数据湖探索

Spark SQL 语法参考

文档版本 01

发布日期 2021-06-16

华为技术有限公司

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

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

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 Spark SQL 常用配置项说明

本章节为您介绍DLI 批作业SQL语法的常用配置项。

表 1-1 常用配置项

名称	默认值	描述
spark.sql.files.maxRecordsPerFile	0	要写入单个文件的最大记录数。如果该值为零或为负，则没有限制。
spark.sql.autoBroadcastJoinThreshold	209715200	配置执行连接时显示所有工作节点的表的最大字节大小。通过将此值设置为“-1”，可以禁用显示。 **说明** 当前仅支持运行命令**ANALYZE TABLE COMPUTE statistics noscan**的配置单元元存储表，和直接根据数据文件计算统计信息的基于文件的数据源表。
spark.sql.shuffle.partitions	4096	为连接或聚合过滤数据时使用的默认分区数。
spark.sql.dynamicPartitionOverwrite.enabled	false	在动态模式下，Spark不会删除前面的分区，只覆盖那些运行时没有写入数据的分区。
spark.sql.files.maxPartitionBytes	134217728	读取文件时要打包到单个分区中的最大字节数。
spark.sql.badRecordsPath	-	Bad Records的路径。

2 Spark SQL 语法概览

本章节介绍了目前DLI所提供的Spark SQL语法列表。参数说明，示例等详细信息请参考具体的语法说明。

表 2-1 批作业 SQL 语法

语法分类	功能描述
数据库相关语法	创建数据库
	删除数据库
	查看指定数据库
	查看所有数据库
创建OBS表相关语法	使用DataSource语法创建OBS表
	使用Hive语法创建OBS表
创建DLI表相关语法	使用DataSource语法创建DLI表
	使用Hive语法创建DLI表
删除表相关语法	删除表
查看表相关语法	查看所有表
	查看建表语句
	查看表属性
	查看指定表所有列
	查看指定表所有分区
	查看表统计信息
修改表相关语法	添加列
分区表相关语法	添加分区（只支持OBS表）
	重命名分区

语法分类	功能描述
	删除分区
	修改表分区位置（只支持OBS表）
	修改表分区SerDe属性（只支持OBS表）
	更新表分区信息（只支持OBS表）
导入数据相关语法	导入数据
插入数据相关语法	插入数据
清空数据相关语法	清空数据
导出查询结果相关语法	导出查询结果
跨源连接HBase表相关语法	创建表关联HBase
	插入数据至HBase表
	查询HBase表
跨源连接OpenTSDB表相关语法	创建表关联OpenTSDB
	插入数据至OpenTSDB
	查询OpenTSDB表
跨源连接DWS表相关语法	创建表关联DWS
	插入数据至DWS表
	查询DWS表
跨源连接RDS表相关语法	创建表关联RDS
	插入数据至RDS表
	查询RDS表
跨源连接CSS表相关语法	创建表关联CSS
	插入数据至CSS表
	查询CSS表
跨源连接DCS表相关语法	创建表关联DCS
	插入数据至DCS表
	查询DCS表
跨源连接DDS表相关语法	创建表关联DDS
	插入数据至DDS表
	查询DDS表
视图相关语法	创建视图

语法分类	功能描述
	删除视图
查看计划相关语法	查看计划
数据权限相关语法	创建角色
	删除角色
	绑定角色
	解绑角色
	显示角色
	分配权限
	回收权限
	显示已授权限
	显示所有角色和用户的绑定关系
自定义函数相关语法	创建函数
	删除函数
	显示函数详情
	显示所有函数

3 数据库相关

3.1 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]
  [WITH DBPROPERTIES (property_name=property_value, ...)];
```

关键字

- IF NOT EXISTS：所需创建的数据库已存在时使用，可避免系统报错。
- COMMENT：对数据库的描述。
- DBPROPERTIES：数据库的属性，且属性名和属性值成对出现。

参数说明

表 3-1 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。
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。

示例

若testdb数据库不存在，则创建数据库testdb。

```
CREATE DATABASE IF NOT EXISTS testdb;
```

3.2 删除数据库

功能描述

删除数据库。

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database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database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。

参数说明

表 3-2 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

若存在testdb数据库，则删除数据库testdb。

```
DROP DATABASE IF EXISTS testdb;
```


3.3 查看指定数据库

功能描述

查看指定数据库的相关信息，包括数据库名称、数据库的描述等。

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信息。

参数说明

表 3-3 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。

注意事项

如果所要查看的数据库不存在，则系统报错。

示例

查看testdb数据库的相关信息。

```
DESCRIBE DATABASE testdb;
```

3.4 查看所有数据库

功能描述

查看当前工程下所有的数据库。

语法格式

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

关键字

无。

参数说明

表 3-4 参数说明

参数	描述
regex_expression	数据库名称。

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

查看当前的所有数据库。

```
SHOW DATABASES;
```

查看当前的所有以test开头的数据库。

```
SHOW DATABASES LIKE "test.*";
```

4 表相关

4.1 创建 OBS 表

4.1.1 使用 DataSource 语法创建 OBS 表

功能描述

使用DataSource语法创建OBS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 4-1 参数说明

参数	描述
db_name	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	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]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（''）包围起来。
col_name	字段名称。
col_type	字段类型。
col_comment	字段描述。
file_format	OBS表数据存储格式，支持orc，parquet，json，csv，carbon，avro类型。
path	数据存储路径。
table_comment	表描述。
select_statement	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 4-2 OPTIONS 参数描述

参数	描述	默认值
path	指定的表路径，即OBS存储路径。	-
multiLevelDirEnable	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	是否需要在删除表或分区时，清除path路径下的数据。	false

当file_format为csv时，还可以设置以下OPTIONS参数。

表 4-3 CSV 数据格式 OPTIONS 参数说明

参数	描述	默认值
delimiter	数据分隔符。	逗号（即",”）

参数	描述	默认值
quote	引用字符。	双引号（即“"”）
escape	转义字符。	反斜杠（即“\”）
multiLine	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	指定CSV文件中date字段的日期格式	yyyy-MM-dd
timestampFormat	指定CSV文件中timestamp字段的日期格式	yyyy-MM-dd HH:mm:ss
mode	指定解析CSV时的模式，有三种模式。 ● PERMISSIVE：宽容模式，遇到错误的字段时，设置该行整行为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	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	设置代表null的字符，例如，nullValue=“\\N”表示设置\N 代表null。	-
comment	设置代表注释开头的字符，例如，comment='#'表示以#开头的行为注释。	-

注意事项

- 表名与列名为大小写不敏感，即不区分大小写。
- 表名及列名的描述仅支持字符串常量。
- 创建表时要声明列名及对应的数据类型，数据类型为原生类型。
- 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。
- 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。用户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。用户可以在建表语句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)。
- OBS存储路径必须为OBS上的目录，该目录必须事先创建好，且为空。
- 创建分区表时，PARTITONED 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7000个。
- CTAS建表语句不能指定表的属性，不支持创建分区表。

示例

- 创建名为parquetTable的OBS表。
  ```
  CREATE TABLE parquetTable (name string, id int) USING parquet OPTIONS (path "obs://bucketName/filePath");
  ```
- 以班级号（classNo）为分区字段，创建一张名为student的表，包含姓名（name）与分数（score）两个字段。
  ```
  CREATE TABLE IF NOT EXISTS student(name STRING, score DOUBLE, classNo INT) USING csv OPTIONS (PATH 'obs://bucketName/filePath') PARTITIONED BY (classNo);
  ```

  说明

  “classNo”为分区字段，在表字段中要放在最后一个，即“student(name STRING, score DOUBLE, classNo INT)”。
- 创建表t1，并将表t2的数据插入到表t1中。
  ```
  CREATE TABLE t1 USING parquet OPTIONS(path 'obs://bucketName/tblPath') AS select * from t2;
  ```

4.1.2 使用 Hive 语法创建 OBS 表

功能描述

使用Hive语法创建OBS表。

语法格式

```
CREATE [EXTERNAL] 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'
  [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

- EXTERNAL：指创建OBS表。
- 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, CARBON格式。

- LOCATION：指定OBS的路径。创建OBS表时必须指定此关键字。
- TBLPROPERTIES：TBLPROPERTIES子句允许用户给表添加key/value的属性。
- AS：使用CTAS创建表。

参数说明

表 4-4 参数说明

参数	描述
db_name	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	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]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（''）包围起来。
col_name	字段名称。
col_type	字段类型。
col_comment	字段描述。
row_format	行数据格式。
file_format	OBS表存储格式，支持TEXTFILE, AVRO, ORC, SEQUENCEFILE, RCFILE, PARQUET, CARBON
table_comment	表描述。
obs_path	OBS存储路径。
select_statement	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。

注意事项

- 表名与列名为大小写不敏感，即不区分大小写。
- 表名及列名的描述仅支持字符串常量。
- 创建表时要声明列名及对应的数据类型，数据类型为原生类型。
- 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。
- 创建分区表时，PARTITONED 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100000个。
- CTAS建表语句不能指定表的属性，不支持创建分区表。

示例

- 创建一张名为student的parquet格式表，该表包含字段id，name，score，其对应的数据类型分别是INT，STRING，FLOAT。
  ```
  CREATE TABLE student (id INT, name STRING, score FLOAT) STORED AS PARQUET LOCATION 'obs://bucketName/filePath';
  ```
- 以班级号（classNo）为分区字段，创建一张名为student的表，包含姓名（name）与分数（score）两个字段。
  ```
  CREATE TABLE IF NOT EXISTS student(name STRING, score DOUBLE) PARTITIONED BY (classNo INT) STORED AS PARQUET LOCATION 'obs://bucketName/filePath';
  ```

  说明

  “classNo”为分区字段，需要在PARTITIONED BY关键字后指定，即“PARTITIONED BY (classNo INT)”，不能放在表名后作为表字段指定。
- 创建表t1，并将表t2的数据插入到表t1中（Hive语法）。
  ```
  CREATE TABLE t1 STORED AS parquet LOCATION 'obs://bucketName/filePath' as select * from t2;
  ```

4.2 创建 DLI 表

4.2.1 使用 DataSource 语法创建 DLI 表

功能描述

使用DataSource语法创建DLI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 (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 4-5 参数描述

参数	描述
db_name	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	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]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（ '' ）包围起来。
col_name	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。
col_type	字段类型。
col_comment	字段描述。
file_format	DLI表数据存储格式，“file_format”包括“parquet”格式。
table_comment	表描述。
select_statement	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DLI表中。

表 4-6 OPTIONS 参数描述

参数	描述	默认值
multiLevelDirEnable	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

注意事项

- 若没有指定分隔符，则默认为逗号（ , ）。
- 创建分区表时，PARTITONED 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7000个。
- CTAS建表语句不能指定表的属性，不支持创建分区表。

示例

- 创建一张名为src的表，该表包含字段key、value，其对应的数据类型分别是INT、STRING，并可根据需要指定属性。
  ```
  CREATE TABLE src(key INT, value STRING) USING PARQUET OPTIONS('key1' = 'value1');
  ```
- 创建一张名为tb_carbon的表，存储数据格式为carbon，该表包含字段key、value，其对应的数据类型分别是INT、STRING，并可根据需要指定属性。
  ```
  CREATE TABLE tb_carbon(key INT, value STRING) USING CARBON OPTIONS('key1' = 'value1');
  ```
- 以班级号（classNo）为分区字段，创建一张名为student的表，包含姓名（name）与分数（score）两个字段，存储格式为parquet。
  ```
  CREATE TABLE student(name STRING, score INT, classNo INT) USING PARQUET OPTIONS('key1' =
  'value1') PARTITIONED BY(classNo) ;
  ```

  说明

  “classNo”为分区字段，在表字段中要放在最后一个，即“student(name STRING, score INT, classNo INT)”。
- 以班级号（classNo）为分区字段，创建一张名为student的表，包含姓名（name）与分数（score）两个字段，存储格式为carbon。
  ```
  CREATE TABLE student(name STRING, score INT, classNo INT) USING CARBON OPTIONS('key1' =
  'value1') PARTITIONED BY(classNo) ;
  ```

  说明

  “classNo”为分区字段，在表字段中要放在最后一个，即“student(name STRING, score INT, classNo INT)”。
- 创建表t1，并将表t2的数据插入到表t1中。
  ```
  CREATE TABLE t1 USING parquet AS select * from t2;
  ```

4.2.2 使用 Hive 语法创建 DLI 表

功能描述

使用Hive语法创建DLI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
  [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, CARBON几种格式。创建DLI表时必须指定此关键字。
- TBLPROPERTIES：TBLPROPERTIES子句允许用户给表添加key/value的属性。
- AS：使用CTAS创建表。

参数说明

表 4-7 参数描述

参数	描述
db_name	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	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]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（'）包围起来。
col_name	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。
col_type	字段类型。
col_comment	字段描述。
row_format	行数据格式。
file_format	DLI表数据存储格式：支持TEXTFILE, AVRO, ORC, SEQUENCEFILE, RCFILE, PARQUET, CARBON。
table_comment	表描述。
select_statement	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DLI表中。

注意事项

- 创建分区表时，PARTITONED 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100000个。
- CTAS建表语句不能指定表的属性，不支持创建分区表。

示例

- 创建一张名为src的表，该表包含字段key、value，其对应的数据类型分别是INT、STRING，并可根据需要指定属性。

```
CREATE TABLE src
  (key INT, value STRING)
  STORED AS PARQUET
  TBLPROPERTIES('key1' = 'value1');
```

- 以班级号（classNo）为分区字段，创建一张名为student的表，包含姓名（name）与分数（score）两个字段。

```
CREATE TABLE student
  (name STRING, score INT)
  STORED AS PARQUET
  TBLPROPERTIES('key1' = 'value1') PARTITIONED BY(classNo INT);
```

- 创建表t1，并将表t2的数据插入到表t1中。

```
CREATE TABLE t1
  STORED AS PARQUET
  AS select * from t2;
```

4.3 删除表

功能描述

删除表。

语法格式

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

关键字

- OBS表：仅删除其元数据信息，不删除存放在OBS上的数据。
- DLI表：删除其数据及相应的元数据信息。

参数说明

表 4-8 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。
table_name	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。

示例

在当前所在数据库下删除名为student的表。

```
DROP TABLE IF EXISTS student;
```

4.4 查看表

4.4.1 查看所有表

功能描述

查看当前数据库下所有的表。显示当前数据库下的所有表及视图。

语法格式

```
SHOW TABLES [IN | FROM db_name] [LIKE regex_expression];
```

关键字

FROM/IN：指定数据库名，显示特定数据库下的表及视图。

参数说明

表 4-9 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。
regex_expression	数据库下的表名称。

注意事项

无。

示例

- 查看当前所在数据库中的所有表与视图。
  ```
  SHOW TABLES;
  ```
- 查看testdb数据库下所有以test开头的表。
  ```
  SHOW TABLES IN testdb LIKE "test*";
  ```

4.4.2 查看建表语句

功能描述

返回对应表的建表语句。

语法格式

```
SHOW CREATE TABLE table_name;
```

关键字

CREATE TABLE：建表语句。

参数说明

表 4-10 参数说明

参数	描述
table_name	表名称。

注意事项

语句所涉及的表必须存在，否则会出错。

示例

返回test表的建表语句。

```
SHOW CREATE TABLE test;
```

4.4.3 查看表属性

功能描述

查看表的属性。

语法格式

```
SHOW TBLPROPERTIES table_name [('property_name')];
```

关键字

TBLPROPERTIES：TBLPROPERTIES子句允许用户给表添加key/value的属性。

参数说明

表 4-11 参数说明

参数	描述
table_name	表名称。

参数	描述
property_name	● 命令中不指定property_name时，将返回所有属性及其值； ● 命令中指定property_name时，将返回该特定property_name所对应的值。

注意事项

property_name大小写敏感，不能同时指定多个property_name，否则会出错。

示例

返回test表中属性property_key1的值。

```
SHOW TBLPROPERTIES test ('property_key1');
```

4.4.4 查看指定表所有列

功能描述

查看指定表中的所有列。

语法格式

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

关键字

- COLUMNS：表中的列。
- FROM/IN：指定数据库，显示指定数据库下的表的列名。FROM和IN没有区别，可替换使用。

参数说明

表 4-12 参数说明

参数	描述
table_name	表名称。
db_name	数据库名称。

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。

示例

查看student表中的所有列。


```
SHOW COLUMNS IN student;
```

4.4.5 查看指定表所有分区

功能描述

查看指定表的所有分区。

语法格式

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

关键字

- PARTITIONS：表中的分区。
- PARTITION：分区。

参数说明

表 4-13 参数描述

参数	描述
db_name	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	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。
partition_specs	分区信息，key=value形式，key为分区字段，value为分区值。若分区字段为多个字段，可以不包含所有的字段，会显示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。

示例

- 查看student表下面的所有的分区。
  ```
  SHOW PARTITIONS student;
  ```
- 查看student表中dt='2010-10-10'的分区。
  ```
  SHOW PARTITIONS student PARTITION(dt='2010-10-10')。
  ```

4.4.6 查看表统计信息

功能描述

查看表统计信息。返回所有列的列名和列数据类型。

语法格式

```
DESCRIBE [EXTENDED|FORMATTED] [db_name.]table_name;
```

关键字

- EXTENDED：显示表的所有元数据，通常只在debug时用到。
- FORMATTED：使用表格形式显示所有表的元数据。

参数说明

表 4-14 参数描述

参数	描述
db_name	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	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。

注意事项

若所查看的表不存在，将会出错。

示例

查看student表的所有列的列名与列数据类型。

```
DESCRIBE student;
```

4.5 修改表

4.5.1 添加列

功能描述

添加一个或多个新列到表上。

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

表 4-15 参数描述

参数	描述
db_name	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	表名称。
col_name	列字段名称。
col_type	列字段类型。
col_comment	列描述。

注意事项

无。

示例

```
ALTER TABLE t1 ADD COLUMNS (column2 int, column3 string);
```

4.6 分区相关

4.6.1 添加分区（只支持 OBS 表）

功能描述

添加分区。

语法格式

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION partition_specs1
  [LOCATION 'obs_path1']
  PARTITION partition_specs2
  [LOCATION 'obs_path2'];
```

关键字

- IF NOT EXISTS：所需创建的分区已存在时使用，可避免系统报错。
- PARTITION：分区。
- LOCATION：分区路径。

参数说明

表 4-16 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 向表中添加分区时，此表必须已存在，而所要添加的分区不能存在，否则将出错。已存在的分区可通过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='Shanghai')**。
- 在添加分区时若指定OBS路径，则该OBS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 'obs_path'之间用空格隔开。例如：

  PARTITION partition_specs LOCATION 'obs_path' PARTITION partition_specs LOCATION 'obs_path'。
- 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。用户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

- 向student表中添加分区dt='2009-09-09',city='Shanghai'。

```
ALTER TABLE student ADD
  PARTITION (dt='2009-09-09',city='Shanghai')
  LOCATION 'obs://bucketName/fileName/students/dt=2009-09-09/city=Shanghai';
```

- 向student表中添加分区dt='2009-09-09',city='Shanghai'及分区dt='2008-08-08',city='Hangzhou'。

```
ALTER TABLE student ADD
  PARTITION(dt='2009-09-09',city='Shanghai')
  PARTITION(dt='2008-08-08',city='Hangzhou');
```

4.6.2 重命名分区

功能描述

重命名分区。

语法格式

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

关键字

- PARTITION：分区。
- RENAME：重命名。

参数说明

表 4-17 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。

注意事项

- 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='Shanghai')**。

示例

将student表中的分区city='Hangzhou',dt='2008-08-08'重命名为city='Wenzhou',dt='2009-09-09'。

```
ALTER TABLE student
  PARTITION (city='Hangzhou',dt='2008-08-08')
  RENAME TO PARTITION (city='Wenzhou',dt='2009-09-09');
```

4.6.3 删除分区

功能描述

删除分区表的一个或多个分区。

语法格式

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


关键字

- DROP：删除表分区。
- IF EXISTS：所要删除的分区必须是已经存在的，否则会出错。
- PARTITION：分区。

参数说明

表 4-18 参数描述

参数	描述
db_name	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	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。
partition_specs	分区信息，key=value形式，key为分区字段，value为分区值。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。“partition_specs”中的参数默认带有“( )”，例如：**PARTITION (dt='2009-09-09',city='Shanghai')**。

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。

示例

将分区表student的分区dt = '2008-08-08', city = 'Hangzhou'删除。

```
ALTER TABLE student
  DROP
  PARTITION (dt = '2008-08-08', city = 'Hangzhou');
```

4.6.4 修改表分区位置（只支持 OBS 表）

功能描述

修改表分区的位置。

语法格式

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

关键字

- PARTITION：分区。

- LOCATION：分区路径。

参数说明

表 4-19 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 所要修改位置的表分区必须是已经存在的，否则将报错。
- “partition_specs”中的参数默认带有“( )”，例如：**PARTITION (dt='2009-09-09',city='Shanghai')**。
- 所指定的新的OBS路径必须是已经存在的绝对路径，否则将报错。
- 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。用户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

将student表的分区dt='2008-08-08',city='Hangzhou'的OBS路径设置为“obs://bucketName/fileName/student/dt=2008-08-08/city=Hangzhou_bk”。

```
ALTER TABLE student
  PARTITION(dt='2008-08-08',city='Hangzhou')
  SET LOCATION 'obs://bucketName/fileName/student/dt=2008-08-08/city=Hangzhou_bk';
```

4.6.5 修改表分区 SerDe 属性（只支持 OBS 表）

功能描述

修改表分区Serde属性。

语法格式

```
ALTER TABLE table_name
 [PARTITION partition_specs]
 SET SERDE serde
 [WITH SERDEPROPERTIES (property_name=property_value,...)];
ALTER TABLE table_name
 [PARTITION partition_specs]
 SET SERDEPROPERTIES (property_name=property_value,...);
```

关键字

- PARTITION：分区。

- SERDEPROPERTIES：Serde属性。

参数说明

表 4-20 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 假如Serde属性已经存在，新的值将会覆盖老的值。
- 仅允许针对OBS表设置Serde属性。

示例

```
alter table test
 set serdeproperties (creator = "test");
```

4.6.6 更新表分区信息（只支持 OBS 表）

功能描述

更新表在元数据库中的分区信息。

语法格式

```
MSCK REPAIR TABLE table_name;
```

或

```
ALTER TABLE table_name RECOVER PARTITIONS;
```

关键字

- PARTITIONS：分区。
- SERDEPROPERTIES：Serde属性。

参数说明

表 4-21 参数描述

参数	描述
table_name	表名称。

参数	描述
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 该命令的主要应用场景是针对分区表，如当手动在OBS上面添加分区目录时，再通过上述命令将该新增的分区信息刷新到元数据库中，通过“SHOW PARTITIONS table_name”命令查看新增的分区。
- 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”。

示例

下述两语句都将更新表ptable在元数据库中的分区信息。

```
MSCK REPAIR TABLE ptable;
```

或

```
ALTER TABLE ptable RECOVER PARTITIONS;
```


5 数据相关

5.1 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。

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

表 5-1 参数描述

参数	描述
folder_path	原始数据文件夹或者文件的OBS路径。
db_name	数据库名称。若未指定，则使用当前数据库。
table_name	需要导入数据的DLI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tab，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'= "'"。
  - DLI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 5-2 MAXCOLUMNS

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5-3**。

  表 5-3 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
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3 日期及时间模式字符定义**。

- 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DLI表，“t”为表名。

```
LOAD DATA INPATH 'obs://dli/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DLI表，“jsontb”为表名。

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

5.2 插入数据

功能描述

将SELECT查询结果或某条数据插入到表中。

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 5-4 INSERT 参数

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DLI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，因为有一定概率发生并发冲突，导致插入失败。
- INSERT INTO命令用于将查询的结果追加到目标表中。
- INSERT OVERWRITE命令用于覆盖源表中已有的数据。
- 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：dli.sql.dynamicPartitionOverwrite.enabled=true，再通过“insert overwrite”语句实现，“dli.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 ...
```

  说明

  在“数据湖探索管理控制台>SQL编辑器”页面，单击编辑窗口右上角“设置”，可配置参数。

- 通过配置“spark.sql.shuffle.partitions”参数可以设置非DLI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

- 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 default.data_source_tab1 PARTITION (p1 = 3, p2 = 4)
 SELECT id FROM RANGE(3, 5);
```

- 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);
```

5.3 清空数据

功能描述

清除DLI表或者OBS表的数据。

语法格式

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


关键字

表 5-5 参数

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

注意事项

只支持清除DLI表或者OBS表的数据。

示例

```
truncate table test PARTITION (class = 'test');
```


6 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、CARBON、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 6-1 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、CARBON、JSON、Avro格式。

说明

file_format为csv时，options参数可以参考表4-3。

注意事项

- 通过配置“spark.sql.shuffle.partitions”参数可以设置非DLI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'。
  - 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'= "'"。
  - DLI表还支持\001（^A）和\017（^Q），例如：'DELIMITER'='\001(^A)'，'DELIMITER'='\017(^Q)'。

示例

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


7 跨源连接相关

7.1 跨源连接 HBase 表

7.1.1 创建 DLI 表关联 HBase

功能描述

使用CREATE TABLE命令创建DLI表并关联HBase上已有的表，该语法支持CloudTable服务HBase集群、MRS服务HBase实例以及自建HBase集群。

前提条件

- 创建DLI表关联HBase之前需要创建跨源连接。管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。
- 请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。
  如何添加IP域名映射，请参见《数据湖探索用户指南》中**增强型跨源连接**章节。

语法格式

- 单个RowKey

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME (
  ATTR1 TYPE,
  ATTR2 TYPE,
  ATTR3 TYPE)
  USING [CLOUDTABLE | HBASE] OPTIONS (
  'ZKHost'='xx',
  'TableName'='TABLE_IN_HBASE',
  'RowKey'='ATTR1',
  'Cols'='ATTR2:CF1.C1, ATTR3:CF1.C2');
```

- 组合RowKey

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME (
  ATTR1 String,
  ATTR2 String,
  ATTR3 TYPE)
  USING [CLOUDTABLE | HBASE] OPTIONS (
  'ZKHost'='xx',
  'TableName'='TABLE_IN_HBASE',
  'RowKey'='ATTR1:2, ATTR2:10',
```



```
'Cols'='ATTR2:CF1.C1, ATTR3:CF1.C2',
'krbauth'='KRB_AUTH_NAME');
```

关键字

表 7-1 CREATE TABLE 参数描述

参数	描述
USING [CLOUDTABLE | HBASE]	指定hbase datasource，"CLOUDTABLE"或"HBASE"二选一，大小写不敏感。
ZKHost	HBase集群的ZK连接地址。 获取ZK连接地址需要先创建跨源连接，管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。 ● 访问CloudTable集群，填写ZK连接地址（内网）。 ● 访问MRS集群，填写ZK所在节点IP与ZK对外端口，格式为："ZK_IP1:ZK_PORT1,ZK_IP2:ZK_PORT2"。 **说明** 访问MRS集群，只支持创建增强型跨源连接并且需要配置主机信息，管理控制台操作请参考**增强型跨源连接**，相关API信息请参考**创建增强型跨源连接**。
TableName	指定在HBase集群中已创建的表名。
RowKey	指定作为rowkey的dli关联表字段，支持单rowkey与组合rowkey。单rowkey支持数值与String类型，不需要指定长度。组合rowkey仅支持String类型定长数据，格式为：属性名1:长度,属性名2:长度。
Cols	通过逗号分隔的DLI表字段与HBase表的列之间的对应关系。其中，冒号前面放置DLI表字段，冒号后面放置HBase表信息，用‘.’分隔HBase表的列族与列名。
krbauth	创建跨源认证的认证名。如果创建的MRS集群未开启kerb认证的集群，请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。

注意事项

- 若所要创建的表已经存在将报错，可以通过添加IF NOT EXISTS参数跳过该错误。
- OPTIONS中的所有参数是必选的，参数名称大小写不敏感，但参数值大小写敏感。
- OPTIONS中引号内的值前后不能带空格，空格也会被当做有效值。
- 表名及列名的描述仅支持字符串常量。
- 创建表时要说明列名及对应的数据类型，目前支持的数据类型为：boolean、short、int、long、float、double和string。
- 作为RowKey的字段（如上述语法格式中的ATTR1），其值不能为null，长度要大于0，小于或等于32767。

- Cols与RowKey中的字段加起来的数量必须与DLI表的字段保持一致，即表中所有的字段都到对应到Cols和RowKey中，但是顺序可以任意。
- 组合Rowkey只支持String类型，在使用组合Rowkey时，每个属性后面必须带上长度。当Rowkey指定的字段只有一个的时候，该字段的类型可以是支持的所有数据类型，并且不需要填写长度。
- 在组合Rowkey的场景中
  - 插入Rowkey数据时，如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要短，则会在数据后面补'\0'字符；如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要长，则会在实际插入HBase的时候进行截断。
  - 读取HBase上的Rowkey数据时，如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要短，则会抛出异常（OutofBoundException）；如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要长，则会在读取时进行截断。

示例

```
CREATE TABLE test_hbase(
ATTR1 int,
ATTR2 int,
ATTR3 string)
using hbase OPTIONS (
'ZKHost'='to-hbase-1174405101-CE1bDm5B.datasource.com:2181',
'TableName'='HBASE_TABLE',
'RowKey'='ATTR1',
'Cols'='ATTR2:CF1.C1, ATTR3:CF1.C2');
```

7.1.2 插入数据至 HBase 表

功能描述

INSERT INTO命令将DLI表中的数据插入到已关联的hbase表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**基本语句**。

参数说明

表 7-2 参数描述

参数	描述
DLI_TABLE	创建的DLI表名称，为插入数据的目的表。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
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

- DLI表必须已经存在。
- 在“**创建表关联HBase**”章节创建的表中，OPTIONS里的Cols指定的列族如果不存在，insert into执行时会报错。
- 如果插入的(rowkey, 列族, 列)已存在，则执行插入操作时，会覆盖hbase中相同的(rowkey, 列族, 列)。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。
- 不支持INSERT OVERWRITE语法。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

7.1.3 查询 HBase 表

SELECT命令用于查询hbase表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表test_ct中的数据。

```
SELECT * FROM test_hbase limit 100;
```

查询下压

通过hbase进行数据过滤，即HBase Client将过滤条件传给HBase服务端进行处理，HBase服务端只返回用户需要的数据，提高了Spark SQL查询的速度。对于HBase不支持的过滤条件，例如组合Rowkey的查询，直接由Spark SQL进行。

- 支持查询下压的场景
  - 数据类型场景
    - Int
    - boolean
    - short
    - long
    - double
    - string

      说明

      float类型数据不支持查询下压。
  - 过滤条件场景
    - 过滤条件为>,<,>=,<=,=,!=,and,or

      例如：

      ```
      select * from tableName where (column1 >= value1 and column2<= value2) or column3 != value3
      ```
    - 过滤条件为like 和 not like，支持前缀，后缀和包含匹配

      例如：

      ```
      select * from tableName where column1 like "%value" or column2 like "value%" or column3 like "%value%"
      ```
    - 过滤条件为IsNotNull()

      例如：

      ```
      select * from tableName where IsNotNull(column)
      ```
    - 过滤条件为in ,not in

      例如：

      ```
      select * from tableName where column1 in (value1,value2,value3)  and column2 not in (value4,value5,value6)
      ```


- 过滤条件为between _ and _

  例如：

  ```
  select * from tableName where column1 between value1 and value2
  ```

- 组合rowkey中的子rowkey过滤

  例如，组合Rowkey为column1+column2+column3，进行子rowkey查询：

  ```
  select * from tableName where column1= value1
  ```

- 不支持查询下压的场景
  - 数据类型场景

    除上述支持的数据类型外，其余复杂数据类型不支持查询下压。
  - 过滤条件场景
    - length，count，max，min，join，groupby，orderby，limit和avg等
    - 过滤条件为列比较

      例如：

      ```
      select * from tableName where column1 > (column2+column3)
      ```

7.2 跨源连接 OpenTSDB 表

7.2.1 创建 DLI 表关联 OpenTSDB

功能描述

使用CREATE TABLE命令创建DLI表并关联OpenTSDB上已有的metric，该语法支持CloudTable服务的OpenTSDB和MRS服务的OpenTSDB。

前提条件

创建DLI表关联OpenTSDB之前需要创建跨源连接。管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。

语法格式

```
CREATE TABLE [IF NOT EXISTS] UQUERY_OPENTSDB_TABLE_NAME
  USING OPENTSDB OPTIONS (
  'host' = 'xx;xx',
  'metric' = 'METRIC_NAME',
  'tags' = 'TAG1,TAG2');
```


关键字

表 7-3 CREATE TABLE 参数描述

参数	描述
host	OpenTSDB连接地址。 获取OpenTSDB连接地址需要先创建跨源连接，管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。 • 访问CloudTable OpenTSDB，填写OpenTSDB链接地址。 • 访问MRS OpenTSDB，若使用增强型跨源连接，填写OpenTSDB所在节点IP与端口，格式为"IP:PORT"，OpenTSDB存在多个节点时，用分号间隔。若使用经典型跨源，填写经典型跨源返回的连接地址，管理控制台操作请参考**经典型跨源连接**。
metric	所创建的DLI表对应的OpenTSDB中的指标名称。
tags	metric对应的标签，用于归类、过滤、快速检索等操作。可以是1个到8个，以“,”分隔，包括对应metric下所有tagk的值。

注意事项

创建DLI表时，不需要指定timestamp和value字段，系统会根据指定的tags自动构建字段，包含以下字段，其中TAG1和TAG2由tags指定。

- TAG1 String
- TAG2 String
- timestamp Timestamp
- value double

示例

```
CREATE table opentsdb_table
  USING OPENTSDB OPTIONS (
  'host' = 'opentsdb-3xcl8dir15m58z3.cloudtable.com:4242',
  'metric' = 'city.temp',
  'tags' = 'city,location');
```

7.2.2 插入数据至 OpenTSDB 表

功能描述

使用INSERT INTO命令将DLI表中的数据插入到已关联的OpenTSDB metric中。

说明

若OpenTSDB上不存在metric，插入数据时会在OpenTSDB上自动创建一个新的metric。

语法格式

```
INSERT INTO TABLE TABLE_NAME SELECT * FROM DLI_TABLE;
INSERT INTO TABLE TABLE_NAME VALUES(XXX);
```


关键字

表 7-4 INSERT INTO 参数描述

参数	描述
TABLE_NAME	所关联的OpenTSDB表名。
DLI_TABLE	创建的DLI表名称。

注意事项

- 插入的数据不能为null；插入的数据相同，会覆盖原数据；插入的数据只有value值不同，也会覆盖原数据。
- 不支持INSERT OVERWRITE语法。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。
- 时间戳格式只支持yyyy-MM-dd hh:mm:ss。

示例

```
INSERT INTO TABLE opentsdb_table VALUES('shenzhen','futian','2018-05-03 00:00:00',21);
```

7.2.3 查询 OpenTSDB 表

SELECT命令用于查询OpenTSDB表中的数据。

说明

- 若OpenTSDB上不存在metric，查询对应的DLI表会报错。
- 若OpenTSDB开了安全模式，则访问时，需要设置conf:dli.sql.mrs.opentsdb.ssl.enabled=true

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表opentsdb_table中的数据。

```
SELECT * FROM opentsdb_table limit 100;
```


7.3 跨源连接 DWS 表

7.3.1 创建 DLI 表关联 DWS

功能描述

使用CREATE TABLE命令创建DLI表并关联DWS上已有的表。

前提条件

创建DLI表关联DWS之前需要创建跨源连接。管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME
  USING DWS OPTIONS (
  'url'='xx',
  'dbtable'='db_name_in_DWS.table_name_in_DWS',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

关键字

表 7-5 CREATE TABLE 参数描述

参数	描述
url	DWS的连接地址，需要先创建跨源连接，管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。 创建经典型跨源连接后，使用经典型跨源连接中返回的连接地址。 创建增强型跨源连接后，可以使用DWS提供的"JDBC连接字符串（内网）"，或者内网地址和内网端口访问，格式为"协议头://内网IP:内网端口/数据库名"，例如："jdbc:postgresql://192.168.0.77:8000/postgres"。 **说明** DWS的连接地址格式为："协议头://访问地址:访问端口/数据库名" 例如： jdbc:postgresql://to-dws-1174405119-ihlUr78j.datasource.com:8000/postgres 如果想要访问DWS中自定义数据库，请在这个连接里将"postgres"修改为对应的数据库名字。
dbtable	指定在DWS关联的表名，或者"模式名.表名"，例如：public.table_name。
user	（已废弃）DWS的用户名。
password	（已废弃）DWS集群的用户密码。

参数	描述
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>《**跨源认证**》。
encryption	使用跨源密码认证时配置为“true”。
partitionColumn	读取数据时，用于设置并发使用的数值型字段。 **说明** • “partitionColumn”、“lowerBound”、“upperBound”、“numPartitions”四个参数必须同时设置，不支持仅设置其中某一个或某几个。 • 为了提升并发读取的性能，建议使用自增列。
lowerBound	partitionColumn设置的字段数据最小值，该值包含在返回结果中。
upperBound	partitionColumn设置的字段数据最大值，该值不包含在返回结果中。
numPartitions	读取数据时并发数。 **说明** 实际读取数据时，会根据“lowerBound”与“upperBound”，平均分配给每个task，获取其中一部分的数据。例如： 'partitionColumn'='id', 'lowerBound'='0', 'upperBound'='100', 'numPartitions'='2' 表示在DLI中会起2个并发task，一个task执行id>=0 and id < 50，另一个task执行id >=50 and id < 100。
fetchsize	读取数据时，每一批次获取数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
batchsize	写入数据时，每一批次写入数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
truncate	执行overwrite时是否不删除原表，直接执行清空表操作，取值范围： • true • false 默认为“false”，即在执行overwrite操作时，先将原表删除再重新建表。
isolationLevel	事务隔离级别，取值范围： • NONE • READ_UNCOMMITTED • READ_COMMITTED • REPEATABLE_READ • SERIALIZABLE 默认值为“READ_UNCOMMITTED”。

注意事项

创建DWS关联表时，不需要指定关联表的Schema。DLI会自动获取DWS中对应参数"dbtable"中的表的Schema。

示例

```
CREATE TABLE IF NOT EXISTS dli_to_dws
  USING DWS OPTIONS (
  'url'='jdbc:postgresql://to-dws-1174405119-ih1Ur78j.datasource.com:8000/postgres',
  'dbtable'='test_dws',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

7.3.2 插入数据至 DWS 表

功能描述

INSERT INTO命令将DLI表中的数据插入到已关联的指定DWS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**基本语句**。

参数说明

表 7-6 参数描述

参数	描述
DLI_TABLE	创建的DLI表名称，为插入数据的目的表。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
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

- DLI表必须已经存在。
- DLI表在创建时不需要指定Schema信息，Schema信息将使用DWS表的信息。如果select子句中选择的字段数量和类型与DWS表的Schema信息不匹配时，系统将报错。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

7.3.3 查询 DWS 表

SELECT命令用于查询DWS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表dli_to_dws中的数据。

```
SELECT * FROM dli_to_dws limit 100;
```

7.4 跨源连接 RDS 表

7.4.1 创建 DLI 表关联 RDS

功能描述

使用CREATE TABLE命令创建DLI表并关联RDS上已有的表。该功能支持访问RDS的MySQL集群和PostGre集群。

前提条件

创建DLI表关联RDS之前需要创建跨源连接。管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME
  USING JDBC OPTIONS (
  'url'='xx',
  'driver'='DRIVER_NAME',
  'dbtable'='db_name_in_RDS.table_name_in_RDS',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

关键字

表 7-7 CREATE TABLE 参数描述

参数	描述
url	RDS的连接地址，需要先创建跨源连接，管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。 创建经典型跨源连接后，使用经典型跨源连接中返回的连接地址。 创建增强型跨源连接后，使用RDS提供的"内网域名"或者内网地址和数据库端口访问，MySQL格式为"协议头://内网IP:内网端口"，PostGre格式为"协议头://内网IP:内网端口/数据库名"。 例如："jdbc:mysql://192.168.0.193:3306"或者"jdbc:postgresql://192.168.0.193:3306/postgres"。 **说明** 经典型跨源的连接地址默认格式为："协议头://访问地址:访问端口" 例如：jdbc:mysql://to-rds-1174405119-oLRHAGE7.datasource.com:3306 如果想要访问RDS的postgre集群，需要将连接地址中的协议头修改为"jdbc:postgresql"，并在连接地址最后加上"/数据库名"。 例如：jdbc:postgresql://to-rds-1174405119-oLRHAGE7.datasource.com:3306/postgreDB
driver	jdbc驱动类名，访问MySQL集群请填写："com.mysql.jdbc.Driver"，访问PostGre集群请填写："org.postgresql.Driver"。
dbtable	• 访问MySQL集群填写"数据库名.表名" • 访问PostGre集群填写"模式名.表名" **说明** 模式名即为数据库模式（schema）的名称。数据库中schema是数据库对象集合，包含了表，视图等多种对象。
user	（已废弃）RDS用户名。
password	（已废弃）RDS用户名密码。
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>《**跨源认证**》。
encryption	使用跨源密码认证时配置为“true”。

参数	描述
partitionColumn	读取数据时，用于设置并发使用的数值型字段。 **说明** • “partitionColumn”、“lowerBound”、“upperBound”、“numPartitions”四个参数必须同时设置，不支持仅设置其中某一个或某几个。 • 为了提升并发读取的性能，建议使用自增列。
lowerBound	partitionColumn设置的字段数据最小值，该值包含在返回结果中。
upperBound	partitionColumn设置的字段数据最大值，该值不包含在返回结果中。

numPartitions	读取数据时并发数。
	说明
	实际读取数据时，会根据“lowerBound”与“upperBound”，平均分配给每个task，获取其中一部分的数据。例如：
	`'partitionColumn'='id',`
	`'lowerBound'='0',`
	`'upperBound'='100',`
	`'numPartitions'='2'`
	表示在DLI中会起2个并发task，一个task执行id>=0 and id < 50，另一个task执行id >=50 and id < 100。

fetchsize	读取数据时，每一批次获取数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
batchsize	写入数据时，每一批次写入数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
truncate	执行overwrite时是否不删除原表，直接执行清空表操作，取值范围： • true • false 默认为“false”，即在执行overwrite操作时，先将原表删除再重新建表。
isolationLevel	事务隔离级别，取值范围： • NONE • READ_UNCOMMITTED • READ_COMMITTED • REPEATABLE_READ • SERIALIZABLE 默认值为“READ_UNCOMMITTED”。

注意事项

创建RDS关联表时，不需要指定关联表的Schema。DLI会自动获取RDS中对应参数"dbtable"中的表的Schema。

示例

访问MySQL

```
CREATE TABLE IF NOT EXISTS dli_to_rds
  USING JDBC OPTIONS (
  'url'='jdbc:mysql://to-rds-117405104-3eAHxnlz.datasource.com:3306',
  'driver'='com.mysql.jdbc.Driver',
  'dbtable'='rds_test.test1',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

访问PostGre

```
CREATE TABLE IF NOT EXISTS dli_to_rds
  USING JDBC OPTIONS (
  'url'='jdbc:postgresql://to-rds-1174405119-oLRHAGE7.datasource.com:3306/postgreDB',
  'driver'='org.postgresql.Driver',
  'dbtable'='pg_schema.test1',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

7.4.2 插入数据至 RDS 表

功能描述

INSERT INTO命令将DLI表中的数据插入到已关联的指定RDS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**基本语句**。

参数说明

表 7-8 参数描述

参数	描述
DLI_TABLE	创建的DLI表名称，为插入数据的目的表。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。

参数	描述
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
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

- DLI表必须已经存在。
- DLI表在创建时不需要指定Schema信息，Schema信息将使用RDS表的信息。如果select子句中选择的字段数量和类型与RDS表的Schema信息不匹配时，系统将报错。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

7.4.3 查询 RDS 表

SELECT命令用于查询RDS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表test_ct中的数据。

```
SELECT * FROM dli_to_rds limit 100;
```


7.5 跨源连接 CSS 表

7.5.1 创建 DLI 表关联 CSS

功能描述

使用CREATE TABLE命令创建DLI表并关联CSS上已有的表。

前提条件

创建DLI表关联CSS之前需要创建跨源连接。管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
  FIELDNAME1 FIELDTYPE1,
  FIELDNAME2 FIELDTYPE2)
  USING CSS OPTIONS (
  'es.nodes'='xx',
  'resource'='type_path_in_CSS',
  'pushdown'='true',
  'strict'='false',
  'batch.size.entries'= '1000',
  'batch.size.bytes'= '1mb',
  'es.nodes.wan.only' = 'true',
  'es.mapping.id' = 'FIELDNAME');
```

关键字

表 7-9 CREATE TABLE 参数描述

参数	描述
es.nodes	CSS的连接地址，需要先创建跨源连接，管理控制台操作请参考**经典型跨源连接**和**增强型跨源连接**。 创建经典型跨源连接后，使用经典型跨源连接中返回的连接地址。 创建增强型跨源连接后，使用CSS提供的"内网访问地址"，格式为"IP1:PORT1,IP2:PORT2"。
resource	指定在CSS关联的资源名，用"/index/type"指定资源位置（可简单理解index为database，type为table，但绝不等同）。 **说明** • ES 6.X版本中，单个Index只支持唯一type，type名可以自定义。 • ES 7.X版本中，单个Index将使用“_doc”作为type名，不再支持自定义。若访问ES 7.X版本时，该参数只需要填写index即可。
pushdown	CSS的下压功能是否开启，默认为“true”。包含大量IO传输的表在有where过滤条件的情况下能够开启pushdown降低IO。
strict	CSS的下压是否是严格的，默认为“false”。精确匹配的场景下比pushdown降低更多IO。

参数	描述
batch.size.entries	单次batch插入entry的条数上限，默认为1000。如果单条数据非常大，在bulk存储设置的数据条数前提前到达了单次batch的总数据量上限，则停止存储数据，以batch.size.bytes为准，提交该批次的数据。
batch.size.bytes	单次batch的总数据量上限，默认为1mb。如果单条数据非常小，在bulk存储到总数据量前提前到达了单次batch的条数上限，则停止存储数据，以batch.size.entries为准，提交该批次的数据。
es.nodes.wan.only	是否仅通过域名访问es节点，默认为false。使用经典型跨源的连接地址作为es.nodes时，该参数需要配置为true；使用css服务提供的原始内网IP地址作为es.nodes时，不需要填写该参数或者配置为false。
es.mapping.id	指定一个字段，其值作为es中Document的id。 **说明** • 相同/index/type下的Document id是唯一的。如果作为Document id的字段存在重复值，则在执行插入es时，重复id的Document将会被覆盖。 • 该特性可以用作容错解决方案。当插入数据执行一半时，DLI作业失败，会有部分数据已经插入到es中，这部分为冗余数据。如果设置了Document id，则在重新执行DLI作业时，会覆盖上一次的冗余数据。
es.net.ssl	连接安全CSS集群，默认值为false
es.certificate.name	连接安全CSS集群，使用的跨源认证信息名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>《跨源认证》。

说明

batch.size.entries和batch.size.bytes分别对数据条数和数据量大小进行限制。

示例

```
CREATE TABLE IF NOT EXISTS dli_to_css (doc_id String, name string, age int)
  USING CSS OPTIONS (
  es.nodes 'to-css-1174404703-LzwpJEyx.datasource.com:9200',
  resource '/dli_index/dli_type',
  pushdown 'false',
  strict 'true',
  es.nodes.wan.only 'true',
  es.mapping.id 'doc_id');
```

7.5.2 插入数据至 CSS 表

功能描述

INSERT INTO命令将DLI表中的数据插入到已关联的指定CSS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
```

```
[WHERE where_condition]
[LIMIT num]
[GROUP BY field]
[ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
 VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**基本语句**。

参数说明

表 7-10 参数描述

参数	描述
DLI_TABLE	创建的DLI表名称，为插入数据的目的表。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
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

- DLI表必须已经存在。
- DLI表在创建时需要指定Schema信息，如果select子句或者values中字段数量与CSS表的Schema字段数量不匹配时，系统将报错。
- 类型不一致时不一定报错，例如插入int类型数据，但CSS中Schema保存的是文本类型，int类型会被转换成文本类型。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
 SELECT ATTR_EXPR
 FROM user
 WHERE user_name='cyz'
 LIMIT 3
 GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
 VALUES (1);
```


7.5.3 查询 CSS 表

SELECT命令用于查询CSS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表dli_to_css中的数据。

```
SELECT * FROM dli_to_css limit 100;
```

7.6 跨源连接 DCS 表

7.6.1 创建 DLI 表关联 DCS

功能描述

使用CREATE TABLE命令创建DLI表并关联DCS上已有的Key。

前提条件

创建DLI表关联DCS之前需要创建跨源连接，绑定队列。管理控制台操作请参考**增强型跨源连接**。

语法格式

- 指定Key

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
  FIELDNAME1 FIELDTYPE1,
  FIELDNAME2 FIELDTYPE2)
 USING REDIS OPTIONS (
 'host'='xx',
 'port'='xx',
 'passwdauth' = 'xxx',
 'encryption' = 'true',
 'table'='namespace_in_redis:key_in_redis',
 'key.column'= 'FIELDNAME1'
);
```

- 通配key

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
  FIELDNAME1 FIELDTYPE1,
  FIELDNAME2 FIELDTYPE2)
 USING REDIS OPTIONS (
 'host'='xx',
```

```
  'port'='xx',
  'passwdauth' = 'xxx',
  'encryption' = 'true',
  'keys.pattern'='key*:*',
  'key.column'= 'FIELDNAME1'
);
```

关键字

表 7-11 CREATE TABLE 参数描述

参数	描述
host	DCS的连接IP，需要先创建跨源连接，管理控制台操作请参考**增强型跨源连接**。 创建增强型跨源连接后，使用DCS提供的"连接地址"。"连接地址"有多个时，选择其中一个即可。 **说明** 访问DCS目前只支持增强型跨源。
port	DCS的连接端口，例如6379。
password	（已废弃）创建DCS集群时填写的密码。访问非安全Redis集群时不需要填写。
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>《**跨源认证**》。
encryption	使用跨源密码认证时配置为“true”。
table	对应Redis中的Key或Hash Key。 ● 插入redis数据时必填。 ● 查询redis数据时与“keys.pattern”参数二选一。
keys.pattern	使用正则表达式匹配多个Key或Hash Key。该参数仅用于查询时使用。查询redis数据时与“table”参数二选一。
key.column	非必填。用于指定schema中的某个字段作为Redis中key的标识。在插入数据时与参数“table”配合使用。
partitions.number	读取数据时，并发task数。
scan.count	每批次读取的数据记录数，默认为100。如果在读取过程中，redis集群中的CPU使用率还有提升空间，可以调大该参数。
iterator.grouping.size	每批次插入的数据记录数，默认为100。如果在插入过程中，redis集群中的CPU使用率还有提升空间，可以调大该参数。
timeout	连接redis的超时时间，单位ms，默认值2000（2秒超时）。

说明

访问DCS时，不支持复杂类型数据（Array、Struct、Map等）。

可以考虑以下几种方式进行复杂类型数据处理：

- 字段扁平化处理，将下一级的字段展开放在同一层Schema字段中。
- 使用二进制方式进行写入与读取，并通过自定义函数进行编解码。

示例

- 指定table

```
create table test_redis(name string, age int) using redis options(
  'host' = '192.168.4.199',
  'port' = '6379',
  'passwdauth' = 'xxx',
  'encryption' = 'true',
  'table' = 'person'
);
```

- 通配table名

```
create table test_redis_keys_patten(id string, name string, age int) using redis options(
  'host' = '192.168.4.199',
  'port' = '6379',
  'passwdauth' = 'xxx',
  'encryption' = 'true',
  'keys.pattern' = 'p*:*',
  'key.column' = 'id'
);
```

7.6.2 插入数据至 DCS 表

功能描述

INSERT INTO命令将DLI表中的数据插入到已关联的DCS Key中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**基本语句**。

参数说明

表 7-12 参数描述

参数	描述
DLI_TABLE	创建的DLI表名称，为插入数据的目的表。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
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

- DLI表必须已经存在。
- DLI表在创建时需要指定Schema信息。
- 如果在建表时指定“key.column”，则在Redis中会以指定字段的值作为Redis Key名称的一部分。例如：

```
create table test_redis(name string, age int) using redis options(
  'host' = '192.168.4.199',
  'port' = '6379',
  'password' = '******',
  'table' = 'test_with_key_column',
  'key.column' = 'name'
);
insert into test_redis values("James", 35), ("Michael", 22);
```

在redis中将会有2个名为test_with_key_column:James和test_with_key_column:Michael的表：

```
192.168.7.238:6379> keys test_with_key_column:*
1) "test_with_key_column:Michael"
2) "test_with_key_column:James"
192.168.7.238:6379>
```

```
192.168.7.238:6379> hgetall "test_with_key_column:Michael"
1) "age"
2) "22"
192.168.7.238:6379> hgetall "test_with_key_column:James"
1) "age"
2) "35"
192.168.7.238:6379>
```

- 如果在建表时没有指定“key.column”，则在Redis中的key name将会使用uuid。例如：

```
create table test_redis(name string, age int) using redis options(
  'host' = '192.168.7.238',
  'port' = '6379',
  'password' = '******',
  'table' = 'test_without_key_column'
);
insert into test_redis values("James", 35), ("Michael", 22);
```


在redis中将会有2个以“test_without_key_column:uuid”命名的表：

```
192.168.7.238:6379> keys test_without_key_column:*
1) "test_without_key_column:b0ce581fa0d548e5b2273f4db1df6dcd"
2) "test_without_key_column:1e80aa7175d747ee9a82cce241767b01"
192.168.7.238:6379>
```

```
192.168.7.238:6379> hgetall "test_without_key_column:b0ce581fa0d548e5b2273f4db1df6dcd"
1) "age"
2) "35"
3) "name"
4) "James"
192.168.7.238:6379> hgetall "test_without_key_column:1e80aa7175d747ee9a82cce241767b01"
1) "age"
2) "22"
3) "name"
4) "Michael"
192.168.7.238:6379>
```

示例

```
INSERT INTO test_redis
  VALUES("James", 35), ("Michael", 22);
```

7.6.3 查询 DCS 表

SELECT命令用于查询DCS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

示例

查询表test_redis中的数据。

```
SELECT * FROM test_redis limit 100;
```

7.7 跨源连接 DDS 表

7.7.1 创建 DLI 表关联 DDS

功能描述

使用CREATE TABLE命令创建DLI表并关联DDS上已有的collection。

前提条件

创建DLI表关联OpenTSDB之前需要创建跨源连接，绑定队列。管理控制台操作请参考**增强型跨源连接**。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
   FIELDNAME1 FIELDTYPE1,
```



```
  FIELDNAME2 FIELDTYPE2)
 USING MONGO OPTIONS (
 'url'='IP:PORT[,IP:PORT]/[DATABASE][.COLLECTION][AUTH_PROPERTIES]',
 'database'='xx',
 'collection'='xx',
 'passwdauth' = 'xxx',
 'encryption' = 'true'
);
```

关键字

表 7-13 CREATE TABLE 参数描述

参数	描述
url	DDS的连接信息，需要先创建跨源连接，管理控制台操作请参考**增强型跨源连接**。 创建增强型跨源连接后，使用DDS提供的"随机连接地址"，格式为： "IP:PORT[,IP:PORT]/[DATABASE][.COLLECTION][AUTH_PROPERTIES]" 例如："192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin"
database	DDS的数据库名，如果在"url"中同时指定了数据库名，则"url"中的数据库名不生效。
collection	DDS中的collection名，如果在"url"中同时指定了collection，则"url"中的collection不生效。
user	（已废弃）访问DDS集群用户名。
password	（已废弃）访问DDS集群密码
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>《**跨源认证**》。
encryption	使用跨源密码认证时配置为“true”。

说明

如果在DDS中已存在collection，则建表可以不指定schema信息，DLI会根据collection中的数据自动生成schema信息。

示例

```
create table 1_datasource_mongo.test_mongo(id string, name string, age int) using mongo options(
 'url' = '192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin',
 'database' = 'test',
 'collection' = 'test',
 'passwdauth' = 'xxx',
 'encryption' = 'true');
```

7.7.2 插入数据至 DDS 表

功能描述

INSERT INTO命令将DLI表中的数据插入到已关联的指定DDS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

- 覆盖插入数据

```
INSERT OVERWRITE TABLE DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

关键字

SELECT对应关键字说明请参考**基本语句**。

参数说明

表 7-14 参数描述

参数	描述
DLI_TABLE	创建的DLI表名称，为插入数据的目的表。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
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

DLI表必须已经存在。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

7.7.3 查询 DDS 表

SELECT命令用于查询DDS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

如果在建表时没有指定schema信息，则查询出来的结果将会包含"_id"字段用于存放doc中的"_id"，例如：

示例

查询表test_mongo中的数据。

```
SELECT * FROM test_mongo limit 100;
```


8 视图相关

8.1 创建视图

功能描述

创建视图。

语法格式

```
CREATE [OR REPLACE] VIEW view_name 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。

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。

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;
```

8.2 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;
```


9 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;
```


10 数据权限相关

10.1 数据权限列表

DLI中SQL语句与数据库、表、角色相关的权限矩阵如表10-1所示。

表 10-1 权限矩阵

分类	SQL语句	权限	说明
Database	DROP DATABASE db1	database.db1的DROP_DATABASE权限	-
	CREATE TABLE tb1(...)	database.db1的CREATE_TABLE权限	-
	CREATE VIEW v1	database.db1的CREATE_VIEW权限	-
	EXPLAIN query	database.db1的EXPLAIN权限	query需要其相应的权限。
Table	SHOW CREATE TABLE tb1	database.db1.tables.tb1的SHOW_CREATE_TABLE权限	-
	DESCRIBE [EXTENDED| FORMATTED] tb1	databases.db1.tables.tb1的DESCRIBE_TABLE权限	-
	DROP TABLE [IF EXISTS] tb1	database.db1.tables.tb1的DROP_TABLE权限	-
	SELECT * FROM tb1	database.db1.tables.tb1的SELECT权限	-
	SELECT count(*) FROM tb1	database.db1.tables.tb1的SELECT权限	-

分类	SQL语句	权限	说明
	SELECT * FROM view1	database.db1.tables.view1的SELECT权限	-
	SELECT count(*) FROM view1	database.db1.tables.view1的SELECT权限	-
	LOAD DLI TABLE	database.db1.tables.tb1的INSERT_INTO_TABLE权限	-
	INSERT INTO TABLE	database.db1.tables.tb1的INSERT_INTO_TABLE权限	-
	INSERT OVERWRITE TABLE	database.db1.tables.tb1的INSERT_OVERWRITE_TABLE权限	-
	ALTER TABLE ADD COLUMNS	database.db1.tables.tb1的ALTER_TABLE_ADD_COLUMNS权限	-
	ALTER TABLE RENAME	database.db1.tables.tb1的ALTER_TABLE_RENAME权限	-
ROLE&PRIVILEGE	CREATE ROLE	db的CREATE_ROLE权限	-
	DROP ROLE	db的DROP_ROLE权限	-
	SHOW ROLES	db的SHOW_ROLES权限	-
	GRANT ROLES	db的GRANT_ROLE权限	-
	REVOKE ROLES	db的REVOKE_ROLE权限	-
	GRANT PRIVILEGE	db或table的GRANT_PRIVILEGE权限	-
	REVOKE PRIVILEGE	db或table的REVOKE_PRIVILEGE权限	-
	SHOW GRANT	db或table的SHOW_GRANT权限	-

Privilege在进行数据库和表赋权或回收权限时，DLI支持的权限类型如下所示。

- DATABASE上可赋权/回收的权限：
  - DROP_DATABASE（删除数据库）
  - CREATE_TABLE（创建表）
  - CREATE_VIEW（创建视图）
  - EXPLAIN（将SQL语句解释为执行计划）
  - CREATE_ROLE（创建角色）
  - DROP_ROLE（删除角色）

  - SHOW_ROLES（显示角色）
  - GRANT_ROLE（绑定角色）
  - REVOKE_ROLE（解除角色绑定）
  - DESCRIBE_TABLE（描述表）
  - DROP_TABLE（删除表）
  - SELECT（查询表）
  - INSERT_INTO_TABLE（插入）
  - INSERT_OVERWRITE_TABLE（重写）
  - GRANT_PRIVILEGE（数据库的赋权）
  - REVOKE_PRIVILEGE（数据库权限的回收）
  - SHOW_PRIVILEGES（查看其他用户具备的数据库权限）
  - ALTER_TABLE_ADD_PARTITION（在分区表中添加分区）
  - ALTER_TABLE_DROP_PARTITION（删除分区表的分区）
  - ALTER_TABLE_RENAME_PARTITION（重命名表分区）
  - ALTER_TABLE_RECOVER_PARTITION（恢复表分区）
  - ALTER_TABLE_SET_LOCATION（设置分区的路径）
  - SHOW_PARTITIONS（显示所有分区）
  - SHOW_CREATE_TABLE（查看建表语句）
- TABLE上可以赋权/回收的权限：
  - DESCRIBE_TABLE（描述表）
  - DROP_TABLE（删除表）
  - SELECT（查询表）
  - INSERT_INTO_TABLE（插入）
  - INSERT_OVERWRITE_TABLE（重写）
  - GRANT_PRIVILEGE（表的赋权）
  - REVOKE_PRIVILEGE（表权限的回收）
  - SHOW_PRIVILEGES（查看其他用户具备的表权限）
  - ALTER_TABLE_ADD_COLUMNS（增加列）
  - ALTER_TABLE_RENAME（重命名表）
  - ALTER_TABLE_ADD_PARTITION（在分区表中添加分区）
  - ALTER_TABLE_DROP_PARTITION（删除分区表的分区）
  - ALTER_TABLE_RENAME_PARTITION（重命名表分区）
  - ALTER_TABLE_RECOVER_PARTITION（恢复表分区）
  - ALTER_TABLE_SET_LOCATION（设置分区的路径）
  - SHOW_PARTITIONS（显示所有分区）
  - SHOW_CREATE_TABLE（查看建表语句）

10.2 创建角色

功能描述

- 在当前database或指定database中创建一个新的角色。
- 只有在database上具有CREATE_ROLE权限的用户才能创建角色。例如：管理员用户、database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database。

语法格式

```
CREATE ROLE [db_name].role_name;
```

关键字

无。

注意事项

- 要创建的role_name必须在当前database或指定database中不存在，否则会报错。
- 当未指定“db_name”时，表示在当前database中创建角色。

示例

```
CREATE ROLE role1;
```

10.3 删除角色

功能描述

在当前database或指定database中删除角色。

语法格式

```
DROP ROLE [db_name].role_name;
```

关键字

无。

注意事项

- 要删除的role_name必须在当前database或指定database中存在，否则会报错。
- 当未指定“db_name”时，表示在当前database中删除角色。

示例

```
DROP ROLE role1;
```


10.4 绑定角色

功能描述

绑定用户和角色。

语法格式

```
GRANT ([db_name].role_name,...) TO (user_name,...);
```

关键字

无。

注意事项

role_name和username必须存在，否则会报错。

示例

```
GRANT role1 TO user_name1;
```

10.5 解绑角色

功能描述

取消用户和角色的绑定。

语法格式

```
REVOKE ([db_name].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;
```


10.6 显示角色

功能描述

显示所有的角色或者显示当前database下绑定到“user_name”的角色。

语法格式

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

关键字

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

- 显示绑定到该用户的所有角色。
  ```
  SHOW ROLES;
  ```
- 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;
  ```

10.7 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 [db_name].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**数据权限列表**。

- resource可以是queue、database、table、view、column，格式分别为：
  - queue的格式为：queues.queue_name
  - 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

示例

给用户user_name1授予数据库db1的删除数据库权限。

```
GRANT DROP_DATABASE 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;
```

10.8 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 [db_name].role_name) | (USER user_name)),...);
```

关键字

ROLE：限定后面的rol e_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**数据权限列表**。
- resource可以是queue、database、table、view、column，格式分别为：
  - queue的格式为：queues.queue_name
  - 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

示例

回收用户user_name1对于数据库db1的删除数据库权限。

```
REVOKE DROP_DATABASE 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;
```

10.9 显示已授权限

功能描述

显示某个用户或角色在resource上已经授予的权限。

语法格式

```
SHOW GRANT ((ROLE [db_name].role_name) | (USER user_name)) ON resource;
```

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

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

- queue的格式为：queues.queue_name
- 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

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;
```

显示角色role_name在数据库db1的表tb1上的权限。

```
SHOW GRANT ROLE role_name ON databases.db1.tables.tb1;
```


10.10 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

变量ROLE必须存在。

示例

```
SHOW PRINCIPALS role1;
```

11 数据类型

11.1 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。

DLI当前只支持原生数据类型。

11.2 原生数据类型

DLI支持原生数据类型，请参见表11-1。

表 11-1 原生数据类型

数据类型	描述	存储空间	范围	OBS表支持情况	DLI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	DLI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
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

说明

- VARCHAR和CHAR在DLI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DLI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，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，$0 \le s \le p$。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隐性转换的数据类型有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。DLI中实际存储为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。DLI中实际存储为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11-2所示。

表 11-2 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，返回空。
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。

11.3 复杂数据类型

Spark SQL支持复杂数据类型，如表11-3所示。

表 11-3 复杂数据类型

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

使用限制

- CSV格式数据不支持复杂数据类型。
- 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。然后将已存在的文本“array_test.txt”导入“array_test”中。操作如下：

1. 创建表。

   CREATE TABLE array_test(name STRING, id ARRAY<INT>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   COLLECTION ITEMS TERMINATED BY ':';

2. 导入数据。

   “array_test.txt”文件路径为“/opt/array_test.txt”，文件内容如下所示：

   ```
   100,1:2:3:4
   101,5:6
   102,7:8:9:10
   ```

   执行如下命令导入数据。

   LOAD DATA LOCAL INPATH '/opt/array_test.txt' INTO TABLE array_test;

3. 查询结果。

   查“array_test”表中的所有数据：

   SELECT * FROM array_test;

   ```
   100 [1,2,3,4]
   101 [5,6]
   102 [7,8,9,10]
   ```

   查“array_test”表中id数组第0个元素的数据。

SELECT id[0] FROM array_test;

```
1
5
7
```

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），然后将已存在的文本“map_test.txt”导入至“map_test”中。操作如下：

1. 创建表。

   CREATE TABLE map_test(id STRING, score map<STRING,INT>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '|'
   COLLECTION ITEMS TERMINATED BY ','
   MAP KEYS TERMINATED BY ':';

2. 导入数据。

   “map_test.txt”文件路径为“/opt/map_test.txt”，文件内容如下所示：

   ```
   1|Math:90,English:89,Physics:86
   2|Math:88,English:90,Physics:92
   3|Math:93,English:93,Physics:83
   ```

   执行如下命令导入数据：

   LOAD DATA LOCAL INPATH '/opt/map_test.txt' INTO TABLE map_test;

3. 查询结果。

   查询“map_test”表里的所有数据。

   SELECT * FROM map_test;

   ```
   1 {"English":89,"Math":90,"Physics":86}
   2 {"English":90,"Math":88,"Physics":92}
   3 {"English":93,"Math":93,"Physics":83}
   ```

   查询“map_test”表中的数学成绩。

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

   ```
   190
   288
   393
   ```

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）。然后将已存在的文本“struct_test.txt”导入至“struct_test”表中。操作如下：

1. 创建表。

   CREATE TABLE struct_test(id INT, info STRUCT<name:STRING,age:INT>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   COLLECTION ITEMS TERMINATED BY ':';

2. 导入数据。

   “struct_test.txt”文件路径为“/opt/struct_test.txt”，文件内容如下所示：

```
1,Lily:26
2,Sam:28
3,Mike:31
4,Jack:29
```

执行如下命令导入数据：

LOAD DATA LOCAL INPATH '/opt/struct_test.txt' INTO TABLE struct_test;

3. 查询结果。

查询“struct_test”表中的所有数据。

SELECT * FROM struct_test;

```
1 {"name":"Lily","age":26}
2 {"name":"Sam","age":28}
3 {"name":"Mike","age":31}
4 {"name":"Jack","age":29}
```

查询“struct_test”表中姓名和年龄。

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

```
1 Lily 26
2 Sam 28
3 Mike 31
4 Jack 29
```


12 自定义函数

12.1 创建函数

功能描述

创建函数。

语法格式

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

resource:
  : (JAR|FILE|ARCHIVE)file_uri
```

关键字

- TEMPORARY：所创建的函数仅在当前会话中可用，并且不会持久化到底层元数据库中（如果有的话）。不能为临时函数指定数据库名称。
- USING <resources>：需要加载的资源。可以是JAR、文件或者URI的列表。

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。

示例

创建函数mergeBill。

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


12.2 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：所删除的函数是否为临时函数。
- 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;
```

12.3 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;
```


12.4 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);
```

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;
```


13 内置函数

13.1 数学函数

DLI所支持的数学函数如表13-1所示。

表 13-1 数学函数

函数	返回值	描述
round(DOUBLE a)	DOUBLE	四舍五入。
round(DOUBLE a, INT d)	DOUBLE	小数部分d位之后数字四舍五入，例如round(21.263,2)，返回21.26。
bround(DOUBLE a)	DOUBLE	HALF_EVEN模式四舍五入，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，增加一位，前一位数字为偶数，舍弃一位。例如：bround(7.5)=8.0，bround(6.5)=6.0
bround(DOUBLE a, INT d)	DOUBLE	保留小数点后d位，d位之后数字以HALF_EVEN模式四舍五入。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，增加一位，前一位数字为偶数，舍弃一位。例如：bround(8.25, 1) = 8.2, bround(8.35, 1) = 8.4。
floor(DOUBLE a)	BIGINT	对给定数据进行向下舍入最接近的整数。例如：floor(21.2)，返回21。
ceil(DOUBLE a), ceiling(DOUBLE a)	BIGINT	将参数向上舍入为最接近的整数。例如：ceil(21.2)，返回22。
rand(), rand(INT seed)	DOUBLE	返回大于或等于0且小于1的平均分布随机数。如果指定种子seed，则会得到一个稳定的随机数序列。

函数	返回值	描述
exp(DOUBLE a), exp(DECIMAL a)	DOUBLE	返回e的a次方。
ln(DOUBLE a), ln(DECIMAL a)	DOUBLE	返回给定数值的自然对数。
log10(DOUBLE a), log10(DECIMAL a)	DOUBLE	返回给定数值的以10为底自然对数。
log2(DOUBLE a), log2(DECIMAL a)	DOUBLE	返回给定数值的以2为底自然对数。
log(DOUBLE base, DOUBLE a) log(DECIMAL base, DECIMAL a)	DOUBLE	返回给定底数及指数返回自然对数。
pow(DOUBLE a, DOUBLE p), power(DOUBLE a, DOUBLE p)	DOUBLE	返回a的p次幂。
sqrt(DOUBLE a), sqrt(DECIMAL a)	DOUBLE	返回数值的平方根。
bin(BIGINT a)	STRING	返回二进制格式。
hex(BIGINT a) hex(STRING a)	STRING	将整数或字符转换为十六进制格式。
conv(BIGINT num, INT from_base, INT to_base), conv(STRING num, INT from_base, INT to_base)	STRING	进制转换，将from_base进制下的num转化为to_base进制下面的数。例如：将5从十进制转换为四进制，conv(5,10,4)=11。
abs(DOUBLE a)	DOUBLE	取绝对值。
pmod(INT a, INT b), pmod(DOUBLE a, DOUBLE b)	INT or DOUBLE	返回a除b的余数的绝对值。
sin(DOUBLE a), sin(DECIMAL a)	DOUBLE	返回给定角度a的正弦值。
asin(DOUBLE a), asin(DECIMAL a)	DOUBLE	返回给定角度a的反正弦值。
cos(DOUBLE a), cos(DECIMAL a)	DOUBLE	返回给定角度a的余弦值。
acos(DOUBLE a), acos(DECIMAL a)	DOUBLE	返回给定角度a的反余弦值。

函数	返回值	描述
tan(DOUBLE a), tan(DECIMAL a)	DOUBLE	返回给定角度a的正切值。
atan(DOUBLE a), atan(DECIMAL a)	DOUBLE	返回给定角度a的反正切值。
degrees(DOUBLE a), degrees(DECIMAL a)	DOUBLE	返回弧度所对应的角度。
radians(DOUBLE a), radians(DECIMAL a)	DOUBLE	返回角度所对应的弧度。
positive(INT a), positive(DOUBLE a)	INT or DOUBLE	返回a的值，例如positive(2)，返回2。
negative(INT a), negative(DOUBLE a)	INT or DOUBLE	返回a的相反数，例如negative(2)，返回-2。
sign(DOUBLE a), sign(DECIMAL a)	DOUBLE or INT	返回a所对应的正负号，a为正返回1.0，a为负，返回-1.0，否则则返回0.0。
e()	DOUBLE	返回e的值。
pi()	DOUBLE	返回pi的值。
factorial(INT a)	BIGINT	返回a的阶乘。
cbrt(DOUBLE a)	DOUBLE	返回a的立方根。
shiftleft(TINYINT|SMALLINT|INT a, INT b) shiftleft(BIGINT a, INT b)	INT BIGINT	有符号左边移，将a的二进制数按位左移b位。
shiftright(TINYINT|SMALLINT|INT a, INT b) shiftright(BIGINT a, INT b)	INT BIGINT	有符号右移，将a的二进制数按位右移b位。

函数	返回值	描述
shiftrightunsigned(TINYINT|SMALLINT|INT a, INT b), shiftrightunsigned(BIGINT a, INT b)	INT BIGINT	无符号右移，将a的二进制数按位右移b位。
greatest(T v1, T v2, ...)	T	返回列表中的最大值。
least(T v1, T v2, ...)	T	返回列表中的最小值。

13.2 日期函数

DLI所支持的日期函数如表13-2所示。

表 13-2 日期/时间函数

函数	返回值	描述
from_unixtime(bigint unixtime[, string format])	STRING	将时间戳转换为时间格式，格式为“yyyy-MM-dd HH:mm:ss”或“yyyyMMddHHmmss.uuuuuu”。 例如：select FROM_UNIXTIME(1608135036,'yyyy-MM-dd HH:mm:ss')
unix_timestamp()	BIGINT	如果不带参数的调用，返回一个Unix时间戳（从“1970-01-01 00:00:00”到现在的秒数）为无符号整数。
unix_timestamp(string date)	BIGINT	指定日期参数调用UNIX_TIMESTAMP()，它返回“1970-01-01 00:00:00”到指定日期的秒数。
unix_timestamp(string date, string pattern)	BIGINT	转换pattern格式的日期到UNIX时间戳：unix_timestamp("2009-03-20", "yyyy-MM-dd") = 1237532400
to_date(string timestamp)	STRING	返回时间中的年月日，例如：to_date("1970-01-01 00:00:00") = "1970-01-01"
year(string date)	INT	返回指定日期中的年份。
quarter(string date/timestamp/string)	INT	返回该date/timestamp/string所在的季度，如quarter('2015-04-01')=2。

函数	返回值	描述
month(string date)	INT	返回指定时间的月份，范围为1至12月。
day(string date) dayofmonth(string date)	INT	返回指定时间的日期。
hour(string date)	INT	返回指定时间的小时，范围为0到23。
minute(string date)	INT	返回指定时间的分钟，范围为0到59。
second(string date)	INT	返回指定时间的秒，范围为0到59。
weekofyear(string date)	INT	返回指定日期是一年中的第几周，范围为0到53。
datediff(string enddate, string startdate)	INT	两个时间参数的天数之差。
date_add(string startdate, int days)	STRING	给定时间，在此基础上加上指定的时间段。
date_sub(string startdate, int days)	STRING	给定时间，在此基础上减去指定的时间段。
from_utc_timestamp(string timestamp, string timezone)	TIMESTAMP	将UTC的时间戳转化为timezone所对应的时间戳，如from_utc_timestamp('1970-01-01 08:00:00','PST') returns 1970-01-01 00:00:00。
to_utc_timestamp(string timestamp, string timezone)	TIMESTAMP	将timezone所对应的时间戳转换为UTC的时间戳，如to_utc_timestamp('1970-01-01 00:00:00','PST') returns 1970-01-01 08:00:00。
current_date()	DATE	返回当前日期，如2016-07-04。
current_timestamp()	TIMESTAMP	返回当前时间，如2016-07-04 11:18:11.685。
add_months(string start_date, int num_months)	STRING	返回start_date在num_months个月之后的date。
last_day(string date)	STRING	返回date所在月份的最后一天，格式为yyyy-MM-dd，如2015-08-31。
next_day(string start_date, string day_of_week)	STRING	返回start_date之后最接近day_of_week的日期，格式为yyyy-MM-dd，day_of_week表示一周内的星期（如Monday、FRIDAY）。
trunc(string date, string format)	STRING	将date按照特定的格式进行清零操作，支持格式为MONTH/MON/MM, YEAR/YYYY/YY，如trunc('2015-03-17', 'MM') = 2015-03-01。
months_between(string date1, string date2)	DOUBLE	返回date1与date2之间的月份差。

函数	返回值	描述
date_format(date/timestamp/string ts, string fmt)	STRING	返回date/timestamp/string的格式化输出，格式支持JAVA的SimpleDateFormat格式，如date_format('2015-04-08', 'y') = '2015'。 其中，y表示Year, 如果是Y，则表示Week year。Week year表示当天所在的周属于的年份，一周从周日开始，周六结束，如果本周跨年，那么这周就算入下一年。

13.3 字符串函数

DLI所支持的字符串函数如表13-3所示。

表 13-3 字符串函数

函数	返回值	描述
ascii(string str)	INT	返回字符串中首字符的数字值。
concat(string A, string B...)	STRING	连接多个字符串，合并为一个字符串，可以接受任意数量的输入字符串。
concat_ws(string SEP, string A, string B...)	STRING	连接多个字符串，字符串之间以指定的分隔符分隔。
encode(string src, string charset)	BINARY	用charset的编码方式对src进行编码。
find_in_set(string str, string strList)	INT	返回字符串str第一次在strlist出现的位置。如果任一参数为NULL，返回NULL。如果第一个参数包含逗号，返回0。
get_json_object(string json_string, string path)	STRING	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。
instr(string str, string substr)	INT	返回substr在str中最早出现的下标。当参数中出现NULL时，返回NULL，但str中不存在substr时返回0，注意下标从1开始。
length(string A)	INT	返回字符串的长度。
locate(string substr, string str[, int pos])	INT	返回在下标pos（从1开始）之后，substr在str中出现的最小下标。
lower(string A) lcase(string A)	STRING	将文本字符串转换成字母全部小写的形式。

函数	返回值	描述
lpad(string str, int len, string pad)	STRING	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。
ltrim(string A)	STRING	删除字符串左边的空格，其他的空格保留。
parse_url(string urlString, string partToExtract [, string keyToExtract])	STRING	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。 例如：parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') 返回 'facebook.com'.。 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，例如：parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') 返回'v1'。
printf(String format, Obj... args)	STRING	将输入按特定格式打印输出。
regexp_extract(string subject, string pattern, int index)	STRING	通过下标返回正则表达式指定的部分。regexp_extract('foothebar', 'foo(.*?)(bar)', 2) 返回：'bar.'
regexp_replace(string A, string B, string C)	STRING	字符串A中的B字符被C字符替代。
repeat(string str, int n)	STRING	重复N次字符串。
reverse(string A)	STRING	返回倒序字符串。
rpad(string str, int len, string pad)	STRING	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从右侧填补。
rtrim(string A)	STRING	删除字符串右边的空格，其他的空格保留。
space(int n)	STRING	返回指定数量的空格。
substr(string A, int start) substring(string A, int start)	STRING	从文本字符串A中截取指定的起始位置后的字符。

函数	返回值	描述
substr(string A, int start, int len) substring(string A, int start, int len)	STRING	从文本字符串A中截取指定的起始位置后指定长度的字符。
substring_index(string A, string delim, int count)	STRING	返回字符串A在delim出现count次之前的子字符串。
translate(string|char|varchar input, string|char|varchar from, string|char|varchar to)	STRING	将input字符串中的所出现的字符或者字符串from用字符或者字符串to替换。例如：将abcde中的bcd替换成BCD，translate("abcde", "bcd", "BCD")
trim(string A)	STRING	删除字符串两端的空格，字符之间的空格保留。
upper(string A) ucase(string A)	STRING	将文本字符串转换成字母全部大写的形式。
initcap(string A)	STRING	将文本字符串转换成首字母大写其余字母小写的形式。
levenshtein(string A, string B)	INT	返回两个字符串之间的Levenshtein距离，如levenshtein('kitten','sitting') =3。
soundex(string A)	STRING	从str返回一个soundex字符串，如soundex('Miller')= M460。

13.4 聚合函数

聚集函数是从一组输入值计算一个结果。例如使用COUNT函数计算SQL查询语句返回的记录行数。聚合函数如**表13-4**所示。

表 13-4 聚合函数表

函数	返回值类型	描述
count(*), count(expr), count(DISTINCT expr[, expr...])	BIGINT	返回记录条数。
sum(col), sum(DISTINCT col)	DOUBLE	求和。
avg(col), avg(DISTINCT col)	DOUBLE	求平均值。

函数	返回值类型	描述
min(col)	DOUBLE	返回最小值。
max(col)	DOUBLE	返回最大值。
variance(col), var_pop(col)	DOUBLE	返回列的方差。
var_samp(col)	DOUBLE	返回指定列的样本方差。
stddev_pop(col)	DOUBLE	返回指定列的偏差。
stddev_samp(col)	DOUBLE	返回指定列的样本偏差。
covar_pop(col1, col2)	DOUBLE	返回两列数值协方差。
covar_samp(col1, col2)	DOUBLE	返回两列数值样本协方差。
corr(col1, col2)	DOUBLE	返回两列数值的相关系数。
percentile(BIGINT col, p)	DOUBLE	返回数值区域的百分比数值点。0<=P<=1,否则返回NULL,不支持浮点型数值。
percentile_approx(DOUBLE col, p [, B])	DOUBLE	返回组内数字列近似的第p位百分数（包括浮点数），p值在[0,1]之间。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。

说明

函数如var_pop，stddev_pop，var_samp，stddev_samp，covar_pop，covar_samp，corr，percentile_approx，不支持非数值数据类型，如TimeStamp。

13.5 分析窗口函数

窗口函数用于在与当前输入值相关的一组值上执行相关函数计算(包括在GROUP BY中使用的聚集函数，如sum函数、max函数、min函数、count函数、avg函数)，此外分析窗口函数还包括如表13-5中所示的函数。窗口是由一个OVER子句定义的多行记录，窗口函数作用于一个窗口。

表 13-5 函数介绍

函数	返回值	描述
first_value(col)	参数的数据类型	返回结果集中某列第一条数据的值。

函数	返回值	描述
last_value(col)	参数的数据类型	返回结果集中某列最后一条数据的值。
lag (col,n,DEFAULT)	参数的数据类型	用于统计窗口内往上第n行值。第一个参数为列名，第二个参数为往上第n行（可选，默认为1），第三个参数为默认值（当往上第n行为NULL时候，取默认值，如不指定，则为NULL）。
lead (col,n,DEFAULT)	参数的数据类型	用于统计窗口内往下第n行值。第一个参数为列名，第二个参数为往下第n行（可选，默认为1），第三个参数为默认值（当往下第n行为NULL时候，取默认值，如不指定，则为NULL）。
row_number() over (order by col_1[,col_2 ...])	INT	为每一行指派一个唯一的编号。
rank()	INT	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。
cume_dist()	DOUBLE	计算某个值在一行中的相对位置。
percent_rank()	DOUBLE	为窗口的ORDER BY子句所指定列中值的返回秩，但以介于0和1之间的小数形式表示，计算方法为 (RANK - 1)/(- 1)。

14 SELECT

14.1 基本语句

功能描述

基本的查询语句，返回查询结果。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_name_list]
  [ORDER BY col_name_list][ASC | DESC]
  [CLUSTER BY col_name_list | DISTRIBUTE BY col_name_list]
  [SORT BY col_name_list]]
  [LIMIT number];
```

关键字

表 14-1 SELECT 参数描述

参数	描述
ALL	返回重复的行。为默认选项。其后只能跟*，否则会出错。
DISTINCT	从结果集移除重复的行。
WHERE	指定查询的过滤条件，支持算术运算符、关系运算符和逻辑运算符。
where_condition	过滤条件。
GROUP BY	指定分组的字段，支持单字段及多字段分组。
col_name_list	字段列表。
ORDER BY	对查询结果进行排序。
ASC/DESC	ASC为升序，DESC为降序，默认为ASC。

参数	描述
CLUSTER BY	为分桶且排序，按照分桶字段先进行分桶，再在每个桶中依据该字段进行排序，即当DISTRIBUTE BY的字段与SORT BY的字段相同且排序为降序时，两者的作用与CLUSTER BY等效。
DISTRIBUTE BY	指定分桶字段，不进行排序。
SORT BY	将会在桶内进行排序。
LIMIT	对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

将表student中，name为Mike的数据记录查询出来，并根据字段score升序排序。

```
SELECT * FROM student
  WHERE name = 'Mike'
  ORDER BY score;
```

14.2 排序

14.2.1 ORDER BY

功能描述

按字段实现查询结果的全局排序。

语法格式

```
SELECT attr_expr_list FROM table_reference
  ORDER BY col_name
  [ASC | DESC] [,col_name [ASC | DESC],...];
```

关键字

- ASC/DESC：ASC为升序，DESC为降序，默认为ASC。
- ORDER BY：对全局进行单列或多列排序。与GROUP BY一起使用时，ORDER BY后面可以跟聚合函数。

注意事项

所排序的表必须是已经存在的，否则会出错。

示例

根据字段score对表student进行升序排序，并返回排序后的结果。


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

14.2.2 SORT BY

功能描述

按字段实现表的局部排序。

语法格式

```
SELECT attr_expr_list FROM table_reference
  SORT BY col_name
  [ASC | DESC] [,col_name [ASC | DESC],...];
```

关键字

- ASC/DESC：ASC为升序，DESC为降序，默认为ASC。
- SORT BY：一般与GROUP BY一起使用，为PARTITION进行单列或多列的局部排序。

注意事项

所排序的表必须是已经存在的，否则会出错。

示例

根据字段score对表student在Reducer中进行升序排序。

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

14.2.3 CLUSTER BY

功能描述

按字段实现表的分桶及桶内排序。

语法格式

```
SELECT attr_expr_list FROM table_reference
  CLUSTER BY col_name [,col_name ,...];
```

关键字

CLUSTER BY：根据指定的字段进行分桶，支持单字段及多字段，并在桶内进行排序。

注意事项

所排序的表必须是已经存在的，否则会出错。

示例

根据字段score对表student进行分桶并进行桶内局部降序排序。


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

14.2.4 DISTRIBUTE BY

功能描述

按字段实现表的分桶。

语法格式

```
SELECT attr_expr_list FROM table_reference
  DISTRIBUTE BY col_name [,col_name ,...];
```

关键字

DISTRIBUTE BY：根据指定的字段进行分桶，支持单字段及多字段，不会在桶内进行排序。与SORT BY配合使用即为分桶后的排序。

注意事项

所排序的表必须是已经存在的，否则会出错。

举例

根据字段score对表student进行分桶。

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

14.3 分组

14.3.1 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，attr_expr_list中包含的字段必须出现在col_name_list中，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，attr_expr_list中出现的字段必须在GROUP BY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;
```

14.3.2 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，attr_expr_list中出现的字段必须包含在GROUP BY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);
```

14.3.3 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;
```

14.3.4 ROLLUP

功能描述

ROLLUP生成聚合行、超聚合行和总计行。可以实现从右到左递减多级的统计，显示统计某一层次结构的聚合。

语法格式

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

关键字

ROLLUP：为GROUP BY的扩展，例如：***SELECT a, b, c, SUM(expression) FROM table GROUP BY a, b, c WITH ROLLUP;***将转换成以下四条查询：

- (a, b, c)组合小计
  ```
  SELECT a, b, c, sum(expression) FROM table
    GROUP BY a, b, c;
  ```
- (a, b)组合小计
  ```
  SELECT a, b, NULL, sum(expression) FROM table
    GROUP BY a, b;
  ```
- (a)组合小计
  ```
  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。

示例

根据group_id与job两个字段生成聚合行、超聚合行和总计行，返回每种聚合情况下的salary总和。


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

14.3.5 GROUPING SETS

功能描述

GROUPING SETS生成交叉表格行，可以实现GROUP BY字段的交叉统计。

语法格式

```
SELECT attr_expr_list FROM table_reference
  GROUP BY col_name_list
  GROUPING SETS(col_name_list);
```

关键字

GROUPING SETS：为对GROUP BY的扩展，例如

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS((a,b));***

  将转换为以下一条查询：

  ```
  SELECT a, b, sum(expression) FROM table
    GROUP BY a, b;
  ```

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS(a,b);***

  将转换为以下两条查询：

  ```
  SELECT a, NULL, sum(expression) FROM table GROUP BY a;
  UNION
  SELECT NULL, b, sum(expression) FROM table GROUP BY b;
  ```

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS((a,b), a);***

  将转换为以下两条查询：

  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b;
  UNION
  SELECT a, NULL, sum(expression) FROM table GROUP BY a;
  ```

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS((a,b), a, b, ());***

  将转换为以下四条查询：

  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b;
  UNION
  SELECT a, NULL, sum(expression) FROM table GROUP BY a, NULL;
  UNION
  SELECT NULL, b, sum(expression) FROM table GROUP BY NULL, b;
  UNION
  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。
- 不同于ROLLUP，GROUPING SETS目前仅支持一种格式。

示例

根据group_id与job两个字段生成交叉表格行，返回每种聚合情况下的salary总和。


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

14.4 连接

14.4.1 内连接

功能描述

仅将两个表中满足连接条件的行组合起来作为结果集。

语法格式

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

关键字

JOIN/INNER JOIN：只显示参与连接的表中满足JOIN条件的记录。

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。

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);
```

14.4.2 左外连接

功能描述

根据左表的记录去匹配右表，返回所有左表记录，没有匹配值的记录的返回NULL。

语法格式

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

关键字

LEFT OUTER JOIN：返回左表的所有记录，没有匹配值的记录将返回NULL。

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。

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);
```

14.4.3 右外连接

功能描述

根据右表的记录去匹配左表，返回所有右表记录，没有匹配值的记录返回NULL。

语法格式

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

关键字

RIGHT OUTER JOIN：返回右表的所有记录，没有匹配值的记录将返回NULL。

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。

示例

右外连接和左外连接相似，但是会将右边表（这里的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);
```

14.4.4 全外连接

功能描述

根据左表与右表的所有记录进行匹配，没有匹配值的记录返回NULL。

语法格式

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

关键字

FULL OUTER JOIN：根据左表与右表的所有记录进行匹配，没有匹配值的记录返回NULL。

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。

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);
```

14.4.5 隐式连接

功能描述

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显示指定连接条件。

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;
```

关键字

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。**语法格式**中仅给出等式条件下的WHERE条件过滤，同时也支持不等式WHERE条件过滤。

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。
- 隐式JOIN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。

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;
```

14.4.6 笛卡尔连接

功能描述

笛卡尔连接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条记录数。

语法格式

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

关键字

join_condition：连接条件，如果该条件恒成立（比如1=1），该连接就是笛卡尔连接。所以，笛卡尔连接输出的记录条数等于被连接表的各记录条数的乘积，若需要进行笛

卡尔积连接，需使用专门的关键词CROSS JOIN。CROSS JOIN是求笛卡尔积的标准方式。

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。

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 ON (1 = 1);
```

14.4.7 左半连接

功能描述

左半连接用来查看左表中符合JOIN条件的记录。

语法格式

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

关键字

LEFT SEMI JOIN：只显示左表中的记录。可通过在LEFT SEMI JOIN， WHERE...IN和WHERE EXISTS中嵌套子查询来实现。左半连接与左外连接的区别是，左半连接将返回左表中符合JOIN条件的记录，而左外连接将返回左表所有的记录，匹配不上JOIN条件的记录将返回NULL值。

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。
- 此处的attr_expr_list中所涉及的字段只能是左表中的字段，否则会出错。

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);
```

14.4.8 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;
```


关键字

non_equi_join_condition：与join_condition类似，只是join条件均为不等式条件。

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。

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);
```

14.5 子句

14.5.1 FROM

功能描述

在FROM子句中嵌套子查询，子查询的结果作为中间过渡表，进而作为外部SELECT语句的数据源。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM (sub_query) [alias];
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。

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，子查询必须要取别名，且别名的命名要早于别名的使用，否则会出错。建议别名不要重名。
- FROM后所跟的子查询结果必须带上前面所取的别名，否则会出错。

示例

返回选了course_info表中课程的学生姓名，并利用DISTINCT关键字进行去重。

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


14.5.2 OVER

功能描述

窗口函数与OVER语句一起使用。OVER语句用于对数据进行分组，并对组内元素进行排序。窗口函数用于给组内的值生成序号。

语法格式

```
SELECT window_func(args) OVER
  ([PARTITION BY col_name, col_name, ...]
   [ORDER BY col_name, col_name, ...]
   [ROWS | RANGE BETWEEN (CURRENT ROW | (UNBOUNDED |[num]) PRECEDING)
  AND (CURRENT ROW | ( UNBOUNDED | [num]) FOLLOWING)]);
```

关键字

- PARTITION BY：可以用一个或多个键分区。和GROUP BY子句类似，PARTITION BY将表按分区键分区，每个分区是一个窗口，窗口函数作用于各个分区。单表分区数最多允许7000个。
- ORDER BY：决定窗口函数求值的顺序。可以用一个或多个键排序。通过ASC或DESC决定升序或降序。窗口由WINDOW子句指定。如果不指定，默认窗口等同于ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW，即窗口从表或分区（如果OVER子句中用PARTITION BY分区）的初始处到当前行。
- WINDOW：通过指定一个行区间来定义窗口。
- CURRENT ROW：表示当前行。
- num PRECEDING：定义窗口的下限，即窗口从当前行向前数num行处开始。
- UNBOUNDED PRECEDING：表示窗口没有下限。
- num FOLLOWING：定义窗口的上限，即窗口从当前行向后数num行处结束。
- UNBOUNDED FOLLOWING：表示窗口没有上限。
- ROWS BETWEEN…和RANGE BETWEEN…的区别：
  - ROW为物理窗口，即根据ORDER BY子句排序后，取前N行及后N行的数据计算（与当前行的值无关，只与排序后的行号相关）。
  - RANGE为逻辑窗口，即指定当前行对应值的范围取值，列数不固定，只要行值在范围内，对应列都包含在内。
- 窗口有以下多种场景，如
  - 窗口只包含当前行。
    ```
    ROWS BETWEEN CURRENT ROW AND CURRENT ROW
    ```
  - 窗口从当前行向前数3行开始，到当前行向后数5行结束。
    ```
    ROWS BETWEEN 3 PRECEDING AND 5 FOLLOWING
    ```
  - 窗口从表或分区的开头开始，到当前行结束。
    ```
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ```
  - 窗口从当前行开始，到表或分区的结尾结束。
    ```
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
    ```
  - 窗口从表或分区的开头开始，到表或分区的结尾结束。
    ```
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
    ```


注意事项

OVER子句包括：PARTITION BY子句、ORDER BY子句和WINDOW子句，可组合使用。OVER子句为空表示窗口为整张表。

示例

上述语句窗口从表或分区的开头开始，到当前行结束，对over_test表按照id字段进行排序，并返回排序好后的id及id所对应的序号。

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

14.5.3 WHERE

功能描述

在WHERE子句中嵌套子查询，利用子查询的结果作为过滤条件。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  WHERE {col_name operator (sub_query) | [NOT] EXISTS sub_query};
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。
- WHERE：WHERE子句嵌套将利用子查询的结果作为过滤条件。
- operator：包含关系运算符中的等式与不等式操作符及IN，NOT IN，EXISTS，NOT EXISTS操作符。
  - 当operator为IN或者NOT IN时，子查询的返回结果必须是单列。
  - 当operator为EXISTS或者NOT EXISTS时，子查询中一定要包含WHERE条件过滤。当子查询中有字段与外部查询相同时，需要在该字段前加上表名。

注意事项

所要查询的表必须是已经存在的表，否则会出错。

示例

先通过子查询在course_info中找到Biology所对应的课程编号，再在student_info表中找到选了该课程编号的学生姓名。

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

14.5.4 HAVING

功能描述

在HAVING子句中嵌套子查询，子查询结果将作为HAVING子句的一部分。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  GROUP BY groupby_expression
  HAVING aggregate_func(col_name) operator (sub_query);
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。
- 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。
- operator：此操作符包含等式操作符与不等式操作符，及IN，NOT IN操作符。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- 此处的sub_query与聚合函数的位置不能左右互换。

示例

对表student_info按字段name进行分组，计算每组中记录数，若其记录数等于子查询中表course_info的记录数，返回表student_info中字段name等于表course_info字段name的记录数。

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

14.5.5 多层嵌套子查询

功能描述

多层嵌套子查询，即在子查询中嵌套子查询。

语法格式

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

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。

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。
- 别名的命名必须在别名的使用之前，否则会出错，建议别名不要重名。

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;
```

14.6 别名 SELECT

14.6.1 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";
  ```
- 将子查询的结果命令为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;
  ```

14.6.2 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";
```

14.7 集合运算 SELECT

14.7.1 UNION

功能描述

UNION返回多个查询结果的并集。

语法格式

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

关键字

UNION：集合运算，以一定条件将表首尾相接，其中每一个SELECT语句返回的列数必须相同，列的类型和列名不一定要相同。

注意事项

- UNION默认是去重的，UNION ALL是不去重的。
- 不能在多个集合运算间（UNION，INTERSECT，EXCEPT）加括号，否则会出错。

示例

返回“SELECT * FROM student _1”查询结果与“SELECT * FROM student _2”查询结果的并集，不包含重复记录。

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

14.7.2 INTERSECT

功能描述

INTERSECT返回多个查询结果的交集。

语法格式

```
select_statement INTERSECT select_statement;
```

关键字

INTERSECT：返回多个查询结果的交集，且每一个SELECT语句返回的列数必须相同，列的类型和列名不一定要相同。INTERSECT默认去重。

注意事项

不能在多个集合运算间（UNION，INTERSECT，EXCEPT）加括号，否则会出错

示例

返回“SELECT * FROM student _1”查询结果与“SELECT * FROM student _2”查询结果的交集，不包含重复记录。

```
SELECT * FROM student _1 INTERSECT SELECT * FROM student _2;
```

14.7.3 EXCEPT

功能描述

返回两个查询结果的差集。

语法格式

```
select_statement EXCEPT select_statement;
```

关键字

EXCEPT：做集合减法。A EXCEPT B将A中所有和B重合的记录扣除，然后返回去重后的A中剩下的记录，EXCEPT默认不去重。与UNION相同，每一个SELECT语句返回的列数必须相同，列的类型和列名不一定要相同。

注意事项

不能在多个集合运算间（UNION，INTERSECT，EXCEPT）加括号，否则会出错

示例

先将“SELECT * FROM student_1”查询结果减去“SELECT * FROM student_2”结果中的重合部分，然后返回剩下来的记录。

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

14.8 WITH...AS

功能描述

通过用WITH...AS定义公共表达式（CTE）来简化查询，提高可阅读性和易维护性。

语法格式

```
WITH cte_name AS (select_statement) sql_containing_cte_name;
```

关键字

- cte_name：公共表达式的名字，不允许重名。
- select_statement：完整的SELECT语句。
- sql_containing_cte_name：包含了刚刚定义的公共表达式的SQL语句

注意事项

- 定义了一个CTE后必须马上使用，否则这个CTE定义将失效。
- 可以通过一次WITH定义多个CTE，中间用逗号连接，后定义的CTE可以引用已经定义的CTE。

示例

将“SELECT courseId FROM course_info WHERE courseName = 'Biology'”定义为公共表达式nv，然后在后续的查询中直接利用nv代替该SELECT语句。

```
WITH nv AS (SELECT courseId FROM course_info WHERE courseName = 'Biology') SELECT DISTINCT courseId FROM nv;
```

14.9 CASE...WHEN

14.9.1 简单 CASE 函数

功能描述

依据input_expression与when_expression的匹配结果跳转到相应的result_expression。

语法格式

```
CASE input_expression WHEN when_expression THEN result_expression [...n] [ELSE else_result_expression] END;
```

关键字

CASE：简单CASE函数中支持子查询，但须注意input_expression与when_expression是可匹配的。

注意事项

如果没有取值为TRUE的input_expression = when_expression，则当指定ELSE子句时，DLI将返回else_result_expression；当没有指定ELSE子句时，返回NULL值。

示例

返回表student中的字段name及与id相匹配的字符。匹配规则如下：

- id为1则返回'a'；
- id为2则返回'b'；
- id为3则返回'c'；
- 否则返回NULL。

```
SELECT name, CASE id WHEN 1 THEN 'a' WHEN 2 THEN 'b' WHEN 3 THEN 'c' ELSE NULL END FROM student;
```


14.9.2 CASE 搜索函数

功能描述

按指定顺序为每个WHEN子句的boolean_expression求值。返回第一个取值为TRUE的boolean_expression的result_expression。

语法格式

```
CASE WHEN boolean_expression THEN result_expression [...n] [ELSE else_result_expression] END;
```

关键字

boolean_expression：可以包含子查询，但整个boolean_expression表达式返回值只能是布尔类型。

注意事项

如果没有取值为TRUE的Boolean_expression，则当指定ELSE子句时，DLI将返回else_result_expression；当没有指定ELSE子句时，返回NULL值。

示例

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;
```


15 标示符

15.1 aggregate_func

格式

无。

说明

聚合函数。

15.2 alias

格式

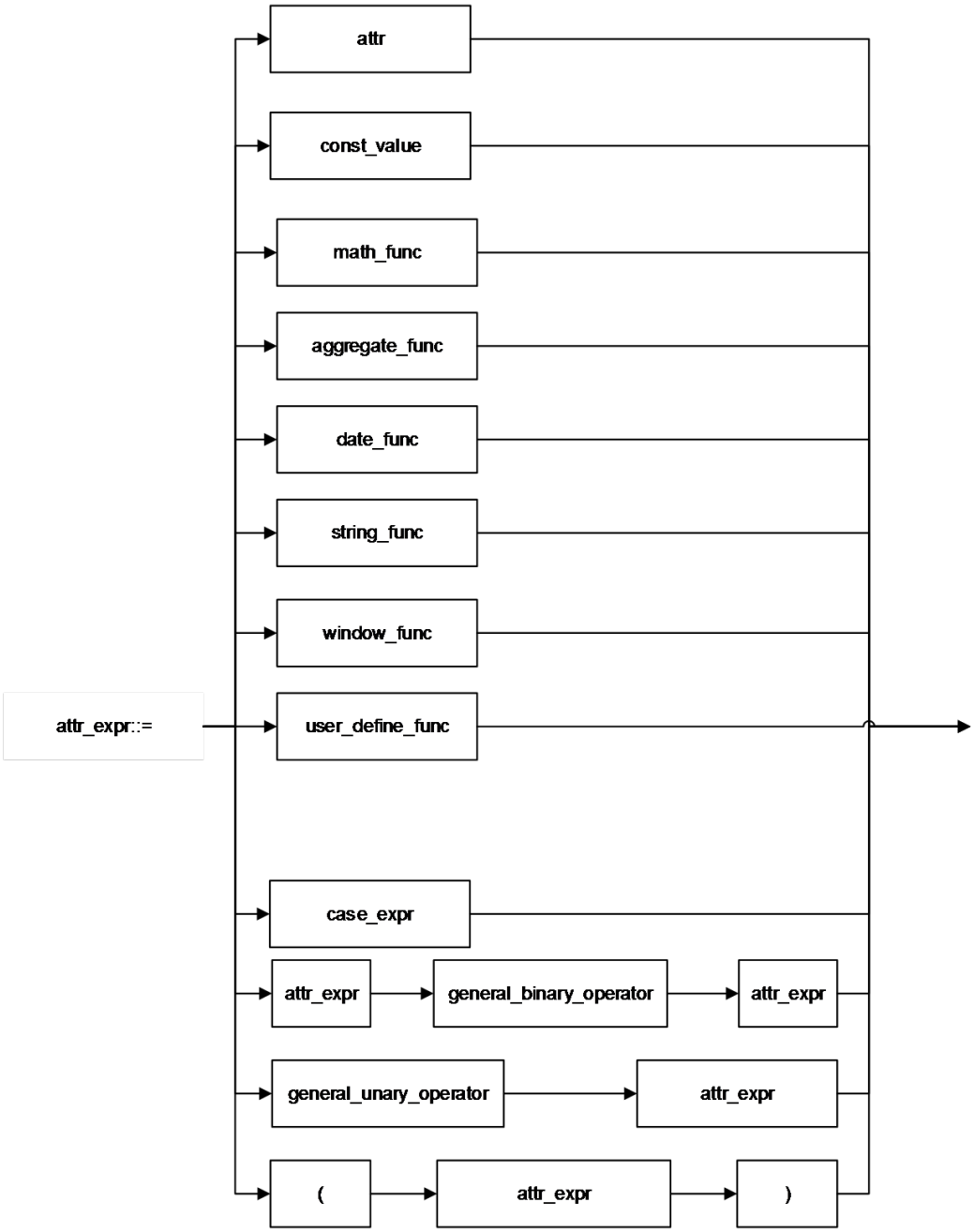
无。

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。

15.3 attr_expr

格式



说明

语法	描述
attr_expr	属性表达式。

语法	描述
attr	表的字段，与col_name相同。
const_value	常量值。
case_expr	case表达式。
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。

15.4 attr_expr_list

格式

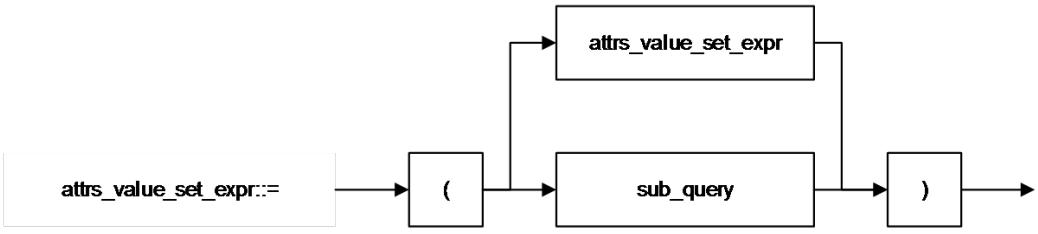
无。

说明

attr_expr列表，以逗号分隔。

15.5 attrs_value_set_expr

格式



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。

15.6 boolean_expression

格式

无。

说明

返回boolean类型的表达式。

15.7 class_name

格式

无。

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。

15.8 col

格式

无。

说明

函数调用时的形参，一般即为字段名称，与col_name相同。

15.9 col_comment

格式

无。

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。

15.10 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。

15.11 col_name_list

格式

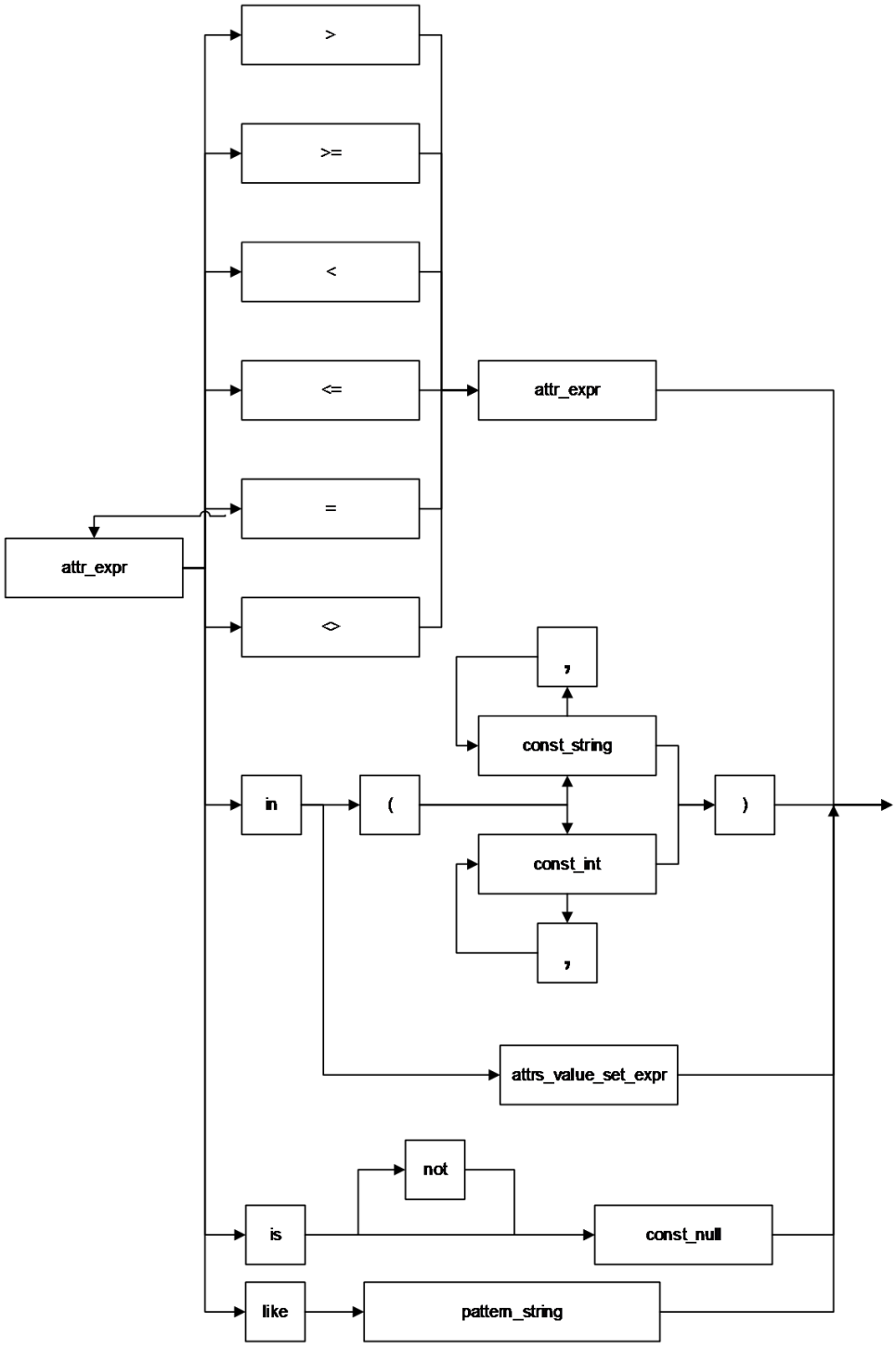
无。

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。

15.12 condition

格式

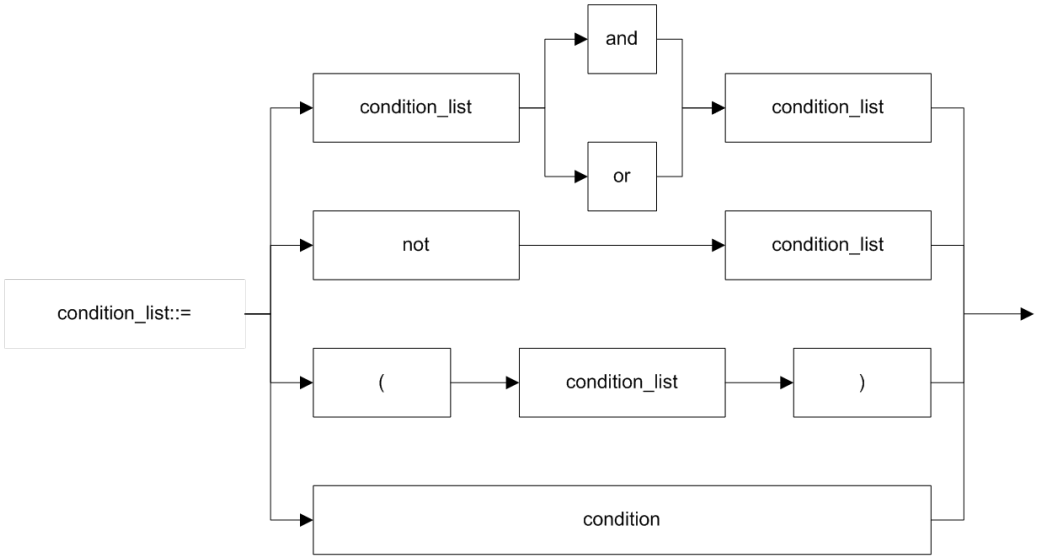


说明

语法	描述
condition	逻辑判断条件。
>	关系运算符：大于。
>=	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。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。
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。

15.13 condition_list

格式



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。

15.14 cte_name

格式

无。

说明

公共表达式的名字。

15.15 data_type

格式

无。

说明

数据类型，当前只支持原生数据类型。

15.16 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。

15.17 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。

15.18 else_result_expression

格式

无。

说明

CASE WHEN语句中ELSE语句后的返回结果。

15.19 file_format

格式

| AVRO

| CARBON

| CSV

| JSON

| ORC

| PARQUET

说明

- 目前包含以上6种格式。
- 指定数据格式的方式有两种，一种是USING，可指定以上6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。
- CARBON不支持tinyint，float，date数据类型。

15.20 file_path

格式

无。

说明

文件路径，该路径是OBS路径。

15.21 function_name

格式

无。

说明

函数名称，仅支持字符串类型。

15.22 groupby_expression

格式

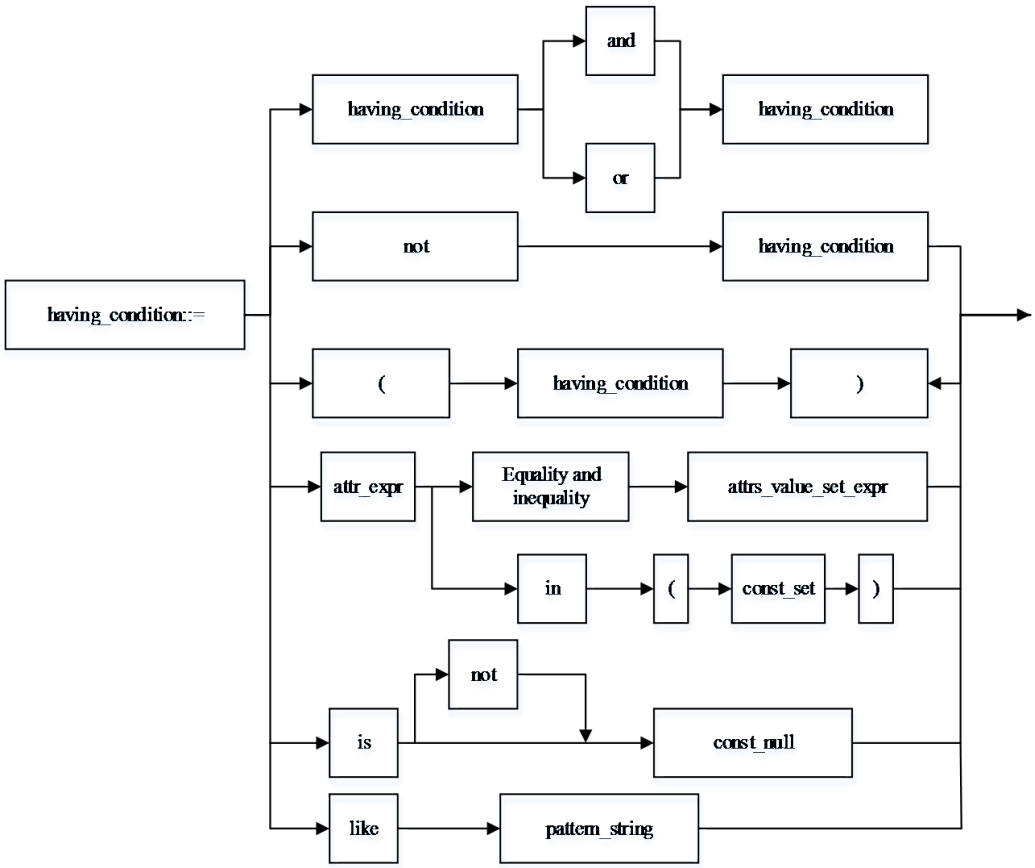
无。

说明

包含GROUP BY的表达式。

15.23 having_condition

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。

语法	描述
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**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。
like	关系运算符：用于通配符匹配。

15.24 hdfs_path

格式

无。

说明

HDFS的路径，如“hdfs:///tmp”。

15.25 input_expression

格式

无。

说明

CASE WHEN的输入表达式。

15.26 input_format_classname

格式

无。

说明

指定输入格式的类名，如***org.apache.hadoop.mapred.TextInputFormat***。

15.27 jar_path

格式

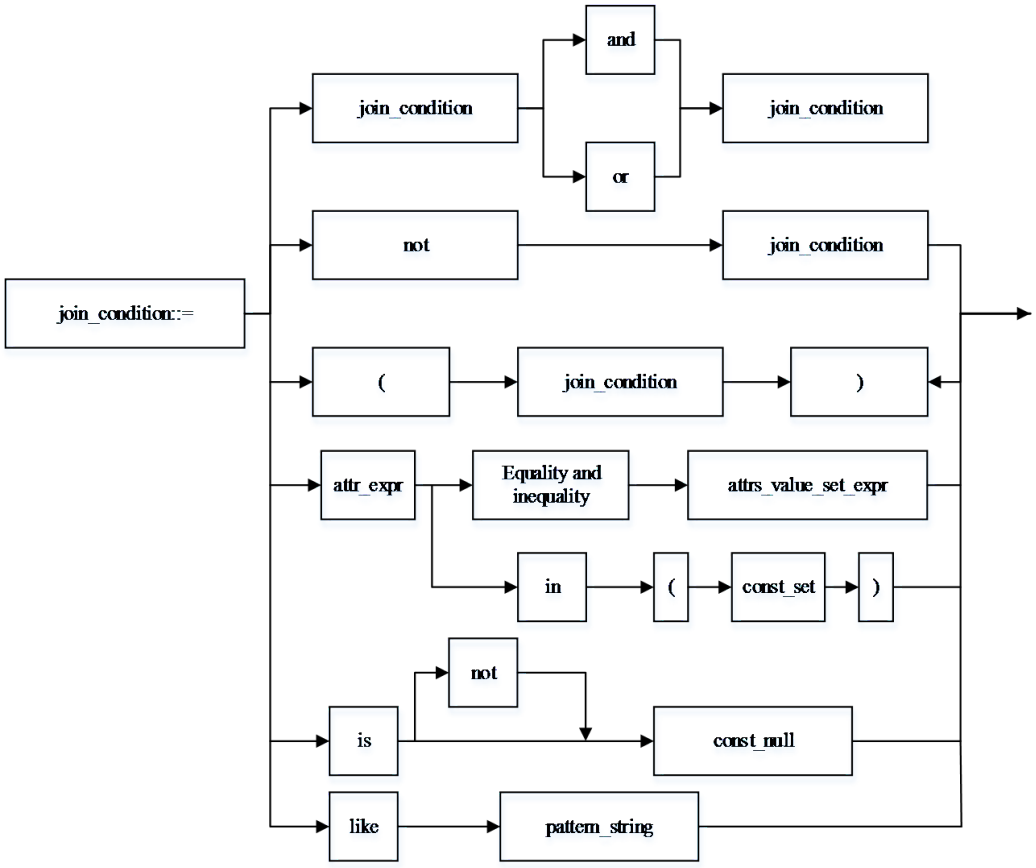
无。

说明

jar包路径，该路径可以是本地路径也可以是HDFS路径。

15.28 join_condition

格式



说明

语法	描述
join_condition	join逻辑判断条件。
and	逻辑运算符：与。

语法	描述
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。
atrrs_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**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。

15.29 non_equi_join_condition

格式

无。

说明

指不等式join条件。

15.30 number

格式

无。

说明

LIMIT限制输出的行数，只支持INT类型。

15.31 num_buckets

格式

无。

说明

分桶的个数，仅支持INT类型。

15.32 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。***

15.33 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。

15.34 partition_col_value

格式

无。

说明

分区列值，即分区字段的值。

15.35 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。

15.36 property_name

格式

无。

说明

属性名称，仅支持字符串类型。

15.37 property_value

格式

无。

说明

属性值，仅支持字符串类型。

15.38 regex_expression

格式

无。

说明

模式匹配字符串，支持通配符匹配。

15.39 result_expression

格式

无。

说明

CASE WHEN语句中THEN语句后的返回结果。

15.40 row_format

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 KEY TERMINATED BY仅用于指定MAP类型中的key与vaule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割符，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' = '')

15.41 select_statement

格式

无。

说明

SELECT基本语句，即查询语句。

15.42 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。

15.43 serde_name

格式

无。

说明

指定serde的名称。

15.44 sql_containing_cte_name

格式

无。

说明

包含了cte_name定义的公共表达式的SQL语句。

15.45 sub_query

格式

无。

说明

指子查询。

15.46 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。

15.47 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。

15.48 table_properties

格式

无。

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。

15.49 table_reference

格式

无。

说明

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。

15.50 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。

15.51 view_properties

格式

无。

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。

15.52 when_expression

格式

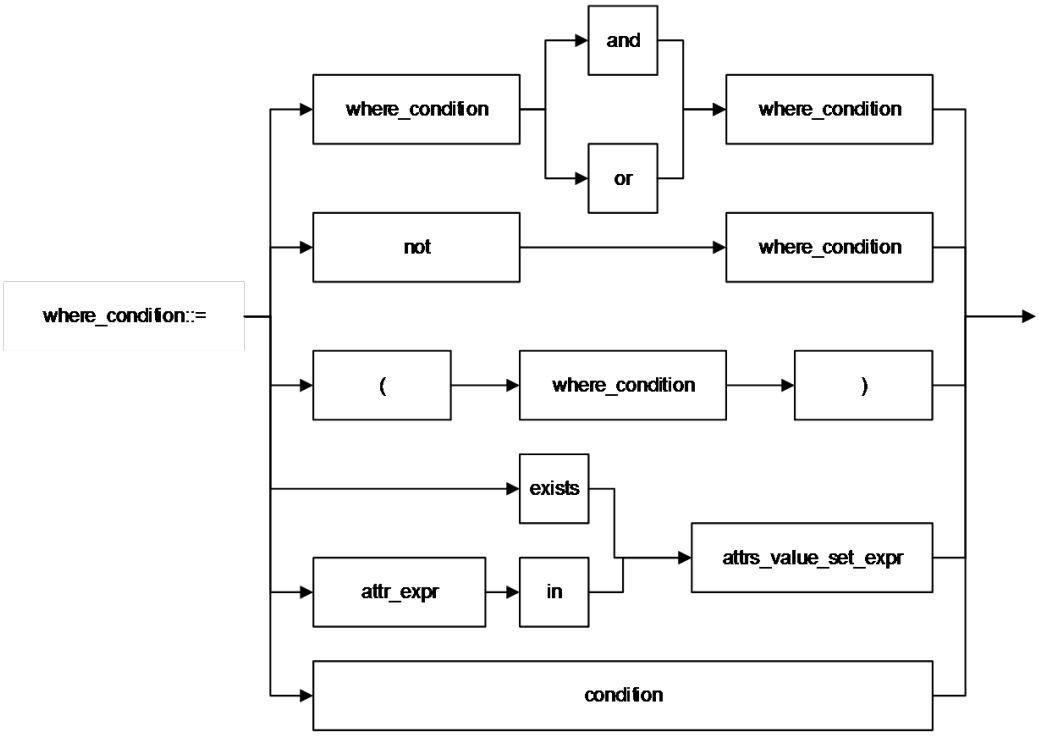
无。

说明

CASE WHEN语句的when表达式，与输入表达式进行匹配。

15.53 where_condition

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。

15.54 window_function

格式

无。

说明

分析窗口函数，详情请参见**分析窗口函数**。

16 运算符

16.1 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。

DLI提供的关系运算符，请参见表16-1。

表 16-1 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FALSE，若A与B都为NULL则返回TRUE，A与B其中一个为NULL则返回FALSE。
A <> B	BOOLEAN	若A与B不相等，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL，该种运算符为标准SQL语法。
A != B	BOOLEAN	与<>逻辑操作符相同，该种运算符为SQL Server语法。
A < B	BOOLEAN	若A小于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A <= B	BOOLEAN	若A小于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A > B	BOOLEAN	若A大于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。

运算符	返回类型	描述
A >= B	BOOLEAN	若A大于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A 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三者中存在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三者中存在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的LIKE操作，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。

16.2 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。DLI所支持的算术运算符如表16-2所示。

表 16-2 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，结果数据之类与操作数据类型相关。

运算符	返回类型	描述
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）。

16.3 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16-3**，表中的A和B代表逻辑表达式。

表 16-3 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。
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。

运算符	返回类型	描述
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。

