导致数据重复或数据不一致等问题。

- 原因分析

在数据的Commit阶段如果出现队列资源重启可能会导致数据已经被修复到正式目录中。如果执行的是Insert into语句，资源重启后触发重试就会有概率导致数据重复写入。

- 解决方案

1. 推荐使用具备ACID能力的Hudi类型数据源。
2. 建议尽量使用insert overwrite这样幂等的语法而不是insert into等非幂等语法插入数据。
3. 如果严格需求数据不能重复，建议在insert into后对表数据执行去重操作，防止数据重复。

4.3 复用子查询

功能描述

复用子查询（Common Table Expression，CTE）通过 WITH ... AS 语句定义命名的临时结果集，在查询的不同部分重复使用，避免重复计算相同的子查询，从而提高查询性能和可读性。

语法格式

单个子查询：

```
WITH cte_name AS (
    SELECT ...
    FROM table_name
    WHERE ...
)
SELECT ...
FROM cte_name
WHERE ...;
```

多个子查询：

```
WITH cte_name1 AS (
    SELECT ...
    FROM table_name
    WHERE ...
),
cte_name2 AS (
    SELECT ...
    FROM table_name
    WHERE ...
)
SELECT ...
FROM cte_name1, cte_name2
WHERE ...;
```

递归CTE：

```
WITH RECURSIVE cte_name AS (
    SELECT ... FROM ...          -- 基础查询
    UNION ALL
    SELECT ... FROM cte_name ... -- 递归查询
)
SELECT ... FROM cte_name;
```

参数说明

参数	说明
cte_name	用户自定义的子查询名称，在同一 WITH 语句中必须唯一。
table_name	执行子查询命令的表的名称。必须是已存在的表。
WITH	CTE 定义关键字，后跟一个或多个命名的子查询。
RECURSIVE	递归 CTE 关键字，允许 CTE 引用自身实现递归查询。
AS	为子查询指定名称的关键字。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- 多个 CTE 之间使用逗号分隔。
- CTE 定义必须在主查询之前。
- CTE 只在其所在的 SELECT 语句中有效，不能跨查询使用。
- 递归 CTE 必须包含锚点查询和递归查询两部分，用 UNION ALL 连接。

示例

- 示例 1：基于表 sales 定义子查询 sales_data，查询销售金额大于 100 的商品，最后查询该子查询的所有数据。

```
WITH sales_data AS (
  SELECT product_name, sales_amount
  FROM sales
  WHERE sales_amount > 100
)
SELECT * FROM sales_data;
```

- 示例 2：基于表 sales 定义子查询 sales_data1 和 sales_data2，其中 sales_data1 查询销售金额大于 100 的商品，sales_data2 查询销售金额小于 20 的商品，最后合并两个子查询的数据。

```
WITH sales_data1 AS (
   SELECT product_name, sales_amount
   FROM sales
   WHERE sales_amount > 100
),
sales_data2 AS (
   SELECT product_name, sales_amount
   FROM sales
   WHERE sales_amount < 20
)
SELECT * FROM sales_data1
UNION ALL
SELECT * FROM sales_data2;
```

- 示例 3：递归CTE生成1到10的序列。

```
WITH RECURSIVE nums AS (
   SELECT 1 AS n
   UNION ALL
   SELECT n + 1 FROM nums WHERE n < 10
)
SELECT * FROM nums;
```

4.4 清空数据

功能描述

清除管理表或者OBS表的数据。

语法格式

```
TRUNCATE TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)];
```

关键字

表 4-4 关键字说明

参数	描述
tablename	需要执行 Truncate 命令的管理表或者 OBS 表的名称。
partcol1	需要删除的管理表或者 OBS 表的分区名称。

注意事项

只支持清除管理表或者OBS表的数据。

示例

```
TRUNCATE TABLE test PARTITION (class = 'test');
```


5 导出查询结果

功能描述

INSERT OVERWRITE DIRECTORY用于将查询结果直接写入到指定的目录，支持按CSV、Parquet、ORC、JSON、Avro格式进行存储。

语法格式

```
INSERT OVERWRITE DIRECTORY path
  USING file_format
  [OPTIONS(key1=value1)]
  select_statement;
```

关键字

- USING：指定所存储格式。
- OPTIONS：导出时的属性列表，为可选项。

参数

表 5-1 INSERT OVERWRITE DIRECTORY 参数描述

参数	描述
path	要将查询结果写入的OBS路径。
file_format	写入的文件格式，支持按CSV、Parquet、ORC、JSON、Avro格式。

注意事项

- 通过配置“spark.sql.shuffle.partitions”参数可以设置非管理表在OBS桶中插入的文件个数，同时，为了避免数据倾斜，在INSERT语句后可加上“distribute by rand()”，可以增加处理作业的并发量。例如：
  ```
  insert into table table_target select * from table_source distribute by cast(rand() * N as int);
  ```
- 配置项为OPTIONS('DELIMITER'=',')时，可以指定分隔符，默认值为“,”。

  对于CSV数据，支持如下所述分隔符：

- 制表符tab，例如：'DELIMITER'='\t'。
- 支持通过unicode编码指定分隔符，例如：'DELIMITER'='\u0001'。
- 单引号（'），单引号必须在双引号（" "）内。例如：'DELIMITER'= "'"。

示例

```
INSERT OVERWRITE DIRECTORY 'obs://bucket/dir'
  USING csv
  OPTIONS(key1=value1)
  select * from db1.tb1;
```


6 视图

6.1 创建视图

功能描述

创建视图。

语法格式

```
CREATE [OR REPLACE] VIEW view_name
  [WITH SCHEMA { BINDING | COMPENSATION | EVOLUTION | TYPE EVOLUTION }]
  AS select_statement;
```

关键字

- CREATE VIEW：基于给定的select语句创建视图，不会将select语句的结果写入磁盘。
- OR REPLACE：指定该关键字后，若视图已经存在将不报错，并根据select语句更新视图的定义。
- WITH SCHEMA：指定视图的Schema管理策略。
- BINDING：严格绑定，底层表结构变化时视图失效。
- COMPENSATION：补偿模式（默认），容忍列类型变化。
- EVOLUTION：演进模式，自动跟随底层表Schema变化。
- TYPE EVOLUTION：类型演进，仅限列类型演进。

注意事项

- 所要创建的视图必须是当前数据库下不存在的，否则会报错。当视图存在时，可通过增加OR REPLACE关键字来避免报错。

- 视图中包含的表或视图信息不可被更改，如有更改可能会造成查询失败。
- 如果创建表和创建视图使用的计算引擎不一致，可能会因为varchar类型不兼容，导致视图查询失败。

示例

先通过对student表中的id和name数据进行查询，并以该查询结果创建视图student_view。

```
CREATE VIEW student_view AS SELECT id, name FROM student;
```

创建带Schema演进策略的视图：

```
CREATE VIEW student_view WITH SCHEMA EVOLUTION AS SELECT id, name FROM student;
```

6.2 删除视图

功能描述

删除视图。

语法格式

```
DROP VIEW [IF EXISTS] [db_name.]view_name;
```

关键字

DROP：删除指定视图的元数据。虽然视图和表有很多共同之处，但是DROP TABLE不能用来删除VIEW。

注意事项

所要删除的视图必须是已经存在的，否则会出错，可以通过IF EXISTS来避免该错误。

示例

删除名为student_view的视图。

```
DROP VIEW student_view;
```

6.3 修改视图

功能描述

ALTER VIEW语句用于修改视图的元数据，包括重命名视图、设置/删除视图属性。

语法

- 重命名视图
  ```
  ALTER VIEW view_identifier RENAME TO view_identifier
  ```
- 设置视图属性
  ```
  ALTER VIEW view_identifier SET TBLPROPERTIES ( property_key = property_val [, ...] )
  ```


- 删除视图属性
  ```
  ALTER VIEW view_identifier UNSET TBLPROPERTIES [ IF EXISTS ] ( property_key [, ...] )
  ```

参数说明

参数	描述
view_identifier	视图名称，可使用 database_name.view_name 限定。
RENAME TO	重命名视图。源数据库和目标数据库必须相同，不支持跨数据库移动视图。
SET TBLPROPERTIES	设置视图属性。如果属性键已存在，则替换值；如果不存在，则添加。
UNSET TBLPROPERTIES	删除视图属性。如果指定 IF EXISTS，属性不存在时不报错。
property_key	属性键名，可由多个以点分隔的部分组成，如 key_part1.key_part2。

示例

重命名视图

```
ALTER VIEW tempdb1.v1 RENAME TO tempdb1.v2;
```

设置视图属性

```
ALTER VIEW tempdb1.v2 SET TBLPROPERTIES ('created.by.user' = 'John', 'created.date' = '01-01-2001');
```

删除视图属性

```
-- 删除已有属性
ALTER VIEW tempdb1.v2 UNSET TBLPROPERTIES ('created.by.user', 'created.date');
-- 使用IF EXISTS避免属性不存在时报错
ALTER VIEW tempdb1.v2 UNSET TBLPROPERTIES IF EXISTS ('non-existent');
```

注意事项

- ALTER VIEW仅修改视图的元数据，不会影响底层数据。
- 重命名视图时，如果新视图名已存在，会抛出 TableAlreadyExistsException。
- 重命名视图会清除该视图及其依赖项的缓存数据。
- ALTER VIEW不支持 SET SERDE 或 SET SERDEPROPERTIES。
- WITH SCHEMA不支持TEMPORARY视图。

6.4 查看视图

功能描述

SHOW VIEWS语句用于列出指定数据库中的所有视图。可以通过模式匹配过滤视图名称。如果不指定数据库，则列出当前数据库的视图。如果指定数据库为全局临时视图数据库，则列出全局临时视图。注意：无论指定哪个数据库，本地临时视图始终会被列出。

语法

```
SHOW VIEWS [ { FROM | IN } database_name ] [ LIKE regex_pattern ]
```

参数说明

参数	描述
FROM / IN database_name	可选。指定要查看视图的数据库名称。如果省略，则使用当前数据库。
LIKE regex_pattern	可选。使用正则表达式过滤视图名称。* 匹配0个或多个字符，`

示例

列出当前数据库的所有视图

```
SHOW VIEWS;
```

结果：

namespace	viewName	isTemporary
default	sam	false
default	sam1	false
default	suj	false
default	temp2	true

列出指定数据库的视图

```
SHOW VIEWS FROM userdb;
```

结果：

namespace	viewName	isTemporary
userdb	user1	false
userdb	user2	false
userdb	temp2	true

列出全局临时视图

```
SHOW VIEWS IN global_temp;
```

结果：

namespace	viewName	isTemporary
global_temp	temp1	true
global_temp	temp2	true

使用模式匹配过滤视图

```
SHOW VIEWS FROM default LIKE 'sam*';
```

结果：

namespace	viewName	isTemporary
default	sam	false
default	sam1	false

使用多模式匹配

```
SHOW VIEWS LIKE 'sam|suj|temp*';
```

结果：

namespace	viewName	isTemporary
default	sam	false
default	suj	false
default	temp2	true

注意事项

- 该语句仅显示视图，不包含普通表。要查看所有表，请使用 SHOW TABLES。
- 本地临时视图始终出现在结果中，无论指定哪个数据库。
- 正则表达式模式中的 * 等价于SQL中的 %，| 表示或关系。
- 如果数据库不存在，会报错。

7 物化视图

7.1 使用场景

物化视图通过物理存储SQL查询的结果，使得未来的查询可以直接读取预计算的结果，而不是每次都重新计算相同的逻辑。这可以显著提高查询性能，特别是对于频繁运行的查询，尤其是涉及连接（joins）和聚合（aggregates）操作的查询。

当物化视图启用时，优化器会自动将符合条件的查询重写为使用物化视图，而不是扫描基础表。

Spark物化视图支持以下场景：

- 聚合场景：物化视图可以预先计算并存储聚合结果，如SUM、COUNT、AVG等。

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW mv AS
  SELECT empid, deptno
  FROM emps WHERE deptno > 5
  GROUP BY empid, deptno
  ```

  原始查询语句：

  ```
  SELECT deptno FROM emps
  WHERE deptno > 10 GROUP BY deptno
  ```

  改写后的查询语句：

  ```
  SELECT deptno FROM mv
  WHERE deptno > 10
  GROUP BY deptno
  ```

- Join场景：如果查询的表连接方式与物化视图匹配或物化视图可以补偿该连接方式。

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW mv AS
  SELECT o.order_id, c.customer_id
  FROM orders o INNER JOIN customers c ON o.cust_id = c.customer_id;
  ```


原始查询语句：

```
SELECT order_id FROM orders INNER JOIN customers
ON orders.cust_id = customers.customer_id;
```

改写后的查询语句：

```
SELECT order_id FROM mv
```

- Predicate/Filter场景：如果查询有可下推的过滤条件，物化视图可以预先计算并存储包含过滤条件的结果。

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW mv AS
  SELECT * FROM sales WHERE region = 'ASIA';
  ```

  原始查询语句：

  ```
  SELECT customer_id
  FROM sales WHERE region = 'ASIA' AND revenue > 1000;
  ```

  改写后的查询语句：

  ```
  SELECT customer_id from mv WHERE revenue > 1000;
  ```

  优化器可以使用物化视图，并应用额外的过滤条件来筛选revenue > 1000的数据。

- Aggregate Rollup场景：当物化视图（MV）包含更细粒度的预聚合数据时，查询可以进行“向上汇总”（roll up）。

  原始查询语句：

  ```
  SELECT deptno, COUNT(*) AS c, SUM(salary) AS s
  FROM emps
  GROUP BY deptno
  ```

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW mv AS
  SELECT empid, deptno, COUNT(*) AS c, SUM(salary) AS s
  FROM emps
  GROUP BY empid, deptno
  ```

  改写后的查询语句：

  ```
  SELECT deptno, SUM(c), SUM(s)
  FROM mv
  GROUP BY deptno
  ```

- Partial Join场景：查询中连接表是物化视图中连接表的超集的场景。

  样例一

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW join_mv1 AS
  SELECT lo_orderkey, lo_linenumber, lo_revenue, lo_partkey, c_name, c_address
  FROM lineorder INNER JOIN customer
  ON lo_custkey = c_custkey;
  ```

  原始查询语句：

  ```
  SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address, p_name
  FROM lineorder INNER JOIN customer ON lo_custkey = c_custkey
  INNER JOIN part ON lo_partkey = p_partkey;
  ```

  改写后的查询语句：

  ```
  SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address, p_name
  FROM join_mv1 INNER JOIN part ON lo_partkey = p_partkey;
  ```

  样例二

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW mv AS
  SELECT empid, deptno, state, SUM(salary) AS s
  FROM emps
  ```



```
JOIN locations ON emps.locationid = locations.locationid
GROUP BY empid, deptno, state
```

原始查询语句：

```
SELECT deptname, state, SUM(salary) AS s
FROM emps
JOIN depts ON emps.deptno = depts.deptno
JOIN locations ON emps.locationid = locations.locationid
GROUP BY deptname, state
```

改写后的查询语句：

```
SELECT deptname, state, SUM(s)
FROM mv
JOIN depts ON mv.deptno = depts.deptno
GROUP BY deptname, state;
```

- 表达式场景：重写带有复杂表达式（如算术运算、字符串函数和OR谓词）的连接查询。

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW join_mv1
  AS
  SELECT lo_orderkey, lo_linenumber, lo_revenue, lo_partkey, c_name, c_address
  FROM lineorder INNER JOIN customer
  ON lo_custkey = c_custkey;
  ```

  原始查询语句：

  ```
  SELECT
  lo_orderkey,
  lo_linenumber,
  (2 * lo_revenue + 1) * lo_linenumber,
  upper(c_name),
  substr(c_address, 3)
  FROM lineorder INNER JOIN customer
  ON lo_custkey = c_custkey;
  ```

- Join传递场景：指物化视图中的连接类型与查询中的连接类型不一致，但物化视图的连接结果却包含查询连接结果的一种场景。

  样例一

  原始查询语句：

  ```
  SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
  FROM lineorder LEFT OUTER JOIN customer
  ON lo_custkey = c_custkey
  WHERE customer.c_address = 'Sb4gxKs7' and customer.c_address is not null;
  ```

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW join_mv4
  AS
  SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
  FROM lineorder INNER JOIN customer
  ON lo_custkey = c_custkey;
  ```

  样例二

  原始查询语句：

  ```
  SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
  FROM lineorder INNER JOIN customer
  ON lo_custkey = c_custkey WHERE  c_custkey IS NOT NULL;
  ```

  创建物化视图示例：

  ```
  CREATE MATERIALIZED VIEW join_mv3
  AS
  SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
  FROM lineorder LEFT OUTER JOIN customer
  ON lo_custkey = c_custkey WHERE c_custkey IS NOT NULL;
  ```


7.2 配置参数

参数	参数说明	参数默认值
spark.sql.materialized.views.location	用于指定普通物化视图的存储位置，对于智能物化视图不生效。 如果没有指定这个参数，Spark会使用默认的存储位置。	-
spark.sql.materialized.views.rewrite.enabled	控制是否启用物化视图查询重写功能。 查询重写是指在执行查询时，Spark会尝试将查询重写为使用物化视图，以提高查询性能。 ● true：启用物化视图的查询重写功能。 ● false：不会尝试将查询重写为使用物化视图。	false
spark.sql.materialized.views.calcite.timeoutSeconds	用于控制在查询重写过程中Calcite规划器的最大超时时间，以秒为单位。 如果在指定的时间内Calcite规划器没有完成查询重写，Spark会放弃使用物化视图并按原样执行查询。	10
spark.sql.cte.mv.enabled	符合条件的CTE定义将在查询执行后异步物化。后续如果出现结构完全相同的 CTE 主体查询，将直接通过简单扫描物化视图（ MV ）来代替重新计算。（ 当前不支持Session级打开/开关，后续会支持Session级开关 ） ● true：启用CTE物化。 ● false：不启用CTE物化。	false
spark.sql.cte.mv.viewPrefix	由CTE物化功能创建的临时视图名称的前缀。	cte
spark.sql.cte.mv.database	存储CTE物化视图的数据库名称。	default
spark.sql.materialized.views.cte.hash.match	用于控制物化视图查询重写时，是否启用hash匹配模式。 hash匹配模式能够更快地命中物化视图，但缺点是被重写的sql必须与物化视图逻辑执行计划完全一致，否则无法命中。 ● true：启用hash匹配模式。 ● false：不启用hash匹配模式。	false

7.3 普通物化视图

7.3.1 语法参考

- 创建普通物化视图

  基于SQL查询创建物化视图：

  ```
  CREATE MATERIALIZED VIEW [IF NOT EXISTS] view_name
  [(<col_name> [COMMENT <col_comment>], ...)]
  [COMMENT '...']
  [PARTITIONED BY col1, col2, ...]
  [ CLUSTERED BY (col_name3, col_name4, ...)
  [ SORTED BY (col_name [ ASC | DESC], ...)]
  INTO num_buckets BUCKETS]
  [TBLPROPERTIES ('key'='value', ...)]
  AS <SELECT query>;
  ```

  样例：

  ```
  CREATE MATERIALIZED VIEW mv_partitioned
  PARTITIONED BY (region, product)
  AS SELECT region, product, SUM(amount) AS total
  FROM sales GROUP BY region, product
  CREATE MATERIALIZED VIEW mv_sorted_bucketed
  CLUSTERED BY (id) SORTED BY (amount ASC) INTO 2 BUCKETS
  AS SELECT * FROM sales
  CREATE MATERIALIZED VIEW mv_sales_with_product AS
  SELECT s.region, p.product_name, SUM(s.amount) AS total_amount
  FROM sales s
  INNER JOIN products p ON s.product_id = p.product_id
  GROUP BY s.region, p.product_name
  ```

- 删除物化视图

  ```
  DROP MATERIALIZED VIEW [IF EXISTS] mv_revenue_per_customer;
  ```

  样例：

  ```
  DROP MATERIALIZED VIEW mv_sales
  ```

- 查询物化视图

  查询当前数据库中的所有物化视图

  ```
  SHOW MATERIALIZED VIEWS;
  ```

  查询指定数据库中的所有物化视图

  ```
  SHOW MATERIALIZED VIEWS FROM example_db;
  ```

  查询符合特定模式的物化视图

  ```
  SHOW MATERIALIZED VIEWS LIKE 'mv_*';
  ```

- 刷新物化视图

  从基表中重新计算物化视图的数据。

  ```
  REFRESH MATERIALIZED VIEW mv_revenue_per_customer;
  ```

7.3.2 权限管理

物化视图可视为一种特殊的表在LakeFormation进行单独授权管理，包括读取、删除、修改等权限。

- 创建视图

  需要database的CREATE_TABLE、DESCRIBE权限 + 所有源表的SELECT、DESCRIBE、ALTER权限。

- 查询视图

  视图可视为一种特殊的表类型，视图创建者拥有视图的全部权限。其他用户查询视图，需要在LakeFormation中单独赋予SELECT、DESCRIBE权限。

- 删除视图

  视图属主会自动拥有视图的删除权限，其他用户想要删除需要单独赋予DESCRIBE、DROP权限。

- 刷新视图

  视图属主会自动拥有视图的刷新权限，其他用户想要刷新视图需要单独赋予INSERT、UPDATE、DELETE、DESCRIBE、ALTER权限。

- 自动改写

  需要拥有所有源表的SELECT、DESCRIBE权限 + 物化视图的SELECT、DESCRIBE权限。

7.3.3 自动改写

物化视图自动改写可在逻辑计划生成阶段匹配执行计划，将其自动改写为物化视图查询语句

- 自动改写鉴权

  自动改写时，需要同时校验所有源表的权限 + 物化视图的权限，才能自动改写。

- 精确匹配（Hash Match）

  将逻辑执行计划的hash值保存在表属性中，直接使用hash值比对判断是否命中，如果命中则改写查询语句。

- 模糊匹配（Plan Match）

  取出视图sql语句重新解析出逻辑计划，在后台详细对比执行计划的所有节点，如果是物化视图执行计划的子树则命中改写查询语句。

  相比于精确匹配，模糊匹配能够覆盖的范围更广，即使执行计划不完全相同也有命中的可能性。

7.3.4 元数据

创建物化视图时，会在源表和物化视图的Table Properties中新增一些参数用于记录相互关联的物化视图和源表信息。

表 7-1

属性名称	类型	参数描述
mv.archived	Boolean	视图是否归档。物化视图允许只创建表schema而不填充数据，刷新时再向其中填入数据。
mv.base.table.valid	Boolean	视图是否有效。视图无效时，不允许自动改写。
mv.base.tables	Array	用于保存视图关联的源表。
mv.select.statement.spark	String	保存Spark引擎物化视图的select语句。

7.3.5 约束与限制

- 模式

  当前仅支持Spark-SQL模式，Spark-JOB模式暂不支持。

- 支持的源表类型

  MANAGED_TABLE 、EXTERNAL_TABLE

- 表格式
  - Managed Materialized View被创建为Iceberg表。
  - 视图源表可支持所有AIDataLake中的可用格式，比如Hive、DataSource、Iceberg等。
- 刷新策略

  仅支持全视图刷新，不支持增量刷新。

- 使用场景
  - 不支持重写包含非确定性函数的查询，包括rand、random、uuid等函数。
  - 不支持将物化视图作为创建物化视图语句中的源表。
  - 不支持重写包含窗口函数的查询。
  - 不支持重写包含GROUPING SETS、WITH CUBE或WITH ROLLUP的查询。
  - 不支持重写包含外层引用的标量子查询或相关子查询的查询。

7.4 智能物化视图

智能物化视图提供更多高阶功能，包括更多类型的刷新策略、更灵活的刷新时效、CTE物化等。

7.4.1 语法参考

- 创建智能物化视图

  基于SQL查询创建物化视图：

```
CREATE LAKE MATERIALIZED VIEW [IF NOT EXISTS] view_name
[(<col_name> [COMMENT <col_comment>], ...)]
[COMMENT '...']
[PARTITIONED BY col1, col2, ...]
[ CLUSTERED BY (col_name3, col_name4, ...)
[ SORTED BY (col_name [ ASC | DESC], ...)]
INTO num_buckets BUCKETS]
[TBLPROPERTIES ('key'='value', ...)]
REFRESH POLICY AUTO | INCREMENTAL | INCREMENTAL STRICT | FULL
SCHEDULE EVERY  MINUTES| HOURS | DAYS
AS <SELECT query>;
```

  REFRESH POLICY AUTO | INCREMENTAL | INCREMENTAL STRICT | FULL 用于指定刷新策略

  SCHEDULE EVERY MINUTES| HOURS | DAYS 用于指定刷新时效，参数限制如下：
  - MINUTES： 1 < = M <= 60
  - HOURS：1 <= H <= 72
  - DAYS：1 <= D <= 7

7.4.2 权限管理

- 创建视图

  创建位置在LakeFormation内部obs桶中。需要database的CREATE_TABLE、DESCRIBE权限 + 所有源表的SELECT、DESCRIBE、ALTER权限。

- 查询视图

  为方便用户使用，LakeFormation为智能物化视图在后台默认开启了权限级联授权。

  如果一个用户对物化视图的源表有SELECT、DESCRIBE权限，当新视图被其他用户创建时，LakeFormation会在后台自动将视图的SELECT、DESCRIBE权限赋予该用户。

- 删除视图

  视图属主会自动拥有视图的删除权限，其他用户想要删除需要单独赋予DESCRIBE、DROP权限。

- 刷新视图

  视图属主会自动拥有视图的刷新权限，其他用户想要刷新视图需要单独赋予INSERT、UPDATE、DELETE、DESCRIBE、ALTER权限。

  智能物化视图支持LakeFormation后台定时刷新，因此刷新权限需要赋予LakeFormation内部用户。

- 自动改写

  需要拥有所有源表的SELECT、DESCRIBE权限 + 物化视图的SELECT、DESCRIBE权限。

7.4.3 视图刷新

7.4.3.1 刷新策略

- 增量刷新

  增量刷新允许物化视图（MVs）通过**仅处理源表中发生变化的数据**来进行更新，而不是重新计算整个视图。这大大减少了大容量数据集的刷新时间和资源消耗。

  使用增量刷新，要求源表必须是 Apache Iceberg 表，且开启了Iceberg快照功能。

- 增量刷新的使用限制
  - 支持的Aggregate Functions：SUM、COUNT(*)、COUNT(col)、MIN、MAX
  - 支持的Join：INNER JOIN、LEFT OUTER JOIN、RIGHT OUTER JOIN、FULL OUTER JOIN
  - 支持的Query：Simple SELECT (Projection)、GROUP BY Aggregation、UNION ALL、JOIN + Aggregation
  - 不支持的场景：源表不是iceberg表、不等值JOIN
- 四种刷新策略

  如下表所示，智能物化视图共支持四种刷新策略，可以在建表语句中使用关键字REFRESH POLICY指定，若未指定默认值为AUTO。亦可在视图创建后，使用Alter语句修改。

刷新策略	描述
INCREMENTAL	优先尝试增量刷新；如果不支持，则回退到全量刷新。
INCREMENTAL STRICT	仅使用增量刷新；如果不支持，则直接报错/失败。
FULL	始终执行全量刷新（重新计算整个视图）。
AUTO (Default)	优先选择增量刷新；如果不满足增量条件，则回退到全量刷新。

7.4.3.2 刷新时效

智能物化视图提供多种刷新时效供用户选择

- 手动刷新
  用户执行REFRESH MATERIALIZED VIEW mv_name命令。
- 定时刷新
  在表属性中设置刷新周期字段，由LakeFormation后台设置定时任务。
  允许用户执行alter命令修改刷新周期字段，Spark内核调用LakeFormation后台接口修改定时任务的周期。
- 实时刷新
  由LakeFormation监控视图新鲜度，一旦发现不新鲜，后台触发异步刷新任务。

7.4.3.3 视图元数据

属性名称	类型	参数描述
mv.base.tables.last.update.time	Long	源表上一次更新的UTC时间戳。
mv.last.refresh.time	Long	物化视图上次刷新UTC时间戳，类型为long，默认值为物化视图创建时间。
mv.refresh.policy	String	枚举类型：FULL、INCREMENTAL、INCREMENTAL STRICT、AUTO。
mv.refresh.interval	Int	单位：分钟。最小1分钟，最大值10080（即7天）。

7.4.4 CTE 物化

在某些情况下，用户可能会执行包含相同 CTE（公用表表达式）子查询的重复查询。此外，不同的用户之间可能也无法察觉彼此正在使用相同的计算逻辑。为了提升用户的计算效率、减少重复计算并加速查询，AIDatalake提供了CTE 物化 (CTE Materialization) 功能，该功能可以为 CTE 自动创建物化视图。

- 启动CTE物化
  SQL语句中开启本特性后，系统将执行以下操作：

a. CTE 子查询的识别与合规性检查：识别查询中的CTE子查询，并仅提取符合条件的CTE子查询进行物化。具体符合条件的标准请参见“限制(Limitations)”章节。
b. 自动创建物化视图：基于第一步的识别结果，对符合条件的CTE子查询进行物化。
c. 查询重写：利用CTE物化视图（MV）的查询重写功能，将CTE子查询重写为直接从上述创建的CTE物化视图表中读取数据。

7.4.5 约束与限制

- 模式
  当前仅支持Spark-SQL模式，Spark-JOB模式暂不支持
- 支持的源表类型
  MANAGED_TABLE
- 表格式
  - LAKE Materialized View被创建为Iceberg表。
  - 视图源表仅支持Iceberg表。
- 刷新策略
  支持全部四种刷新策略。
- 使用场景
  - 不支持重写包含非确定性函数的查询，包括rand、random、uuid等函数。
  - 不支持将物化视图作为创建物化视图语句中的源表。
  - 不支持重写包含窗口函数的查询。
  - 不支持重写包含GROUPING SETS、WITH CUBE或WITH ROLLUP的查询。
  - 不支持重写包含外层引用的标量子查询或相关子查询的查询。
  - 不支持CTE自引用

8 查看计划

功能描述

执行该语句将返回该SQL语句的逻辑计划与物理执行计划。

语法格式

```
EXPLAIN [EXTENDED | CODEGEN] statement;
```

关键字

EXTENDED：指定该关键字后，会同时输出逻辑计划与物理执行计划。

CODEGEN：指定该关键字后，若有codegen产生的代码也将输出。

注意事项

无。

示例

返回“SELECT * FROM test”SQL语句的逻辑计划与物理执行计划。

```
EXPLAIN EXTENDED select * from test;
```


9 数据权限语法

9.1 数据权限列表

权限矩阵

平台中SQL语句与数据库、表、角色相关的权限矩阵如**表9-2**所示。

表 9-1 表 1：支持权限矩阵

授权类型	LakeFormation权限类型	权限说明
database	ALL	数据库的所有操作权限。
	ALTER	修改数据库。
	DROP	删除数据库。
	DESCRIBE	查看数据库的元数据信息或切换数据库。
	LIST_TABLE	查看数据库下资源列表。

授权类型	LakeFormation权限类型	权限说明
	LIST_FUNC	查看某一数据库下的函数。
	CREATE_TABLE	在数据库中创建表。
	CREATE_FUNC	在数据库中创建函数。
table	ALL	表的所有操作权限。
	ALTER	修改表。
	DROP	删除表。
	DESCRIBE	查看表的元数据信息。
	UPDATE	更新表数据。
	INSERT	插入表数据。
	SELECT	查询表内数据。
	DELETE	删除表的数据。
column	SELECT	查询表内的列数据。
function	ALL	函数的所有操作权限。
	ALTER	修改函数。
	DROP	删除函数。
	DESCRIBE	查看函数的元数据信息。
	EXEC	执行函数。

表 9-2 常用 SQL 权限说明

分类	SQL语句	所需权限
Database	DROP DATABASE db1	db1的DROP权限。
	CREATE TABLE tb1(...)	db1下的CREATE_TABLE权限。
	CREATE VIEW v1	db1下的CREATE_VIEW权限。
Table	SHOW CREATE TABLE tb1	tb1的DESCRIBE权限。
	DESCRIBE [EXTENDED| FORMATTED] tb1	tb1的DESCRIBE权限。
	DROP TABLE [IF EXISTS] tb1	tb1的DROP权限。

分类	SQL语句	所需权限
	SELECT count(*) FROM tb1	tb1的SELECT权限。
	SELECT * FROM view1	view1的SELECT权限。
	SELECT count(*) FROM view1	view1的SELECT权限。
	LOAD DATA TABLE	tb1的INSERT权限。
	INSERT INTO TABLE	tb1的INSERT权限。
	INSERT OVERWRITE TABLE	tb1的INSERT权限。
	ALTER TABLE ADD COLUMNS	tb1的ALTER权限。
	ALTER TABLE RENAME	tb1的ALTER权限。
column	SELECT user_id FROM db.users;	db.users表中user_id列的SELECT权限。
function	ALTER FUNCTION db.calc_tax RENAME TO db.calc_tax_v2;	db下calc_tax函数的ALTER权限。
function	DROP FUNCTION IF EXISTS db.calc_tax;	db下calc_tax函数的DROP权限。
function	DESCRIBE FUNCTION EXTENDED db.calc_tax;	db下calc_tax函数的DESCRIBE 权限。
function	SELECT db.calc_tax("xxx");	db下calc_tax函数的SELECT权限。

注意事项

- 权限操作本质是通过平台管理的API实现，而非直接在Spark SQL引擎中执行GRANT/REVOKE语句来实现。
- 同一数据库下的表和视图共享权限命名空间。
- 具有GRANT_PRIVILEGE权限的用户可以为其他用户赋权。
- 具有REVOKE_PRIVILEGE权限的用户可以回收其他用户的权限。

9.2 创建角色

功能描述

- 创建一个新的角色。
- 只有在数据库上具有CREATE_ROLE权限的用户才能创建角色。例如：管理员用户、数据库的owner用户和被赋予了CREATE_ROLE权限的其他用户。
- 每个角色必须属于且只能属于一个数据库。

语法格式

```
CREATE ROLE role_name;
```

关键字

无。

示例

```
CREATE ROLE role1;
```

9.3 显示角色

功能描述

显示所有的角色或者显示当前数据库下绑定到“user_name”的角色。

语法格式

```
SHOW [ALL] ROLES [user_name];
```

参数说明

ALL：显示所有的角色。

注意事项

ALL关键字与user_name不可同时存在。

示例

- 显示project下的所有角色。
  ```
  SHOW ALL ROLES;
  ```

  说明

  只有管理员才有权限执行show all roles语句。
- 显示绑定到用户名为user_name1的所有角色。
  ```
  SHOW ROLES user_name1;
  ```


9.4 删除角色

功能描述

删除角色。

语法格式

```
DROP ROLE role_name;
```

关键字

无。

示例

```
DROP ROLE role1;
```

9.5 绑定角色

功能描述

绑定用户和角色。

语法格式

```
GRANT (role_name,...) TO (user_name,...);
```

关键字

无。

注意事项

role_name和user_name必须存在，否则会报错。

示例

```
GRANT role1 TO user_name1;
```

9.6 解绑角色

功能描述

取消用户和角色的绑定。

语法格式

```
REVOKE (role_name,...) FROM (user_name,...);
```


关键字

无。

注意事项

role_name和user_name必须存在，且user_name绑定了该role_name。

示例

取消用户user_name1和role1的绑定。

```
REVOKE role1 FROM user_name1;
```

9.7 分配权限

功能描述

授予用户或角色权限。

语法格式

```
GRANT (privilege,...) ON (resource,..) TO ((ROLE role_name) | (USER user_name)),...);
```

关键字

- ROLE：限定后面的role_name是一个角色。
- USER：限定后面的user_name是一个用户。

注意事项

- privilege必须是可授权限中的一种。且如果赋权对象在resource或上一级resource上已经有对应权限时，则会赋权失败。Privilege支持的权限类型可参见**9.1 数据权限列表**。
- resource可以是database、table、view、column、function格式分别为：
  - database的格式为：databases.db_name。

    database支持的Privilege权限类型可参见**9.1 数据权限列表**。
  - table的格式为：databases.db_name.tables.table_name。

    table支持的Privilege权限类型可参见**9.1 数据权限列表**。
  - view的格式为：databases.db_name.tables.view_name。

    view支持的Privilege权限类型和table一样，具体可以参考**9.1 数据权限列表**中table的权限列表描述。
  - column的格式为：databases.db_name.tables.table_name.columns.column_name。

    column支持的Privilege权限类型仅为：SELECT。
  - function的格式为：databases.db_name.functions.function_name。

    function支持的Privilege权限类型可参见**9.1 数据权限列表**。

示例

- 给用户user_name1授予数据库db1的删除数据库权限。
  ```
  GRANT DROP ON databases.db1 TO USER user_name1;
  ```
- 给用户user_name1授予数据库db1的表tb1的SELECT权限。
  ```
  GRANT SELECT ON databases.db1.tables.tb1 TO USER user_name1;
  ```
- 给角色role_name授予数据库db1的表tb1的SELECT权限。
  ```
  GRANT SELECT ON databases.db1.tables.tb1 TO ROLE role_name;
  ```

9.8 回收权限

功能描述

回收已经授予用户或角色的权限。

语法格式

```
REVOKE (privilege,...) ON (resource,..) FROM ((ROLE role_name) | (USER user_name)),...);
```

关键字

- ROLE：限定后面的role_name是一个角色。
- USER：限定后面的user_name是一个用户。

注意事项

- privilege必须为赋权对象在resource中的已授权限，否则会回收失败。Privilege支持的权限类型可参见**9.1 数据权限列表**。
- resource可以是database、table、view、column，function格式分别为：
  - database的格式为：databases.db_name
  - table的格式为：databases.db_name.tables.table_name
  - view的格式为：databases.db_name.tables.view_name
  - column的格式为：databases.db_name.tables.table_name.columns.column_name
  - function的格式为：databases.db_name.functions.function_name。

示例

- 回收用户user_name1对于数据库db1的删除数据库权限。
  ```
  REVOKE DROP ON databases.db1 FROM USER user_name1;
  ```
- 回收用户user_name1对于数据库db1的表tb1的SELECT权限。
  ```
  REVOKE SELECT ON databases.db1.tables.tb1 FROM USER user_name1;
  ```
- 回收角色role_name对于数据库db1的表tb1的SELECT权限。
  ```
  REVOKE SELECT ON databases.db1.tables.tb1 FROM ROLE role_name;
  ```

9.9 显示已授权限

功能描述

显示某个用户在resource上已经授予的权限。

语法格式

```
SHOW GRANT USER user_name ON resource;
```

关键字

USER：限定后面的user_name是一个用户。

注意事项

resource可以是database、table、column、view，function格式分别为：

- database的格式为：databases.db_name
- table的格式为：databases.db_name.tables.table_name
- column的格式为：databases.db_name.tables.table_name.columns.column_name
- view的格式为：databases.db_name.tables.view_name
- function的格式为：databases.db_name.functions.function_name

示例

显示用户user_name1在数据库db1上的权限。

```
SHOW GRANT USER user_name1 ON databases.db1;
```

9.10 显示所有角色和用户的绑定关系

功能描述

在当前database显示角色与某用户的绑定关系。

语法格式

```
SHOW PRINCIPALS ROLE;
```

关键字

无。

注意事项

角色名必须存在。

示例

```
SHOW PRINCIPALS role1;
```


10 数据类型

10.1 概述

数据类型是数据的一个基本属性，用于区分不同类型的数据。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表中。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。

10.2 原生数据类型

AI DataLake支持原生数据类型，请参见**表10-1**。

表 10-1 原生数据类型

数据类型	描述	存储空间	范围	OBS表支持情况	管理表支持情况
INT	有符号整数	4字节	-2147483648～2147483647	是	是
STRING	字符串	-	-	是	是
FLOAT	单精度浮点型	4字节	-	是	是
DOUBLE	双精度浮点型	8字节	-	是	是

数据类型	描述	存储空间	范围	OBS表支持情况	管理表支持情况
DECIMAL(precision,scale)	10进制精确数字类型。固定有效位数和小数位数的数据类型，例如：3.5。 precision：表示最多可以表示多少位的数字。 scale：表示小数部分的位数。	-	1<=precision<=38 0<=scale<=38 若不指定precision和scale，则默认为decimal(38,38)。	是	是
BOOLEAN	布尔类型	1字节	TRUE/FALSE	是	是
SMALLINT/SHORT	有符号整数	2字节	-32768~32767	是	是
TINYINT	有符号整数	1字节	-128~127	是	否
BIGINT/LONG	有符号整数	8字节	-9223372036854775808～9223372036854775807	是	是
TIMESTAMP	时间戳，表示日期和时间，格式为原始数据。例如：1621434131222	-	-	是	是
TIMESTAMP_NTZ(p)	无时区时间戳，p为秒小数精度（0-6），不带时区信息	-	-	是	是
TIMESTAMP_LTZ(p)	有时区时间戳，p为秒小数精度（0-6），带时区信息	-	-	是	是

数据类型	描述	存储空间	范围	OBS表支持情况	管理表支持情况
CHAR	固定长度字符串	-	-	是	是
VARCHAR	可变长度字符串	-	-	是	是
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 ~ 9999-12-31。	是	是
VARIANT	适用于半结构化数据处理，只支持iceberg表。	-	-	-	-

说明

- VARCHAR和CHAR在实际存储是STRING型，因此超出长度的字符串不会被截断。
- FLOAT类型在实际存储是DOUBLE型。

INT

有符号整数，存储空间为4字节，-2147483648～2147483647，在NULL情况下，默认值为0。

STRING

字符串类型。

FLOAT

单精度浮点型，存储空间为4字节，在NULL情况下，计算时默认值为0。

由于浮点类型的数据在计算机中的存储方式的限制，在比较两个浮点类型的数据是否相等时，因存在精度问题，不能直接采用“a==b”的方式进行比较，建议使用“(a-b)的绝对值<=EPSILON”这种方式进行比较，EPSILON为允许的误差范围，一般为1.19209290E-07F。若两个浮点数的差值的绝对值在这个范围内就认为相等。

DOUBLE

双精度浮点型，存储空间为8字节，在NULL情况下，采用计算值默认值为0。

由于浮点类型的数据在计算机中的存储方式的限制，在比较两个浮点类型的数据是否相等时，因存在精度问题，不能直接采用“a==b”的方式进行比较，建议使用“(a-b)

的绝对值<=EPSILON”这种方式进行比较，EPSILON为允许的误差范围，一般为2.2204460492503131E-16。若两个浮点数的差值的绝对值在这个范围内就认为相等。

DECIMAL

Decimal(p,s)表示数值中共有p位数，其中整数p-s位，小数s位。p表示可储存的最大十进制数的位数总数，小数点左右两侧都包括在内。有效位数p必须是1至最大有效位数38之间的值。s表示小数点右侧所能储存的最大十进制数的位数。小数位数必须是从0到p的值。只有在指定了有效位数时，才能指定小数位数。因此，0 ≤ s ≤ p。例如：decimal(10,6)，表示数值中共有10位数，其中整数占4位，小数占6位。

BOOLEAN

布尔类型，包括TRUE与FALSE。

SMALLINT/SHORT

有符号整数，存储空间为2字节，范围为-32768～32767。当为NULL情况下，采用计算值默认为0。

TINYINT

有符号整数，存储空间为1字节，范围为-128～127。当为NULL情况下，采用计算值默认为0。

BIGINT/LONG

有符号整数，存储空间为8字节，范围为-9223372036854775808～9223372036854775807，不支持科学计数法。当为NULL情况下，采用计算值默认为0。

TIMESTAMP

支持传统的UNIX TIMESTAMP，提供达到微秒级别精度的选择。TIMESTAMP是以指定时间和UNIX epoch（ UNIX epoch时间为1970年1月1日00:00:00 ）之间的秒数差定义的。可以向TIMESTAMP隐式转换的数据类型有STRING（ 必须具有"yyyy-MM-dd HH:mm:ss[.ffffff]"格式。小数点后精度可选 ）。

CHAR

CHAR的长度是固定的，使用指定长度的固定长度表示字符串。实际存储为STRING类型。

VARCHAR

VARCHAR生成时会带有一个长度指定数，用来定义字符串中的最大字符数。如果一个向VARCHAR转换的STRING型中的字符个数超过了长度指定数，那么这个STRING会被自动缩短。和STRING类型一样，VARCHAR末尾的空格数是有意义的，会影响比较结果。实际存储为STRING类型。

DATE

DATE类型只能和DATE、TIMESTAMP和STRING进行显式转换（cast），具体如表10-2所示。

表 10-2 cast 函数转换

显式转换	转换结果
cast(date as date)	相同DATE值。
cast(timestamp as date)	根据本地时区从TIMESTAMP得出年/月/日，将其作为DATE值返回。
cast(string as date)	如果字符串的形式是“yyyy-MM-dd”，将对应年/月/日作为DATE值返回。如果字符串不具有这种形式，返回NULL。
cast(date as timestamp)	根据本地时区生成并返回对应DATE的年/月/日零点的TIMESTAMP值。
cast(date as string)	根据DATE的年/月/日值生成并返回“yyyy-MM-dd”格式的字符串。

10.3 复杂数据类型

Spark SQL支持复杂数据类型，如表10-3所示。

表 10-3 复杂数据类型

数据类型	描述	使用格式
ARRAY	一组有序字段，使用指定的值构造ARRAY数组。可以为任意类型，要求所有字段的数据类型必须相同。	array(<value>,<value>[, ...]) 具体使用示例详见：**ARRAY示例**。
MAP	一组无序的键/值对，使用给定的Key和Value对生成MAP。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。	map(K <key1>, V <value1>, K <key2>, V <value2>[, ...]) 具体使用示例详见：**MAP示例**。
STRUCT	一组命名的字段，字段的数据类型可以不同。	struct(<value1>,<value2>[, ...]) 具体使用示例详见：**STRUCT示例**。

使用限制

- 创建含有复杂数据类型字段的表时，该表存储格式不支持CSV（txt）。
- 如果表中含有复杂数据类型字段时，该表不支持CSV（txt）格式的文件数据导入。
- MAP数据类型建表必须指定schema，且不支持date、short、timestamp数据类型。
- 对于JSON格式OBS表，MAP的键类型只支持STRING类型。
- 由于MAP类型的键不能为NULL，MAP键不支持对插入数据进行可能出现NULL值类型之间的隐式转换，如：STRING类型转换为其他原生类型、FLOAT类型转换为TIMESTAMP类型、其他原生类型转换为DECIMAL类型等。
- STRUCT数据类型不支持double，boolean数据类型。

ARRAY 示例

创建表“array_test”，将“id”参数定义为“ARRAY<INT>”数据类型，“name”参数定义为“STRING”数据类型。建表成功后插入测试数据到“array_test”中。操作如下：

1. 创建表。

   CREATE TABLE array_test(name STRING, id ARRAY < INT >) USING PARQUET;

2. 插入测试数据。

   INSERT INTO array_test VALUES ('test',array(1,2,3,4));

   INSERT INTO array_test VALUES ('test2',array(4,5,6,7));

   INSERT INTO array_test VALUES ('test3',array(7,8,9,0));

3. 查询结果。

   查“array_test”表中的所有数据：

   SELECT * FROM array_test;

   ```
   test3    [7,8,9,0]
   test2    [4,5,6,7]
   test    [1,2,3,4]
   ```

   查“array_test”表中id数组第0个元素的数据。

   SELECT id[0] FROM array_test;

   ```
   7
   4
   1
   ```

MAP 示例

创建表“map_test”，将“score”参数定义为“map<STRING,INT>”数据类型（键为STRING类型，值为INT类型）。建表成功后插入测试数据至“map_test”中。操作如下：

1. 创建表。

   CREATE TABLE map_test(id STRING, score map<STRING,INT>) USING PARQUET;

2. 插入测试数据。

   INSERT INTO map_test VALUES ('test4',map('math',70,'chemistry',84));

INSERT INTO map_test VALUES ('test5',map('math',85,'chemistry',97));

INSERT INTO map_test VALUES ('test6',map('math',88,'chemistry',80));

3. 查询结果。

   查询“map_test”表里的所有数据。

   SELECT * FROM map_test;

```
test6    {"chemistry":80,"math":88}
test5    {"chemistry":97,"math":85}
test4    {"chemistry":84,"math":70}
```

   查询“map_test”表中的数学成绩。

   SELECT id, score['math'] FROM map_test;

```
test6    88
test5    85
test4    70
```

STRUCT 示例

创建表“struct_test”，将info定义为“STRUCT<name:STRING, age:INT>”数据类型（由name和age构成的字段，其中name为STRING类型，age为INT类型）。建表成功后插入测试数据至“struct_test”表中。操作如下：

1. 创建表。

   CREATE TABLE struct_test(id INT, info STRUCT<name:STRING,age:INT>) USING PARQUET;

2. 插入测试数据。

   INSERT INTO struct_test VALUES (8, struct('user1',23));

   INSERT INTO struct_test VALUES (9, struct('user2',25));

   INSERT INTO struct_test VALUES (10, struct('user3',26));

3. 查询结果。

   查询“struct_test”表中的所有数据。

   SELECT * FROM struct_test;

```
8{"name":"user1","age":23}
10{"name":"user2","age":26}
9{"name":"user3","age":25}
```

   查询“struct_test”表中的**name**和**age**数据。

   SELECT id,info.name,info.age FROM struct_test;

```
8       user1    23
10      user2    26
9     user3     25
```


11 Iceberg 表相关语法

Apache Iceberg是一种面向海量分析数据集的开放表格式，通过高性能表格式为计算引擎添加表功能，使用方式与SQL表完全一致。

Iceberg专为超大规模表而构建，已在实际业务环境中得到应用，单个表可容纳数十PB数据，且即使如此庞大的表也无需分布式SQL引擎即可读取。

Spark引擎支持Iceberg的相关语法，关于Iceberg表相关语法的更多信息，请参见**Iceberg on Spark**。

12 函数语法

12.1 SQL 自定义函数

功能描述

SQL自定义函数允许使用SQL表达式定义函数逻辑，支持标量函数（返回单个值）和表值函数（返回表结果集）。

语法格式

- **创建标量函数**

```
CREATE [OR REPLACE] FUNCTION function_name ([parameter_name data_type [{ DEFAULT | = } default_value]] [, ...])
  RETURNS return_data_type
  RETURN expression;
```

- **创建表值函数**

```
CREATE [OR REPLACE] FUNCTION function_name ([parameter_name data_type [{ DEFAULT | = } default_value]] [, ...])
  RETURNS TABLE (col_name col_type [, ...])
  RETURN query_statement;
```

- **删除函数**

```
DROP FUNCTION [IF EXISTS] function_name;
```

- **查看函数定义**

```
DESCRIBE FUNCTION [EXTENDED] function_name;
```

关键字

- OR REPLACE：如果函数已存在则替换。

- RETURNS：指定返回值类型。
- RETURN：函数体，SQL表达式或查询语句。
- DEFAULT / =：指定参数默认值。

参数说明

参数	是否必选	描述
function_name	是	函数名称
parameter_name	否	参数名称
data_type	否	参数数据类型，若省略时，系统从默认值推断类型
default_value	否	参数默认值
return_data_type	是	返回值数据类型
expression	是	标量函数体，SQL表达式
query_statement	是	表值函数体，SELECT查询语句

示例

- **创建简单标量函数**

```
CREATE FUNCTION hello() RETURNS STRING RETURN 'Hello World';

SELECT hello();
-- 输出: Hello World
```

- **创建带参数的标量函数**

```
CREATE FUNCTION add_tax(price DOUBLE) RETURNS DOUBLE RETURN price * 1.13;

SELECT add_tax(100.0);
-- 输出: 113.0
```

- **创建带默认参数的函数**

```
CREATE FUNCTION greet(name STRING DEFAULT 'World') RETURNS STRING
  RETURN CONCAT('Hello, ', name, '!');

SELECT greet();
-- 输出: Hello, World!

SELECT greet('Spark');
-- 输出: Hello, Spark!
```

- **创建表值函数**

```
CREATE FUNCTION range_n(n INT)
  RETURNS TABLE (id INT, value STRING)
  RETURN SELECT id, CONCAT('val_', id) FROM EXPLODE(SEQUENCE(1, n)) AS t(id);

SELECT * FROM range_n(3);
-- 输出: (1, val_1), (2, val_2), (3, val_3)
```

- **替换已有函数**

```
CREATE OR REPLACE FUNCTION add_tax(price DOUBLE) RETURNS DOUBLE
  RETURN price * 1.2;
```

- **删除函数**

```
DROP FUNCTION IF EXISTS add_tax;
```


注意事项

- SQL自定义函数在Spark 4.0中为新增功能
- 标量函数的RETURN后跟单个SQL表达式
- 表值函数的RETURN后跟SELECT查询语句
- 带DEFAULT参数的函数，调用时可省略该参数
- 函数体内可引用已声明的会话变量

12.2 创建函数

功能描述

平台支持创建使用UDF和UDTF等自定义函数应用于Spark作业开发中。

语法格式

```
CREATE FUNCTION [db_name.]function_name AS class_name
  [USING resource,...]

resource:
  : JAR file_uri
```

或

```
CREATE OR REPLACE FUNCTION [db_name.]function_name AS class_name
  [USING resource,...]

resource:
  : JAR file_uri
```

注意事项

- 如果在数据库中存在同名的函数，系统将会报错。
- 只支持Hive语法创建函数。
- 请注意避免该场景：如果创建的自定义函数F1指定类C1，程序包名JAR1，创建自定义函数F2也指定类C1，程序包JAR2，因为F2和F1使用相同的类名，导致功能相互冲突，影响作业执行。
- UDF热加载功能处于受限使用阶段，如需使用请提交工单申请开通。

关键字

- USING ： 需要加载的资源。可以是JAR、文件或者URI的列表。
- OR REPLACE：支持自定义函数热加载功能。
  - 如果创建自定义函数时**不携带OR REPLACE**，则需要注意以下场景：

表 12-1 不携带 OR REPLACE 场景说明

场景说明	场景举例	生效机制	操作影响
场景一：修改了原有程序包类的实现逻辑，重新创建的函数指定的JAR包名和类名保持和原有一致。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。后续对J1包中函数实现做了逻辑修改，重新执行创建函数F2，指定类名C1，Jar包名J1。注意，如果步骤2继续使用函数名F1，则会因为函数名重复导致创建失败。这时可以考虑使用OR REPLACE，或者替换所有作业中的函数F1为F2。	需要重启Spark SQL队列后新创建的自定义函数F2生效	需要重启Spark SQL队列，影响当前运行的作业。重启队列后，影响F1原有功能，F1的功能变为和F2一样。
场景二：在原有程序包类的基础上新增了类，新创建的函数指定为新增的类，包名不变。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。J1的Jar包中新增了类C2，C1保持不变，新创建自定义函数F2，指定类名C2，Jar包名J1。	需要重启Spark SQL队列后新创建的自定义函数F2生效	需要重启Spark SQL队列，影响当前运行的作业。重启队列后，F1的功能不变。
场景三：原有程序包类的实现逻辑不变，重新打包程序包名。新创建的函数指定新JAR包名，类名保持不变。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。重新打包Jar包为J2，功能逻辑不变。新创建的自定义函数F2，指定类名C1，Jar包名J2。	新创建的自定义函数F2立即生效	无影响。

- 如果创建自定义函数**携带OR REPLACE**，表示需要对已有的函数内容进行功能替换并实时生效。

  该功能处于受限使用阶段，如需使用请提交工单申请开通。

  - 如果要在所有**SQL**队列上立即生效，需要分别选择**SQL**队列执行一遍：**CREATE OR** REPLACE xxx **FUNCTION** ...，否则没有执行的队列可能延迟0-12小时生效。
  - 如果当前运行的作业中使用自定义函数F1，该F1函数指定类名C1，程序包名J1，作业运行了一半后，重新修改J1程序包逻辑，CREATE OR REPLACE FUNCTION F1后，后续作业使用新的F1函数逻辑，但是已执行完一半的作业使用旧的函数逻辑。

表 12-2 携带 OR REPLACE 场景说明

场景说明	场景举例	生效机制	操作影响
场景一：修改了原有程序包类的实现逻辑，重新创建的函数指定的JAR包名和类名保持和原有一致。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。后续对J1包中函数实现做了逻辑修改，重新执行创建函数F1，指定类名C1，Jar包名J1。	立即生效	F1的功能变成新修改的逻辑。
场景二：在原有程序包类的基础上新增了类，新创建的函数指定为新增的类，包名不变。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。J1的Jar包中新增了类C2，C1保持不变，新创建自定义函数F2，指定类名C2，Jar包名J1。因为是第一次创建指定C2的函数，这时该F2函数不能立即生效。	两种生效方式：1. CREATE OR REPLACE FUNCTION F2创建语句再执行一次生效；2. 重启Spark SQL队列后生效	如果是选择重启Spark SQL队列，则会导致当前运行的作业受影响。
场景三：修改了原有程序包类的类名，新创建的函数指定为新类名，包名不变。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。J1的Jar包中修改原C1为C2，C1已经不存在，新创建自定义函数F2，指定类名C2，Jar包名J1。	立即生效	F1功能失效，F2功能立即生效。
场景四：原有程序包类的实现逻辑不变，重新打包程序包名。新创建的函数指定新JAR包名，类名保持不变。	在Spark SQL队列下已创建自定义函数F1，指定类名C1，Jar包名J1。重新打包Jar包为J2，功能逻辑不变。新创建的自定义函数F2，指定类名C1，Jar包名J2。	立即生效	无影响。

示例

创建函数mergeBill。

```
CREATE FUNCTION mergeBill AS 'com.xxx.hiveudf.MergeBill'
  using jar 'obs://onlyci-7/udf/MergeBill.jar';
```


12.3 删除函数

功能描述

删除函数。

语法格式

```
DROP [TEMPORARY] FUNCTION [IF EXISTS] [db_name.] function_name;
```

关键字

- TEMPORARY：指定删除临时函数。
- IF EXISTS：所删除的函数不存在时使用，可避免系统报错。

注意事项

- 删除一个已存在的函数。如果要删除的函数不存在，则系统报错。
- 只支持Hive语法。

示例

删除函数mergeBill。

```
DROP FUNCTION mergeBill;
```

12.4 显示函数详情

功能描述

查看指定函数的相关信息。

语法格式

```
DESCRIBE FUNCTION [EXTENDED] [db_name.] function_name;
```

关键字

EXTENDED：显示扩展使用信息。

注意事项

返回已有函数的元数据（实现类和用法），如果函数不存在，则系统报错。

示例

查看函数mergeBill的相关信息。

```
DESCRIBE FUNCTION mergeBill;
```


12.5 显示所有函数

功能描述

查看当前工程下所有的函数。

语法格式

```
SHOW [USER|SYSTEM|ALL] FUNCTIONS ([LIKE] regex | [db_name.] function_name);
```

其中regex为正则表达式，可以参考如下：

表 12-3 regex 参数举例说明

regex表达式	匹配含义
'xpath*'	表示匹配所有xpath开头的函数名。 例如：SHOW FUNCTIONS LIKE 'xpath*' ，表示可以匹配到：xpath、xpath_int、xpath_string等等xpath开头的函数。
'x[a-z]+'	表示匹配以x开头，后面是a到z范围的一个到多个字符的函数名。如可以匹配到：xpath、xtest等。
'x.*h'	匹配以x开头，h结尾，中间为一个或多个字符的函数名。如可以匹配到：xpath、xtesth等。

关键字

LIKE：此限定符仅为兼容性而使用，没有任何实际作用。

注意事项

显示与给定正则表达式或函数名匹配的函数。如果未提供正则表达式或名称，则显示所有函数。如果声明了USER或SYSTEM，那么将分别显示用户定义的Spark SQL函数和系统定义的Spark SQL函数。

示例

查看当前的所有函数。

```
SHOW FUNCTIONS;
```

12.6 DESCRIBE FUNCTION

功能描述

DESCRIBE FUNCTION语句用于查看函数的元数据信息，包括函数名称、实现类和用法说明。如果指定 EXTENDED 选项，还会显示扩展的使用信息和示例。

语法格式

```
{ DESC | DESCRIBE } FUNCTION [ EXTENDED ] function_name
```

参数说明

参数	描述
EXTENDED	可选。显示函数的扩展信息，包括使用方法和示例。
function_name	函数名称。可以是内置函数或自定义函数。可使用 database_name.function_name 限定。如果不指定数据库名，则使用当前数据库。

注意事项

- 不使用 EXTENDED 时，仅显示函数名称、实现类和基本用法。
- 使用 EXTENDED 时，额外显示示例和详细使用说明。
- 如果函数不存在，会报错。
- DESC 是 DESCRIBE 的缩写形式。

示例

- **查看内置标量函数**

```
DESC FUNCTION abs; -- +----------------------------------------------------------------------------+-- |
function_desc                                                                |--
+------------------------------------------------------------------+-- |Function:
abs                                                          |-- |Class:
org.apache.spark.sql.catalyst.expressions.Abs                |-- |Usage: abs(expr) - Returns the absolute
value of the numeric value.|-- +----------------------------------------------------------------------------+
```

- **查看内置标量函数（扩展信息）**

```
DESC FUNCTION EXTENDED abs; -- +----------------------------------------------------------------------------+-- |
function_desc                                                                |--
+------------------------------------------------------------------+-- |Function:
abs                                                          |-- |Class:
org.apache.spark.sql.catalyst.expressions.Abs                |-- |Usage: abs(expr) - Returns the absolute
value of the numeric value.|-- |Extended Usage:                                                                |-- |
Examples:                                                    |-- |     > SELECT abs(-1);
|-- |       1                                                    |--
+------------------------------------------------------------------+
```

- **查看内置聚合函数**

```
DESC FUNCTION max; -- +------------------------------------------------------------------+-- |
function_desc                                                |--
+------------------------------------------------------------------+-- |Function:
max                                                          |-- |Class:
org.apache.spark.sql.catalyst.expressions.aggregate.Max|-- |Usage: max(expr) - Returns the maximum
value of `expr`.       |-- +------------------------------------------------------------------+
```

- **查看生成器函数（扩展信息）**

```
DESC FUNCTION EXTENDED explode; -- +------------------------------------------------------------------+-- |
function_desc                                                |--
+------------------------------------------------------------------+-- |Function:
explode                                                      |-- |Class: org.apache.spark.sql.catalyst.expressions.Explode
|-- |Usage: explode(expr) - Separates the elements of array `expr`  |-- | into multiple rows, or the
elements of map `expr` into        |-- | multiple rows and columns.                                    |-- |
Extended Usage:                                              |-- |    Examples:                                                  |--
|     > SELECT explode(array(10, 20));                       |-- |       10
```



```
/-- /      20                                          /--
+-------------------------------------------------------------+
```


13 查询语法

13.1 Select

功能描述

SELECT语句用于从表中查询数据，是Spark SQL中最基本的查询语句。Spark 4.0支持完整的SELECT语法，包括LATERAL、PIVOT/UNPIVOT、OFFSET分页等特性。

语法格式

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] { * | projectItem [, ...] }
  FROM table_reference [, ...]
  [ WHERE boolean_expression ]
  [ GROUP BY { expression | GROUPING SETS ( ... ) | ROLLUP ( ... ) | CUBE ( ... ) } [, ...] ]
  [ HAVING boolean_expression ]
  [ PIVOT ( aggregate_expr FOR pivot_column IN ( value1 [, ...] ) ) ]
  [ UNPIVOT ( value_column FOR name_column IN ( column1 [, ...] ) ) ]
  [ LATERAL VIEW function_expr AS alias [, ...] ]
  [ LATERAL ( subquery ) ]

  [ ORDER BY { expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] } [, ...] ]
  [ SORT BY { expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] } [, ...] ]
  [ CLUSTER BY expression [, ...] ]
  [ DISTRIBUTE BY expression [, ...] ]
```



```
[ OFFSET integer_expression ]
[ LIMIT integer_expression ]
```

其中 projectItem 为：

```
expression [ [ AS ] alias ]
```

table_reference 为：

```
{ table_name [ TABLE ] [ alias ]
  | join_table  | LATERAL ( subquery ) [ alias ]
  | ( subquery ) [ alias ]
  | VALUES ( { expression } [, ...] ) [, ...] [ alias ]
  | UNNEST(expression) [ alias ] }
```

参数说明

表 13-1 参数说明

参数	说明
WITH with_query	定义公共表表达式（CTE），用于简化复杂查询，提高可读性。详见WITH...AS。
ALL	返回所有行，包括重复行。为默认选项。
DISTINCT	从结果集中移除重复的行。
*	查询所有列。
projectItem	指定查询的列或表达式，可使用AS指定别名。
FROM table_reference	指定查询的数据源，可以是表名、子查询、JOIN结果、LATERAL子查询等。详见FROM。
WHERE	指定过滤条件，仅返回满足条件的行。详见WHERE。
GROUP BY	按指定表达式分组，通常与聚合函数配合使用。支持GROUPING SETS、ROLLUP、CUBE扩展。
HAVING	对GROUP BY的结果进行过滤。详见HAVING。
PIVOT	行转列操作，根据指定列的值将行数据旋转为列。
UNPIVOT	列转行操作，将列数据旋转为行。
LATERAL VIEW	与表生成函数（如explode）配合使用，将函数输出展开为多行。
LATERAL (subquery)	LATERAL子查询，允许子查询引用外部查询的列。
ORDER BY	全局排序。详见ORDER BY。
SORT BY	局部排序，每个Reducer内排序。详见SORT BY。
CLUSTER BY	分桶并桶内排序，等效于DISTRIBUTE BY + SORT BY同字段。详见CLUSTER BY。
DISTRIBUTE BY	按指定字段分桶，将相同键值的数据分配到同一Reducer。详见DISTRIBUTE BY。

参数	说明
OFFSET	跳过前N行结果，用于分页查询。
LIMIT	限制返回的行数。
NULLS FIRST / NULLS LAST	指定NULL值在排序结果中的位置。NULLS FIRST将NULL排在最前，NULLS LAST将NULL排在最后。

注意事项

- 所查询的表必须是已经存在的表，否则提示查询错误。
- 在WHERE子句中使用NOT IN时，必须保证子查询结果集不包含任何NULL值。一旦存在NULL，整个查询结果将为空集。详见下方常见问题。

示例

- 基本查询：
```
SELECT * FROM student;
```
- 带DISTINCT去重：
```
SELECT DISTINCT name FROM student;
```
- 带WHERE条件：
```
SELECT name, score FROM student WHERE score > 90;
```
- 带GROUP BY和HAVING：
```
SELECT class_id, AVG(score) AS avg_score FROM student
  GROUP BY class_id
  HAVING AVG(score) > 80;
```
- 带OFFSET分页（Spark 4.0）：
```
SELECT * FROM student ORDER BY score DESC LIMIT 10 OFFSET 20;
```
- 带PIVOT行转列（Spark 4.0）：
```
SELECT * FROM sales
  PIVOT (SUM(amount) FOR year IN (2023, 2024, 2025));
```
- 带LATERAL子查询（Spark 4.0）：
```
SELECT * FROM orders, LATERAL (SELECT * FROM items WHERE items.order_id = orders.id);
```

:: 类型转换

```
SELECT '123' :: INT;
 -- 等价于 CAST('123' AS INT)
```

COLLATE 排序规则

```
SELECT name FROM t ORDER BY name COLLATE unicode_ci;
```

常见问题：NOT IN 子查询中有 NULL 值，查询结果为空怎么办？

- **问题概述**：SQL语法中只要NOT IN子查询中有一个NULL值，整个查询结果就会是空集（无数据返回）。
- **示例**：如下示例中期望的结果是3，但是反馈的是空集。
```
-- 表A       -- 表B
id          value
---         -----
1           1
2           2
3
```


执行以下SQL语句：

```
SELECT * FROM A
WHERE id NOT IN (SELECT value FROM B);
```

- **原因分析**：

  ```
  id NOT IN (1, 2, NULL) 等价于 id <> 1 AND id <> 2 AND id <> NULL
  ```

  id <> NULL 的结果是UNKNOWN（SQL的三值逻辑），整个表达式为FALSE，因此无数据返回。

- **解决方案**：

  方案1：用NOT EXISTS替代NOT IN

  ```
  SELECT * FROM A
  WHERE NOT EXISTS (
    SELECT 1 FROM B WHERE B.value = A.id
  );
  ```

  方案2：在子查询中过滤掉NULL

  ```
  SELECT * FROM A
  WHERE id NOT IN (
    SELECT value FROM B WHERE value IS NOT NULL
  );
  ```

13.2 排序

13.2.1 ORDER BY

功能描述

ORDER BY子句用于对查询结果进行全局排序。Spark 4.0增强了排序功能，支持NULLS FIRST/LAST指定NULL值排序位置，并可与OFFSET配合实现分页。

语法格式

```
SELECT projectItem [, ...] FROM table_reference
  ORDER BY { expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] } [, ...]
```

参数说明

参数	说明
expression	排序表达式，可以是列名、列别名或表达式。
ASC	升序排序，为默认选项。
DESC	降序排序。
NULLS FIRST	将NULL值排在排序结果的最前面。
NULLS LAST	将NULL值排在排序结果的最后面。默认情况下，升序时NULL排在最后，降序时NULL排在最前。

注意事项

- ORDER BY执行全局排序，所有数据会汇聚到一个Reducer中处理。
- 与GROUP BY一起使用时，ORDER BY后面可以跟聚合函数。
- ORDER BY可以引用SELECT列表中的别名。
- 可与LIMIT和OFFSET配合实现分页查询。

示例

按单列升序排序：

```
SELECT * FROM student ORDER BY score;
```

按单列降序排序：

```
SELECT * FROM student ORDER BY score DESC;
```

按多列排序：

```
SELECT * FROM student ORDER BY class_id ASC, score DESC;
```

指定NULL值排序位置：

```
SELECT * FROM student ORDER BY score DESC NULLS LAST;
```

使用列别名排序：

```
SELECT name, score * 1.5 AS weighted_score FROM student   ORDER BY weighted_score DESC;
```

分页查询（与OFFSET配合，Spark 4.0）：

```
SELECT * FROM student ORDER BY score DESC LIMIT 10 OFFSET 20;
```

13.2.2 SORT BY

功能描述

SORT BY子句用于在每个Reducer内对数据进行局部排序。与ORDER BY的全局排序不同，SORT BY只保证每个Reducer内的数据有序。

语法格式

```
SELECT projectItem [, ...] FROM
  table_reference SORT BY { expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] } [, ...]
```

参数说明

参数	说明
expression	排序表达式，可以是列名或表达式。
ASC	升序排序，为默认选项。
DESC	降序排序。
NULLS FIRST	将NULL值排在排序结果的最前面。
NULLS LAST	将NULL值排在排序结果的最后面。

注意事项

- SORT BY是局部排序，仅保证每个Reducer内的数据有序，不保证全局有序。
- 如需全局排序，请使用ORDER BY。
- SORT BY通常与DISTRIBUTE BY配合使用，先分桶再桶内排序。
- SORT BY支持NULLS FIRST/LAST指定NULL值的排序位置。

示例

按score在Reducer内升序排序：

```
SELECT * FROM student SORT BY score;
```

按score在Reducer内降序排序：

```
SELECT * FROM student SORT BY score DESC;
```

与DISTRIBUTE BY配合使用：

```
SELECT * FROM student DISTRIBUTE BY class_id SORT BY score DESC;
```

多字段排序并指定NULL值位置：

```
SELECT * FROM student SORT BY class_id ASC NULLS LAST, score DESC NULLS LAST;
```

13.2.3 CLUSTER BY

功能描述

CLUSTER BY子句用于根据指定字段对数据进行分桶，并在桶内按同一字段进行排序。CLUSTER BY等效于DISTRIBUTE BY和SORT BY使用相同字段的情况。

语法格式

```
SELECT projectItem [, ...] FROM table_reference   CLUSTER BY expression [, ...]
```

参数说明

参数	说明
expression	分桶及排序表达式。支持单字段及多字段。CLUSTER BY根据指定字段的哈希值将数据分配到不同的桶中，并在桶内按同一字段排序。

注意事项

- CLUSTER BY只能按升序排序，不能指定ASC或DESC。如需指定排序方向，请使用DISTRIBUTE BY + SORT BY组合。
- CLUSTER BY等效于DISTRIBUTE BY和SORT BY使用相同字段。

示例

根据字段score进行分桶并桶内排序：

```
SELECT * FROM student CLUSTER BY score;
```


根据多字段分桶排序：

```
SELECT * FROM student CLUSTER BY class_id, score;
```

当需要指定降序时，使用DISTRIBUTE BY + SORT BY替代：

```
SELECT * FROM student DISTRIBUTE BY score SORT BY score DESC;
```

13.2.4 DISTRIBUTE BY

功能描述

DISTRIBUTE BY子句用于根据指定字段对数据进行分桶（重新分区），将相同键值的数据分配到同一个Reducer中。DISTRIBUTE BY不会在桶内进行排序。

语法格式

```
SELECT projectItem [, ...] FROM table_reference
  DISTRIBUTE BY expression [, ...]
```

参数说明

参数	说明
expression	分桶表达式。支持单字段及多字段。根据表达式的哈希值将数据分配到不同的Reducer中，相同键值的数据一定在同一个Reducer中。

注意事项

- DISTRIBUTE BY仅负责分桶，不负责桶内排序。
- 如需在分桶后排序，请配合SORT BY使用。DISTRIBUTE BY和SORT BY可以使用不同字段。
- CLUSTER BY是DISTRIBUTE BY和SORT BY使用相同字段的简写。

示例

根据字段score进行分桶：

```
SELECT * FROM student DISTRIBUTE BY score;
```

分桶后排序：

```
SELECT * FROM student DISTRIBUTE BY class_id SORT BY score DESC;
```

多字段分桶：

```
SELECT * FROM student DISTRIBUTE BY class_id, name;
```

13.3 分组

13.3.1 按列 GROUP BY

功能描述

按列对表进行分组操作。

语法格式

```
SELECT attr_expr_list FROM table_reference
 GROUP BY col_name_list;
```

关键字

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，col_name_list中包含的字段必须出现在attr_expr_list的字段内，attr_expr_list中可以使用多个聚合函数，比如count()，sum()，聚合函数中可以包含其他字段。
- 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，同样，GROUP BY中出现的字段必须在attr_expr_list的字段内，attr_expr_list也可以使用聚合函数。

注意事项

所要分组的表必须是已经存在的表，否则会出错。

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。

```
SELECT score, count(name) FROM student
  GROUP BY score,name;
```

13.3.2 按表达式 GROUP BY

功能描述

按表达式对表进行分组操作。

语法格式

```
SELECT attr_expr_list FROM table_reference
 GROUP BY groupby_expression [, groupby_expression, ...];
```

关键字

groupby_expression：可以是单字段，多字段，也可以是聚合函数，字符串函数等。

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 同单列分组，GROUP BY中出现的字段必须包含在attr_expr_list的字段中，表达式支持内置函数，自定义函数等。

示例

先利用substr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。

```
SELECT substr(name,6),count(name) FROM student
  GROUP BY substr(name,6);
```

13.3.3 GROUP BY 中使用 HAVING

功能描述

利用HAVING子句在表分组后实现过滤。

语法格式

```
SELECT attr_expr_list FROM table_reference
 GROUP BY groupby_expression[, groupby_expression… ]
 HAVING having_expression;
```

关键字

groupby_expression：可以是单字段，多字段，也可以是聚合函数，字符串函数等。

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 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子句中可支持算术运算，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。

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

13.3.4 ROLLUP

功能描述

ROLLUP是GROUP BY的扩展，用于生成多级聚合行、超聚合行和总计行。ROLLUP按照从右到左的顺序递减分组粒度，实现层次化的聚合统计。

语法格式

```
SELECT projectItem [, ...] FROM table_reference
  GROUP BY ROLLUP ( expression [, ...] )
```

或使用WITH语法：


```
SELECT projectItem [, ...] FROM table_reference
  GROUP BY expression [, ...] WITH ROLLUP;
```

参数说明

参数	说明
ROLLUP	按指定列从右到左递减的层次结构进行分组聚合。例如ROLLUP(a, b, c)等价于GROUPING SETS ((a, b, c), (a, b), (a), ())。
expression	分组表达式，可以是列名或表达式。

ROLLUP 等价转换

GROUP BY ROLLUP(a, b, c) 等价于以下四个分组查询的UNION ALL：

- (a, b, c) 分组小计：
  ```
  SELECT a, b, c, sum(expression) FROM table GROUP BY a, b, c;
  ```
- (a, b) 分组小计：
  ```
  SELECT a, b, NULL, sum(expression) FROM table GROUP BY a, b;
  ```
- (a) 分组小计：
  ```
  SELECT a, NULL, NULL, sum(expression) FROM table GROUP BY a;
  ```
- () 总计：
  ```
  SELECT NULL, NULL, NULL, sum(expression) FROM table;
  ```

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- ROLLUP的分组粒度从右到左递减，最右侧的列最先被去除。
- 可以配合GROUPING()函数判断某列是否参与了当前行的分组。

示例

根据group_id与job生成多级聚合：

```
SELECT group_id, job, SUM(salary) FROM group_test
  GROUP BY ROLLUP (group_id, job);
```

等价于WITH语法：

```
SELECT group_id, job, SUM(salary) FROM group_test
  GROUP BY group_id, job WITH ROLLUP;
```

配合GROUPING()函数标识分组级别：

```
SELECT group_id, job, SUM(salary),
  GROUPING(group_id) AS g1, GROUPING(job) AS g2
  FROM group_test
  GROUP BY ROLLUP (group_id, job);
```

13.3.5 GROUPING SETS

功能描述

GROUPING SETS是GROUP BY的扩展，用于在一次查询中实现多组不同粒度的分组统计，相当于将多个GROUP BY查询通过UNION ALL合并的结果。

语法格式

```
SELECT projectItem [, ...] FROM table_reference
  GROUP BY { expression | GROUPING SETS ( grouping_set [, ...] ) } [, ...]
```

其中 grouping_set 为：

```
{ ( expression [, ...] ) | expression }
```

参数说明

参数	说明
GROUPING SETS	指定多个分组集合，每个分组集合定义一组用于GROUP BY的列。未参与分组的列在结果中以NULL填充。
grouping_set	一个分组集合，用括号包围的列组合表示。单个列可以省略括号。

GROUPING SETS 等价转换

- GROUP BY a, b GROUPING SETS ((a,b)) 等价于：
  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b;
  ```
- GROUP BY a, b GROUPING SETS (a, b) 等价于：
  ```
  SELECT a, NULL, sum(expression) FROM table GROUP BY a
  UNION ALL
  SELECT NULL, b, sum(expression) FROM table GROUP BY b;
  ```
- GROUP BY a, b GROUPING SETS ((a,b), a) 等价于：
  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b
  UNION ALL
  SELECT a, NULL, sum(expression) FROM table GROUP BY a;
  ```
- GROUP BY a, b GROUPING SETS ((a,b), a, b, ()) 等价于：
  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b
  UNION ALL
  SELECT a, NULL, sum(expression) FROM table GROUP BY a
  UNION ALL
  SELECT NULL, b, sum(expression) FROM table GROUP BY b
  UNION ALL
  SELECT NULL, NULL, sum(expression) FROM table;
  ```

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 未参与分组的列在结果中以NULL填充。
- 可以配合GROUPING()函数判断某列是否参与了当前行的分组。

示例

根据group_id与job两个字段生成交叉表格行：

```
SELECT group_id, job, SUM(salary) FROM group_test
  GROUP BY group_id, job
  GROUPING SETS (group_id, job);
```

多分组集合包含总计：

```
SELECT group_id, job, SUM(salary), GROUPING(group_id), GROUPING(job)
  FROM group_test
```



```
GROUP BY group_id, job
GROUPING SETS ((group_id, job), group_id, job, ());
```

13.4 连接

13.4.1 内连接

功能描述

内连接（INNER JOIN）将两个表中满足连接条件的行组合起来作为结果集。仅返回两表中都满足 JOIN 条件的记录。

语法格式

```
SELECT attr_expr_list FROM table_reference
 {JOIN | INNER JOIN} table_reference ON join_condition;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。
JOIN / INNER JOIN	内连接关键字，只显示参与连接的表中满足 JOIN 条件的记录。两者语义相同。
join_condition	连接条件，通常为两表之间列的等值比较表达式。
ON	指定连接条件的关键字。

注意事项

- 所要进行 JOIN 连接的表必须是已经存在的表，否则会出错。
- 在一次查询中可以连接两个以上的表，使用多个 JOIN 子句依次连接。

示例

通过将 student_info 与 course_info 两张表中的课程编号匹配建立 JOIN 连接，来查看学生姓名及所选课程名称。

```
SELECT student_info.name, course_info.courseName FROM student_info JOIN course_info ON
student_info.courseId = course_info.courseId;
```

等价写法：

```
SELECT student_info.name, course_info.courseName FROM student_info INNER JOIN course_info ON
student_info.courseId = course_info.courseId;
```


13.4.2 左外连接

功能描述

左外连接（LEFT OUTER JOIN）根据左表的记录去匹配右表，返回左表的所有记录。对于右表中没有匹配值的记录，结果集中相应列返回 NULL。

语法格式

```
SELECT attr_expr_list FROM table_reference
 LEFT OUTER JOIN table_reference ON join_condition;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。
LEFT OUTER JOIN	左外连接关键字，返回左表的所有记录，右表中没有匹配值的记录将返回 NULL。LEFT JOIN 是 LEFT OUTER JOIN 的简写。
join_condition	连接条件，通常为两表之间列的等值比较表达式。
ON	指定连接条件的关键字。

注意事项

- 所要进行 JOIN 连接的表必须是已经存在的表，否则会出错。
- LEFT JOIN 是 LEFT OUTER JOIN 的简写形式，两者语义完全相同。

示例

左外连接时利用 student_info 表中的 courseId 与 course_info 中的 courseId 进行匹配，返回已经选课的学生姓名及所选的课程名称，没有匹配值的右表记录将返回 NULL。

```
SELECT student_info.name, course_info.courseName FROM student_info LEFT OUTER JOIN course_info ON student_info.courseId = course_info.courseId;
```

等价写法：

```
SELECT student_info.name, course_info.courseName FROM student_info LEFT JOIN course_info ON student_info.courseId = course_info.courseId;
```


13.4.3 右外连接

功能描述

右外连接（RIGHT OUTER JOIN）根据右表的记录去匹配左表，返回右表的所有记录。对于左表中没有匹配值的记录，结果集中相应列返回 NULL。

语法格式

```
SELECT attr_expr_list FROM table_reference
  RIGHT OUTER JOIN table_reference ON join_condition;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。
RIGHT OUTER JOIN	右外连接关键字，返回右表的所有记录，左表中没有匹配值的记录将返回 NULL。RIGHT JOIN 是 RIGHT OUTER JOIN 的简写。
join_condition	连接条件，通常为两表之间列的等值比较表达式。
ON	指定连接条件的关键字。

注意事项

- 所要进行 JOIN 连接的表必须是已经存在的表，否则会出错。
- RIGHT JOIN 是 RIGHT OUTER JOIN 的简写形式，两者语义完全相同。
- 右外连接与左外连接对称：A RIGHT OUTER JOIN B ON ... 等价于 B LEFT OUTER JOIN A ON ...。

示例

右外连接和左外连接相似，但是会将右边表（course_info）中的所有记录返回，没有匹配值的左表记录将返回 NULL。

```
SELECT student_info.name, course_info.courseName FROM student_info RIGHT OUTER JOIN course_info ON student_info.courseId = course_info.courseId;
```

等价写法：

```
SELECT student_info.name, course_info.courseName FROM student_info RIGHT JOIN course_info ON student_info.courseId = course_info.courseId;
```

13.4.4 全外连接

功能描述

全外连接（FULL OUTER JOIN）根据左表与右表的所有记录进行匹配，返回两表中的所有记录。对于没有匹配值的记录，结果集中相应列返回 NULL。

语法格式

```
SELECT attr_expr_list FROM table_reference
  FULL OUTER JOIN table_reference ON join_condition;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。
FULL OUTER JOIN	全外连接关键字，返回左表与右表的所有记录，没有匹配值的记录返回 NULL。FULL JOIN 是 FULL OUTER JOIN 的简写。
join_condition	连接条件，通常为两表之间列的等值比较表达式。
ON	指定连接条件的关键字。

注意事项

- 所要进行 JOIN 连接的表必须是已经存在的表，否则会出错。
- FULL JOIN 是 FULL OUTER JOIN 的简写形式，两者语义完全相同。
- 全外连接的结果集等于左外连接与右外连接的并集。

示例

利用全外连接可以将两张表中的所有记录返回，没有匹配值的左表及右表记录将返回 NULL。

```
SELECT student_info.name, course_info.courseName FROM student_info FULL OUTER JOIN course_info ON student_info.courseId = course_info.courseId;
```

等价写法：

```
SELECT student_info.name, course_info.courseName FROM student_info FULL JOIN course_info ON student_info.courseId = course_info.courseId;
```

13.4.5 隐式连接

功能描述

隐式连接与内连接功能相同，返回两表中满足 WHERE 条件的结果集，但不用 JOIN 关键字显式指定连接条件，而是通过 WHERE 子句实现。

语法格式

```
SELECT table_reference.col_name, table_reference.col_name, ...
FROM table_reference, table_reference
WHERE table_reference.col_name = table_reference.col_name;
```


参数说明

参数	说明
col_name	列名，指定需要返回的列。
table_reference	进行连接的表名。必须是已存在的表，否则会出错。
WHERE	隐式连接利用 WHERE 条件实现类似 JOIN ... ON ... 的连接，返回匹配的记录。支持等值条件和不等值条件。

注意事项

- 所要进行连接的表必须是已经存在的表，否则会出错。
- 隐式连接的命令中不含有 JOIN ... ON ... 关键词，而是通过 WHERE 子句作为连接条件将两张表连接。
- 隐式连接等价于 INNER JOIN，建议优先使用显式 JOIN 语法以提升可读性。

示例

返回 courseId 匹配的学生姓名及课程名称。

```
SELECT student_info.name, course_info.courseName FROM student_info, course_info WHERE
student_info.courseId = course_info.courseId;
```

不等式隐式连接示例：

```
SELECT a.name, b.name FROM student_info_1 a, student_info_2 b WHERE a.score > b.score;
```

13.4.6 笛卡尔连接

功能描述

笛卡尔连接（CROSS JOIN）把第一个表的每一条记录和第二个表的所有记录相连接。如果第一个表的记录数为m，第二个表的记录数为n，则会产生m * n条记录。

语法格式

```
SELECT attr_expr_list FROM table_reference
  CROSS JOIN table_reference [ON join_condition];

SELECT attr_expr_list FROM table_reference
  CROSS JOIN table_reference;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。

参数	说明
CROSS JOIN	笛卡尔连接关键字，求笛卡尔积的标准方式。
join_condition	可选的连接条件。如果该条件恒成立（如1=1），该连接即为笛卡尔连接。不带ON子句的CROSS JOIN也表示笛卡尔积。

注意事项

- 所要进行JOIN连接的表必须是已经存在的表，否则会出错。
- CROSS JOIN是求笛卡尔积的标准方式。如果不带ON子句，则直接计算两表的笛卡尔积。
- 笛卡尔连接可能产生大量记录，请谨慎使用。

示例

返回 student_info 与 course_info 两张表中学生姓名与课程名称的所有组合。

```
SELECT student_info.name, course_info.courseName FROM student_info CROSS JOIN course_info;
```

等价写法（使用恒成立的 ON 条件）：

```
SELECT student_info.name, course_info.courseName FROM student_info CROSS JOIN course_info ON (1 = 1);
```

13.4.7 左半连接

功能描述

左半连接（LEFT SEMI JOIN）用来查看左表中符合 JOIN 条件的记录。左半连接只返回左表中的记录，不返回右表的列。

语法格式

```
SELECT attr_expr_list FROM table_reference
  LEFT SEMI JOIN table_reference ON join_condition;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。所涉及的字段只能是左表中的字段，否则会出错。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。
LEFT SEMI JOIN	左半连接关键字，只显示左表中满足 JOIN 条件的记录。
join_condition	连接条件，通常为两表之间列的等值比较表达式。
ON	指定连接条件的关键字。

注意事项

- 所要进行 JOIN 连接的表必须是已经存在的表，否则会出错。
- attr_expr_list 中所涉及的字段只能是左表中的字段，否则会出错。
- 左半连接与左外连接的区别：左半连接只返回左表中符合 JOIN 条件的记录，而左外连接返回左表所有的记录，匹配不上 JOIN 条件的记录右表列返回 NULL。
- 左半连接的效果等价于 WHERE ... IN 或 WHERE EXISTS 子查询。

示例

返回选课学生的姓名及其所选的课程编号。

```
SELECT student_info.name, student_info.courseId FROM student_info LEFT SEMI JOIN course_info ON student_info.courseId = course_info.courseId;
```

等价写法（使用 WHERE IN）：

```
SELECT name, courseId FROM student_info WHERE courseId IN (SELECT courseId FROM course_info);
```

等价写法（使用 WHERE EXISTS）：

```
SELECT name, courseId FROM student_info WHERE EXISTS (SELECT 1 FROM course_info WHERE student_info.courseId = course_info.courseId);
```

13.4.8 不等值连接

功能描述

不等值连接中，多张表通过不相等的连接值进行连接，并返回满足不等式条件的结果集。

语法格式

```
SELECT attr_expr_list FROM table_reference
  JOIN table_reference ON non_equi_join_condition;
```

参数说明

参数	说明
attr_expr_list	查询列表达式列表，指定需要返回的列。
table_reference	进行连接的表名或子查询。必须是已存在的表，否则会出错。
JOIN	连接关键字，可以是 INNER JOIN、LEFT OUTER JOIN 等类型。
non_equi_join_condition	不等值连接条件，使用不等式运算符（如 <>、<、>、<=、>=）替代等号进行连接。

注意事项

- 所要进行JOIN连接的表必须是已经存在的表，否则会出错。

- 不等值连接的连接条件均为不等式条件，与等值连接（使用=比较）不同。
- 不等值连接可能导致结果集较大，请谨慎使用。

示例

返回student_info_1与 student_info_2两张表中的所有学生姓名对组合，但不包含相同姓名的姓名对。

```
SELECT student_info_1.name, student_info_2.name FROM student_info_1 JOIN student_info_2 ON student_info_1.name <> student_info_2.name;
```

使用范围条件的不等值连接示例：

```
SELECT a.name, b.name FROM student_info_1 a JOIN student_info_2 b ON a.score > b.score;
```

13.5 子句

13.5.1 FROM

功能描述

FROM子句用于指定查询的数据源。Spark 4.0增强了FROM子句，支持LATERAL子查询、TABLE关键字、VALUES行构造等多种数据源形式。

语法格式

```
FROM table_reference [, ...]
```

其中 table_reference 为：

```
{ table_name [ TABLE ] [ alias ]
  | join_table
  | LATERAL ( subquery ) [ alias ]
  | ( subquery ) [ alias ]
  | VALUES ( { expression } [, ...] ) [, ...] [ alias ]
  }
```

join_table 为：

```
table_reference [ join_type ] JOIN table_reference [ ON join_condition | USING ( column [, ...] ) ]
```

join_type 为：

```
{ INNER | CROSS | LEFT [ OUTER ] | RIGHT [ OUTER ] | FULL [ OUTER ] | LEFT SEMI | LEFT ANTI }
```

参数说明

参数	说明
table_name	表名，指定要查询的表。可使用TABLE关键字引用表值。
alias	为表或子查询指定别名。子查询必须指定别名。
LATERAL (subquery)	LATERAL子查询，允许子查询引用同一FROM子句中先前出现的表的列。Spark 4.0增强支持。
(subquery)	嵌套子查询，子查询结果作为中间过渡表。子查询必须指定别名。

参数	说明
VALUES	行构造器，直接构造行数据作为数据源。
join_type	连接类型，包括INNER、CROSS、LEFT OUTER、RIGHT OUTER、FULL OUTER、LEFT SEMI、LEFT ANTI。
ON join_condition	JOIN连接条件。
USING (column)	使用同名字段进行JOIN的简写方式。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- FROM嵌套子查询中，子查询必须指定别名。
- LATERAL子查询可以引用同一FROM子句中左侧表的列，普通子查询不能引用外部列。

示例

子查询作为数据源：

```
SELECT DISTINCT name FROM (SELECT name FROM student_info   JOIN course_info ON
student_info.courseId = course_info.courseId) temp;
```

LATERAL子查询引用外部列：

```
SELECT * FROM orders o,   LATERAL (SELECT * FROM order_items WHERE order_id = o.id) items;
```

VALUES构造数据：

```
SELECT * FROM VALUES (1, 'a'), (2, 'b'), (3, 'c') AS t(id, name);
```

LATERAL VIEW与explode配合：

```
SELECT name, course FROM student   LATERAL VIEW EXPLODE(courses) t AS course;
```

13.5.2 OVER

功能描述

OVER子句与窗口函数配合使用，用于定义窗口的分区、排序和范围。窗口函数为每行数据基于其所在窗口计算一个值，而不像普通聚合函数那样将多行合并为一行。

语法格式

```
window_func ( [ expression [, ...] ] ) OVER
 ( [ PARTITION BY expression [, ...] ]
   [ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] ]
   [ window_frame ] )
```

其中 window_frame 为：

```
{ ROWS | RANGE } BETWEEN frame_start AND frame_end
```


frame_start 和 frame_end 为：

```
{ CURRENT ROW
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | offset PRECEDING
  | offset FOLLOWING }
```

参数说明

参数	说明
window_func	窗口函数，包括聚合窗口函数（SUM、AVG、COUNT、MAX、MIN等）和排名窗口函数（RANK、DENSE_RANK、ROW_NUMBER、NTILE、LEAD、LAG、FIRST_VALUE、LAST_VALUE、NTH_VALUE等）。
PARTITION BY	按一个或多个表达式对数据进行分区，窗口函数在各分区内独立计算。类似于GROUP BY，但不合并行。
ORDER BY	决定窗口函数计算的顺序。可用ASC或DESC指定升序或降序。
ROWS	物理窗口，基于行号偏移定义窗口范围。
RANGE	逻辑窗口，基于ORDER BY表达式的值偏移定义窗口范围。
CURRENT ROW	当前行。
UNBOUNDED PRECEDING	窗口起点，从分区第一行开始。
UNBOUNDED FOLLOWING	窗口终点，到分区最后一行结束。
offset PRECEDING	从当前行向前偏移offset行（ROWS）或值（RANGE）。
offset FOLLOWING	从当前行向后偏移offset行（ROWS）或值（RANGE）。
NULLS FIRST	NULL值排在最前。
NULLS LAST	NULL值排在最后。

窗口范围示例

窗口范围	说明
ROWS BETWEEN CURRENT ROW AND CURRENT ROW	窗口只包含当前行
ROWS BETWEEN 3 PRECEDING AND 5 FOLLOWING	窗口从当前行向前3行到向后5行
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW	窗口从分区开头到当前行（默认窗口）
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING	窗口从当前行到分区结尾

窗口范围	说明
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING	窗口覆盖整个分区

ROWS 与 RANGE 的区别

- **ROWS**为物理窗口，根据行号偏移计算，与当前行的值无关。
- **RANGE**为逻辑窗口，根据ORDER BY表达式的值偏移计算，包含值在范围内的所有行。

注意事项

- OVER子句包括PARTITION BY、ORDER BY和窗口范围三部分，可组合使用。
- 不指定ORDER BY时，窗口默认覆盖整个分区。
- 不指定窗口范围时，默认窗口为ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW。
- OVER子句为空时表示窗口为整张表。

示例

累计计数：

```
SELECT id, COUNT(id) OVER (ORDER BY id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
  FROM over_test;
```

分区排名：

```
SELECT name, class_id, score,
  RANK() OVER (PARTITION BY class_id ORDER BY score DESC) AS class_rank
  FROM student;
```

移动平均：

```
SELECT date, amount,
  AVG(amount) OVER (ORDER BY date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg
  FROM sales;
```

使用窗口别名（Spark 4.0）：

```
SELECT name, score,
  RANK() OVER w AS rnk,
  SUM(score) OVER w AS total
  FROM student
  WINDOW w AS (PARTITION BY class_id ORDER BY score DESC);
```

13.5.3 WHERE

功能描述

WHERE子句用于指定过滤条件，仅返回满足条件的行。Spark 4.0增强了WHERE中的子查询能力，支持LATERAL子查询和相关子查询。

语法格式

```
SELECT [ ALL | DISTINCT ] projectItem [, ...] FROM table_reference [, ...]
  WHERE boolean_expression
```

其中 boolean_expression 可以包含子查询：

```
{ expression operator ( subquery )
  | [ NOT ] EXISTS ( subquery )
  | expression [ NOT ] IN ( subquery | value [, ...] ) }
```

参数说明

参数	说明
ALL	返回所有行，包括重复行。为默认选项。
DISTINCT	从结果集中移除重复的行。
boolean_expression	过滤条件表达式，结果为布尔值。
operator	比较运算符，包括 =, !=, <, >, <=, >= 等。
IN / NOT IN	判断值是否在子查询结果或值列表中。当使用子查询时，子查询必须返回单列。
EXISTS / NOT EXISTS	判断子查询是否返回行。EXISTS为真当子查询至少返回一行。子查询中可以引用外部查询的列（相关子查询）。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- 使用IN或NOT IN时，子查询的返回结果必须是单列。
- 使用EXISTS或NOT EXISTS时，子查询可以引用外部查询的列形成相关子查询。

示例

基本条件过滤：

```
SELECT name, score FROM student WHERE score > 90;
```

子查询作为过滤条件：

```
SELECT name FROM student_info
  WHERE courseId = (SELECT courseId FROM course_info WHERE courseName = 'Biology');
```

使用IN子查询：

```
SELECT name FROM student_info
  WHERE courseId IN (SELECT courseId FROM course_info WHERE credits > 3);
```

使用EXISTS相关子查询：

```
SELECT name FROM student_info s
  WHERE EXISTS (SELECT 1 FROM course_info c WHERE c.courseId = s.courseId AND c.courseName =
'Biology');
```

使用NOT EXISTS：


```
SELECT name FROM student_info s
  WHERE NOT EXISTS (SELECT 1 FROM course_info c WHERE c.courseId = s.courseId);
```

13.5.4 HAVING

功能描述

HAVING子句用于对GROUP BY分组后的结果进行过滤。与WHERE不同，HAVING可以使用聚合函数作为过滤条件。Spark 4.0增强了HAVING子句中的子查询能力。

语法格式

```
SELECT [ ALL | DISTINCT ] projectItem [, ...] FROM table_reference
  GROUP BY { expression [, ...] | GROUPING SETS (...) | ROLLUP (...) | CUBE (...) }
  HAVING boolean_expression
```

参数说明

参数	说明
ALL	返回所有行，包括重复行。为默认选项。
DISTINCT	从结果集中移除重复的行。
GROUP BY	按指定表达式分组，支持GROUPING SETS、ROLLUP、CUBE扩展。
HAVING	对分组后的结果进行过滤，条件中可以使用聚合函数和子查询。
boolean_expression	过滤条件表达式，通常包含聚合函数。可以包含子查询。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- HAVING中的子查询与聚合函数的位置不能互换，聚合函数应在比较运算符的左侧。
- HAVING与WHERE的区别：WHERE在分组前过滤行，HAVING在分组后过滤组。

示例

基本HAVING过滤：

```
SELECT class_id, AVG(score) AS avg_score FROM student
  GROUP BY class_id
  HAVING AVG(score) > 80;
```

子查询作为HAVING条件：

```
SELECT name FROM student_info
  GROUP BY name
  HAVING count(name) = (SELECT count(*) FROM course_info);
```

与ROLLUP配合使用：


```
SELECT group_id, job, SUM(salary) FROM group_test
  GROUP BY ROLLUP (group_id, job)
  HAVING SUM(salary) > 10000;
```

13.5.5 多层嵌套子查询

功能描述

多层嵌套子查询是指在子查询中嵌套子查询，即在一个 SELECT 语句的 FROM 子句中使用另一个 SELECT 语句作为数据源，可以多层嵌套。

语法格式

```
SELECT attr_expr FROM (
  SELECT attr_expr FROM (
    SELECT attr_expr FROM ...
  ) [alias]
) [alias];
```

LATERAL 子查询语法（Spark 4.0 支持）：

```
SELECT attr_expr FROM table_reference
  LATERAL VIEW OUTER explode(array_col) alias AS col_name;

SELECT attr_expr FROM table_reference,
  LATERAL (SELECT ... FROM ... WHERE ...) AS alias;
```

参数说明

参数	说明
attr_expr	查询列表达式，指定需要返回的列。
alias	子查询的别名。在嵌套查询中必须为每个子查询指定别名，否则会出错。
LATERAL VIEW	LATERAL VIEW 关键字，用于将表中某列展开为多行，常与 explode、json_tuple 等表生成函数配合使用。
LATERAL (subquery)	LATERAL 子查询，允许在 FROM 子句中引用前面表中的列。Spark 4.0 新增支持。
OUTER	可选关键字。当 LATERAL VIEW 展开结果为空时，保留原行并将展开列置为 NULL。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- 在嵌套查询中必须指定子查询的别名，否则会出错。
- 别名必须先定义后引用，建议别名不要重名。
- LATERAL 子查询可以引用 FROM 子句中出现在其前面的表的列。

示例

通过三次子查询，最终返回 user_info 中的 name 字段。


```
SELECT name
FROM (
  SELECT name, acc_num FROM (
    SELECT name, acc_num, password FROM (
      SELECT name, acc_num, password, bank_acc FROM user_info
    ) a
  ) b
) c;
```

LATERAL VIEW 示例，将数组列展开为多行：

```
SELECT name, course
FROM student_info
LATERAL VIEW OUTER explode(courses) courses AS course;
```

LATERAL 子查询示例（Spark 4.0）：

```
SELECT o.order_id, s.total
FROM orders o,
  LATERAL (SELECT SUM(amount) AS total FROM order_items i WHERE i.order_id = o.order_id) s;
```

13.6 别名 SELECT

13.6.1 列别名

功能描述

给列起别名。

语法格式

```
SELECT attr_expr [AS] alias, attr_expr [AS] alias, ... FROM table_reference;
```

关键字

- alias：用于对attr_expr中的字段名称起别名。
- AS：是否添加此关键字不会影响结果。

注意事项

- 所要查询的表必须是已经存在的，否则会出错。
- 别名的命名必须在别名的使用之前，否则会出错。此外，建议不要重名。

示例

先通过子查询SELECT name AS n FROM simple_table WHERE score > 90获得结果，在子查询中给name起的别名n可直接用于外部SELECT语句。

```
SELECT n FROM (SELECT name AS n FROM simple_table WHERE score > 90) m WHERE n = "xiaoming";
```

13.7 集合运算 select

13.7.1 UNION

功能描述

UNION返回多个查询结果的并集。UNION默认去除重复行，使用UNION ALL保留重复行。

语法格式

```
select_statement UNION [ ALL ] select_statement;
```

参数说明

参数	说明
UNION	集合并运算，将多个查询结果合并。默认去除重复行。
ALL	加上ALL关键字后保留重复行，即不去重。UNION ALL的性能通常优于UNION，因为不需要去重操作。

注意事项

- 每一个SELECT语句返回的列数必须相同。
- 对应列的数据类型必须兼容。
- 结果集的列名由第一个SELECT语句决定。
- UNION默认去重，UNION ALL不去重。在不需要去重的场景下，推荐使用UNION ALL以获得更好的性能。
- 多个集合运算（UNION、INTERSECT、EXCEPT）可以通过括号控制优先级（Spark 4.0）。

示例

UNION去重并集：

```
SELECT * FROM student_1 UNION SELECT * FROM student_2;
```

UNION ALL不去重并集：

```
SELECT * FROM student_1 UNION ALL SELECT * FROM student_2;
```

使用括号控制优先级（Spark 4.0）：

```
(SELECT * FROM student_1 UNION ALL SELECT * FROM student_2)
  INTERSECT SELECT * FROM student_3;
```

多表UNION：

```
SELECT name FROM student_1
  UNION ALL
SELECT name FROM student_2
  UNION ALL
SELECT name FROM student_3;
```


13.7.2 INTERSECT

功能描述

INTERSECT返回多个查询结果的交集，即同时存在于所有查询结果中的行。
INTERSECT默认去除重复行，使用INTERSECT ALL保留重复行。

语法格式

```
select_statement INTERSECT [ ALL ] select_statement;
```

参数说明

参数	说明
INTERSECT	集合交运算。返回同时存在于左侧和右侧查询结果中的行。
ALL	加上ALL关键字后保留重复行。不加ALL时默认去除重复行。

注意事项

- 两个SELECT语句返回的列数必须相同。
- 对应列的数据类型必须兼容。
- Spark 4.0支持INTERSECT ALL语法，保留重复行。
- 多个集合运算（UNION、INTERSECT、EXCEPT）可以通过括号控制优先级。

示例

基本INTERSECT交集（去重）：

```
SELECT * FROM student_1 INTERSECT SELECT * FROM student_2;
```

INTERSECT ALL保留重复行（Spark 4.0）：

```
SELECT * FROM student_1 INTERSECT ALL SELECT * FROM student_2;
```

使用括号控制优先级（Spark 4.0）：

```
(SELECT * FROM student_1 INTERSECT SELECT * FROM student_2)  UNION SELECT * FROM student_3;
```

13.7.3 EXCEPT

功能描述

EXCEPT返回两个查询结果的差集，即存在于第一个查询结果中但不存在于第二个查询结果中的行。EXCEPT默认去除重复行，使用EXCEPT ALL保留重复行。

语法格式

```
select_statement EXCEPT [ ALL ] select_statement;
```


参数说明

参数	说明
EXCEPT	集合差运算。返回左侧查询结果中存在但右侧查询结果中不存在的行。
ALL	加上ALL关键字后保留重复行。不加ALL时默认去除重复行。

注意事项

- 每一个SELECT语句返回的列数必须相同。
- 对应列的数据类型必须兼容。
- Spark 4.0支持EXCEPT ALL语法，保留重复行。
- 多个集合运算（UNION、INTERSECT、EXCEPT）可以通过括号控制优先级。

示例

基本EXCEPT差集（去重）：

```
SELECT * FROM student_1 EXCEPT SELECT * FROM student_2;
```

EXCEPT ALL保留重复行（Spark 4.0）：

```
SELECT * FROM student_1 EXCEPT ALL SELECT * FROM student_2;
```

使用括号控制优先级（Spark 4.0）：

```
(SELECT * FROM student_1 EXCEPT SELECT * FROM student_2)
  UNION SELECT * FROM student_3;
```

13.8 CASE...WHEN

功能描述

CASE...WHEN是条件表达式，用于根据条件返回不同的值。Spark SQL支持简单CASE函数和CASE搜索函数两种形式。

语法格式

简单CASE函数：依据expression与value1的匹配结果跳转到相应的result1。

```
CASE expression
  WHEN value1 THEN result1
  [ WHEN value2 THEN result2 ] ...
  [ ELSE else_result ]
 END
```

CASE搜索函数：按指定顺序为每个WHEN子句的condition1求值。返回第一个取值为TRUE的condition1的result1。

```
CASE
 WHEN condition1 THEN result1
 [ WHEN condition2 THEN result2 ] ...
```



```
  [ ELSE else_result ]
END
```

参数说明

参数	说明
expression	简单CASE中被比较的表达式。
WHEN value	简单CASE中，当expression等于value时，返回对应的THEN结果。
WHEN condition	CASE搜索函数中，当condition为真时，返回对应的THEN结果。
THEN result	满足条件时返回的结果。所有THEN结果的类型应一致。
ELSE else_result	所有WHEN条件都不满足时返回的结果。如省略ELSE，则返回NULL。

注意事项

- 简单CASE函数使用等值比较（ = ），CASE搜索函数支持任意布尔条件。
- CASE表达式从上到下依次判断，返回第一个满足条件的THEN结果，后续条件不再判断。
- 所有THEN和ELSE返回值的类型应一致或可隐式转换。
- CASE表达式可以用于SELECT列表、WHERE条件、ORDER BY、GROUP BY等任意需要表达式的地方。

示例

- **简单CASE函数**

  返回表student中的字段name及与class_id相匹配的字符。

  ```
  SELECT name,
    CASE class_id
      WHEN 1 THEN 'Class A'
      WHEN 2 THEN 'Class B'
      WHEN 3 THEN 'Class C'
      ELSE 'Other'
    END AS class_name
    FROM student;
  ```

- **CASE搜索函数**

  对表student进行查询，返回字段name及与score对应的结果，score大于等于90返回EXCELLENT，score在（80,90）之间的返回GOOD，否则返回BAD。

  ```
  SELECT name,
   CASE
    WHEN score >= 90 THEN 'EXCELLENT'
    WHEN 80 < score AND score < 90 THEN 'GOOD'
    ELSE 'BAD'
   END AS level
   FROM student;
  ```


- **CASE在ORDER BY中使用**

```
SELECT name, score FROM student
  ORDER BY CASE WHEN score >= 60 THEN 0 ELSE 1 END, score DESC;
```

- **CASE在聚合中使用**

```
SELECT class_id,
  COUNT(CASE WHEN score >= 90 THEN 1 END) AS excellent_count,
  COUNT(CASE WHEN score < 60 THEN 1 END) AS fail_count
  FROM student
  GROUP BY class_id;
```

13.9 Pipe 语法（|>）

功能描述

Pipe语法（管道操作符 |>）是Spark 4.0的SQL语法扩展，允许以管道（pipeline）方式组织SQL查询。Pipe语法将查询从嵌套的函数调用风格改为从上到下的线性流程，使复杂查询更易读写。

Pipe语法与现有SQL语法完全兼容，可以在任何查询中混合使用。

语法格式

```
FROM table_source
|> WHERE condition
|> SELECT select_expr [, ...]
|> ORDER BY col_name [ASC|DESC] [, ...]
|> LIMIT num_rows
```

参数说明

操作符	语法	说明
FROM / TABLE	FROM 或 TABLE	从源表开始，返回所有行
SELECT	|> SELECT [[AS] alias], ...	计算表达式，生成新列（不支持聚合函数，聚合用 AGGREGATE）
EXTEND	|> EXTEND [[AS] alias], ...	在输入表基础上追加新列
SET	|> SET = , ...	替换已有列的值
DROP	|> DROP , ...	删除指定列
AS	|> AS	为输入表引入新别名
WHERE	|> WHERE	过滤行（无需单独的 HAVING 或 QUALIFY）
LIMIT	|> [LIMIT] [OFFSET]	限制返回行数和跳过行数
AGGREGATE	|> AGGREGATE <agg_expr> GROUP BY <group_expr>	聚合操作（全表或分组）

操作符	语法	说明
JOIN	|> [LEFT|RIGHT|FULL|CROSS|SEMI|ANTI] JOIN ON	连接操作
ORDER BY	|> ORDER BY [ASC|DESC], ...	排序
UNION ALL	|> UNION ALL	合并结果集
TABLESAMPLE	|> TABLESAMPLE ({ROWS|PERCENT})	采样
PIVOT	|> PIVOT (agg_expr FOR col IN (val1, ...))	行转列
UNPIVOT	|> UNPIVOT (value_col FOR key_col IN (col1, ...))	列转行

注意事项

- Pipe语法以FROM子句开头。
- 每个 |> 操作符处理前一步的输出，形成管道链。
- WHERE 操作符可用于任何位置，无需单独的 HAVING 或 QUALIFY。
- Pipe语法不改变查询的语义，只是提供了不同的书写方式。
- EXTEND/JOIN操作符在简单内联查询上可能触发 toAttribute on unresolved object 内部错误，建议使用 SELECT * FROM (subquery) AS alias \|> ... 的子表别名方式。
- |> SET 操作符仅支持单段式键（如 key_name），不支持多段式配置键（如 spark.sql.ansi.enabled）。

示例 1：基本管道查询

```
FROM employees
|> WHERE department = 'Engineering'
|> SELECT name, salary
|> ORDER BY salary DESC
|> LIMIT 10;
```

等价于：

```
SELECT name, salary
FROM employees
WHERE department = 'Engineering'
ORDER BY salary DESC
LIMIT 10;
```

示例 2：使用 EXTEND 添加计算列

```
VALUES (0), (1) tab(col)
|> EXTEND col * 2 AS result;
```

结果：

col	result
0	0
1	2

示例 3：使用 SET 修改列值

```
VALUES (0), (1) tab(col)
|> SET col = col * 2;
```

结果：

col
0
2

示例 4：使用 AGGREGATE 分组聚合

```
-- 全表聚合
VALUES (0), (1) tab(col)
|> AGGREGATE COUNT(col) AS count;
-- 分组聚合
VALUES (0, 1), (0, 2) tab(col1, col2)
|> AGGREGATE COUNT(col2) AS count GROUP BY col1;
```

示例 5：使用 JOIN

```
SELECT 0 AS a, 1 AS b
|> AS lhs
|> JOIN VALUES (0, 2) rhs(a, b) ON (lhs.a = rhs.a);
```

示例 6：复杂管道（TPC-H Q13 改写）

标准SQL：

```
SELECT c_count, COUNT(*) AS custdist
FROM
  (SELECT c_custkey, COUNT(o_orderkey) c_count
   FROM customer
   LEFT OUTER JOIN orders ON c_custkey = o_custkey
     AND o_comment NOT LIKE '%unusual%packages%' GROUP BY c_custkey
   ) AS c_orders
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
```

管道语法：

```
FROM customer
|> LEFT OUTER JOIN orders ON c_custkey = o_custkey    AND o_comment NOT LIKE '%unusual%packages
%'
|> AGGREGATE COUNT(o_orderkey) c_count    GROUP BY c_custkey
|> AGGREGATE COUNT(*) AS custdist    GROUP BY c_count
|> ORDER BY custdist DESC, c_count DESC;
```


核心特性

独立性与互操作性

- 每个管道操作符接收一个输入表，独立地对行进行操作。
- 对于任何N个管道操作符的有效链，前M个（ M ≤ N ）操作符也构成有效查询。
- 可以在任何有效的标准SQL查询后面追加管道操作符。
- 表子查询可以使用标准SQL语法或管道语法编写。
- 视图、DDL、DML语句中都可以包含管道语法查询。

投影操作

- SELECT：计算新表达式，生成新表。
- EXTEND：在保留原列的基础上追加新列。
- SET：替换已有列的值。
- DROP：删除指定列。

聚合操作

- AGGREGATE 合并了 GROUP BY + SELECT 的功能，无需重复表达式。
- 分组表达式支持直接在 GROUP BY 中指定别名。
- 输出列自动包含分组列和聚合列。

14 导出查询结果

功能描述

INSERT OVERWRITE DIRECTORY用于将查询结果直接写入到指定的目录，支持按CSV、Parquet、ORC、JSON、Avro格式进行存储。

语法格式

```
INSERT OVERWRITE DIRECTORY path
  USING file_format
  [OPTIONS(key1=value1)]
  select_statement;
```

关键字

- USING：指定所存储格式。
- OPTIONS：导出时的属性列表，为可选项。

参数

表 14-1 INSERT OVERWRITE DIRECTORY 参数描述

参数	描述
path	要将查询结果写入的OBS路径。
file_format	写入的文件格式，支持按CSV、Parquet、ORC、JSON、Avro格式。

注意事项

- 通过配置“spark.sql.shuffle.partitions”参数可以设置非Managed表在OBS桶中插入的文件个数，同时，为了避免数据倾斜，在INSERT语句后可加上“distribute by rand()”，可以增加处理作业的并发量。例如：
  `INSERT INTO TABLE table_target SELECT * FROM table_source DISTRIBUTE BY cast(rand() * N AS INT);`
- 配置项为OPTIONS('DELIMITER'=',')时，可以指定分隔符，默认值为“,”。
  对于CSV数据，支持如下所述分隔符：

- 制表符tab，例如：'DELIMITER'='\t'。
- 支持通过unicode编码指定分隔符，例如：'DELIMITER'='\u0001'。
- 单引号（'），单引号必须在双引号（" "）内。例如：'DELIMITER'= "'"。

示例

```
INSERT OVERWRITE DIRECTORY 'obs://bucket/dir'
  USING csv
  OPTIONS(key1=value1)
  select * from db1.tb1;
```


15 内置函数

15.1 日期函数

15.1.1 add_months

add_months函数用于计算日期值增加指定月数后的日期。即start_date在num_months个月之后的date。

命令格式

```
add_months(string start_date, int num_months)
```


参数说明

表 15-1 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE或STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
num_months	是	INT	代表需要增加月的数量。

返回值说明

返回开始日期start_date增加num_months个月后的日期，返回值格式为yyyy-mm-dd。

返回值date类型的日期值。

说明

- start_date非DATE或STRING时，返回报错，错误信息：data type mismatch。
- start_date为DATE或STRING，但不符合日期值的入参格式时，返回NULL。
- start_date值为NULL时，返回报错。
- num_months值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select add_months('2023-02-26',3);
  ```
  返回2023-05-26。
- 示例2
  ```
  select add_months('2023-02-14 21:30:00',3);
  ```
  返回2023-05-14。
- 示例3
  ```
  select add_months('2023-08-15 20:00:00',null);
  ```
  返回NULL。

15.1.2 current_date

current_date函数用于返回当前日期值。返回值格式为yyyy-mm-dd。

相似函数：**15.1.15 getdate**，getdate函数用于返回当前系统时间。返回值格式为yyyy-mm-dd hh:mi:ss。

命令格式

```
current_date()
```


参数说明

无

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd

示例代码

```
select current_date();
```

返回2023-08-16。

15.1.3 current_timestamp

CURRENT_TIMESTAMP函数用于返回当前时间戳。

命令格式

```
current_timestamp()
```

参数说明

无

返回值说明

返回TIMESTAMP类型的时间戳。

示例代码

```
select current_timestamp();
```

返回1692002816300。

15.1.4 date_add

date_add函数用于计算按照days幅度递增start_date日期的天数。

如需要获取当前日期基础上指定变动幅度的日期，可结合**15.1.2 current_date**或**15.1.15 getdate**函数共同使用。

请注意date_add函数与**15.1.6 date_sub**函数逻辑相反。

命令格式

```
date_add(string startdate, int days)
```


参数说明

表 15-2 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE 或 STRING	代表起始日期。 支持以下格式： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
days	是	BIGINT	代表需要增加天的数量。 ● days大于0，则为增加天数。 ● days小于0，则减去天数。 ● days等于0，即加0天，日期不变。 ● days值为NULL时，返回NULL。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- start_date非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- start_date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- start_date值为NULL时，返回NULL。
- days值为NULL时，返回NULL。

示例代码

- 示例1
```
select date_add('2023-02-28 00:00:00', 1);
```
返回2023-03-01。加1天，结果超出当年2月份的最后1天，实际值为下个月的第1天。
- 示例2
```
select date_add(date '2023-02-28', -1);
```
返回2023-02-27。减1天。
- 示例3
```
select date_add('2023-02-28 00:00:00', 20);
```
返回2023-03-20。
- 示例4
```
select date_add(getdate(),-1);
```
假设当前时间为2023-08-14 16:00:00，返回2023-08-13。
- 示例5
```
select date_add('2023-02-28 00:00:00', null);
```
返回NULL。

15.1.5 dateadd1

dateadd1函数用于按照指定的单位datepart和幅度delta修改date的值。

如需要获取当前日期基础上指定变动幅度的日期，可结合15.1.2 current_date或15.1.15 getdate函数共同使用。

命令格式

```
dateadd1(string date, bigint delta, string datepart)
```

参数说明

表 15-3 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
delta	是	BIGINT	代表修改幅度。
datepart	是	STRING	代表修改的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回STRING类型的日期值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- delta或datepart值为NULL时，返回NULL。

示例代码

返回2023-08-15 17:00:00。加1天。

```
select dateadd1( '2023-08-14 17:00:00', 1, 'dd');
```

返回2025-04-14 17:00:00。加20个月，月份溢出，年份加2。

```
select dateadd1('2023-08-14 17:00:00', 20, 'mm');
```

返回2023-09-14 17:00:00。

```
select dateadd1('2023-08-14 17:00:00', 1, 'mm');
```

返回2023-09-14。

```
select dateadd1('2023-08-14', 1, 'mm');
```

假设当前时间为2023-08-14 17:00:00，返回2023-08-13 17:00:00。

```
select dateadd1(getdate(),-1,'dd');
```

返回NULL。

```
select dateadd1(date '2023-08-14', 1, null);
```

15.1.6 date_sub

date_sub函数按照days幅度递减start_date日期的天数。

如需要获取当前日期基础上指定变动幅度的日期，可结合**15.1.2 current_date**或**15.1.15 getdate**函数共同使用。

请注意date_sub函数与**15.1.4 date_add**函数逻辑相反。

命令格式

```
date_sub(string startdate, int days)
```

参数说明

表 15-4 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE 或 STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

参数	是否必选	参数类型	说明
days	是	INT	代表需要减少的天数。 ● 当days大于0时，从start_date中减少指定的天数。 ● 当days小于0时，向start_date中增加指定的天数。 ● 当days等于0时，即加0天，日期不变。 ● 当days值为NULL时，返回NULL。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- start_date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- start_date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- start_date值为NULL时，返回NULL。
- days值为NULL时，返回NULL。

示例代码

- 示例1
```
select date_sub('2023-08-14 17:00:00', 2);
```
返回2023-08-12。减2天。
- 示例2
```
select date_sub(date'2023-08-14', -1);
```
返回2023-08-15。增1天。
- 示例3
```
select date_sub(getdate(),1);
```
假设当前时间为2023-08-14 17:00:00，返回2023-08-13。
- 示例4
```
select date_sub('2023-08-14 17:00:00', null);
```
返回NULL。

15.1.7 date_format

date_format函数用于将date按照format指定的格式转换为字符串。

命令格式

```
date_format(string date, string format)
```


参数说明

表 15-5 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表待转换的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的格式。 格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • MM代表月份。 • dd代表天。 • HH代表24小时制时。 • hh代表12小时制时。 • mm代表分钟。 • ss代表秒。

返回值说明

按指定类型返回STRING类型的日期。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回2023-08-14。

```
select date_format('2023-08-14','yyyy-MM-dd');
```

返回2023-08。

```
select date_format('2023-08-14','yyyy-MM')
```

返回20230814。

```
select date_format('2023-08-14','yyyyMMdd')
```


15.1.8 datediff

datediff函数用于计算两个时间date1、date2的日期差值。

相似函数：**15.1.9 datediff1**，datediff1函数用于计算两个时间date1、date2的差值，将差值以指定的时间单位datepart表示。

命令格式

```
datediff(string date1, string date2)
```

参数说明

表 15-6 参数说明

参数	是否必选	参数类型	说明
date1	是	DATE 或 STRING	被减数日期。 格式： yyyy-mm-dd yyyy-mm-dd hh:mi:ss yyyy-mm-dd hh:mi:ss.ff3
date2	是	DATE 或 STRING	减数日期。 格式： yyyy-mm-dd yyyy-mm-dd hh:mi:ss yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回BIGINT类型。

说明

- date1、date2非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date1、date2为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- 如果date1小于date2，返回值为负数。
- date1或date2值为NULL时，返回NULL。

示例代码

返回10。

```
select datediff('2023-06-30 00:00:00', '2023-06-20 00:00:00');
```

返回11。

```
select datediff(date '2023-05-21', date '2023-05-10');
```

返回NULL。


```
select datediff(date '2023-05-21', null);
```

15.1.9 datediff1

datediff1函数用于计算两个时间date1、date2的差值，将差值以指定的时间单位datepart表示。

相似函数：**15.1.8 datediff**，datediff函数用于计算两个时间date1、date2的日期差值，不支持指定返回的时间单位。

命令格式

```
datediff1(string date1, string date2, string datepart)
```

参数说明

表 15-7 参数说明

参数	是否必选	参数类型	说明
date1	是	DATE 或 STRING	计算两个时间date1、date2的日期差值中的被减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
date2	是	DATE 或 STRING	计算两个时间date1、date2的日期差值的减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
datepart	是	STRING	代表需要返回的时间单位。 参数datepart支持扩展的日期格式： 年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BIGINT类型。

说明

- date1、date2非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- date1、date2为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- 如果date1小于date2，返回值为负数。
- date1或date2值为NULL时，返回NULL。
- datepart值为NULL时，返回NULL。

示例代码

返回14400。

```
select datediff1('2023-06-30 00:00:00', '2023-06-20 00:00:00', 'mi');
```

返回10。

```
select datediff1(date '2023-06-21', date '2023-06-11', 'dd');
```

返回NULL。

```
select datediff1(date '2023-05-21', date '2023-05-10', null);
```

返回NULL。

```
select datediff1(date '2023-05-21', null, 'dd');
```

15.1.10 datepart1

datepart1函数用于计算日期date中符合指定时间单位datepart1的值。

命令格式

```
datepart1(string date,string datepart)
```

参数说明

表 15-8 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表起始日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

参数	是否必选	参数类型	说明
datepart	是	STRING	代表需要返回的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BIGINT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- date值为NULL时，返回NULL。
- datepart值为NULL时，返回NULL。

示例代码

返回2023。

```
select datepart1(date '2023-08-14 17:00:00', 'yyyy');
```

返回2023。

```
select datepart1('2023-08-14 17:00:00', 'yyyy');
```

返回59。

```
select datepart1('2023-08-14 17:59:59', 'mi')
```

返回NULL。

```
select datepart1(date '2023-08-14', null);
```

15.1.11 datetrunc

datetrunc函数用于计算将日期date按照datepart指定的时间单位进行截取后的日期值。

截取datepart之前的部分，除截取的部分外自动填充为默认值。可参考**示例代码**。

命令格式

```
datetrunc (string date, string datepart)
```


参数说明

表 15-9 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表起始日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
datepart	是	STRING	代表需要返回的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回DATE或STRING类型的日期值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- datepart值为NULL时，返回NULL。
- 当datepart指定为时（hh）、分（mi）、秒（ss）时，按天（DD）级别截取返回。

示例代码

静态数据示例

- 示例1：
  ```
  select datetrunc('2023-08-14 17:00:00', 'yyyy');
  ```
  返回2023-01-01 00:00:00。
- 示例2：
  ```
  select datetrunc('2023-08-14 17:00:00', 'month');
  ```
  返回2023-08-01 00:00:00。
- 示例3：
  ```
  select datetrunc('2023-08-14 17:00:00', 'DD');
  ```
  返回2023-08-14。

- 示例4：
  ```
  select datetrunc('2023-08-14', 'yyyy');
  ```
  返回2023-01-01。
- 示例5：
  ```
  select datetrunc('2023-08-14 17:11:11', 'hh');
  ```
  返回2023-08-14 17:00:00。
- 示例6：
  ```
  select datetrunc('2023-08-14', null);
  ```
  返回None。

15.1.12 day/dayofmonth

day函数用于返回指定日期的天。

命令格式

```
day(string date)、dayofmonth(string date)
```

参数说明

表 15-10 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING类型	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- **示例1**
  ```
  select day('2023-08-01');
  ```
  返回1。
- **示例2**
  ```
  select day(null);
  ```
  返回NULL。

15.1.13 from_unixtime

from_unixtime函数用于计算将数字型的UNIX值代表的时间戳转换为日期值。

命令格式

```
from_unixtime(bigint unixtime)
```

参数说明

表 15-11 参数说明

参数	是否必选	参数类型	说明
unixtime	是	BIGINT	UNIX格式的时间戳。代表需要转换的时间戳。 此处参数应填正常UNIX格式时间戳前十位。

返回值说明

将时间戳转换为时间格式，返回STRING类型的日期值，格式为“yyyy-MM-dd HH:mm:ss”或“yyyyMMddHHmmss.uuuuuu”或 "yyyyMMdd"。

说明

unixtime值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select from_unixtime(1692149997);
  ```
  返回2023-08-16 09:39:57。
- 示例2
  ```
  select from_unixtime(NULL);
  ```
  返回NULL。

15.1.14 from_utc_timestamp

from_utc_timestamp函数用于计算将UTC的时间戳转化为timezone所对应的本地时间戳。

命令格式

```
from_utc_timestamp(string timestamp, string timezone)
```


参数说明

表 15-12 参数说明

参数	是否必选	参数类型	说明
timestamp	是	DATE、TIMESTAMP或STRING	UTC时间戳。 格式： yyyy-MM-dd yyyy-MM-dd HH:mm:ss yyyy-MM-dd HH:mm:ss.SSS。
timezone	是	STRING	代表需要转换的目标时区。

返回值说明

返回TIMESTAMP类型的时间戳。

说明

- timestamp非DATE、TIMESTAMP或STRING类型时，返回报错，错误信息：data type mismatch；
- timestamp为DATE、TIMESTAMP或STRING类型时，但不符合日期值的入参格式时，返回NULL；
- timestamp值为NULL时，返回NULL。
- timezone值为NULL时，返回NULL。

示例代码

返回1691978400000（代表2023-08-14 10:00:00）。

```
select from_utc_timestamp('2023-08-14 17:00:00','PST');
```

返回1691917200000（代表2023-08-13 17:00:00）。

```
select from_utc_timestamp(date '2023-08-14 00:00:00','PST');
```

返回NULL。

```
select from_utc_timestamp('2023-08-13',null);
```

15.1.15 getdate

getdate函数用于返回当前系统时间。返回值格式为yyyy-mm-dd hh:mi:ss。

相似函数：**15.1.2 current_date**，current_date函数用于返回当前日期值。返回值格式为yyyy-mm-dd。

命令格式

```
getdate()
```


参数说明

无

返回值说明

返回STRING类型的日期值，格式为yyyy-mm-dd hh:mi:ss。

示例代码

假设当前时间为2023-08-10 10:54:00，返回2023-08-10 10:54:00。

```
select getdate();
```

15.1.16 hour

hour函数用于返回指定时间的小时，范围为0到23。

命令格式

```
hour(string date)
```

参数说明

表 15-13 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select hour('2023-08-10 10:54:00');
  ```
  返回10。
- 示例2
  ```
  select hour('12:00:00');
  ```


返回12。

- 示例3
  ```
  select hour(null);
  ```
- 返回NULL。

15.1.17 isdate

isdate函数用于判断一个日期字符串能否根据指定的格式转换为一个日期值。

命令格式

```
isdate(string date , string format)
```

参数说明

表 15-14 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要判断的字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，会隐式转换为STRING类型后参与运算。 date参数为任意字符串格式。
format	是	STRING	代表需要转换的目标日期格式。 STRING类型常量，不支持日期扩展格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BOOLEAN类型的值。

说明

date或format值为NULL时，返回NULL。

示例代码

返回true。


```
select isdate('2023-08-10','yyyy-mm-dd');
```

返回false。

```
select isdate(123456789,'yyyy-mm-dd');
```

15.1.18 last_day

last_day函数用于返回date所在月份的最后一天。

相似函数：**15.1.19 last_day1**，lastday函数用于返回date所在月的最后一天，截取到天，时分秒部分为00:00:00。

命令格式

```
last_day(string date)
```

参数说明

表 15-15 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select last_day('2023-08-15');
  ```
  返回2023-08-31。
- 示例2
  ```
  select last_day('2023-08-10 10:54:00');
  ```
  返回2023-08-31。

15.1.19 last_day1

last_day1函数用于返回date所在月的最后一天，截取到天，时分秒部分为00:00:00。

相似函数：15.1.18 last_day，last_day函数用于返回date所在月份的最后一天。返回值格式为：yyyy-mm-dd。

命令格式

```
last_day1(string date)
```

参数说明

表 15-16 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回STRING类型的日期值。格式为yyyy-mm-dd hh:mi:ss。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 返回2023-08-31。
  ```
  select last_day1('2023-08-10');
  ```
- 返回2023-08-31 00:00:00。
  ```
  select last_day1('2023-08-10 10:54:00');
  ```
- 返回NULL。
  ```
  select last_day1(null);
  ```

15.1.20 minute

minute函数用于返回指定时间的分钟，范围为0到59。

命令格式

```
minute(string date)
```


参数说明

表 15-17 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select minute('2023-08-10 10:54:00');
  ```
  返回54。
- 示例2
  ```
  select minute('10:54:00');
  ```
  返回54。
- 示例3
  ```
  select minute(null);
  ```
  返回NULL。

15.1.21 month

month函数用于返回指定时间的月份，范围为1至12月。

命令格式

```
month(string date)
```

参数说明

表 15-18 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 date取值为STRING类型格式时，至少要包含yyyy-mm-dd。且不含多余的字符。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select month('2023-08-10 10:54:00');
  ```
  返回8。
- 示例2
  ```
  select month(null);
  ```
  返回NULL。

15.1.22 months_between

months_between函数用于返回date1与date2之间的月份差。

命令格式

```
months_between(string date1, string date2[, roundOff])
```

参数说明

表 15-19 参数说明

参数	是否必选	参数类型	说明
date1	是	DATE 或 STRING	代表被减数。 格式为： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
date2	是	DATE 或 STRING	代表减数。 格式为： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
roundOff	否	BOOLEAN	是否对结果进行四舍五入，默认为false。

返回值说明

返回DOUBLE类型的值。

说明

- date1、date2非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date1、date2为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- 当date1晚于date2时，返回值为正。当date2晚于date1时，返回值为负。
- 当date1和date2分别对应两个月的最后一天，返回整数月；否则计算方式为date1减去date2的天数除以31天，结果是否取整由roundOff参数决定。
- date1或date2值为NULL时，返回NULL。

示例代码

- 返回0.0563172。
  ```
  select months_between('2023-08-16 10:54:00', '2023-08-14 17:00:00');
  ```
- 返回0.06451613。
  ```
  select months_between('2023-08-16','2023-08-14');
  ```
- 返回NULL。
  ```
  select months_between('2023-08-16',null);
  ```
- 返回3.0。

  说明：2025-05-31和2025-02-28分别是各自月份的最后一天，所以返回整数月3
  ```
  select months_between('2025-05-31','2025-02-28')
  ```
- 返回3.0。

  说明：2025-05-28和2025-02-28的天数差为89天，89除以31等于2.87，向上取整后为3。


```
select months_between('2025-05-28','2025-02-28')
```

15.1.23 next_day

next_day函数用于返回起始日期之后最接近指定星期的日期。

命令格式

```
next_day(string start_date, string day_of_week)
```

参数说明

表 15-20 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE 或 STRING	代表起始日期。 start_date取值为STRING类型格式时，至少要包含yyyy-mm-dd且不含多余的字符串。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
day_of_week	是	STRING	星期名称的前2个或3个字母缩写，或者星期的全名。例如TU代表星期二。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- start_date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- start_date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- start_date值为NULL时，返回NULL。
- day_of_week值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select next_day('2023-08-16','TU');
  ```
  返回2023-08-22。
- 示例2
  ```
  select next_day('2023-08-16 10:54:00','TU');
  ```
  返回2023-08-22。
- 示例3
  ```
  select next_day('2023-08-16 10:54:00','WE');
  ```
  返回2023-08-23。

- 示例4
  ```
  select next_day('20230816 20:00:00',null);
  ```
  返回NULL。

15.1.24 quarter

quarter函数用于返回该date所在的季度，范围为1~4。

命令格式

```
quarter(string date)
```

参数说明

表 15-21 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回3。

```
select quarter('2023-08-16 10:54:00');
```

返回3。

```
select quarter('2023-08-16');
```

返回NULL。

```
select quarter(null);
```

15.1.25 second

second函数用于返回指定时间的秒，范围为0到59。

命令格式

```
second(string date)
```

参数说明

表 15-22 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 示例1
  ```
  select second('2023-08-16 10:54:36');
  ```
  返回36。
- 示例2
  ```
  select second('10:54:36');
  ```
  返回36。
- 示例3
  ```
  select second(null);
  ```
  返回NULL。

15.1.26 to_char1

to_char1函数用于将日期按照指定格式转换为字符串。

命令格式

```
to_char1(string date, string format)
```


参数说明

表 15-23 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的目标日期格式。 STRING类型常量，不支持日期扩展格式。 format格式为代表年月日时分秒的时间单位与任意字符的组合，其中： ● yyyy代表年份。 ● mm代表月份。 ● dd代表天。 ● hh代表小时。 ● mi代表分钟。 ● ss代表秒。

返回值说明

返回STRING类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回静态数据示例2023-08*16。

```
select to_char1('2023-08-16 10:54:36', '静态数据示例yyyy-mm*dd');
```

返回20230816。

```
select to_char1('2023-08-16 10:54:36', 'yyyymmdd');
```

返回NULL。

```
select to_char1('静态数据示例2023-08-16', '静态数据示例yyyy-mm*dd');
```

返回NULL。

```
select to_char1('20230816', 'yyyy');
```


返回NULL。

```
select to_char1('2023-08-16 10:54:36', null);
```

15.1.27 to_date

to_date函数用于返回时间中的年月日。

相似函数：**15.1.28 to_date1**，to_date1函数用于将指定格式的字符串转换为日期值，支持指定转换的日期格式。

命令格式

```
to_date(string timestamp)
```

参数说明

表 15-24 参数说明

参数	是否必选	参数类型	说明
timestamp	是	DATE 或 STRING	代表待处理的时间。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- timestamp非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- timestamp为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。

示例代码

返回2023-08-16。

```
select to_date('2023-08-16 10:54:36');
```

返回NULL。

```
select to_date(null);
```

15.1.28 to_date1

to_date1函数用于将指定格式的字符串转换为日期值。

相似函数：**15.1.27 to_date**，to_date函数用于返回时间中的年月日，不支持指定转换的日期格式。

命令格式

```
to_date1(string date, string format)
```

参数说明

表 15-25 参数说明

参数	是否必选	参数类型	说明
date	是	STRING	要转换的字符串。 格式： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的日期格式。 STRING类型常量，不支持日期扩展格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： ● yyyy代表年份。 ● mm代表月份。 ● dd代表天。 ● hh代表小时。 ● mi代表分钟。 ● ss代表秒。

返回值说明

返回DATE类型的日期值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回yyyy-mm-dd格式的日期值。

示例代码

返回2023-08-16 10:54:36。

```
select to_date1('2023-08-16 10:54:36','yyyy-mm-dd hh:mi:ss');
```

返回2023-08-16 00:00:00。

```
select to_date1('2023-08-16','yyyy-mm-dd');
```

返回NULL。


```
select to_date1(null);
```

返回2023-08-16。

```
select to_date1('2023-08-16 10:54:36');
```

15.1.29 to_utc_timestamp

to_utc_timestamp函数用于将timezone所对应的时间戳转换为UTC的时间戳。

命令格式

```
to_utc_timestamp(string timestamp, string timezone)
```

参数说明

表 15-26 参数说明

参数	是否必选	参数类型	说明
timestamp	是	DATE、TIMESTAMP或STRING	代表待转换的时间戳。 格式： yyyy-MM-dd yyyy-MM-dd HH:mm:ss yyyy-MM-dd HH:mm:ss.SSS
timezone	是	STRING	代表需要转换的目标时区。

返回值说明

返回BIGINT类型值。

说明

- timestamp非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- timestamp为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- timestamp值为NULL时，返回NULL。
- timezone值为NULL时，返回NULL。

示例代码

返回1692028800000。

```
select to_utc_timestamp('2023-08-14 17:00:00','PST');
```

返回null。

```
select to_utc_timestamp(null, 'PST');
```

15.1.30 trunc

trunc函数用于将date按照特定的格式进行清零操作。

清零操作即返回默认值，年、月、日的默认值为01，时、分、秒、毫秒 的默认值为00。

命令格式

```
trunc(string date, string format)
```

参数说明

表 15-27 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的目标日期格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： ● yyyy代表年份。 ● MM代表月份。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回2023-08-01。

```
select trunc('2023-08-16', 'MM');
```

返回2023-08-01。

```
select trunc('2023-08-16 10:54:36', 'MM');
```

返回NULL。

```
 select trunc(null, 'MM');
```


15.1.31 trunc_numeric

trunc_numeric函数用于将输入值number截取到指定小数点位置。

命令格式

```
trunc_numeric(<number>[, bigint<decimal_places>])
```

参数说明

表 15-28 参数说明

参数	是否必选	参数类型	说明
number	是	DOUBLE、BIGINT、DECIMAL、STRING	需要截取的数据。
decimal_places	否	BIGINT	默认为0，截取到小数点后的第N位。

返回值说明

返回DOUBLE或DECIMAL类型。

说明

返回规则如下：

- number为DOUBLE、DECIMAL类型时会返回相应的类型。
- number为STRING、BIGINT类型时，返回DOUBLE类型。
- decimal_places非BIGINT类型时，返回报错。
- number值为NULL时，返回NULL。

示例代码

- 示例1

  返回 3.141。

  ```
  select trunc_numeric(3.1415926, 3);
  ```

- 示例2

  返回 3。

  ```
  select trunc_numeric(3.1415926);
  ```

15.1.32 unix_timestamp

unix_timestamp函数用于将日期值转化为数字型的UNIX格式的日期值。

函数返回值将返回正常UNIX格式时间戳前十位。

命令格式

```
unix_timestamp(string timestamp, string pattern)
```

参数说明

表 15-29 参数说明

参数	是否必选	参数类型	说明
timestamp	否	DATE或STRING	代表待转换的日期值。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
pattern	否	STRING	代表需要转换的格式。 pattern为空时，默认为yyyy-mm-dd hh:mi:ss格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BIGINT类型的值。

说明

- timestamp值为NULL时，返回NULL。
- timestamp和pattern都为空时，返回从“1970-01-01 00:00:00”到现在的秒数代表的时间戳。

示例代码

- 返回1692149997。
  ```
  select unix_timestamp('2023-08-16 09:39:57')
  ```
- 假设当前系统时间为2023-08-16 10:23:16，返回1692152596。
  ```
  select unix_timestamp();
  ```

15.1.33 weekday

weekday函数用于返回日期值所在周的第几天（周一为0）

命令格式

```
weekday (string date)
```

参数说明

表 15-30 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- 周一作为一周的第一天，返回值为0。其他日期依次递增，周日返回6。
- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回2。

```
select weekday ('2023-08-16 10:54:36');
```

返回NULL。

```
select weekday (null);
```

15.1.34 weekofyear

weekofyear函数用于返回指定日期是一年中的第几周，范围为0到53。

命令格式

```
weekofyear(string date)
```


参数说明

表 15-31 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 返回33。
```
select weekofyear('2023-08-16 10:54:36');
```
- 返回NULL。
```
select weekofyear(null);
```

15.1.35 year

year函数用于返回指定日期中的年份。

命令格式

```
year(string date)
```

参数说明

表 15-32 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

- 返回2023。
```
select year('2023-08-16 10:54:36');
```
- 返回NULL。
```
select year(null);
```

15.2 字符串函数

15.2.1 ascii

ascii函数用于返回字符串str第一个字符的ASCII码。

命令格式

```
ascii(string )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会自动转换为STRING类型后参与运算。 例如“ABC”

返回值说明

返回BIGINT的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

- 返回字符串ABC第一个字符的ASCII码。命令示例如下。

  返回65。
```
select ascii('ABC');
```


- 输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select ascii(null);
  ```

15.2.2 concat

concat函数用于拼接数组或字符串。

命令格式

输入为ARRAY数组：将多个ARRAY数组中的所有元素连接在一起，生成一个新的ARRAY数组。

```
concat(array , array [,...])
```

输入为字符串：将多个字符串连接在一起，生成一个新的字符串。

```
concat(string , string [,...])
```

参数说明

输入为ARRAY数组

参数	是否必选	参数类型	说明
a、b	是	ARRAY	ARRAY数组。array中的T指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。a和b中元素的数据类型必须一致。数组中的元素为NULL值时会参与运算。

输入为字符串

参数	是否必选	参数类型	说明
str1、str2	是	STRING	字符串。如果输入参数为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会自动转换为STRING类型后参与运算，其他类型会返回报错。

返回值说明

返回ARRAY数组或STRING的值。

说明

- 返回ARRAY类型。如果任一输入ARRAY数组为NULL，返回结果为NULL。
- 返回STRING类型。如果没有参数或任一参数为NULL，返回结果为NULL。

示例代码

- 连接ARRAY数组array(1, 2)和array(2, -2)。命令示例如下。
  返回[1,2,2,-2]。
  ```
  select concat(array(1, 2), array(2, -2));
  ```
- 连接字符串ABC和DEF。命令示例如下。
  返回ABCDEF。
  ```
  select concat('ABC','DEF');
  ```
- 输入为空。命令示例如下。
  返回NULL。
  ```
  select concat();
  ```
- 任一字符串输入为NULL。命令示例如下。
  返回NULL。
  ```
  select concat('abc', 'def', null);
  ```

15.2.3 concat_ws

连接多个字符串，字符串之间以指定的分隔符分隔。

命令格式

```
concat_ws(string <separator>, string <str1>, string <str2>[,...])
```

或

```
concat_ws(string <separator>, array<string> <a>)
```

参数说明

参数	是否必选	参数类型	说明
separator	是	STRING	分隔符。
str1、str2	是	STRING	至少要指定2个字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
a	是	ARRAY	数组中的元素为STRING类型。

返回值说明

返回STRING类型的值。

说明

- str1或str2非STRING、BIGINT、DECIMAL、DOUBLE或DATETIME类型时，返回报错。
- 如果参数（待拼接字符）为NULL，则会忽略这个参数。
- 如果没有输入参数（待拼接字符），返回NULL。

示例代码

将字符串ABC和DEF通过:连接，返回ABC:DEF。

```
select concat_ws(':','ABC','DEF');
```

任一输入参数为NULL，返回avg:18。

```
select concat_ws(':','avg',null,'18');
```

将ARRAY数组array('name', 'lilei')中的元素通过:连接，返回name:lilei。

```
select concat_ws(':',array('name', 'lilei'));
```

15.2.4 char_matchcount

char_matchcount函数用于计算str1中有多少个字符出现在str2中。

命令格式

```
char_matchcount(string , string )
```

参数说明

参数	是否必选	参数类型	说明
str1、str2	是	STRING	待计算的字符串str1、str2。

返回值说明

返回BIGINT类型。

说明

str1或str2值为NULL时，返回NULL。

示例代码

返回3。

```
select char_matchcount('abcz','abcde');
```

返回NULL。

```
select char_matchcount(null,'abcde');
```

15.2.5 encode

encode函数用于使用charset的编码方式对str进行编码。

命令格式

```
encode(string , string )
```


参数说明

参数	是否必选	参数类型	说明
str	是	STRING	至少要指定2个字符串。 STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
charset	是	STRING	编码格式。 取值范围为：UTF-8、UTF-16、UTF-16LE、UTF-16BE、ISO-8859-1、US-ASCII。

返回值说明

返回BINARY类型的值。

说明

str或charset值为NULL时，返回NULL。

示例代码

- 将字符串abc按照UTF-8格式编码。命令示例如下。

  返回abc。

  ```
  select encode("abc", "UTF-8");
  ```

- 任一输入参数为NULL。命令示例如下。

  返回结果为NULL。

  ```
  select encode("abc", null);
  ```

15.2.6 find_in_set

find_in_set函数用于查找字符串str1在以逗号（,）分隔的字符串str2中的位置，从1开始计数。

命令格式

```
find_in_set(string , string )
```

参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待查找的字符串。
str2	是	STRING	以逗号（,）分隔的字符串。

返回值说明

返回BIGINT类型的值。

说明

- 当str2中无法匹配到str1或str1中包含逗号（,）时，返回0。
- 当str1或str2值为NULL时，返回NULL。

示例代码

- 查找字符串ab在字符串abc,123,ab,c中的位置。命令示例如下。
  返回3。
  ```
  select find_in_set('ab', 'abc,123,ab,c');
  ```
- 查找字符串hi在字符串abc,123,ab,c中的位置。命令示例如下。
  返回0。
  ```
  select find_in_set('hi', 'abc,123,ab,c');
  ```
- 任一输入参数为NULL。命令示例如下。
  返回NULL。
  ```
  select find_in_set(null, 'abc,123,ab,c');
  ```

15.2.7 get_json_object

get_json_object函数用于根据所给路径对json对象进行解析，当json对象非法时将返回NULL。

命令格式

```
get_json_object( json_string , path  )
```

参数说明

参数	是否必选	参数类型	说明
json_string	是	STRING	标准的JSON格式对象，格式为{Key:Value, Key:Value,...}
path	是	STRING	JSONPath 表达式，用于指定要提取的元素位置。不同字符的含义如下： ● $ 表示根对象。 ● .key 表示访问对象的属性。 ● []： 下标操作符，用于提取JSON数组中的特定元素（从0开始计数），如 $.tags[1]。

返回值说明

返回STRING类型的值。

说明

- 如果json为空或非法的json格式，返回NULL。
- 如果json合法，path也存在，则返回对应字符串。

示例代码

- 提取JSON对象src_json.json中的信息。示例数据如下。

```
jsonString = {"store": {"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}], "bicycle":
{"price":19.95,"color":"red"} }, "email":"amy@only_for_json_udf_test.net", "owner":"Tony" }
```

从上述示例中提取owner字段信息，返回Tony。

```
select get_json_object(jsonString, '$.owner');
```

提取store.fruit字段第一个数组信息，返回\{"weight":8,"type":"apple"\}。

```
select get_json_object(jsonString, '$.store.fruit[0]');
```

提取不存在的字段信息，返回NULL。

```
select get_json_object(jsonString, '$.non_exist_key');
```

- 提取数组型JSON对象的信息。命令示例如下。

返回22。

```
select get_json_object('{"array":[["a",11],["b",22],["c",33]]}','$.array[1][1]');
```

返回["h00","h11","h22"]。

```
select get_json_object('{"a":"b","c":{"d":"e","f":"g","h":["h00","h11","h22"]},"i":"j"}','$.c.h[*]');
```

返回["h00","h11","h22"]。

```
select get_json_object('{"a":"b","c":{"d":"e","f":"g","h":["h00","h11","h22"]},"i":"j"}','$.c.h');
```

返回h11。

```
select get_json_object('{"a":"b","c":{"d":"e","f":"g","h":["h00","h11","h22"]},"i":"j"}','$.c.h[1]');
```

- 提取带有.的JSON对象中的信息。命令示例如下。

创建一张表。

```
create table json_table (id string, json string);
```

向表中插入数据，Key带 "."

```
insert into table json_table (id, json) values ("1", "{\"city1\":{\"region\":{\"rid\":6}}}");
```

向表中插入数据，Key不带 "."

```
insert into table json_table (id, json) values ("2", "{\"city1\":{\"region\":{\"rid\":7}}}");
```

取rid的值，查询key为city1，返回6。由于包含.，只能用\['']\来解析。

```
select get_json_object(json, "$['city1'].region['id']") from json_table where id =1;
```

取rid的值，查询key为city1，返回7。查询方法有如下两种。

```
select get_json_object(json, "$['city1'].region['id']") from json_table where id =2;      select
get_json_object(json, "$.city1.region['id']") from json_table where id =2;
```

- JSON输入为空或非法格式。命令示例如下。

返回NULL。

```
select get_json_object('','$.array[2]');
```

返回NULL。

```
select get_json_object('"array":["a",1],"b":["c",3]','$.array[1][1]');
```

- JSON字符串涉及转义。命令示例如下。

返回"3"。

```
select get_json_object('{"a":"\\"3\\"","b":"6"}', '$.a');
```

返回'3'。


```
select get_json_object('{"a":"\'3\'","b":"6"}', '$.a');
```

- 一个JSON对象中可以出现相同的Key，可以成功解析。

  返回1。

  ```
  select get_json_object('{"b":"1","b":"2"}', '$.b');
  ```

- 输出结果按照JSON字符串的原始排序方式输出。

  返回{"b":"3","a":"4"}。

  ```
  select get_json_object('{"b":"3","a":"4"}', '$');
  ```

15.2.8 initcap

initcap函数用于将文本字符串转换成首字母大写其余字母小写的形式。

命令格式

```
initcap(string A)
```

参数说明

参数	是否必选	参数类型	说明
A	是	STRING	待转换的文本字符串。

返回值说明

返回一个STRING类型字符串，字符串中每个单词首字母大写，其余变为小写。

示例代码

返回Fli Sql

```
SELECT initcap("fLI sql");
```

15.2.9 instr

instr函数用于返回substr在str中最早出现的下标。

当参数中出现NULL时，返回NULL，当str中不存在substr时返回0，注意下标从1开始。

相似函数：instr1，instr1函数用于计算子串str2在字符串str1中的位置，instr1函数支持指定起始搜索位置和匹配次数。

命令格式

```
instr(string , string )
```


参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待搜索的目标字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
substr	是	STRING	待匹配的子串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。

返回值说明

返回BIGINT类型的值。

说明

- 如果在str1中未找到str2，则返回0。
- 如果str2为空串，则总能匹配成功，例如select instr('abc','');会返回1。
- str1或str2值为NULL时，返回NULL。

示例代码

- 计算字符b在字符串abc中的位置。命令示例如下。

  返回2。

  ```
  select instr('abc', 'b');
  ```

- 任一输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select instr('abc', null)
  ```

15.2.10 instr1

instr1函数用于计算子串str2在字符串str1中的位置。

相似函数：instr，instr函数用于返回substr在str中最早出现的下标。但是instr不支持指定起始搜索位置和匹配次数。

命令格式

```
instr1(string <str1>, string <str2>[, bigint <start_position>[, bigint <nth_appearance>]])
```

参数说明

表 1 参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待搜索的目标字符串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
str2	是	STRING	待匹配的子串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
start_position	否	BIGINT	表示从str1的第几个字符开始搜索，默认起始位置是第一个字符位置1。不支持指定负数。
nth_appearance	否	BIGINT	表示str2在str1中第nth_appearance次匹配的位置。如果nth_appearance为其他类型或小于等于0，则返回报错。

返回值说明

返回BIGINT类型。

说明

- 如果在str1中未找到str2，则返回0。
- 如果str2为空串，则总能匹配成功。
- str1、str2、start_position或nth_appearance值为NULL时，返回NULL。

示例代码

返回 10

```
select instr1('Tech on the net', 'h', 5, 1);
```

返回2。

```
select instr1('abc', 'b');
```

返回NULL。

```
select instr1('abc', null);
```

15.2.11 keyvalue

keyvalue函数用于计算将字符串str按照split1进行切分，并按split2将每组变成Key-Value对，返回key所对应的Value。

命令格式

```
keyvalue(string <str>,[string <split1>,string <split2>,] string <key>)
```

参数说明

表 1 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待拆分的字符串。
split1、split2	否	STRING	用于作为分隔符的字符串，按照指定的两个分隔符拆分源字符串。如果表达式中没有指定这两项，默认split1为";"，split2为":"。当某个被split1拆分后的字符串中有多个split2时，返回结果未定义。
key	是	STRING	将字符串按照split1和split2拆分后，返回key值对应的Value。

返回值说明

返回STRING类型。

说明

- split1或split2值为NULL时，返回NULL。
- str或key值为NULL或没有匹配的key时，返回NULL。
- 如果有多个Key-Value匹配，返回第一个匹配上的key对应的Value。

示例代码

返回2。

```
select keyvalue('a:1;b:2', 'b');
```

15.2.12 length

length函数用于返回字符串的长度。

相似函数：lengthb，lengthb函数用于计算字符串str以字节为单位的长度，返回STRING类型的值。

命令格式

```
length(string )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待计算长度的字符串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。

返回值说明

返回BIGINT类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

- 计算字符串abc的长度。命令示例如下。
  返回3。
  ```
  select length('abc');
  ```
- 输入参数为NULL。命令示例如下。
  返回NULL。
  ```
  select length(null);
  ```

15.2.13 lengthb

lengthb函数用于计算字符串str以字节为单位的长度。

相似函数：length，length函数用于返回字符串的长度，返回BIGINT类型的值。

命令格式

```
lengthb(string )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	输入的字符串。

返回值说明

返回BIGINT类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

返回5。

```
select lengthb('hello');
```

返回NULL。

```
select lengthb(null);
```

15.2.14 levenshtein

levenshtein函数用于返回两个字符串之间的Levenshtein距离，如levenshtein('kitten','sitting') =3。

说明

Levenshtein距离，是编辑距离的一种。指两个字串之间，由一个转成另一个所需的最少编辑操作次数。

命令格式

```
levenshtein(string A, string B)
```

参数说明

参数	是否必选	参数类型	说明
A、B	是	STRING	计算Levenshtein距离需要输入的字符串。

返回值说明

返回INT类型的值。

示例代码

返回3

```
SELECT levenshtein('kitten','sitting');
```

15.2.15 locate

在str中查找substr的位置，从1开始计数。可以通过start_pos指定开始查找的位置。

命令格式

```
locate(string <substr>, string <str>[, bigint <start_pos>])
```

参数说明

参数	是否必选	参数类型	说明
substr	是	STRING	待匹配的子串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
str	是	STRING	待搜索的目标字符串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
start_pos	否	BIGINT	指定查找的起始位置，从1开始。不指定时默认为1。

返回值说明

返回INT类型的值。

说明

- str中无法匹配到substr时，返回0。
- str或substr值为NULL时，返回NULL。
- start_pos值为NULL时，返回0。

示例代码

查找字符串ab在字符串abhiab中的位置，返回1。

```
select locate('ab', 'abhiab');
```

从第2个位置开始查找，返回5。

```
select locate('ab', 'abhiab', 2);
```

start_pos为NULL，返回0。

```
select locate('ab', 'abhiab', null);
```

查找字符串hi在字符串hanmeimei and lilei中的位置，返回0。

```
select locate('hi', 'hanmeimei and lilei');
```

15.2.16 lower/lcase

lower函数用于将文本字符串转换成字母全部小写的形式。

命令格式

```
lower(string A) / lcase(string A)
```

参数说明

参数	是否必选	参数类型	说明
A	是	STRING	待转换的文本字符串。

返回值说明

返回为STRING类型的值。

说明

- 入参非 STRING、BIGINT、DOUBLE、DECIMAL 或 DATETIME 类型时，返回报错。
- 入参值为NULL时，返回NULL。

示例代码

将字符串中的大写字符转换为小写字符。命令示例如下。

返回 abc。

```
select lower('ABC');
```

输入参数为NULL。命令示例如下。

返回NULL。

```
select lower(null);
```

15.2.17 lpad

返回指定长度的字符串。给定字符串str长度小于指定长度len时，由指定字符pad从左侧填补；给定字符串str长度大于指定长度len时，从左截取len个字符。

命令格式

```
lpad(string <str>, int <len>[, string <pad>])
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待补位的字符串。
len	是	INT	结果字符串的长度。
pad	否	STRING	用于补位的字符串。不指定时默认为空格。

返回值说明

返回STRING类型的值。

说明

- 如果len小于str的字符数，则返回str从左开始截取len位的字符串。
- 如果len为0，则返回空串。
- 如果没有输入参数或任一输入参数值为NULL，返回NULL。

示例代码

用字符串ZZ将字符串abcdefgh向左补足到10位，返回ZZabcdefgh。

```
select lpad('abcdefgh', 10, 'ZZ');
```

用字符串ZZ将字符串abcdefgh向左补足到5位，返回abcde。

```
select lpad('abcdefgh', 5, 'ZZ');
```

length为0，返回空串。

```
select lpad('abcdefgh', 0, 'ZZ');
```

任一输入参数为NULL，返回NULL。

```
select lpad(null, 0, 'ZZ');
```


15.2.18 ltrim

ltrim函数用于从str的左端去除字符：

- 如果未指定trimChars，则默认去除空格。
- 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的左端去除尽可能长的所有字符都在集合trimChars中的子串。

相似函数：

- rtrim，rtrim函数用于从str的右端去除字符。
- trim，trim函数用于从str的左右两端去除字符。

命令格式

```
ltrim([,] string )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待去除左端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
trimChars	是	STRING	待去除的字符。

返回值说明

返回为STRING类型。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str或trimChars值为NULL时，返回NULL。

示例代码

- 去除字符串" abc"的左边空格。命令示例如下。

  返回字符串abc 。

  ```
  select ltrim('  abc');
  ```

  等效于如下语句。

  ```
  select trim(leading from '  abc');
  ```

  leading代表去除字符串前面的空格。
- 输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select ltrim(null);  select ltrim('xy', null);  select ltrim(null, 'xy');
  ```


- 去除字符串yxlucyxx左端所有字符都在集合xy中的子串。
  返回lucyxx，只要左端遇到x或者y就会被去掉。

  ```
  select ltrim( 'xy','yxlucyxx');
  ```

  等效于如下语句。

  ```
  select trim(leading 'xy' from 'yxlucyxx');
  ```

15.2.19 parse_url

返回给定URL的指定部分。partToExtract的有效值包括HOST、PATH、QUERY、REF、PROTOCOL、AUTHORITY、FILE和USERINFO。当第二个参数为QUERY时，可以使用第三个参数提取特定参数的值。

命令格式

```
parse_url(string <urlString>, string <partToExtract [, string <keyToExtract>])
```

参数说明

参数	是否必选	参数类型	说明
urlString	是	STRING	URL链接。
partToExtract	是	STRING	取值包含：HOST、PATH、QUERY、REF、PROTOCOL、AUTHORITY、FILE和USERINFO，不区分大小写。
keyToExtract	否	STRING	当partToExtract取值为QUERY时，根据key值取出对应的Value值。

返回值说明

返回STRING类型的值。

说明

- urlString、partToExtract或keyToExtract值为NULL时，返回NULL。
- partToExtract取值不符合要求时，返回NULL。

示例代码

返回example.com。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'HOST');
```

返回/over/there/index.dtb。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'PATH');
```

返回animal。


```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'QUERY', 'type');
```

返回nose。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'REF');
```

返回file。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'PROTOCOL');
```

返回username@example.com:666。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'AUTHORITY');
```

返回username。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'USERINFO');
```

15.2.20 regexp_count1

计算source中从start_position位置开始，匹配指定pattern的子串数。

命令格式

```
regexp_count1(string <source>, string <pattern>[, bigint <start_position>])
```

参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待搜索的字符串。
pattern	是	STRING	正则表达式。pattern为空串时返回报错。
start_position	否	BIGINT	搜索的开始位置，必须大于0。不指定时默认为1，表示从source的第一个字符开始匹配。

返回值说明

返回BIGINT类型的值。

说明

- 如果没有匹配成功，返回0。
- source或pattern值为NULL时，返回NULL。

示例代码

返回4。

```
select regexp_count1('ab0a1a2b3c', '[0-9]');
```

返回3。


```
select regexp_count1('ab0a1a2b3c', '[0-9]', 4);
```

返回NULL。

```
select regexp_count1('ab0a1a2b3c', null);
```

15.2.21 regexp_extract

将字符串source按照pattern的分组规则进行字符串匹配，返回第groupid个组匹配到的字符串内容。

命令格式

```
regexp_extract(string <source>, string <pattern>[, bigint <groupid>])
```

参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待匹配的字符串。
pattern	是	STRING	正则表达式。待匹配的模型。
groupid	否	BIGINT	BIGINT类型常量，必须大于等于0。不指定时默认为1，表示返回第一个组。如果groupid等于0，则返回满足整个pattern的子串。

返回值说明

返回STRING类型的值。

说明

- 如果pattern为空串或pattern中没有分组，返回报错。
- groupid非BIGINT类型或小于0时，返回报错。
- source、pattern或groupid值为NULL时，返回NULL。

示例代码

将basketball按照bas(.*?)(ball)拆分，返回ket。

```
select regexp_extract('basketball', 'bas(.*?)(ball)');
```

返回basketball。

```
select regexp_extract('basketball', 'bas(.*?)(ball)', 0);
```

返回99。在SQL中需要使用两个"\"作为转义字符。

```
select regexp_extract('8d99d8', '8d(\\d+)d8');
```


15.2.22 regexp_instr1

计算字符串source从start_position开始，与pattern第occurrence次匹配的子串的起始或结束位置。

命令格式

```
regexp_instr1(string <source>, string <pattern>[, bigint <start_position>[, bigint <occurrence>[, bigint <return_option>]]])
```

参数说明

参数	是否必选	参数类型	说明
source	是	STRING	源字符串。
pattern	是	STRING	正则表达式。待匹配的模型。pattern为空串时返回报错。
start_position	否	BIGINT	搜索的开始位置。不指定时默认值为1。
occurrence	否	BIGINT	指定匹配次数。不指定时默认值为1，表示搜索第一次出现的位置。
return_option	否	BIGINT	指定返回的位置类型。值为0或1，不指定时默认值为0。0表示返回匹配的开始位置，1表示返回匹配的结束位置。

返回值说明

返回BIGINT类型的值。return_option指定匹配的子串在source中的开始或结束位置。

说明

- 如果pattern为空串，返回报错。
- start_position或occurrence非BIGINT类型或小于等于0时，返回报错。
- source、pattern、start_position、occurrence或return_option值为NULL时，返回NULL。

示例代码

返回6。

```
select regexp_instr1('a1b2c3d4', '[0-9]', 3, 2);
```

返回NULL。

```
select regexp_instr1('a1b2c3d4', null, 3, 2);
```


15.2.23 regexp_replace

regexp_replace函数用于将source字符串中匹配pattern的子串替换成指定字符串replacement后，返回结果字符串。Spark 4.0版本支持position参数，可指定从第N次匹配开始替换。

命令格式

```
regexp_replace(str, pattern, replacement[, position])
```

参数说明

表 1 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待替换的字符串。
pattern	是	STRING	正则表达式模式。pattern为空串时返回报错。
replacement	是	STRING	将匹配pattern的子串替换为该字符串。
position	否	BIGINT	指定从第position次匹配开始替换之后的所有匹配。为0或1时替换所有匹配。为其他类型或小于0时，返回报错。默认值为0（替换所有匹配）。

返回值说明

返回STRING类型的值。

说明

- 如果pattern为空串，返回报错。
- 当引用不存在的组时，不进行替换。
- 如果replacement值为NULL且pattern有匹配，返回NULL。
- str、pattern或position值为NULL时，返回NULL。

示例代码

替换所有匹配，返回num-num。

```
SELECT regexp_replace('100-200', '(\\d+)', 'num');
```

替换所有匹配，返回2222。

```
SELECT regexp_replace('abcd', '[a-z]', '2');
```

从第2次匹配开始替换，返回a222。

```
SELECT regexp_replace('abcd', '[a-z]', '2', 2);
```

从第3次匹配开始替换，返回ab22。


```
SELECT regexp_replace('abcd', '[a-z]', '2', 3);
```

从第4次匹配开始替换，返回abc2。

```
SELECT regexp_replace('abcd', '[a-z]', '2', 4);
```

15.2.24 regexp_replace1

regexp_replace1函数用于将source字符串中第occurrence次匹配pattern的子串，替换成指定字符串replace_string后，返回结果字符串。

说明

regexp_replace1函数只适用于Spark 2.4.5及之前的版本。

相似函数：**regexp_replace**，**regexp_replace**函数针对不同的Spark版本，功能略有差异，请参考**regexp_replace**查看详细的功能说明。

命令格式

```
regexp_replace1(string , string  , string [, bigint ])
```

参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待替换的字符串。
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。pattern为空串时返回报错。
replace_string	是	STRING	将匹配pattern的字符串替换后的字符串。
occurrence	否	BIGINT	必须大于等于1，表示将第occurrence次匹配的字符串替换为replace_string，为1时表示替换所有匹配的子串。为其他类型或小于1时，返回报错。默认值为1。

返回值说明

返回STRING类型的值。

说明

- 当引用不存在的组时，不进行替换。
- 如果replace_string值为NULL且pattern有匹配，返回NULL。
- 如果replace_string值为NULL但pattern不匹配，返回NULL。
- source、pattern或occurrence值为NULL时，返回NULL。

示例代码

返回 2222。


```
select regexp_replace1('abcd', '[a-z]', '2');
```

返回 2bcd。

```
select regexp_replace1('abcd', '[a-z]', '2', 1);
```

返回 a2cd。

```
select regexp_replace1('abcd', '[a-z]', '2', 2);
```

返回 ab2d。

```
select regexp_replace1('abcd', '[a-z]', '2', 3);
```

返回 abc2。

```
select regexp_replace1('abcd', '[a-z]', '2', 4);
```

15.2.25 regexp_substr1

计算从start_position位置开始，source中第occurrence次匹配指定pattern的子串。

命令格式

```
regexp_substr1(string <source>, string <pattern>[, bigint <start_position>[, bigint <occurrence>]])
```

参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待搜索的字符串。
pattern	是	STRING	正则表达式。待匹配的模型。pattern为空串时返回报错。
start_position	否	BIGINT	起始位置，必须大于0。不指定时默认为1，表示从source的第一个字符开始匹配。
occurrence	否	BIGINT	匹配次数，必须大于0。不指定时默认为1，表示返回第一次匹配的子串。

返回值说明

返回STRING类型的值。

说明

- 如果pattern为空串，返回报错。
- 没有匹配时，返回NULL。
- start_position或occurrence非BIGINT类型或小于等于0时，返回报错。
- source、pattern、start_position或occurrence值为NULL时，返回NULL。

示例代码

返回a。


```
select regexp_substr1('a1b2c3', '[a-z]');
```

返回b。

```
select regexp_substr1('a1b2c3', '[a-z]', 2, 1);
```

返回c。

```
select regexp_substr1('a1b2c3', '[a-z]', 2, 2);
```

返回NULL。

```
select regexp_substr1('a1b2c3', null);
```

15.2.26 repeat

repeat函数用于返回将str重复n次后的字符串。

命令格式

```
repeat(string , bigint )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待重复的字符串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
n	是	BIGINT	重复次数。

返回值说明

返回STRING类型。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- n为空时，返回报错。
- str或n值为NULL时，返回NULL。

示例代码

将字符‘123’重复2次，返回 123123。

```
SELECT repeat('123', 2);
```

15.2.27 replace

用 new 字符串替换 str 字符串中与 old 字符串完全重合的部分并返回替换后的字符串。如果没有重合的字符串，返回原 str。

命令格式

```
replace(string <str>, string <old>, string <new>)
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待替换的字符串。
old	是	STRING	待比较的字符串。
new	是	STRING	替换后的字符串。

返回值说明

返回 STRING 类型。

说明

如果任一输入参数值为 NULL，返回 NULL。

示例代码

返回 AA123AA。

```
SELECT replace('abc123abc', 'abc', 'AA');
```

返回 NULL。

```
SELECT replace('abc123abc', null, 'AA');
```

15.2.28 reverse

reverse函数用于返回倒序字符串。

命令格式

```
reverse(string )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待倒序的字符串。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

返回值说明

返回STRING类型。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

返回 LQS krapS。

```
SELECT reverse('Spark SQL');
```

返回[3,4,1,2]。

```
SELECT reverse(array(2, 1, 4, 3));
```

15.2.29 rpad

将字符串 str1 向右补足到 length 位，使用字符串 str2 进行补位。

命令格式

```
rpad(string <str1>, int <length>, string <str2>)
```

参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待向右补位的字符串。
length	是	INT	向右补位后的目标长度。
str2	是	STRING	用于补位的字符串。

返回值说明

返回 STRING 类型。

说明

- 如果 length 小于 str1 的长度，则返回 str1 从左开始截取 length 位的字符串。
- 如果 length 为 0，则返回空串。
- 如果任一输入参数值为 NULL，返回 NULL。

示例代码

返回 hi???。

```
SELECT rpad('hi', 5, '??');
```

返回 h。

```
SELECT rpad('hi', 1, '??');
```


15.2.30 rtrim

rtrim函数用于从str的右端去除字符：

- 如果未指定trimChars，则默认去除空格字符。
- 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的右端去除尽可能长的所有字符都在集合trimChars中的子串。

相似函数：

- ltrim，ltrim函数用于从str的左端去除字符。
- trim，trim函数用于从str的左右两端去除字符。

命令格式

```
rtrim([, ]string )
```

或

```
rtrim(trailing [] from )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待去除右端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
trimChars	是	STRING	待去除的字符。

返回值说明

返回为STRING类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str或trimChars值为NULL时，返回NULL。

示例代码

- 去除字符串 yxabcxx 的右边空格。命令示例如下。

  返回字符串 yxabcxx。

  ```
  select rtrim('yxabcxx     ');
  ```

  等效于如下语句。

  ```
  select trim(trailing from ' yxabcxx ');
  ```

- 去除字符串yxabcxx右端所有字符都在集合xy中的子串。

  返回yxabc，只要右端遇到x或者y就会被去掉。

  ```
  select rtrim('xy', 'yxabcxx');
  ```


等效于如下语句。

```
select trim(trailing 'xy' from 'yxabcxx');
```

- 输入参数为NULL。命令示例如下。

  返回NULL。

```
select rtrim(null);      select rtrim('yxabcxx', 'null');
```

15.2.31 space

space函数用于返回指定数量的空格。

命令格式

```
space(bigint )
```

参数说明

参数	是否必选	参数类型	说明
n	是	BIGINT	用于指定空格数量。

返回值说明

返回STRING类型。

说明

- n为空时，返回报错。
- n值为NULL时，返回NULL。

示例代码

返回6。

```
select length(space(6));
```

15.2.32 split_part1

split_part函数用于依照分隔符separator拆分字符串str，返回从start部分到end部分的子串（闭区间）。

命令格式

```
split_part1(string <str>, string <separator>, bigint <start>[, bigint <end>])
```

参数说明

表 1 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待拆分的字符串。
separator	是	STRING	STRING类型常量。拆分用的分隔符，可以是一个字符，也可以是一个字符串。
start	是	BIGINT	BIGINT类型常量，必须大于0。表示返回段的开始编号（从1开始）。
end	否	BIGINT	BIGINT类型常量，大于等于start。表示返回段的截止编号，可省略，缺省时表示和start取值相等，返回start指定的段。

返回值说明

返回STRING类型的值。

说明

- 如果start的值大于切分后实际的分段数，例如字符串拆分完有4个片段，start大于4，返回空串。
- 如果separator不存在于str中，且start指定为1，返回整个str。如果str为空串，则输出空串。
- 如果separator为空串，则返回原字符串str。
- 如果end大于片段个数，返回从start开始的子串。
- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- 除separator外，如果任一参数值为NULL，返回NULL。

示例代码

返回aa。

```
select split_part1('aa,bb,cc,dd', ',', 1);
```

返回aa,bb。

```
select split_part1('aa,bb,cc,dd', ',', 1, 2);
```

返回空串。

```
select split_part1('aa,bb,cc,dd', ',', 10);
```

返回aa,bb,cc,dd。

```
select split_part1('aa,bb,cc,dd', ':', 1);
```

返回空串。

```
select split_part1('aa,bb,cc,dd', ':', 2);
```

返回aa,bb,cc,dd。

```
select split_part1('aa,bb,cc,dd', '', 1);
```

返回bb,cc,dd。


```
select split_part1('aa,bb,cc,dd', ',', 2, 6);
```

返回None。

```
select split_part1('aa,bb,cc,dd', ',', null);
```

15.2.33 substr/substring

返回字符串 str 从 start_position 开始、长度为 length 的子串。substr 与 substring 功能相同。

命令格式

```
substr(string <str>, bigint <start_position>[, bigint <length>])
```

或

```
substring(string <str>, bigint <start_position>[, bigint <length>])
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待截取子串的字符串。
start_position	是	BIGINT	起始位置，从 1 开始。取值为正时从左边开始，取值为负时从右边开始。
length	否	BIGINT	子串的长度值，必须大于 0。省略时返回到 str 结尾的子串。

返回值说明

返回 STRING 类型。

说明

- 当 length 被省略时，返回到 str 结尾的子串。
- str、start_position 或 length 值为 NULL 时，返回 NULL。

示例代码

返回 k SQL。

```
SELECT substr('Spark SQL', 5);
```

返回 SQL。

```
SELECT substr('Spark SQL', -3);
```

返回 k。

```
SELECT substr('Spark SQL', 5, 1);
```


15.2.34 substring_index

substring_index函数用于截取字符串str第count个分隔符之前的字符串。如果count为正，则从左边开始截取。如果count为负，则从右边开始截取。

命令格式

```
substring_index(string <str>, string <separator>, int <count>)
```

参数说明

表 1 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待截取的字符串。
separator	是	STRING	STRING类型的分隔符。
count	是	INT	指定分隔符位置。

返回值说明

返回STRING类型。

说明

如果任一输入参数值为NULL，返回NULL。

示例代码

返回 hello.world。

```
SELECT substring_index('hello.world.people', '.', 2);
```

返回world.people。

```
select substring_index('hello.world.people', '.', -2);
```

15.2.35 translate

将 input 字符串中出现的 from 中的字符逐个替换为 to 中对应位置的字符。

例如：将 abcde 中的 bcd 替换成 BCD：

```
translate("abcde", "bcd", "BCD")
```

命令格式

```
translate(string <input>, string <from>, string <to>)
```


参数说明

参数	是否必选	参数类型	说明
input	是	STRING	待替换的字符串。
from	是	STRING	待替换的字符集合。
to	是	STRING	替换后的字符集合。

返回值说明

返回 STRING 类型。

说明

如果任一输入参数值为 NULL，返回 NULL。

示例代码

返回 A1B2C3。

```
SELECT translate('AaBbCc', 'abc', '123');
```

15.2.36 trim

trim函数用于从str的左右两端去除字符：

- 如果未指定trimChars，则默认去除空格字符。
- 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的左右两端去除尽可能长的所有字符都在集合trimChars中的子串。

相似函数：

- ltrim，ltrim函数用于从str的左端去除字符。
- rtrim，rtrim函数用于从str的右端去除字符。

命令格式

```
trim([,]string )
```

或

```
trim([BOTH] [] from )
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待去除左右两端字符的字符串。 如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。

参数	是否必选	参数类型	说明
trimChars	否	STRING	待去除的字符。

返回值说明

返回为STRING类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str或trimChars值为NULL时，返回NULL。

示例代码

- 去除字符串 yxabcxx 的左右空格。命令示例如下。
  返回字符串yxabcxx。
  ```
  select trim(' yxabcxx ');
  ```
  等效于如下语句。
  ```
  select trim(both from ' yxabcxx ');
  ```
- 去除字符串yxabcxx左右两端所有字符都在集合xy中的子串。
  返回abc，只要左右两端遇到x或者y就会被去掉。
  ```
  select trim('xy', 'yxabcxx');
  ```
  等效于如下语句。
  ```
  select trim(both 'xy' from 'yxabcxx');
  ```
- 输入参数为NULL。命令示例如下。
  返回NULL。
  ```
  select trim(null);      select trim(null, 'yxabcxx');
  ```

15.2.37 upper/ucase

upper函数用于将文本字符串转换成字母全部大写的形式。

命令格式

```
upper(string A)
```

或

```
ucase(string A)
```

参数说明

参数	是否必选	参数类型	说明
A	是	STRING	待转换的文本字符串。

返回值说明

返回STRING类型。

说明

- 入参非 STRING、BIGINT、DOUBLE、DECIMAL 或 DATETIME 类型时，返回报错。
- 入参值为NULL时，返回NULL。

示例代码

将字符串中的小写字符转换为大写字符。命令示例如下。

返回ABC。

```
select upper('abc');
```

输入参数为NULL。命令示例如下。

返回NULL。

```
select upper(null);
```

15.2.38 url_decode1

url_decode函数用于将字符串从application/x-www-form-urlencoded MIME格式转为常规字符。

命令格式

```
url_decode1(string [, string ])
```

参数说明

参数	是否必选	参数类型	说明
input	是	STRING	要输入的字符串。
encoding	否	STRING	指定编码格式，支持GBK或UTF-8等标准编码格式，不输入默认为UTF-8。

返回值说明

返回STRING类型的值。

说明

STRING类型UTF-8编码的字符串。

示例代码

返回 Example for url_decode :// dsf(fasfs)。

```
select url_decode1('Example+for+url_decode+%3A%2F%2F+dsf%28fasfs%29', 'GBK');
```


15.2.39 url_encode1

url_encode函数用于将字符串编码为application/x-www-form-urlencoded MIME格式。

命令格式

```
url_encode1(string [, string ])
```

参数说明

参数	是否必选	参数类型	说明
input	是	STRING	要输入的字符串。
encoding	否	STRING	指定编码格式，支持GBK或UTF-8等标准编码格式，不输入默认为UTF-8。

返回值说明

返回STRING类型的值。

说明

input或encoding值为NULL时，返回NULL。

示例代码

返回Example+for+url_encode%3A%2F%2F+dsf%28fasfs%29

```
select url_encode1('Example for url_encode:// dsf(fasfs)', 'GBK');
```

15.2.40 printf

printf函数用于将输入按特定格式打印输出。

命令格式

```
printf(String format, Obj... args)
```

参数说明

参数	是否必选	参数类型	说明
format	是	STRING	用于定义输出格式。
Obj	否	STRING	其他输入参数。

返回值说明

返回STRING类型的值。

将Obj中的参数填入format后打印输出。

示例代码

返回字符串：姓名：user1，年龄：20，性别：女，籍贯：城市1。

```
SELECT printf('姓名：%s，年龄：%d，性别：%s，籍贯：%s', "user1", 20, "女", "城市1");
```

15.3 数学函数

15.3.1 abs

abs函数用于计算入参的绝对值。

命令格式

abs(DOUBLE a)

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE、INT或DECIMAL类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回NULL。

```
select abs(null);
```

返回1。

```
select abs(-1);
```

返回3.1415926。

```
select abs(-3.1415926);
```

15.3.2 acos

acos函数用于返回给定数值a的反余弦值。

命令格式

```
acos(DOUBLE a)
```

参数说明

表 15-33 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a取值范围为[-1,1]，a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型，值在0~π之间。

说明

- a的值不在[-1,1]范围内时，返回NaN。
- a为NULL，则返回NULL。

示例代码

返回3.141592653589793。

```
select acos(-1);
```

返回0。

```
select acos(1);
```

返回NULL。

```
select acos(null);
```

返回NaN。

```
select acos(10);
```

15.3.3 asin

asin函数用于返回给定数值a的反正弦值。

命令格式

```
asin(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a取值范围为[-1,1]，a的格式包括浮点数格式、整数格式、字符串格式。参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型，值在-π/2~π/2之间。

说明

- a的值不在[-1,1]范围内时，返回NaN。
- a为NULL，则返回NULL。

示例代码

返回1.5707963267948966。

```
select asin(1);
```

返回0.6435011087932844。

```
select asin(0.6);
```

返回NULL。

```
select asin(null);
```

返回NaN。

```
select asin(10);
```

15.3.4 atan

atan函数用于返回给定数值a的反正切值。

命令格式

```
atan(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型，值在-π/2~π/2之间。

说明

a为NULL，则返回NULL。

示例代码

返回0.7853981633974483。

```
select atan(1);
```

返回0.5404195002705842。

```
select atan(0.6);
```

返回NULL。

```
select atan(null);
```

15.3.5 bin

bin函数用于返回a的二进制格式。

命令格式

```
bin(BIGINT a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式为整数格式。 参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回STRING类型。

说明

a值为NULL时，返回NULL。

示例代码

返回1。

```
select bin(1);
```

返回NULL。

```
select bin(null);
```

返回1000。


```
select bin(8);
```

返回1000。

```
select bin(8.123456);
```

15.3.6 bround

bround函数用于返回一个数值，该数值是按照指定d位小数进行四舍五入运算的结果。

命令格式

```
bround(DOUBLE a, INT d)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 代表需要被四舍五入的值。 该命令与传统四舍五入方式的区别在于，对数字5进行操作时，由前一位数字来决定，前一位数字为奇数，则进位，前一位数字为偶数，则舍去。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
d	否	DOUBLE、BIGINT、DECIMAL、STRING类型。	代表需要四舍五入到的位数。 参数d非INT类型时，会隐式转换为INT类型后参与运算。

返回值说明

返回DOUBLE类型。

说明

a或d值为NULL时，返回NULL。

示例代码

返回123.4。

```
select bround(123.45,1);
```

返回123.6。

```
select bround(123.55,1);
```

返回NULL。

```
select bround(null);
```

返回123.457。

```
select bround(123.456789,3.123456);
```


15.3.7 cbrt

cbrt函数用于返回a的立方根。

命令格式

```
cbrt(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型。

说明

a为NULL，则返回NULL。

示例代码

返回3。

```
select cbrt(27);
```

返回3.3019272488946267。

```
select cbrt(36);
```

返回NULL。

```
select cbrt(null);
```

15.3.8 ceil

ceil函数用于返回大于或等于a的最小整数。

命令格式

```
ceil(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DECIMAL类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回2。

```
select ceil(1.3);
```

返回-1。

```
select ceil(-1.3);
```

返回NULL。

```
select ceil(null);
```

15.3.9 conv

conv函数用于进制转换，将from_base进制下的num转化为to_base进制下面的数。

命令格式

```
conv(BIGINT num, INT from_base, INT to_base)
```

参数说明

参数	是否必选	参数类型	说明
num	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	需要进行转换进制的数。 参数num格式为浮点数格式、整数格式、字符串格式。
from_base	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	被转换的进制from_base。 参数from_base格式为浮点数格式、整数格式、字符串格式。

参数	是否必选	参数类型	说明
to_base	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	转化至的进制to_base。 参数to_base格式为浮点数格式、整数格式、字符串格式。

返回值说明

返回STRING类型。

说明

- num、from_base或to_base值为NULL时，返回NULL。
- 转换过程以64位精度工作，溢出时返回NULL。
- num如果输入的是小数，会转为整数值后进行进制转换，小数部分会被舍弃。

示例代码

返回8。

```
select conv('1000', 2, 10);
```

返回B。

```
select conv('1011', 2, 16);
```

返回703710。

```
select conv('ABCDE', 16, 10);
```

返回27。

```
select conv(1000.123456, 3.123456, 10.123456);
```

返回18446744073709551589。

```
select conv(-1000.123456, 3.123456, 10.123456);
```

返回NULL。

```
select conv('1100', null, 10);
```

15.3.10 cos

cos函数用于计算a的余弦值，输入为弧度

命令格式

```
cos(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回-0.9999999999999986

```
select cos(3.1415926);
```

返回NULL。

```
select cos(null);
```

15.3.11 cot1

cot1函数用于计算a的余切值，输入为弧度。

命令格式

```
cot1(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE或DECIMAL类型。

说明

a为NULL，则返回NULL。

示例代码

返回1.0000000000000002。

```
select cot1(pi()/4);
```

返回NULL。

```
select cot1(null);
```

15.3.12 degrees

degrees函数用于计算返回弧度所对应的角度。

命令格式

```
degrees(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回90.0。

```
select degrees(1.5707963267948966);
```

返回0。

```
select degrees(0);
```

返回NULL。

```
select degrees(null);
```

15.3.13 e

e函数用于返回自然常数e的值。

命令格式

```
e()
```

返回值说明

返回DOUBLE类型。

示例代码

返回2.718281828459045。

```
select e();
```

15.3.14 exp

exp函数用于计算返回e的a次方的值。

命令格式

```
exp(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回7.38905609893065。

```
select exp(2);
```

返回20.085536923187668。

```
select exp(3);
```

返回NULL。

```
select exp(null);
```

15.3.15 factorial

factorial函数用于返回a的阶乘。

命令格式

```
factorial(INT a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	BIGINT、INT、SMALLINT、TINYINT类型。	参数a的格式为整数格式。 参数a非INT类型时，会隐式转换为INT类型后参与运算。 字符串会转为对应的ASCII码。

返回值说明

返回BIGINT类型。

说明

- a值为0时，返回1。
- a值为NULL或[0,20]之外的值，返回NULL。

示例代码

返回720。

```
select factorial(6);
```

返回1。

```
select factorial(1);
```

返回120。

```
select factorial(5.123456);
```

返回NULL。

```
select factorial(null);
```

返回NULL。

```
select factorial(21);
```

15.3.16 floor

floor函数用于返回小于或等于a的最大整数。

命令格式

```
floor(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回BIGINT类型。

说明

a为NULL，则返回NULL。

示例代码

返回1。

```
select floor(1.2);
```

返回-2。

```
select floor(-1.2);
```

返回NULL。

```
select floor(null);
```

15.3.17 greatest

greatest函数用于返回列表中的最大值。

命令格式

```
greatest(T v1, T v2, ...)
```

参数说明

参数	是否必选	参数类型	说明
v1	是	DOUBLE、BIGINT、DECIMAL类型。	参数v1的格式包括浮点数格式、整数格式和DECIMAL格式。
v2	是	DOUBLE、BIGINT、DECIMAL类型。	参数v2的格式包括浮点数格式、整数格式和DECIMAL格式。

返回值说明

返回DOUBLE类型的值。

说明

任一参数为NULL时，则返回NULL。

示例代码

返回4.0。

```
select greatest(1,2.0,3,4.0);
```

返回NULL。

```
select greatest(null);
```

15.3.18 hex

hex函数用于将整数或字符转换为十六进制格式。

命令格式

```
hex(BIGINT a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。 字符串会转为对应的ASCII码。

返回值说明

返回STRING类型。

说明

- a值为0时，返回0。
- a值为NULL时，返回NULL。

示例代码

返回0。

```
select hex(0);
```

返回61。

```
select hex('a');
```

返回10。

```
select hex(16);
```

返回31。


```
select hex('1');
```

返回3136。

```
select hex('16');
```

返回NULL。

```
select hex(null);
```

15.3.19 least

least函数用于返回列表中的最小值。

命令格式

```
least(T v1, T v2, ...)
```

参数说明

参数	是否必选	参数类型	说明
v1	是	DOUBLE、BIGINT、DECIMAL类型。	参数v1的格式包括浮点数格式、整数格式和DECIMAL格式。
v2	是	DOUBLE、BIGINT、DECIMAL类型。	参数v2的格式包括浮点数格式、整数格式和DECIMAL格式。

返回值说明

返回DOUBLE类型的值。

说明

- v1、v2...为String类型时，返回报错。
- 所有参数都为NULL时，返回NULL。

示例代码

返回1.0。

```
select least(1,2.0,3,4.0);
```

返回NULL。

```
select least(null);
```

15.3.20 ln

ln函数用于返回给定数值的自然对数。

命令格式

```
ln(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

- a值为负数或0时，返回NULL。
- a值为NULL时，返回NULL。

示例代码

返回1.144729868791239。

```
select ln(3.1415926);
```

返回1。

```
select ln(2.718281828459045);
```

返回NULL。

```
select ln(null);
```

15.3.21 log

log函数根据给定底数及真数返回对数值。

命令格式

```
log(DOUBLE base, DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
base	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数base的格式包括浮点数格式、整数格式、字符串格式。 参数base非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

- base或a为NULL时，返回NULL。
- base或a为负数或0时，返回NULL。
- 如果base为1（会引发一个除零行为），会返回NULL。

示例代码

返回2。

```
select log(2, 4);
```

返回NULL。

```
select log(2, null);
```

返回NULL。

```
select log(null, 4);
```

15.3.22 log10

log10函数用于返回给定数值的以10为底的对数（常用对数）。

命令格式

```
log10(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a值为0、负数或NULL时，返回NULL。

示例代码

返回NULL。

```
select log10(null);
```


返回NULL。

```
select log10(0);
```

返回0.9542425094393249。

```
select log10(9);
```

返回1。

```
select log10(10);
```

15.3.23 log2

log2函数用于返回给定数值的以2为底的对数。

命令格式

```
log2(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a值为0、负数或NULL时，返回NULL。

示例代码

返回NULL。

```
select log2(null);
```

返回NULL。

```
select log2(0);
```

返回3.1699250014423126。

```
select log2(9);
```

返回4。

```
select log2(16);
```

15.3.24 negative

negative函数用于返回a的相反数。

命令格式

```
negative(INT a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回DECIMAL或INT类型。

说明

a为NULL，则返回NULL。

示例代码

返回-1。

```
SELECT negative(1);
```

返回3。

```
SELECT negative(-3);
```

15.3.25 pi

pi函数用于返回圆周率π的值。

命令格式

```
pi()
```

返回值说明

返回DOUBLE类型。

示例代码

返回3.141592653589793。

```
select pi();
```

15.3.26 pmod

pmod函数用于返回a除b的正余数。

命令格式

```
pmod(INT a, INT b)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回DECIMAL或INT类型。

说明

- a或b为NULL，则返回NULL。
- b为0时，返回NULL

示例代码

返回2。

```
select pmod(2,5);
```

返回1。

```
select pmod(5,2);
```

返回0.877。

```
select pmod(5.123,2.123);
```

15.3.27 positive

positive函数用于返回a的值。

命令格式

```
positive(INT a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回 DECIMAL、DOUBLE或INT类型。

说明

a为NULL，则返回NULL。

示例代码

返回3。

```
SELECT positive(3);
```

返回-3。

```
SELECT positive(-3);
```

返回123。

```
SELECT positive('123');
```

15.3.28 pow

pow函数用于计算返回a的p次幂。

命令格式

```
pow(DOUBLE a, DOUBLE p),
power(DOUBLE a, DOUBLE p)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
p	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数p的格式包括浮点数格式、整数格式、字符串格式。 参数p非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a、p为NULL，则返回NULL。

示例代码

返回16。

```
select pow(2, 4);
```

返回NULL。

```
select pow(2, null);
```

返回17.429460393524256。

```
select pow(2, 4.123456);
```

15.3.29 radians

radians函数用于返回角度所对应的弧度。

命令格式

```
radians(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回1.0471975511965976。

```
select radians(60);
```

返回0。

```
select radians(0);
```

返回NULL。

```
select radians(null);
```

15.3.30 rand

rand函数用于返回大于或等于0且小于1的平均分布随机数。

命令格式

```
rand(INT seed)
```

参数说明

参数	是否必选	参数类型	说明
seed	否	INT类型。	参数seed的格式为整数格式。 如果指定种子seed，在相同运行环境下，将会得到一个稳定的随机数序列。

返回值说明

返回DOUBLE类型的值。

示例代码

返回0.3668915240363728。

```
select rand();
```

返回0.25738143505962285。

```
select rand(3);
```

15.3.31 round

round函数用于计算a的四舍五入到d位的值。

命令格式

```
round(DOUBLE a, INT d)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	代表需要被四舍五入的值。 参数a的格式包括浮点数格式、整数格式、字符串格式。
d	否	INT类型。	默认值：0。 代表需要四舍五入到的位数。 参数d非INT类型时，会隐式转换为INT类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

- d为负数时，对小数点左侧第|d|位进行四舍五入。
- a或d值为NULL时，返回NULL。

示例代码

返回123.0。

```
select round(123.321);
```

返回123.4。

```
select round(123.396, 1);
```

返回NULL。

15.3.32 shiftleft

shiftleft函数用于有符号左移，将a的二进制数按位左移b位。

命令格式

```
shiftleft(BIGINT a, BIGINT b)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。 当参数b非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回INT类型。

说明

a或b值为NULL时，返回NULL。

示例代码

返回8。

```
select shiftleft(1,3);
```

返回48。

```
select shiftleft(6,3);
```

返回48。


```
select shiftleft(6.123456,3.123456);
```

返回NULL。

```
select shiftleft(null,3);
```

15.3.33 shiftright

shiftright函数用于有符号右移，将a的二进制数按位右移b位。

命令格式

```
shiftright(BIGINT a, BIGINT b)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。 当参数b非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回INT类型。

说明

a或b值为NULL时，返回NULL。

示例代码

返回2。

```
select shiftright(16,3);
```

返回4。

```
select shiftright(36,3);
```

返回4。

```
select shiftright(36.123456,3.123456);
```

返回NULL。

```
select shiftright(null,3);
```

15.3.34 shiftrightunsigned

shiftrightunsigned函数用于无符号右移，将a的二进制数按位右移b位。

命令格式

```
shiftrightunsigned(BIGINT a, BIGINT b)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。 当参数b非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回INT类型。

说明

a或b值为NULL时，返回NULL。

示例代码

返回2。

```
select shiftrightunsigned(16,3);
```

返回536870910。

```
select shiftrightunsigned(-16,3);
```

返回2。

```
select shiftrightunsigned(16.123456,3.123456);
```

返回NULL。

```
select shiftrightunsigned(null,3);
```

15.3.35 sign

sign函数用于返回a所对应的正负号。

命令格式

```
sign(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回DOUBLE类型。

说明

- a值为正数时，返回1。
- a值为负数时，返回-1。
- a值为0时，返回0。
- a值为NULL时，返回NULL。

示例代码

返回-1。

```
select sign(-3);
```

返回1。

```
select sign(3);
```

返回0。

```
select sign(0);
```

返回1。

```
select sign(3.1415926);
```

返回NULL。

```
select sign(null);
```

15.3.36 sin

sin函数用于计算a的正弦值，输入为弧度。

命令格式

```
sin(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回1。

```
select sin(pi()/2);
```

返回NULL。

```
select sin(null);
```

15.3.37 soundex

soundex函数用于从str返回一个soundex字符串，如soundex('Miller')= M460。

命令格式

```
soundex(string str)
```

参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待转换的字符串。

返回值说明

返回STRING类型的值。

说明

str值为NULL时，返回NULL。

示例代码

返回M460


```
SELECT soundex('Miller');
```

15.3.38 sqrt

sqrt函数用于返回数值的平方根。

命令格式

```
sqrt(DOUBLE a)
```

参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回2.8284271247461903。

```
select sqrt(8);
```

返回4。

```
select sqrt(16);
```

返回NULL。

```
select sqrt(null);
```

15.3.39 tan

tan函数用于计算a的正切值，输入为弧度。

命令格式

```
tan(DOUBLE a)
```


参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回0.9999999999999999。

```
select tan(pi()/4);
```

返回NULL。

```
select tan(null);
```

15.4 聚合函数

15.4.1 avg

功能描述

avg函数用于计算平均值。

语法格式

```
avg(col), avg(DISTINCT col)
```

参数说明

参数	是否必选	说明
col	是	列值支持所有数据类型，可以转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

如果col值为NULL时，该列不参与计算。

示例

- 计算所有仓库的平均商品数（items）。命令示例如下：
```
select avg(items) from warehouse;
```
返回结果如下：
```
_c0
100.0
```
- 与group by配合使用，计算每个仓库中所有商品的平均库存。命令示例如下：
```
select warehouseId, avg(items) from warehouse group by warehouseId;
```
返回结果如下：
```
warehouseId _c1
city1    155
city2    101
city3    194
```

15.4.2 corr

功能描述

corr函数用于返回两列数值的相关系数。

语法格式

```
corr(col1, col2)
```

参数说明

参数	是否必选	参数类型	说明
col1	是	DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL类型	数据类型为数值的列。其他类型返回NULL。
col2	是	DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL类型	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）和价格（price）的相关系数。命令示例如下：
```
select corr(items,price) from warehouse;
```


返回结果如下：

```
_c0
1.242355
```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）和价格（price）的相关系数。命令示例如下：

```
select warehouseId, corr(items,price) from warehouse group by warehouseId;
```

返回结果如下：

```
warehouseId _c1
city1   0.43124
city2   0.53344
city3   0.73425
```

15.4.3 count

功能描述

count函数用于返回记录条数。

语法格式

```
count([distinct|all] <colname>)
```

参数说明

参数	是否必选	说明
distinct或all	否	表示在计数时是否去除重复记录，默认为all，即计算全部记录。 如果指定distinct，则只计算唯一值数量。
colname	是	列值可以为任意类型。colname可以为*，即count(*)，返回所有行数。

返回值说明

返回BIGINT类型。

说明

colname值为NULL时，该行不参与计算。

示例

- 计算所有仓库表中的记录数。命令示例如下：

```
select count(*) from warehouse;
```

返回结果如下：

```
_c0
10
```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，计算各仓库（warehouseId）的商品数。命令示例如下：

```
select warehouseId, count(*) from warehouse group by warehouseId;
```

返回结果如下：


```
warehouseId _c1
city1   6
city2   5
city3   6
```

- 通过distinct去重，计算仓库数量。命令示例如下：

```
select count(distinct warehouseId) from warehouse;
```

返回结果如下：

```
_c0
3
```

15.4.4 covar_pop

功能描述

covar_pop函数用于返回两列数值协方差。

语法格式

```
covar_pop(col1, col2)
```

参数说明

参数	是否必选	说明
col1	是	数据类型为数值的列，其他类型返回NULL。
col2	是	数据类型为数值的列，其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）和价格（price）的协方差。命令示例如下：

```
select covar_pop(items,price) from warehouse;
```

返回结果如下：

```
_c0
1.242355
```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）和价格（price）的协方差。命令示例如下：

```
select warehouseId, covar_pop(items,price) from warehouse group by warehouseId;
```

返回结果如下：

```
warehouseId _c1
city1   1.13124
city2   1.13344
city3   1.53425
```

15.4.5 covar_samp

功能描述

covar_samp函数用于返回两列数值样本协方差。

语法格式

```
covar_samp(col1, col2)
```

参数说明

参数	是否必选	说明
col1	是	数据类型为数值的列，其他类型返回NULL。
col2	是	数据类型为数值的列，其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）和价格（price）的样本协方差。命令示例如下：
  ```
  select covar_samp(items,price) from warehouse;
  ```
  返回结果如下：
  ```
  _c0
  1.242355
  ```
- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）和价格（price）的样本协方差。命令示例如下：
  ```
  select warehouseId, covar_samp(items,price) from warehouse group by warehouseId;
  ```
  返回结果如下：
  ```
  warehouseId _c1
  city1    1.03124
  city2    1.03344
  city3    1.33425
  ```

15.4.6 max

功能描述

max函数用于返回最大值。

语法格式

```
max(col)
```

参数说明

参数	是否必选	说明
col	是	列值可以为除BOOLEAN外的任意类型。

返回值说明

返回DOUBLE类型的值。

说明

返回值的类型与col类型相同。返回规则如下：

- col值为NULL时，该行不参与计算。
- col为BOOLEAN类型时，不允许参与运算。

示例

- 计算所有商品的最高库存（items）。命令示例如下：

```
select max(items) from warehouse;
```

返回结果如下：

```
_c0
900
```

- 与group by配合使用，求每个仓库的最高库存。命令示例如下：

```
select warehouseId, max(items) from warehouse group by warehouseId;
```

返回结果如下：

```
warehouseId _c1
city1    200
city2    300
city3    400
```

15.4.7 min

功能描述

min函数用于返回最小值。

语法格式

```
min(col)
```

参数说明

参数	是否必选	参数类型	说明
col	是	除BOOLEAN外的任意类型。	列值可以为除BOOLEAN外的任意类型。

返回值说明

返回DOUBLE类型的值。

说明

返回值的类型与col类型相同。返回规则如下：

- col值为NULL时，该行不参与计算。
- col为BOOLEAN类型时，不允许参与运算。

示例

- 计算所有商品的最低库存（items）。命令示例如下：
  ```
  select min(items) from warehouse;
  ```
  返回结果如下：
  ```
  _c0
  600
  ```
- 与group by配合使用，求每个仓库的最低库存。命令示例如下：
  ```
  select warehouseId, min(items) from warehouse group by warehouseId;
  ```
  返回结果如下：
  ```
  warehouseId _c1
  city1    15
  city2    10
  city3    19
  ```

15.4.8 percentile

功能描述

percentile函数用于计算精确百分位数，适用于小数据量。先对指定列升序排列，然后取第p位百分数的精确值。

语法格式

```
percentile(colname,DOUBLE p)
```

参数说明

参数	是否必选	参数类型	说明
colname	是	STRING类型	代表需要排序的列名。 列中元素只能为整数类型。
p	是	DOUBLE类型	p的范围为0-1。参数p的格式包括浮点数格式。

返回值说明

返回DOUBLE或ARRAY类型。

说明

- 列名不存在时，返回报错。
- p为NULL或在[0,1]之外时，返回报错。

示例

假设列int_test中的元素为1、2、3、4，类型为INT类型。

返回3.0999999999999996。

```
select percentile(int_test,0.7) FROM int_test;
```

返回2.5。


```
select percentile(int_test,0.5) FROM int_test;
```

返回[1.3,1.9,2.5,2.8,3.7]。

```
select percentile (int_test,ARRAY(0.1,0.3,0.5,0.6,0.9)) FROM int_test;
```

15.4.9 percentile_approx

功能描述

percentile_approx函数用于计算近似百分位数，适用于大数据量。先对指定列升序排列，然后取第p位百分数最靠近的值。

语法格式

```
percentile_approx(DOUBLE col, p [, B])
```

参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。
p	是	0<=P<=1，否则返回NULL。
B	否	参数B控制近似的精确度，B值越大，近似度越高，默认值为10000。当列中非重复值的数量小于B时，返回精确的百分数。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）的0.5百分位，精确度100。命令示例如下：假设列int_test中的元素为1、2、3、4，类型为INT类型。

  ```
  select PERCENTILE_APPROX(items,0.5,100) from warehouse;
  ```

  返回结果如下：

  ```
  +------------+
  | _c0        |
  +------------+
  | 10         |
  +------------+
  ```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）的 0.5 百分位，精确度100。命令示例如下：

  ```
  select warehouseId, PERCENTILE_APPROX(items, 0.5, 100) from warehouse group by warehouseId;
  ```

  返回结果如下：

  ```
  +------------+------------+
  | warehouseId| _c1        |
  +------------+------------+
  | city1    | 3     |
  | city2    | 20     |
  | city3    | 200     |
  +------------+------------+
  ```


15.4.10 stddev_pop

功能描述

stddev_pop函数用于返回指定列的总体标准差。

语法格式

```
stddev_pop(col)
```

参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）的总体标准差。命令示例如下：
  ```
  select stddev_pop(items) from warehouse;
  ```
  返回结果如下：
  ```
  _c0
  91.49358659976605
  ```
- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）的偏差。命令示例如下：
  ```
  select warehouseId, stddev_pop(items) from warehouse group by warehouseId;
  ```
  返回结果如下：
  ```
  warehouseId _c1
  city1    1.4142135623730951
  city2    11.180339887498949
  city3    81.64965809277261
  ```

15.4.11 stddev_samp

功能描述

stddev_samp函数用于返回指定列的样本标准差。

语法格式

```
stddev_samp(col)
```


参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）的样本标准差。命令示例如下：
```
select stddev_samp(items) from warehouse;
```
返回结果如下：
```
+------------+
| _c0        |
+------------+
| 1.342355   |
+------------+
```
- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）的样本标准差。命令示例如下：
```
select warehouseId, stddev_samp(items) from warehouse group by warehouseId;
```
返回结果如下：
```
+------------+------------+
| warehouseId| _c1        |
+------------+------------+
| city1    | 1.23124   |
| city2    | 1.23344   |
| city3    | 1.43425   |
+------------+------------+
```

15.4.12 sum

功能描述

sum函数用于求和。

语法格式

```
sum(col),
sum(DISTINCT col)
```

参数说明

参数	是否必选	说明
col	是	列值支持所有数据类型，可以转换为DOUBLE类型后参与运算。 列值可以为DOUBLE、DECIMAL或BIGINT等类型。 如果输入为STRING类型，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

如果col值为NULL时，该行不参与计算。

示例

- 计算所有仓库的商品（items）总和。命令示例如下：
```
select sum(items) from warehouse;
```
返回结果如下：
```
_c0
55357
```
- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，计算各仓库商品的总数（items）总和。命令示例如下：
```
select warehouseId, sum(items) from warehouse group by warehouseId;
```
返回结果如下：
```
warehouseId| _c1
city1    15500
city2    10175
city3    19400
```

15.4.13 variance/var_pop

功能描述

variance/var_pop函数用于返回列的总体方差。

语法格式

```
variance(col),
var_pop(col)
```

参数说明

参数	是否必选	说明
col	是	数据类型为数值的列，参数为其他类型的列返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）的方差。命令示例如下：


```
select variance(items) from warehouse;
--等效于如下语句。
select var_pop(items) from warehouse;
```

返回结果如下：

```
_c0
203.42352
```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）的方差。命令示例如下：

```
select warehouseId, variance(items) from warehouse group by warehouseId;
--等效于如下语句。
select warehouseId, var_pop(items) from warehouse group by warehouseId;
```

返回结果如下：

```
warehouseId _c1
city1    19.23124
city2    17.23344
city3    12.43425
```

15.4.14 var_samp

功能描述

var_samp函数用于返回指定列的样本方差。

语法格式

```
var_samp(col)
```

参数说明

参数	是否必选	说明
col	是	数据类型为数值的列，其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例

- 计算所有商品库存（items）的样本方差。命令示例如下：

```
select var_samp(items) from warehouse;
```

返回结果如下：

```
_c0
294.342355
```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，并计算同组商品库存（items）的样本方差。命令示例如下：

```
select warehouseId, var_samp(items) from warehouse group by warehouseId;
```

返回结果如下：

```
warehouseId _c1
city1    18.23124
city2    16.23344
city3    11.43425
```


15.4.15 median

功能描述

median函数用于计算入参的中位数。

语法格式

```
median(colname)
```

参数说明

参数	是否必选	参数类型	说明
colname	是	DOUBLE、DECIMAL、STRING、BIGINT类型。	代表需要排序的列名， 列中元素为DOUBLE类型。 当列中元素非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE或DECIMAL类型。

说明

列名不存在时，返回报错。

示例

假设列int_test中的元素为1、2、3、4，类型为INT类型。

返回2.5。

```
select median(int_test) FROM int_test;
```

15.5 分析窗口函数

15.5.1 cume_dist

功能描述

cume_dist函数用于求累计分布，相当于求分区中大于等于或小于等于当前行的数据在分区中的占比。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。

- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
cume_dist() over([partition_clause] [orderby_clause])
```

参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回DOUBLE类型的值。

示例

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept STRING, --部门名称
userid string, -- 员工ID
sal INT -- 薪水
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
d1,user1,1000
d1,user2,2000
d1,user3,3000
d2,user4,4000
d2,user5,5000
```

- 统计小于等于当前薪水的人数占比

```
select dept, userid, sal,
       cume_dist() over(order by sal) as cume1
from salary;
--  结果：
d1 user1 1000 0.2
d1 user2 2000 0.4
d1 user3 3000 0.6
d2 user4 4000 0.8
d2 user5 5000 1.0
```

- 按部门分组统计小于等于当前薪水的人数的比例

```
select dept, userid, sal,
       cume_dist() over (partition by dept order by sal) as cume2
from salary;
-- 结果：
d1 user1 1000 0.3333333333333333
d1 user2 2000 0.6666666666666666
d1 user3 3000 1.0
```



```
d2 user4 4000 0.5
d2 user5 5000 1.0
```

- 按照sal降序排序后，结果就是统计 大于等于 当前薪水的人数的比例

```
select dept, userid, sal,
       cume_dist() over(order by sal desc) as cume3
from salary;
 -- 结果：
d2 user5 5000 0.2
d2 user4 4000 0.4
d1 user3 3000 0.6
d1 user2 2000 0.8
d1 user1 1000 1.0 select dept, userid, sal,
       cume_dist() over(partition by dept order by sal desc) as cume4
from salary;
 -- 结果：
d1 user3 3000 0.3333333333333333
d1 user2 2000 0.6666666666666666
d1 user1 1000 1.0
d2 user5 5000 0.5
d2 user4 4000 1.0
```

15.5.2 first_value

功能描述

first_value函数用于取当前行所对应窗口的第一条数据的值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
first_value([, <ignore_nulls>]) over ([partition_clause] [orderby_clause] [frame_clause])
```

参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
ignore_nulls	否	BOOLEAN类型，表示是否忽略NULL值。默认值为False。 当参数的值为True时，返回窗口中第一条非NULL的值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。
frame_clause	否	用于确定数据边界。

返回值说明

参数的数据类型。

示例

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表logs，并添加数据，命令示例如下：

```
create table logs(
  cookieid string,
  createtime string,
  url string )
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```

示例：将所有记录根据cookieid分组，并按createtime升序排列，返回每组中的第一行数据。命令示例如下

```
SELECT cookieid, createtime, url,
       FIRST_VALUE(url) OVER (PARTITION BY cookieid ORDER BY createtime) AS first
FROM logs;
返回结果如下：
cookieid createtime         url  first
cookie1 2015-04-10 10:00:00 url1 url1
cookie1 2015-04-10 10:00:02 url2 url1
cookie1 2015-04-10 10:03:04 url3 url1
cookie1 2015-04-10 10:10:00 url4 url1
cookie1 2015-04-10 10:50:01 url5 url1
cookie1 2015-04-10 10:50:05 url6 url1
cookie1 2015-04-10 11:00:00 url7 url1
cookie2 2015-04-10 10:00:00 url11 url11
cookie2 2015-04-10 10:00:02 url22 url11
cookie2 2015-04-10 10:03:04 url33 url11
cookie2 2015-04-10 10:10:00 url44 url11
cookie2 2015-04-10 10:50:01 url55 url11
cookie2 2015-04-10 10:50:05 url66 url11
cookie2 2015-04-10 11:00:00 url77 url11
```

15.5.3 last_value

功能描述

last_value函数用于取当前行所对应窗口的最后一条数据的值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
last_value(<expr>[, <ignore_nulls>]) over ([partition_clause] [orderby_clause] [frame_clause])
```

参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
ignore_nulls	否	BOOLEAN类型，表示是否忽略NULL值。默认值为False。当参数的值为True时，返回窗口中最后一条非NULL的值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。
frame_clause	否	用于确定数据边界。

返回值说明

参数的数据类型。

示例

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表logs，并添加数据，命令示例如下：

```
create table logs(
  cookieid string,
  createtime string,
  url string
)
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```


示例：将所有记录根据cookieid分组，并按createtime升序排列，返回每组中的最后一行数据。命令示例如下

```
SELECT cookieid, createtime, url,
       LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last
FROM logs;
-- 返回结果：
cookieid createtime         url last
cookie1 2015-04-10 10:00:00 url1 url1
cookie1 2015-04-10 10:00:02 url2 url2
cookie1 2015-04-10 10:03:04 url3 url3
cookie1 2015-04-10 10:10:00 url4 url4
cookie1 2015-04-10 10:50:01 url5 url5
cookie1 2015-04-10 10:50:05 url6 url6
cookie1 2015-04-10 11:00:00 url7 url7
cookie2 2015-04-10 10:00:00 url11 url11
cookie2 2015-04-10 10:00:02 url22 url22
cookie2 2015-04-10 10:03:04 url33 url33
cookie2 2015-04-10 10:10:00 url44 url44
cookie2 2015-04-10 10:50:01 url55 url55
cookie2 2015-04-10 10:50:05 url66 url66
cookie2 2015-04-10 11:00:00 url77 url77
```

说明： 截止到当前行的最后一个值，其实就是它本身。

15.5.4 lag

功能描述

lag函数用于统计窗口内往上第n行值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
lag(<expr>[, bigint <offset>[, <default>]]) over([partition_clause] orderby_clause)
```

参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
offset	否	偏移量，BIGINT类型常量，取值大于等于0。值为0时表示当前行，为1时表示前一行，以此类推。默认值为1。输入值为STRING类型、DOUBLE类型则隐式转换为BIGINT类型后进行运算。
default	否	常量，默认值为NULL。当offset指定的范围越界时的缺省值，需要与expr对应的数据类型相同。如果expr非常量，则基于当前行进行求值。

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

参数的数据类型。

示例

示例数据

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：

```
create table logs(
  cookieid string,
  createtime string,
  url string
)
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```

将所有记录根据cookieid分组，并按createtime升序排列，返回窗口内往上第2行的值。命令示例如下

示例1：

```
SELECT cookieid, createtime, url,
       LAG(createtime, 2) OVER (PARTITION BY cookieid ORDER BY createtime) AS last_2_time
FROM logs;
-- 返回结果：
cookieid createtime         url  last_2_time
cookie1 2015-04-10 10:00:00 url1 NULL
cookie1 2015-04-10 10:00:02 url2 NULL
cookie1 2015-04-10 10:03:04 url3 2015-04-10 10:00:00
cookie1 2015-04-10 10:10:00 url4 2015-04-10 10:00:02
cookie1 2015-04-10 10:50:01 url5 2015-04-10 10:03:04
cookie1 2015-04-10 10:50:05 url6 2015-04-10 10:10:00
cookie1 2015-04-10 11:00:00 url7 2015-04-10 10:50:01
cookie2 2015-04-10 10:00:00 url11 NULL
cookie2 2015-04-10 10:00:02 url22 NULL
cookie2 2015-04-10 10:03:04 url33 2015-04-10 10:00:00
cookie2 2015-04-10 10:10:00 url44 2015-04-10 10:00:02
```



```
cookie2 2015-04-10 10:50:01 url55 2015-04-10 10:03:04
cookie2 2015-04-10 10:50:05 url66 2015-04-10 10:10:00
cookie2 2015-04-10 11:00:00 url77 2015-04-10 10:50:01
```

说明：因为没有设置默认值，当没有上两行时显示为NULL。

示例2：

```
SELECT cookieid, createtime, url,
       LAG(createtime,1,'1970-01-01 00:00:00') OVER (PARTITION BY cookieid ORDER BY createtime) AS
last_1_time
FROM cookie4;
-- 结果：
cookieid createtime          url last_1_time cookie1 2015-04-10 10:00:00 url1 1970-01-01 00:00:00 (显示默认
值)
cookie1 2015-04-10 10:00:02 url2 2015-04-10 10:00:00
cookie1 2015-04-10 10:03:04 url3 2015-04-10 10:00:02
cookie1 2015-04-10 10:10:00 url4 2015-04-10 10:03:04
cookie1 2015-04-10 10:50:01 url5 2015-04-10 10:10:00
cookie1 2015-04-10 10:50:05 url6 2015-04-10 10:50:01
cookie1 2015-04-10 11:00:00 url7 2015-04-10 10:50:05
cookie2 2015-04-10 10:00:00 url11 1970-01-01 00:00:00 (显示默认值)
cookie2 2015-04-10 10:00:02 url22 2015-04-10 10:00:00
cookie2 2015-04-10 10:03:04 url33 2015-04-10 10:00:02
cookie2 2015-04-10 10:10:00 url44 2015-04-10 10:03:04
cookie2 2015-04-10 10:50:01 url55 2015-04-10 10:10:00
cookie2 2015-04-10 10:50:05 url66 2015-04-10 10:50:01
cookie2 2015-04-10 11:00:00 url77 2015-04-10 10:50:05
```

15.5.5 lead

功能描述

lead函数用于获取窗口内往下第n行值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
lead(<expr>[, bigint <offset>[, <default>]]) over([partition_clause] orderby_clause)
```

参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
offset	否	偏移量，BIGINT类型常量，取值大于等于0。值为0时表示当前行，为1时表示下一行，以此类推。默认值为1。输入值为STRING类型、DOUBLE类型则隐式转换为BIGINT类型后进行运算。

参数	是否必选	说明
default	否	常量，默认值为NULL。 当offset指定的范围越界时的缺省值，需要与expr对应的数据类型相同。如果expr非常量，则基于当前行进行求值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

参数的数据类型。

示例

示例数据

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：

```
create table logs(
  cookieid string,
  createtime string,
  url string )
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```

将所有记录根据cookieid分组，并按createtime升序排列，返回窗口内往下第2行和第1行的值。命令示例如下

```
SELECT cookieid, createtime, url,
       LEAD(createtime, 2) OVER(PARTITION BY cookieid ORDER BY createtime) AS next_2_time,
       LEAD(createtime, 1, '1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY createtime) AS
next_1_time
FROM logs;

-- 返回结果：
cookieid createtime         url  next_2_time          next_1_time
cookie1 2015-04-10 10:00:00 url1 2015-04-10 10:03:04  2015-04-10 10:00:02
cookie1 2015-04-10 10:00:02 url2 2015-04-10 10:10:00  2015-04-10 10:03:04
cookie1 2015-04-10 10:03:04 url3 2015-04-10 10:50:01  2015-04-10 10:10:00
cookie1 2015-04-10 10:10:00 url4 2015-04-10 10:50:05  2015-04-10 10:50:01
```



```
cookie1 2015-04-10 10:50:01 url5 2015-04-10 11:00:00  2015-04-10 10:50:05
cookie1 2015-04-10 10:50:05 url6 NULL                  2015-04-10 11:00:00
cookie1 2015-04-10 11:00:00 url7 NULL                  1970-01-01 00:00:00
cookie2 2015-04-10 10:00:00 url11 2015-04-10 10:03:04 2015-04-10 10:00:02
cookie2 2015-04-10 10:00:02 url22 2015-04-10 10:10:00 2015-04-10 10:03:04
cookie2 2015-04-10 10:03:04 url33 2015-04-10 10:50:01 2015-04-10 10:10:00
cookie2 2015-04-10 10:10:00 url44 2015-04-10 10:50:05 2015-04-10 10:50:01
cookie2 2015-04-10 10:50:01 url55 2015-04-10 11:00:00 2015-04-10 10:50:05
cookie2 2015-04-10 10:50:05 url66 NULL                  2015-04-10 11:00:00
cookie2 2015-04-10 11:00:00 url77 NULL                  1970-01-01 00:00:00
```

15.5.6 percent_rank

功能描述

percent_rank函数为窗口的ORDER BY子句所指定列中值的返回值，但以介于0和1之间的小数形式表示，计算方法为 (分组内当前行的RANK值-1)/(分组内总行数-1)。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
percent_rank() over([partition_clause] [orderby_clause])
```

参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回DOUBLE类型的值。

示例

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept STRING, -- 部门名称
userid string, -- 员工ID
sal INT -- 薪水
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：


```
d1,user1,1000
d1,user2,2000
d1,user3,3000
d2,user4,4000
d2,user5,5000
```

示例：计算员工薪水在部门内的百分比排名。

```
select dept, userid, sal,
      percent_rank() over(partition by dept order by sal) as pr2
from salary;
-- 结果分析：
d1 user1 1000 0.0    -- (1-1)/(3-1)=0.0
d1 user2 2000 0.5    -- (2-1)/(3-1)=0.5
d1 user3 3000 1.0    -- (3-1)/(3-1)=1.0
d2 user4 4000 0.0    -- (1-1)/(2-1)=0.0
d2 user5 5000 1.0    -- (2-1)/(2-1)=1.0
```

15.5.7 rank

功能描述

rank函数用于计算一个值在一组值中的排位。如果出现并列的情况，RANK函数会在排名序列中留出空位。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
rank() over ([partition_clause] [orderby_clause])
```

参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回INT类型的值。

示例

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：


```
CREATE TABLE logs (
cookieid string,
createtime string,
pv INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
cookie1 2015-04-10 1
cookie1 2015-04-11 5
cookie1 2015-04-12 7
cookie1 2015-04-13 3
cookie1 2015-04-14 2
cookie1 2015-04-15 4
cookie1 2015-04-16 4
cookie2 2015-04-10 2
cookie2 2015-04-11 3
cookie2 2015-04-12 5
cookie2 2015-04-13 6
cookie2 2015-04-14 3
cookie2 2015-04-15 9
cookie2 2015-04-16 7
```

示例：将所有记录根据cookieid分组，并按pv降序排列，返回组内每行的序号。命令示例如下:

```
select cookieid, createtime, pv,
      rank() over(partition by cookieid order by pv desc) as rank
from logs
where cookieid = 'cookie1';
-- 结果:
cookie1 2015-04-12 7 1
cookie1 2015-04-11 5 2
cookie1 2015-04-16 4 3 (并列第三)
cookie1 2015-04-15 4 3
cookie1 2015-04-13 3 5 (跳过4，从5开始)
cookie1 2015-04-14 2 6
cookie1 2015-04-10 1 7
```

15.5.8 row_number

功能描述

row_number函数用于计算行号。从1开始递增。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

语法格式

```
row_number() over([partition_clause] [orderby_clause])
```

参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回INT类型的值。

示例

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：

```
CREATE TABLE logs (
cookieid string,
createtime string,
pv INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
cookie1 2015-04-10 1
cookie1 2015-04-11 5
cookie1 2015-04-12 7
cookie1 2015-04-13 3
cookie1 2015-04-14 2
cookie1 2015-04-15 4
cookie1 2015-04-16 4
cookie2 2015-04-10 2
cookie2 2015-04-11 3
cookie2 2015-04-12 5
cookie2 2015-04-13 6
cookie2 2015-04-14 3
cookie2 2015-04-15 9
cookie2 2015-04-16 7
```

示例：将所有记录根据cookieid分组，并按pv降序排列，返回组内每行的序号。命令示例如下:

```
select cookieid, createtime, pv,
      row_number() over (partition by cookieid order by pv desc) as index
from logs;

-- 返回结果：
cookie1 2015-04-12 7 1
cookie1 2015-04-11 5 2
cookie1 2015-04-16 4 3
cookie1 2015-04-15 4 4
cookie1 2015-04-13 3 5
cookie1 2015-04-14 2 6
cookie1 2015-04-10 1 7
cookie2 2015-04-15 9 1
cookie2 2015-04-16 7 2
cookie2 2015-04-13 6 3
cookie2 2015-04-12 5 4
cookie2 2015-04-11 3 5
```



```
cookie2 2015-04-14 3 6
cookie2 2015-04-10 2 7
```

15.6 其他函数

15.6.1 decode1

功能描述

decode1函数实现if-then-else分支选择的功能。

命令格式

```
decode1(expression, search, result [, search, result]...[, default])
```

参数说明

参数	是否必选	参数类型	说明
expression	是	所有数据类型	要比较的表达式。
search	是	与expression一致	与expression进行比较的搜索项。
result	是	所有数据类型	search和expression的值匹配时的返回值。
default	否	与result一致	如果所有的搜索项都不匹配，则返回default值，如果未指定，则返回NULL。

返回值说明

result 和 default 为返回值，支持返回所有的数据类型。

说明

- 如果匹配，返回result。
- 如果没有匹配，返回default。
- 如果没有指定default，返回NULL。
- 如果search选项有重复且匹配时，会返回第一个值。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE IF NOT EXISTS salary (
  dept_id STRING,  -- 部门ID
  userid STRING,   -- 员工ID
  sal INT          -- 工资
```



```
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'obs://your-bucket-name/aidatalake/salary/';-- 【必填】替换为OBS桶路径（平台外表时必须指定）
```

添加数据如下：

```
INSERT INTO salary VALUES
('d1','user1',1000),
('d1','user2',2000),
('d1','user3',3000),
('d2','user4',4000),
('d2','user5',5000),
('d3','user6',6000),
('d4','user7',7000),
('d5','user8',8000);
```

回显结果

返回每个部门的名称

当 dept_id 的值为d1时，返回平台；值为d2时，返回MRS；其他场景返回Others。

```
select dept_id,
decode1(dept_id, 'd1', '平台', 'd2', 'MRS', 'Others') as result from salary
GROUP BY dept_id;  -- 按部门ID去重，确保每个部门仅返回一行
```

返回结果：

```
+-----+-------+
| dept_id| result|
+-----+-------+
|   d1|    平台|
|   d2|    MRS|
|   d3| Others|
|   d4| Others|
|   d5| Others|
+-----+-------+
```

15.6.2 ordinal

ordinal函数用于将输入变量按从小到大排序后，返回nth指定位置的值。

命令格式

```
ordinal(bigint nth, var1, var2 [, ...])
```

参数说明

参数	是否必选	参数类型	说明
nth	是	BIGINT类型	指定要返回的位置值。
var	是	BIGINT、DOUBLE、DATETIME或STRING类型	待排序的值。

返回值说明

DOUBLE或DECIMAL类型。

说明

- 排在第nth位的值，当不存在隐式转换时返回值同输入参数数据类型。
- 当有类型转换时，DOUBLE、BIGINT、STRING之间的转换返回DOUBLE类型；STRING、DATETIME之间的转换返回DATETIME类型。不允许其他的隐式转换。
- NULL为最小值。

示例代码

返回2。

```
select ordinal(3, 1, 3, 2, 5, 2, 4, 9);
```

15.6.3 trans_array

trans_array函数用于将一行数据转为多行的UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。

使用限制

- 所有作为key的列必须位于在前面，而要转置的列必须放在后面。
- 在一个select中只能有一个UDTF，不可以再出现其他的列。
- 不可以与group by、cluster by、distribute by、sort by一起使用。

命令格式

```
trans_array(num_keys, separator, key1, ..., col1, col2, ...) as (key1, ..., col1, col2, ...)
```

参数说明

参数	是否必选	参数类型	说明
num_keys	是	BIGINT类型	BIGINT类型常量，值必须>=0。在转为多行时作为转置key的列的个数。
separator	是	STRING类型	STRING类型常量，用于将字符串拆分成多个元素的分隔符。为空时返回报错。
keys	是	STRING类型	转置时作为key的列， 个数由num_keys指定。如果num_keys指定所有的列都作为key（即num_keys等于所有列的个数），则只返回一行。
cols	是	STRING类型	要转为行的数组，keys之后的所有列视为要转置的数组，必须为STRING类型

返回值说明

参数的数据类型。

说明

- 返回转置后的行，新的列名由as指定。
- 作为key的列类型保持不变，其余所有的列是STRING类型。
- 拆分成的行数以个数多的数组为准，不足的补NULL。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept_id STRING, -- 部门
user_id STRING, -- 员工
sal STRING -- 薪水
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
d1,user1/user4,1000/6000
d1,user2/user5,2000/7000
d1,user3/user6,3000
d2,user4/user7,4000
d2,user5/user8,5000/8000
```

执行SQL（在SQL语句中使用斜杠(/)时，平台控制台会自动进行转义处理，但通过API调用时需要您手动转义，否则会提示解析失败。）

```
select trans_array(1, "/", dept_id, user_id, sal) as (dept_id, user_id, sal) from salary;
```

返回结果如下：

```
d1,user1,1000
d1,user4,6000
d1,user2,2000
d1,user5,7000
d1,user3,3000
d1,user6,NULL
d2,user4,4000
d2,user7,NULL
d2,user5,5000
d2,user8,8000
```

15.7 位运算函数

功能描述

位运算函数用于对整数类型进行按位操作。支持多个位运算函数和操作符，支持在SQL中直接进行位级别的数据处理。

函数/操作符列表

函数/操作符	描述
expr1 & expr2	按位与运算。返回两个整数按位与的结果。
expr1 | expr2	按位或运算。返回两个整数按位或的结果。

函数/操作符	描述
expr1 ^ expr2	按位异或运算。返回两个整数按位异或的结果。
~ expr	按位取反运算。返回整数按位取反的结果。
base << exp	按位左移操作符。等价于 shiftleft(base, exp)。
base >> exp	按位有符号右移操作符。等价于 shiftright(base, exp)。
base >>> exp	按位无符号右移操作符。等价于 shiftrightunsigned(base, exp)。
shiftleft(base, shift)	按位左移。
shiftright(base, shift)	按位右移（带符号）。
shiftrightunsigned(base, shift)	按位无符号右移。
bit_count(expr)	返回整数中1的位数（即汉明重量/人口计数）。
bit_get(expr, pos)	返回整数指定位置上的位值（0或1）。位置从右往左编号，从0开始。
getbit(expr, pos)	bit_get的别名。返回整数指定位置上的位值。

示例

按位与运算（&）

```
SELECT 3 & 5;
-- 1（二进制：011 & 101 = 001）
```

按位或运算（|）

```
SELECT 3 | 5;
-- 7（二进制：011 | 101 = 111）
```

按位异或运算（^）

```
SELECT 3 ^ 5;
-- 6（二进制：011 ^ 101 = 110）
```

按位取反（~）

```
SELECT ~ 0;
-- -1
```

按位左移（<< / shiftleft）

```
SELECT 2 << 1;
-- 4

SELECT shiftleft(2, 1);
-- 4
```

按位有符号右移（>> / shiftright）

```
SELECT 4 >> 1;
-- 2
```



```
SELECT shiftright(4, 1);
-- 2
```

按位无符号右移（ >>> / shiftrightunsigned ）

```
SELECT 4 >>> 1;
-- 2

SELECT shiftrightunsigned(4, 1);
-- 2
```

bit_count — 统计1的位数

```
SELECT bit_count(0);
-- 0

SELECT bit_count(7);
-- 3（二进制：111，有3个1）

SELECT bit_count(8);
-- 1（二进制：1000，有1个1）
```

bit_get / getbit — 获取指定位的值

```
SELECT bit_get(11, 0);
-- 1（11的二进制为1011，第0位为1）

SELECT bit_get(11, 2);
-- 0（11的二进制为1011，第2位为0）

SELECT getbit(11, 0);
-- 1
```

注意事项

- 位运算函数仅适用于整数类型（TINYINT、SMALLINT、INT、BIGINT）。
- <<、>>、>>> 操作符与 shiftleft、shiftright、shiftrightunsigned 函数等价。
- bit_get/getbit 的位置参数从0开始，从最低有效位（最右侧）计数。位置参数不能为负数。

15.8 CSV 函数

功能描述

CSV函数用于处理CSV格式的数据。支持原生CSV解析和生成函数，可以在SQL中直接读取和生成CSV格式的字符串。

函数列表

函数	描述
from_csv(csv_str, schema [, options])	将CSV字符串解析为结构体。使用指定的schema解析CSV字符串。
schema_of_csv(csv [, options])	推断CSV字符串的DDL格式schema。
to_csv(expr [, options])	将结构体转换为CSV字符串。

示例

from_csv — 解析CSV字符串

```
SELECT from_csv('1,Alice,30', 'id INT, name STRING, age INT');
-- {1, Alice, 30}

SELECT from_csv('1|Alice|30', 'id INT, name STRING, age INT', map('delimiter', '|'));
-- {1, Alice, 30}
```

schema_of_csv — 推断CSV的schema

```
SELECT schema_of_csv('1,Alice,30');
-- STRUCT<col0:INT, col1:STRING, col2:INT>

SELECT schema_of_csv('1,Alice', map('header', 'true'));
-- STRUCT<id:INT, name:STRING>
```

to_csv — 将结构体转为CSV

```
SELECT to_csv(named_struct('a', 1, 'b', 2, 'c', 3));
-- 1,2,3

SELECT to_csv(named_struct('a', 1, 'b', 2), map('delimiter', '|'));
-- 1|2
```

注意事项

- from_csv 和 to_csv 是互逆操作。
- options 参数是一个MAP类型，支持CSV格式的各种选项，如 delimiter（分隔符）、header（是否有表头）、quote（引号字符）等。
- schema_of_csv 推断的列名为 col0、col1 等，除非指定 header 选项为 true。
- 这些函数在Spark 3.0+引入，Spark 4.0中功能更完善。

15.9 XML 函数

功能描述

XML函数用于处理XML格式的数据。支持原生XML解析和生成函数，可以在SQL中直接读取和生成XML格式的字符串。

函数列表

函数	描述
from_xml(xml_str, schema [, options])	将XML字符串解析为结构体。使用指定的schema解析XML字符串。
schema_of_xml(xml [, options])	推断XML字符串的DDL格式schema。
to_xml(expr [, options])	将结构体转换为XML字符串。

函数	描述
xpath(xml, xpath)	返回XML节点中匹配XPath表达式的值数组。
xpath_boolean(xml, xpath)	如果XPath表达式求值为true或找到匹配节点，则返回true。
xpath_double(xml, xpath)	返回XPath表达式求值的double值。
xpath_float(xml, xpath)	返回XPath表达式求值的float值。
xpath_int(xml, xpath)	返回XPath表达式求值的int值。
xpath_long(xml, xpath)	返回XPath表达式求值的long值。
xpath_number(xml, xpath)	返回XPath表达式求值的数值（同xpath_double）。
xpath_short(xml, xpath)	返回XPath表达式求值的short值。
xpath_string(xml, xpath)	返回XPath表达式匹配的第一个文本节点。

示例

from_xml — 解析XML字符串

```
SELECT from_xml('<person><id>1</id><name>Alice</name></person>', 'id INT, name STRING');
-- {1, Alice}
```

schema_of_xml — 推断XML的schema

```
SELECT schema_of_xml('<person><id>1</id><name>Alice</name></person>');
-- STRUCT<id:INT, name:STRING>
```

to_xml — 将结构体转为XML

```
SELECT to_xml(named_struct('id', 1, 'name', 'Alice'));
-- <ROW><id>1</id><name>Alice</name></ROW>
```

xpath — 查询XML节点

```
SELECT xpath('<a><b>1</b><b>2</b></a>', 'a/b/text()');
-- ["1", "2"]
```

xpath_string — 查询XML文本

```
SELECT xpath_string('<a><b>hello</b></a>', 'a/b');
-- hello
```

注意事项

- from_xml 和 to_xml 是互逆操作。
- options 参数是一个MAP类型，支持XML格式的各种选项。

- 使用XML函数前，需要确保Spark已加载XML支持模块（spark-xml）。

15.10 哈希函数

功能描述

哈希函数用于计算输入值的哈希摘要。支持多种哈希算法，包括MD5、SHA-1、SHA-224、SHA-256、SHA-384、SHA-512和CRC32。

函数列表

函数	描述
md5(expr)	返回字符串的MD5哈希值（128位，32个十六进制字符）。
sha1(expr)	返回字符串的SHA-1哈希值（160位，40个十六进制字符）。
sha2(expr, bitLength)	返回字符串的SHA-2族哈希值。bitLength可选值：0（等同256）、224、256、384、512。
sha(expr)	sha1的别名。
crc32(expr)	返回字符串的CRC32校验值。
hash(expr [, ...])	返回参数的哈希值。使用Murmur3哈希算法。
xxhash64(expr [, ...])	返回参数的64位xxHash哈希值。

示例

md5

```
SELECT md5('Spark');
-- 8cde774d6f7333752ed72cacddb05126
```

sha1 / sha

```
SELECT sha1('Spark');
-- 85f5955f4b27a9a4c2aab6ffe0d7cb9d2f54c0fc

SELECT sha('Spark');
-- 85f5955f4b27a9a4c2aab6ffe0d7cb9d2f54c0fc
```

sha2

```
SELECT sha2('Spark', 256);
-- 529bc3b07127ecb7e53a4dcf1991d9152cfcbb08f2f268c0a8214e1c0a3d2e0f

SELECT sha2('Spark', 512);
-- 3f9c71d6b5a8e4c1b1c5c5e4a8b9c3d2e1f0a9b8c7d6e5f4a3b2c1d0e9f8a7b6
```

crc32

```
SELECT crc32('Spark');
-- 1557323817
```

hash


```
SELECT hash('Spark', 123);
-- -1153545427
```

xxhash64

```
SELECT xxhash64('Spark');
-- 3511078467956106287
```

注意事项

- md5、sha1、sha2 接受STRING或BINARY类型输入。
- sha2 的 bitLength 参数仅支持0、224、256、384、512，其他值会返回NULL。
- hash 和 xxhash64 不是加密哈希函数，不应用于安全场景。
- hash 使用Murmur3算法，适用于分布式场景下的数据分桶和分区。

15.11 谓词函数

功能描述

谓词函数用于判断条件是否成立，返回BOOLEAN值。Spark 4.0将常用条件判断函数归类为谓词函数，包括比较运算符、空值判断、模式匹配和逻辑运算等。

函数/操作符列表

函数/操作符	描述
expr1 = expr2	等于。如果expr1等于expr2，返回true。
expr1 == expr2	等于（ =的别名 ）。
expr1 != expr2 / expr1 <> expr2	不等于。
expr1 < expr2	小于。
expr1 <= expr2	小于等于。
expr1 > expr2	大于。
expr1 >= expr2	大于等于。
expr1 <=> expr2	NULL安全等于。非NULL操作数与=相同，但两个NULL返回true，一个NULL返回false。
equal_null(expr1, expr2)	与<=>相同。非NULL操作数与=相同，两个NULL返回true。
expr1 AND expr2	逻辑与。
expr1 OR expr2	逻辑或。
NOT expr / ! expr	逻辑非。
expr1 IN(expr2, expr3, ...)	如果expr等于任一值，返回true。

函数/操作符	描述
str LIKE pattern [ESCAPE escape]	模式匹配。_匹配单个字符，%匹配任意字符序列。
str ILIKE pattern [ESCAPE escape]	不区分大小写的模式匹配。
regexp(str, regexp)	如果str匹配正则表达式，返回true。
regexp_like(str, regexp)	如果str匹配正则表达式，返回true。
rlike(str, regexp)	regexp的别名。如果str匹配正则表达式，返回true。
isnan(expr)	如果值为NaN（Not a Number），返回true。
isnull(expr)	如果值为NULL，返回true。
isnotnull(expr)	如果值不为NULL，返回true。
nanvl(expr1, expr2)	如果expr1为NaN，返回expr2；否则返回expr1。
nullif(expr1, expr2)	如果expr1等于expr2，返回NULL；否则返回expr1。
nvl(expr1, expr2)	如果expr1为NULL，返回expr2；否则返回expr1。
nvl2(expr1, expr2, expr3)	如果expr1不为NULL，返回expr2；否则返回expr3。
coalesce(expr1, expr2, ...)	返回第一个非NULL值。
assert_true(expr)	如果expr为true，返回NULL；否则抛出异常。
assert_true(expr, message)	如果expr为true，返回NULL；否则抛出带指定消息的异常。
raise_error(expr)	抛出指定错误消息的异常。

示例

比较运算符

```
SELECT 1 < 2;
-- true

SELECT 2 <= 2;
-- true

SELECT 2 > 1;
-- true

SELECT 2 >= 1;
-- true

SELECT 1 = '1';
-- true
```



```
SELECT 1 != 2;
-- true
```

NULL安全等于（ <=> / equal_null ）

```
SELECT NULL <=> NULL;
-- true

SELECT NULL = NULL;
-- NULL

SELECT equal_null(NULL, NULL);
-- true

SELECT equal_null(3, 3);
-- true
```

逻辑运算

```
SELECT true AND false;
-- false

SELECT true OR false;
-- true

SELECT NOT true;
-- false

SELECT ! true;
-- false
```

LIKE / ILIKE 模式匹配

```
SELECT like('Spark', '_park');
-- true

SELECT ilike('Spark', '_Park');
-- true（不区分大小写）
```

正则表达式匹配

```
SELECT regexp('%SystemDrive%\\Users\\John', r'%SystemDrive%\\Users.*');
-- true

SELECT regexp_like('%SystemDrive%\\Users\\John', r'%SystemDrive%\\Users.*');
-- true

SELECT rlike('%SystemDrive%\\Users\\John', r'%SystemDrive%\\Users.*');
-- true
```

IN 运算符

```
SELECT 1 IN(1, 2, 3);
-- true

SELECT 1 IN(2, 3, 4);
-- false
```

空值判断

```
SELECT isnan(cast('NaN' AS DOUBLE));
-- true

SELECT isnull(NULL);
-- true

SELECT isnotnull(1);
-- true
```

nullif / nvl / nvl2 / coalesce


```
SELECT nullif(1, 1);
-- NULL

SELECT nullif(1, 2);
-- 1

SELECT nvl(NULL, 123);
-- 123

SELECT nvl2(NULL, 'yes', 'no');
-- no

SELECT nvl2(1, 'yes', 'no');
-- yes

SELECT coalesce(NULL, NULL, 3, 5);
-- 3
```

nanvl

```
SELECT nanvl(cast('NaN' AS DOUBLE), 123);
-- 123.0
```

assert_true / raise_error

```
SELECT assert_true(1 = 1);
-- NULL
```

注意事项

- 谓词函数在WHERE、HAVING和CASE WHEN等条件表达式中特别有用。
- isnan仅适用于浮点类型（FLOAT、DOUBLE）。
- coalesce接受多个参数，返回第一个非NULL值。所有参数类型必须兼容。
- LIKE区分大小写，ILIKE不区分大小写。
- regexp、regexp_like、rlike功能相同，均为正则表达式匹配。
- assert_true和aise_error可用于数据验证场景。

15.12 类型转换函数

功能描述

类型转换函数用于在不同数据类型之间进行转换。Spark 4.0将常用类型转换函数归类，包括显式转换、格式化转换和特殊类型转换。

函数列表

函数	描述
cast(expr AS type)	将表达式转换为指定类型。标准SQL类型转换。也支持 expr :: type 语法。
try_cast(expr AS type)	将表达式转换为指定类型。转换失败时返回NULL而非报错。
typeof(expr)	返回表达式的数据类型名称（DDL格式字符串）。
bigint(expr)	将值转换为BIGINT类型。

函数	描述
binary(expr)	将值转换为BINARY类型。
boolean(expr)	将值转换为BOOLEAN类型。
date(expr)	将值转换为DATE类型。
decimal(expr)	将值转换为DECIMAL类型。
double(expr)	将值转换为DOUBLE类型。
float(expr)	将值转换为FLOAT类型。
int(expr)	将值转换为INT类型。
smallint(expr)	将值转换为SMALLINT类型。
string(expr)	将值转换为STRING类型。
timestamp(expr)	将值转换为TIMESTAMP类型。
tinyint(expr)	将值转换为TINYINT类型。

示例

cast — 标准类型转换

```
SELECT cast('123' AS INT);
-- 123

SELECT cast(3.14 AS INT);
-- 3

SELECT cast('2024-01-01' AS DATE);
-- 2024-01-01
```

使用 :: 语法（PostgreSQL风格）

```
SELECT '10' :: INT;
-- 10

SELECT '2024-01-01' :: DATE;
-- 2024-01-01
```

try_cast — 安全类型转换

```
SELECT try_cast('123' AS INT);
-- 123

SELECT try_cast('abc' AS INT);
-- NULL（转换失败返回NULL，不报错）
```

typeof — 查看数据类型

```
SELECT typeof(1);
-- int
```



```
SELECT typeof('hello');
-- string

SELECT typeof(array(1, 2, 3));
-- array<int>
```

简写类型转换函数

```
SELECT int('123');
-- 123

SELECT string(3.14);
-- 3.14

SELECT date('2024-01-01');
-- 2024-01-01
```

注意事项

- cast 在转换失败时会报错，try_cast 在转换失败时返回NULL。
- try_cast 是Spark 3.0+新增的函数，适用于数据清洗场景，避免因个别脏数据导致整个查询失败。
- 简写类型转换函数（如 int()、string()）等价于 cast(expr AS type)。
- Spark 4.0支持PostgreSQL风格的 :: 类型转换语法，如 '10' :: INT。
- typeof 返回的是DDL格式的类型字符串，可用于调试和动态schema处理。

15.13 生成器函数

功能描述

生成器函数（Generator Functions）用于将单个输入行展开为多行输出。生成器函数与LATERAL VIEW配合使用，可以将数组或Map展开为多行。

函数列表

函数	描述
collations()	返回所有Spark SQL字符串排序规则（collation）信息。
explode(expr)	将数组展开为多行，或将Map展开为多行（key/value列）。
explode_outer(expr)	与explode类似，但输入为NULL或空数组时也输出一行（值为NULL）。
inline(expr)	将结构体数组展开为多行。每个结构体的字段成为独立的列。
inline_outer(expr)	与inline类似，但输入为NULL或空数组时也输出一行（值为NULL）。
posexplode(expr)	将数组展开为多行，同时输出元素的位置（从0开始）。
posexplode_outer(expr)	与posexplode类似，但输入为NULL或空数组时也输出一行。

函数	描述
stack(n, expr1, ..., exprk)	将expr1, ..., exprk分为n行输出。
sql_keywords()	返回Spark SQL关键字列表。
json_tuple(jsonStr, p1, p2, ..., pn)	从JSON字符串中提取指定字段的值，每个字段输出为一列。

示例

explode — 展开数组

```
SELECT explode(array(10, 20, 30));
-- 10
-- 20
-- 30
```

explode — 展开Map

```
SELECT explode(map('a', 1, 'b', 2));
-- key | value
-- a   | 1
-- b   | 2
```

explode_outer — 保留NULL行

```
SELECT explode_outer(array(NULL, 1, 2));
-- NULL
-- 1
-- 2

-- 对比：explode遇到NULL数组元素会跳过
SELECT explode(array(1, 2));
-- 1
-- 2
```

posexplode — 带位置的展开

```
SELECT posexplode(array('a', 'b', 'c'));
-- pos | val
-- 0   | a
-- 1   | b
-- 2   | c
```

inline — 展开结构体数组

```
SELECT inline(array(named_struct('a', 1, 'b', 2), named_struct('a', 3, 'b', 4)));
-- a | b
-- 1 | 2
-- 3 | 4
```

stack — 行转列

```
SELECT stack(2, 1, 2, 3, 4);
-- col0 | col1
-- 1    | 2
-- 3    | 4
```

collations — 查看排序规则

```
SELECT * FROM collations() WHERE NAME = 'UTF8_BINARY';
-- CATALOG=SYSTEM, SCHEMA=BUILTIN, NAME=UTF8_BINARY, ...
```


sql_keywords — 查看SQL关键字

```
SELECT * FROM sql_keywords() LIMIT 2;
-- keyword=ADD, reserved=false
-- keyword=AFTER, reserved=false
```

json_tuple — 提取JSON字段

```
SELECT json_tuple('{"a":1, "b":2, "c":3}', 'a', 'b');
-- a | b-- 1 | 2
```

注意事项

- 生成器函数只能在SELECT列表中使用，且不能嵌套在其他表达式中。
- explode 与 explode_outer 的区别：当输入为NULL时，explode 不输出任何行，explode_outer 输出一行NULL。
- posexplode 的位置从0开始。
- stack 的参数个数必须是行数n的整数倍。
- 在LATERAL VIEW中使用生成器函数时，需要为展开的列指定别名。

16 标识符

16.1 aggregate_func

格式

无。

说明

聚合函数。

aggregate_func通常在数据库查询中用于对一组值进行计算并返回单个结果。

16.2 alias

格式

无。

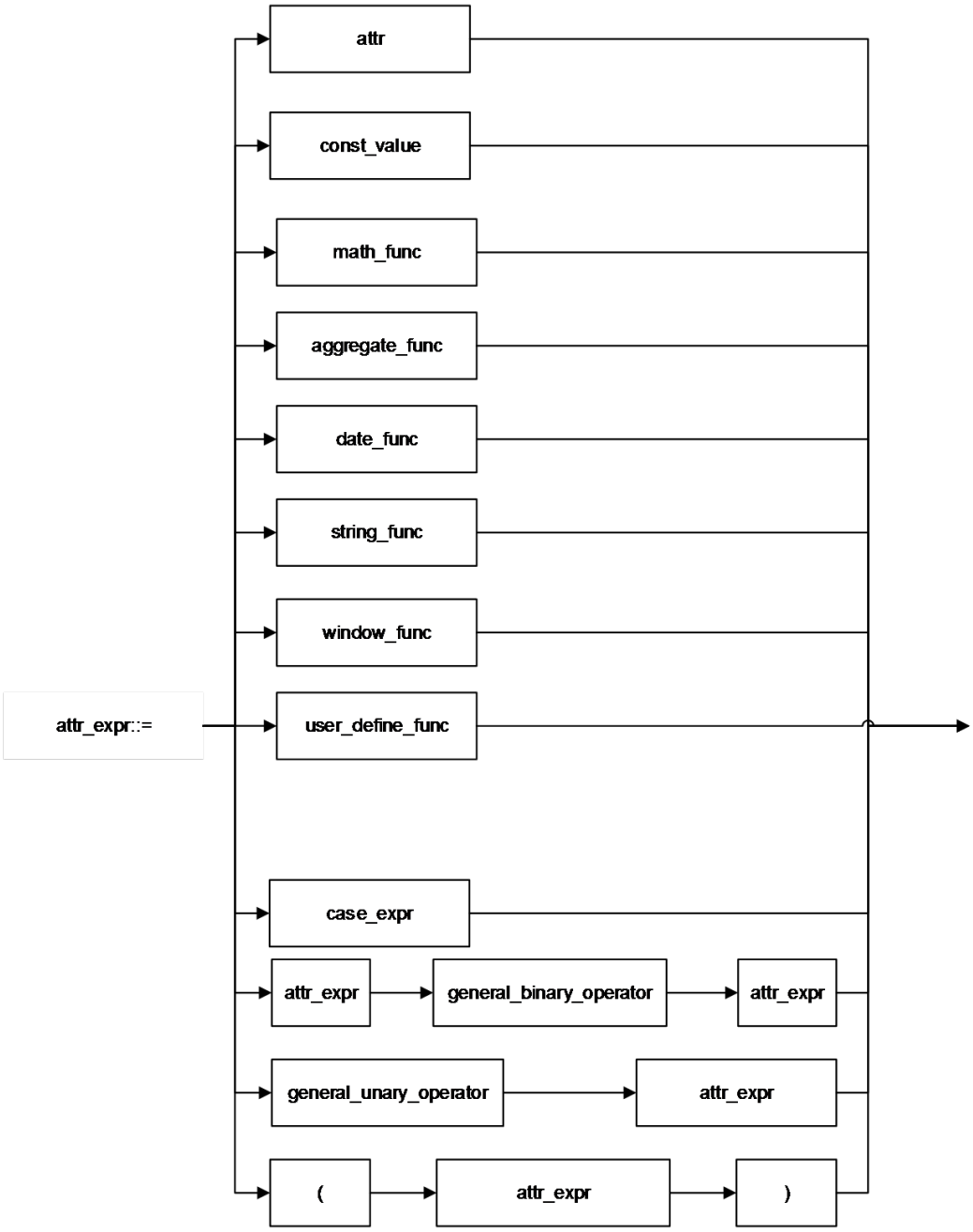
说明

别名，可给字段、表、视图、子查询起别名，仅支持字符串类型。

16.3 attr_expr

属性表达式，用于SELECT列表、WHERE条件等场景。可包含：字段名(attr)、常量(const_value)、CASE表达式、函数调用(数学/日期/字符串/聚合/窗口/UDF)、二元/一元运算符、子表达式(括号包围)。

格式



说明

语法	描述
attr_expr	属性表达式。
attr	表的字段，与col_name相同。
const_value	常量值。
case_expr	case表达式。

语法	描述
math_func	数学函数。
date_func	日期函数。
string_func	字符串函数。
aggregate_func	聚合函数。
window_func	分析窗口函数。
user_define_func	用户自定义函数。
general_binary_operator	普通二元操作符。
general_unary_operator	普通一元操作符。
(	指定子属性表达式开始。
)	指定子属性表达式结束。

16.4 attr_expr_list

属性表达式列表。

格式

```
attr_expr [, attr_expr ...]
```

说明

attr_expr列表，多个属性表达式以逗号分隔。

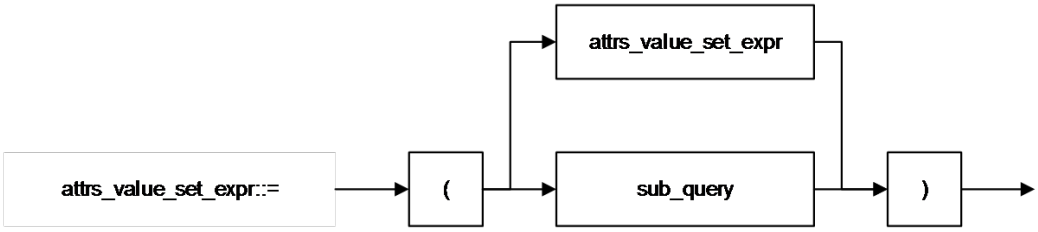
16.5 attrs_value_set_expr

属性值集合表达式，用于INSERT语句的VALUES子句。

格式

每组值用括号包围，多组之间用逗号分隔。

```
(value1, value2, ...) [, (value1, value2, ...)]
```



说明

语法	描述
attrs_value_set_expr	属性值集合。
sub_query	子查询语句。
(	指定子查询表达式开始。
)	指定子查询表达式结束。

16.6 boolean_expression

布尔表达式，返回TRUE、FALSE或NULL。

格式

可包含：比较运算符、逻辑运算符(AND/OR/NOT)、IN/EXISTS子查询、IS NULL等。

说明

返回boolean类型的表达式。

16.7 class_name

Java类的全限定类名，用于UDF或SerDe。

格式

例如org.apache.spark.sql.udf.MyUDF。

说明

函数所依赖的类名，注意类名需要包含类所在包的完整路径。

16.8 col

表的字段，与col_name相同。

格式

无。

说明

函数调用时的形参，一般即为字段名称。

16.9 col_comment

列注释，对列的描述信息。

格式

字符串常量，用单引号包围。

说明

对列（字段）的描述，仅支持字符串类型，描述长度不能超过256字节。

16.10 col_name

格式

无。

说明

列名，即字段名称，仅支持字符串类型，名称长度不能超过128个字节。

16.11 col_name_list

格式

```
col_name [, col_name ...]
```

说明

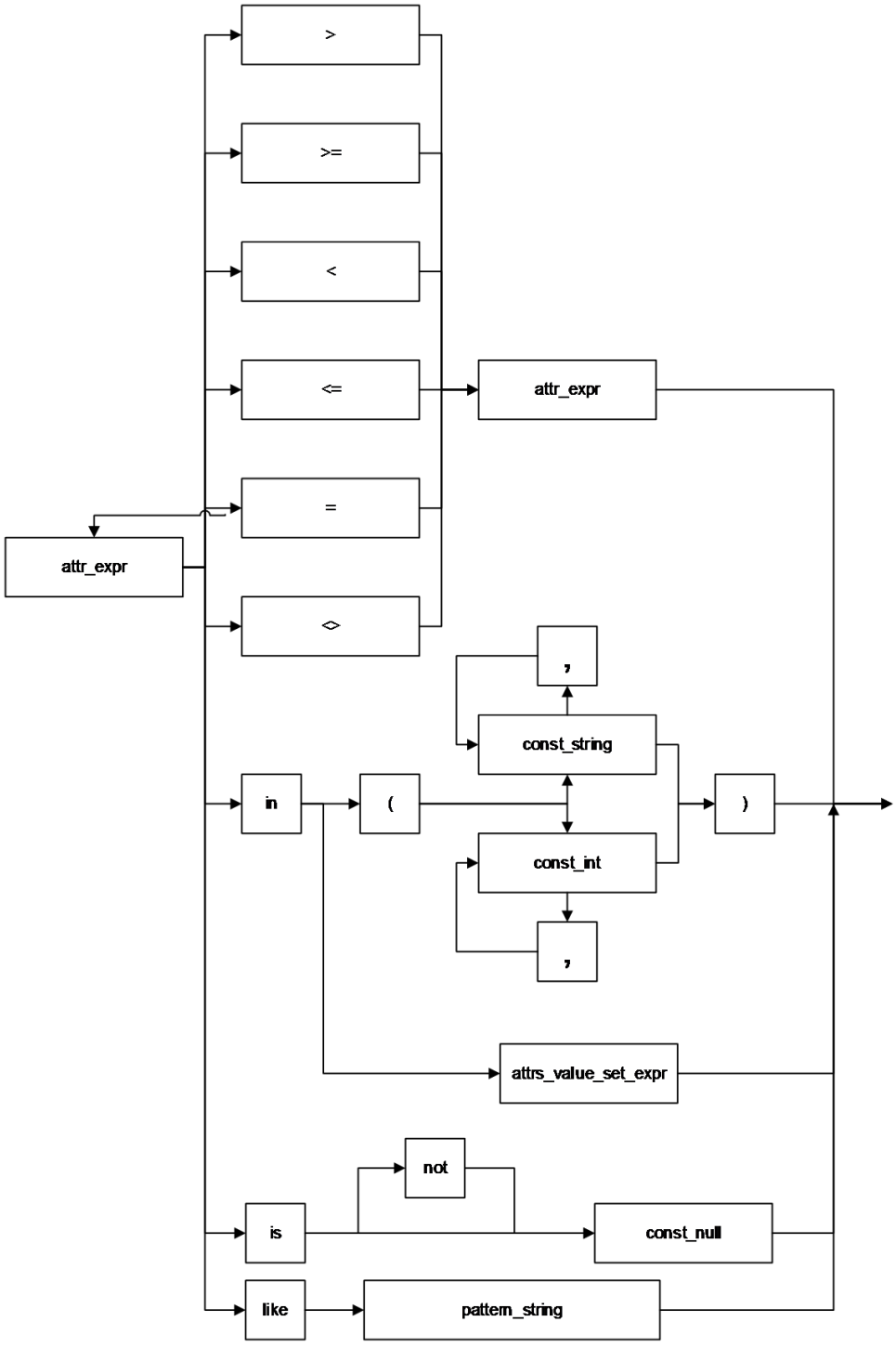
字段列表，可由一个或多个col_name构成，多个col_name之间用逗号分隔。

16.12 condition

条件表达式，用于WHERE、HAVING等子句。

可包含：比较运算、逻辑运算、BETWEEN、IN、LIKE、IS NULL、EXISTS子查询等。

格式



说明

语法	描述
condition	逻辑判断条件。
>	关系运算符：大于。

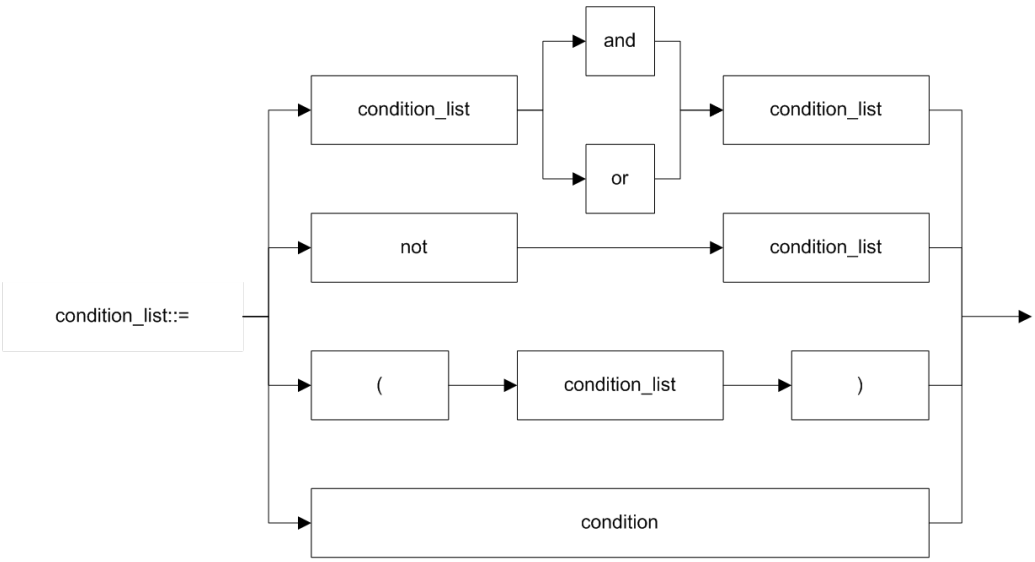
语法	描述
>=	关系运算符：大于等于。
<	关系运算符：小于。
<=	关系运算符：小于等于。
=	关系运算符：等于。
<>	关系运算符：不等于。
is	关系运算符：是。
is not	关系运算符：不是。
const_null	常量：空值。
like	关系运算符：用于通配符匹配。
pattern_string	模式匹配字符串，支持通配符匹配。LIKE条件过滤时，支持SQL通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。
attr_expr	属性表达式。
attrs_value_set_expr	属性值集合。
in	关键字，用于判断属性是否在一个集合中。
const_string	字符串常量。
const_int	整型常量。
(	指定常量集合开始。
)	指定常量集合结束。
,	逗号分隔符。

16.13 condition_list

条件列表，多个条件以AND/OR连接。

格式

```
condition [AND|OR condition ...]
```

说明

语法	描述
condition_list	逻辑判断条件列表。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。

16.14 cte_name

格式

无。

说明

公共表表达式的名字。

16.15 data_type

数据类型，指定列或表达式的数据类型。如INT、BIGINT、DOUBLE、STRING、DATE、TIMESTAMP等。

格式

无。

说明

当前只支持原生数据类型。

16.16 db_comment

格式

无。

说明

对数据库的描述，仅支持字符串类型，描述长度不能超过256字节。

16.17 db_name

格式

无。

说明

数据库名称，仅支持字符串类型，名称长度不能超过128字节。

16.18 else_result_expression

格式

无。

说明

CASE WHEN语句中ELSE语句后的返回结果。

16.19 file_format

数据文件格式。

格式

| AVRO

| CSV

| JSON

| ORC

| PARQUET

说明

- 目前包含以上5种格式。
- 指定数据格式的方式有两种，一种是USING，可指定以上5种数据格式，另一种是STORED AS，只能指定ORC和PARQUET。
- ORC对RCFile做了优化，可以提供一种高效的方法来存储Hive数据。
- PARQUET是面向分析型业务的列式存储格式。

16.20 file_path

格式

无。

说明

文件路径，指数据文件的存储位置。该路径是OBS路径。

16.21 function_name

格式

无。

说明

函数名称，仅支持字符串类型。

16.22 groupby_expression

格式

无。

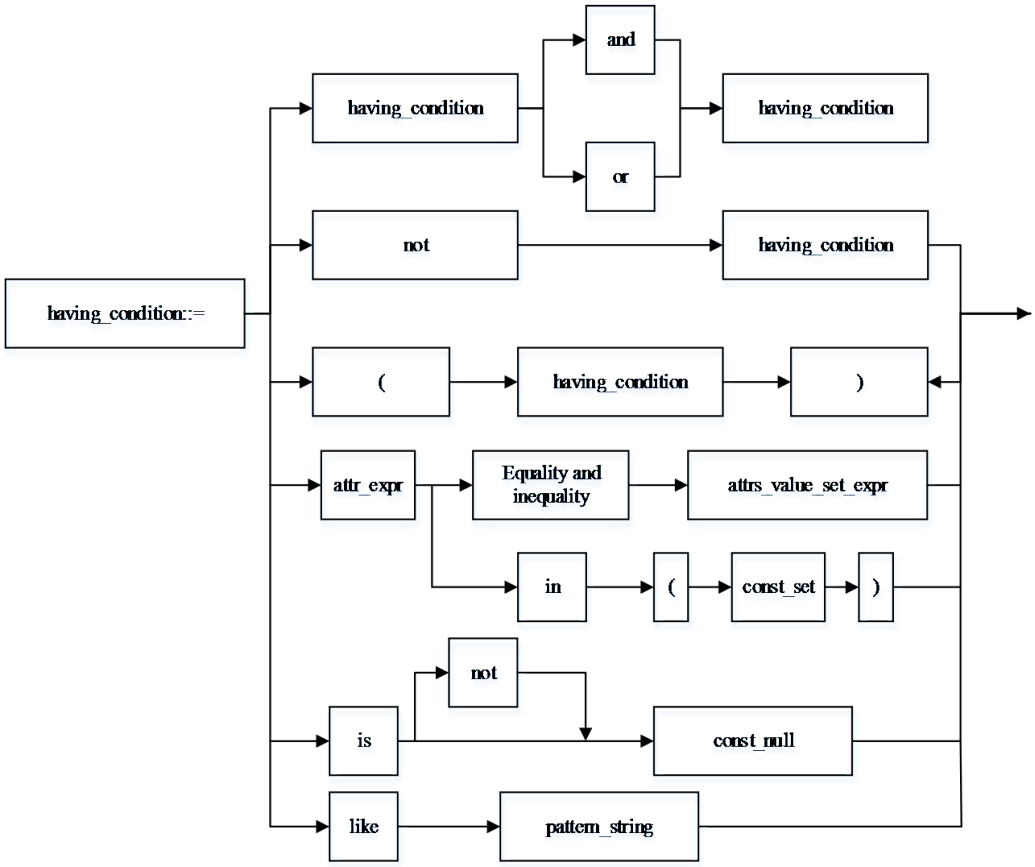
说明

包含GROUP BY的表达式。

16.23 having_condition

HAVING条件，对分组后的结果进行过滤。

格式



说明

语法	描述
having_condition	having逻辑判断条件。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。
const_set	常量集合，元素间逗号分隔。
in	关键字，用于判断属性是否在一个集合中。
attrs_value_set_expr	属性值集合。
attr_expr	属性表达式。
Equality and inequality	等式与不等式，详情请参见**17.1 关系运算符**。

语法	描述
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE条件过滤时，支持SQL通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。
like	关系运算符：用于通配符匹配。

16.24 hdfs_path

格式

无。

说明

HDFS的路径，如“hdfs:///tmp”。

16.25 input_expression

格式

无。

说明

CASE WHEN的输入表达式。

16.26 input_format_classname

格式

无。

说明

指定输入格式的类名，如***org.apache.hadoop.mapred.TextInputFormat。***

16.27 jar_path

格式

无。

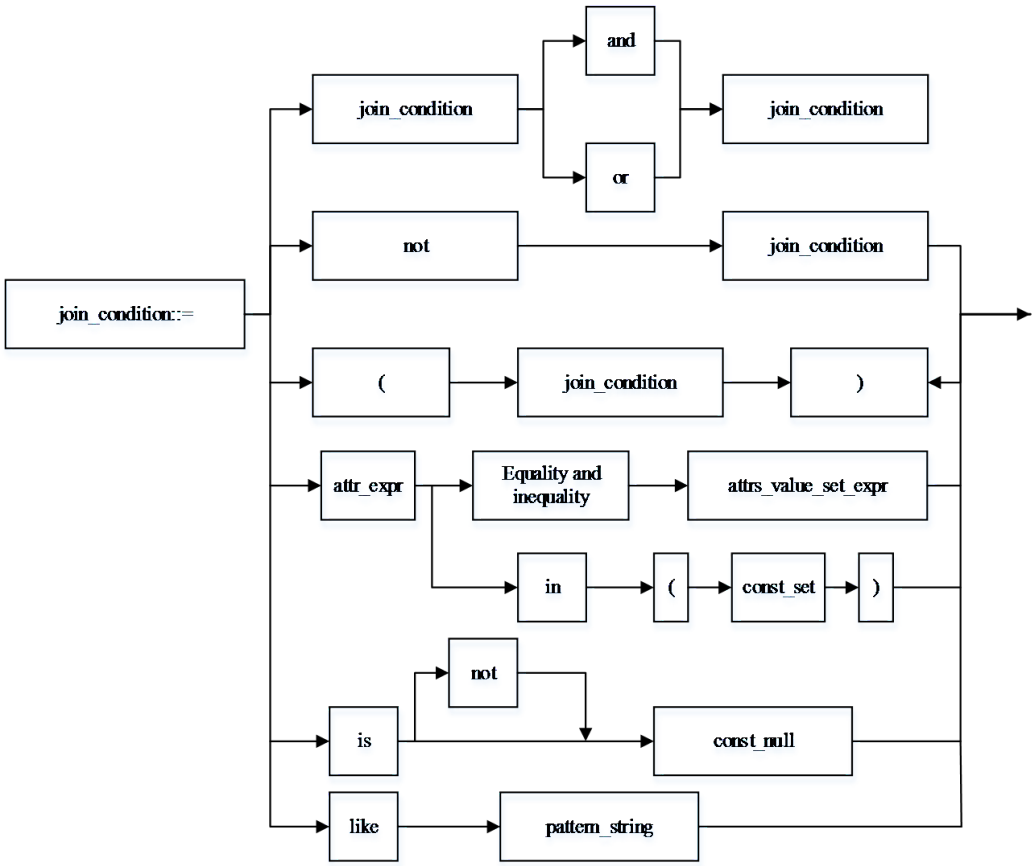
说明

jar包路径，该路径可以是本地路径也可以是HDFS路径。

16.28 join_condition

JOIN连接条件，指定两表之间的关联规则。

格式



说明

语法	描述
join_condition	join逻辑判断条件。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。
const_set	常量集合，元素间逗号分隔。

语法	描述
in	关键字，用于判断属性是否在一个集合中。
attrs_value_set_expr	属性值集合。
attr_expr	属性表达式。
Equality and inequality	等式与不等式，详情请参见17.1 关系运算符。
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE条件过滤时，支持SQL通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。

16.29 non_equi_join_condition

格式

无。

说明

指非等值连接条件。

包含<、>、<=、>=、<>等比较运算符的JOIN条件。

16.30 number

格式

无。

说明

LIMIT限制输出的行数，只支持INT类型。

16.31 num_buckets

格式

无。

说明

分桶的个数，仅支持INT类型。

用于CLUSTERED BY ... INTO N BUCKETS。

16.32 output_format_classname

格式

无。

说明

指定输出格式的类名，如 ***org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat。***

16.33 partition_col_name

格式

无。

说明

分区列名，即分区字段名称，仅支持字符串类型。

16.34 partition_col_value

格式

无。

说明

分区列值，即分区字段的值。

常量值，用单引号包围（字符串类型）或直接写（数值类型）。

16.35 partition_specs

格式

```
partition_specs : (partition_col_name = partition_col_value, partition_col_name = partition_col_value, ...);
```

说明

表的分区列表，以key=value的形式表现，key为partition_col_name ，value为partition_col_value ，若存在多个分区字段，每组key=value之间用逗号分隔。

16.36 property_name

格式

无。

说明

属性名称，用于TBLPROPERTIES或DBPROPERTIES。

仅支持字符串类型。

16.37 property_value

格式

无。

说明

属性值，仅支持字符串类型。

16.38 regex_expression

格式

无。

说明

模式匹配字符串，支持正则表达式匹配。

16.39 result_expression

格式

无。

说明

CASE WHEN语句中THEN语句后的返回结果。

16.40 row_format

行格式，指定Hive表的行序列化/反序列化方式。

格式

ROW FORMAT DELIMITED

[FIELDS TERMINATED BY separator]

[COLLECTION ITEMS TERMINATED BY separator]

[MAP KEYS TERMINATED BY separator] [LINES TERMINATED BY separator]

[NULL DEFINED AS separator]

| SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value, property_name=property_value, ...)]

说明

- separator指语法中的分隔符或替代符，仅支持CHAR类型。
- FIELDS TERMINATED BY指定表中字段级别的分隔符，仅支持CHAR类型。
- COLLECTION ITEMS TERMINATED BY指定集合级别的分隔符，仅支持CHAR类型。
- MAP KEYS TERMINATED BY仅用于指定MAP类型中的key与value之间的分隔符号，仅支持CHAR类型。
- LINES TERMINATED BY指定行与行之间的分隔符，目前只支持“\n”。
- 使用NULL DEFINED AS子句可以指定NULL的格式。
- SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value, property_name=property_value, ...)]可利用以下语句实现NULL值转换为空字符串。

  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe' with serdeproperties('serialization.null.format' = '')

16.41 select_statement

格式

无。

说明

SELECT基本语句，即查询语句。

16.42 separator

格式

无。

说明

分隔符，仅支持CHAR类型，支持用户自定义，如逗号、分号、冒号等。

16.43 serde_name

格式

无。

说明

SerDe类的全限定类名。

指定SerDe的名称。如org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe。

16.44 sql_containing_cte_name

格式

```
WITH cte_name AS (...) SELECT ...
```

说明

包含了cte_name定义的公共表表达式的SQL语句。

16.45 sub_query

子查询，嵌套在另一个查询中的SELECT语句。

格式

用括号包围的SELECT语句。

说明

子查询。

16.46 table_comment

格式

无。

说明

对表的描述，仅支持字符串类型，描述长度不能超过256字节。

16.47 table_name

格式

无。

说明

表名称，支持字符串类型和“$”符号，名称长度不能超过128字节。

可使用db_name.table_name格式指定数据库。

16.48 table_properties

格式

```
('property_name' = 'property_value' [, ...])
```

说明

表的属性列表，以key=value的形式表示，key为property_name，value为property_value，列表中每组key=value之间用逗号分隔。

16.49 table_reference

格式

无。

说明

表引用，FROM子句中引用的数据源。

即表或视图的名称，仅支持字符串类型，也可为子查询，当为子查询时，必须加别名。

16.50 view_name

格式

无。

说明

视图名称，仅支持字符串类型，名称长度不能超过128个字节。

16.51 view_properties

格式

```
('property_name' = 'property_value' [, ...])
```

说明

视图的属性列表，以key=value的形式表示，key为property_name，value为property_value，列表中每组key=value之间用逗号分隔。

16.52 when_expression

格式

无。

说明

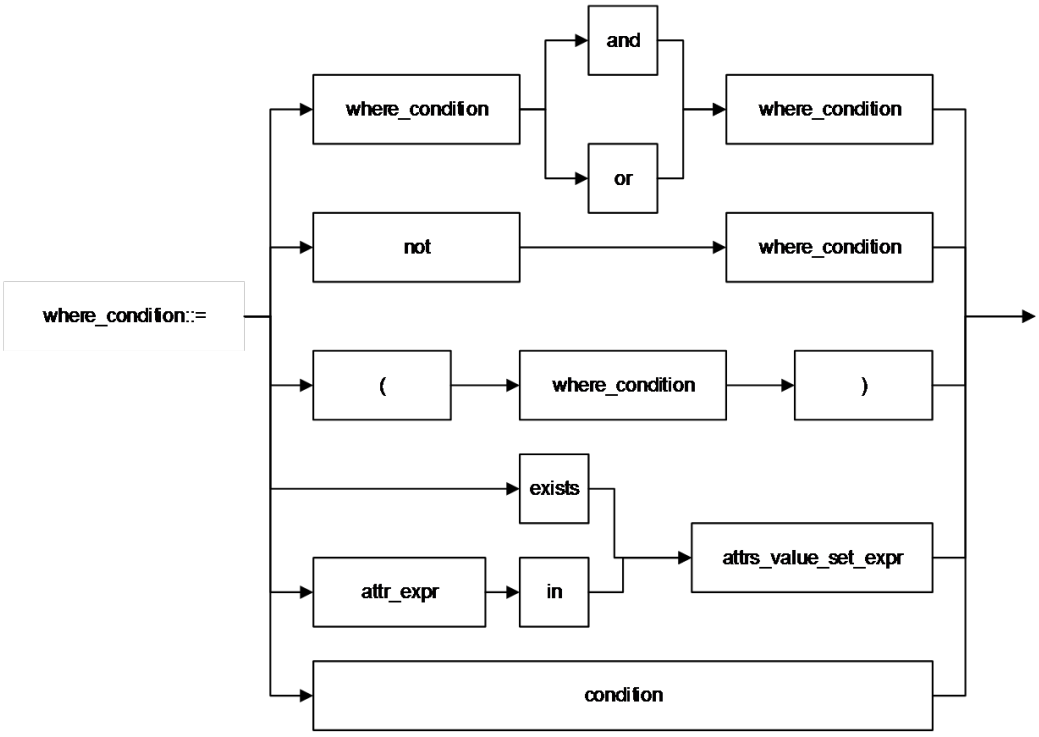
CASE语句中WHEN后的条件或值。

简单CASE：与input_expression比较的值；搜索CASE：布尔条件。

16.53 where_condition

WHERE条件，用于过滤行。

格式



说明

语法	描述
where_condition	where逻辑判断条件。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。
exists	关键字，用于判断是否存在一个不为空的集合，若exists后面跟的为子查询，子查询中须包含逻辑判断条件。
in	关键字，用于判断属性是否在一个集合中。
attrs_value_set_expr	属性值集合。
attr_expr	属性表达式。

16.54 window_function

格式

无。

说明

窗口函数，在OVER窗口上执行计算。

如row_number、rank、lag、lead等。

17 运算符

17.1 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。

表 17-1 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相等，返回TRUE，否则返回FALSE。不能用于赋值操作。
A <=> B	BOOLEAN	若A与B相等，返回TRUE，否则返回FALSE。若A与B都为NULL则返回TRUE，其中一个为NULL则返回FALSE。
A <> B	BOOLEAN	若A与B不相等，返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。标准SQL语法。
A != B	BOOLEAN	与<>逻辑操作符相同。SQL Server语法。
A < B	BOOLEAN	若A小于B，返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A <= B	BOOLEAN	若A小于或等于B，返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。

运算符	返回类型	描述
A > B	BOOLEAN	若A大于B，返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A >= B	BOOLEAN	若A大于或等于B，返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A BETWEEN B AND C	BOOLEAN	若A大于等于B且小于等于C则返回TRUE，否则返回FALSE。若A、B、C中存在NULL，则返回NULL。
A NOT BETWEEN B AND C	BOOLEAN	若A小于B或大于C则返回TRUE，否则返回FALSE。若A、B、C中存在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。
A LIKE B	BOOLEAN	若字符串A与字符串B相匹配则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A NOT LIKE B	BOOLEAN	若字符串A与字符串B不相匹配则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A RLIKE B	BOOLEAN	Java正则匹配，若A或其子字符串与B相匹配则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A REGEXP B	BOOLEAN	与A RLIKE B结果相同。

注意事项

- = 和 == 的区别：= 可用于赋值操作，== 不能用于赋值操作。
- <=> 是NULL安全的等值运算符，当两边都为NULL时返回TRUE。
- <> 是标准SQL的不等于运算符，!= 是SQL Server语法，两者逻辑相同。
- ANSI模式下（Spark 4.0默认开启），涉及NULL的比较会返回NULL而非静默处理。

示例 1：等值比较

```
SELECT 1 = 1;       -- TRUE
SELECT 1 == 2;      -- FALSE
SELECT 1 <=> 1;     -- TRUE
SELECT NULL <=> NULL; -- TRUE
```

示例 2：不等值比较

```
SELECT 1 <> 2;      -- TRUE
SELECT 1 != 1;      -- FALSE
SELECT 1 < 2;       -- TRUE
SELECT 2 >= 2;      -- TRUE
```


示例 3：范围比较和 NULL 判断

```
SELECT 2 BETWEEN 1 AND 3;    -- TRUE
SELECT 5 NOT BETWEEN 1 AND 3; -- TRUE
SELECT NULL IS NULL;          -- TRUE
SELECT 1 IS NOT NULL;         -- TRUE
```

示例 4：模糊匹配

```
SELECT 'hello' LIKE 'h%';     -- TRUE
SELECT 'hello' NOT LIKE 'a%';  -- TRUE
SELECT 'hello' RLIKE 'h.*o';   -- TRUE
SELECT 'hello' REGEXP 'h.*o';  -- TRUE
```

17.2 算术运算符

算术运算符包括双目运算与单目运算，这些运算符都将返回数字类型。

表 17-2 算术运算符

运算符	返回类型	描述
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。
A % B	所有数字类型	A对B取余数，结果数据类型与操作数据类型相关。
A & B	所有数字类型	查看两个参数的二进制表示法的值，并执行按位”与”操作。两个表达式的一位均为1时，则结果的该位为1。否则，结果的该位为0。
A | B	所有数字类型	查看两个参数的二进制表示法的值，并执行按位”或”操作。只要任一表达式的一位为1，则结果的该位为 1。否则，结果的该位为0。
A ^ B	所有数字类型	查看两个参数的二进制表示法的值，并执行按位”异或”操作。当且仅当只有一个表达式的某位上为1时，结果的该位才为1。否则结果的该位为0。
~A	所有数字类型	对一个表达式执行按位”非”操作（取反）。

注意事项

- 整数除法使用div函数，/ 运算符始终返回DOUBLE类型。
- ANSI模式下（Spark 4.0默认开启），整数除以零将抛出异常而非返回NULL。
- 位运算符（&、|、^、~）仅适用于整数类型。

- % 取余运算符与 mod() 函数等效。

示例 1：基本四则运算

```
SELECT 1 + 2;    -- 3
SELECT 5 - 3;    -- 2
SELECT 2 * 3;    -- 6
SELECT 10 / 3;   -- 3.333...
```

示例 2：取余运算

```
SELECT 10 % 3;   -- 1
SELECT 7 % 2;    -- 1
```

示例 3：位运算

```
SELECT 5 & 3;    -- 1   (0101 & 0011 = 0001)
SELECT 5 | 3;    -- 7   (0101 | 0011 = 0111)
SELECT 5 ^ 3;    -- 6   (0101 ^ 0011 = 0110)
SELECT ~5;       -- -6  (按位取反)
```

示例 4：混合类型运算

```
SELECT 1 + 2.5;      -- 3.5 (整数+浮点=浮点)
SELECT 10 / 3;       -- 3.333... (除法返回DOUBLE)
SELECT CAST(10 AS DOUBLE) / 3; -- 3.333...
```

17.3 逻辑运算符

常用的逻辑操作符有AND、OR和NOT，它们的运算结果有三个值，分别为TRUE、FALSE和NULL，其中NULL代表未知。优先级顺序为：NOT>AND>OR。

运算规则请参见**表17-3**，表中的A和B代表逻辑表达式。

表 17-3 逻辑运算符

运算符	返回类型	描述
A AND B	BOOLEAN	若A与B都为TRUE则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A OR B	BOOLEAN	若A或B为TRUE，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。一个为TRUE，另一个为NULL时，返回TRUE。
NOT A	BOOLEAN	若A为FALSE则返回TRUE，若A为NULL则返回NULL，否则返回FALSE。
! A	BOOLEAN	与NOT A相同。
A IN (val1, val2, ...)	BOOLEAN	若A与(val1, val2, ...)中任意值相等则返回TRUE，否则返回FALSE。
A NOT IN (val1, val2, ...)	BOOLEAN	若A与(val1, val2, ...)中任意值都不相等则返回TRUE，否则返回FALSE。

运算符	返回类型	描述
EXISTS (subquery)	BOOLEAN	若子查询返回结果至少包含一行则返回TRUE，否则返回FALSE。
NOT EXISTS (subquery)	BOOLEAN	若子查询返回结果一行都不包含则返回TRUE，否则返回FALSE。

18 约束与限制

目前普通格式的表和Icerbeg表都暂不支持执行如下SQL语句：

- SET
- USE
- USE DATABASE
- REFRESH RESOURCE
- CACHE TABLE
- UNCACHE TABLE
- CLEAR CACHE
- REFRESH TABLE
- RESET
- LIST JARS
- LIST FILES
- ADD JAR
- ADD FILE

