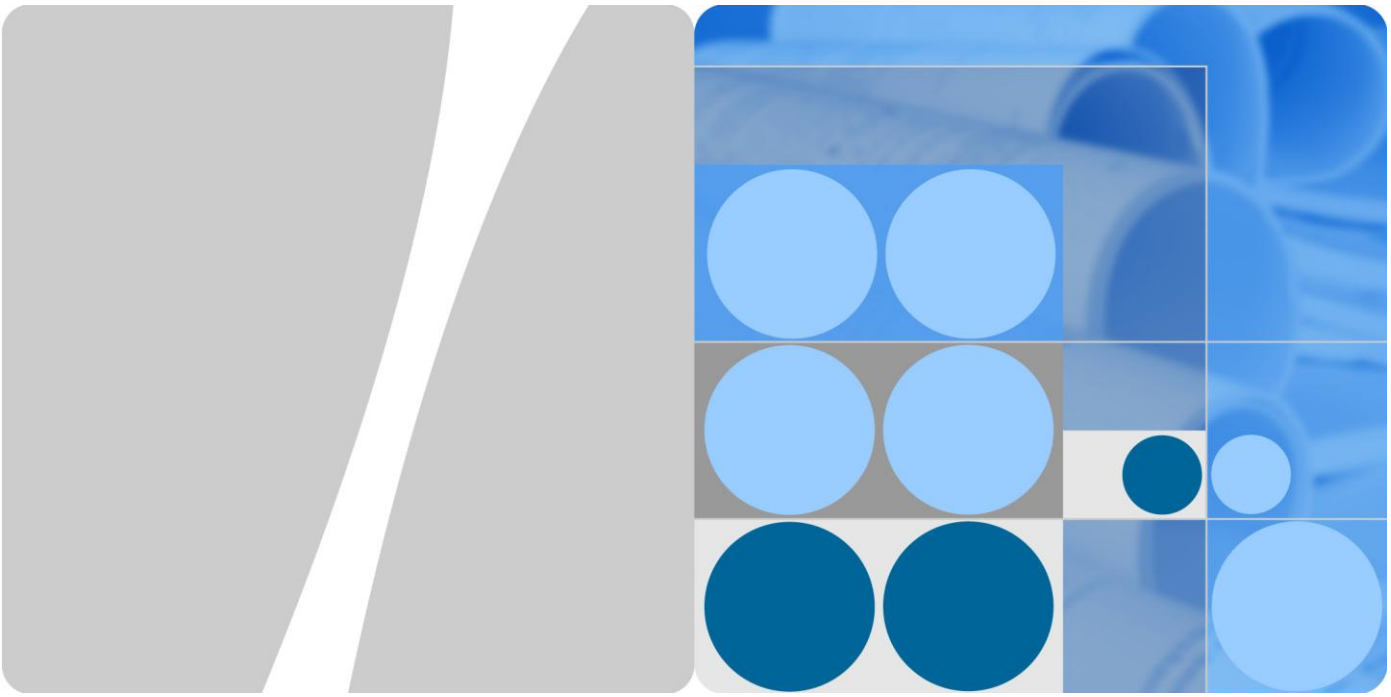


FusionInsight HD
V100R002C60U10

Spark SQL 语法参考

文档版本 02

发布日期 2016-09-30

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

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

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

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： http://e.huawei.com

目 录

1 标示符

1.1 aggregate_func

格式

无。

说明

聚合函数，详情请参见 4.4 聚合函数。

1.2 alias

格式

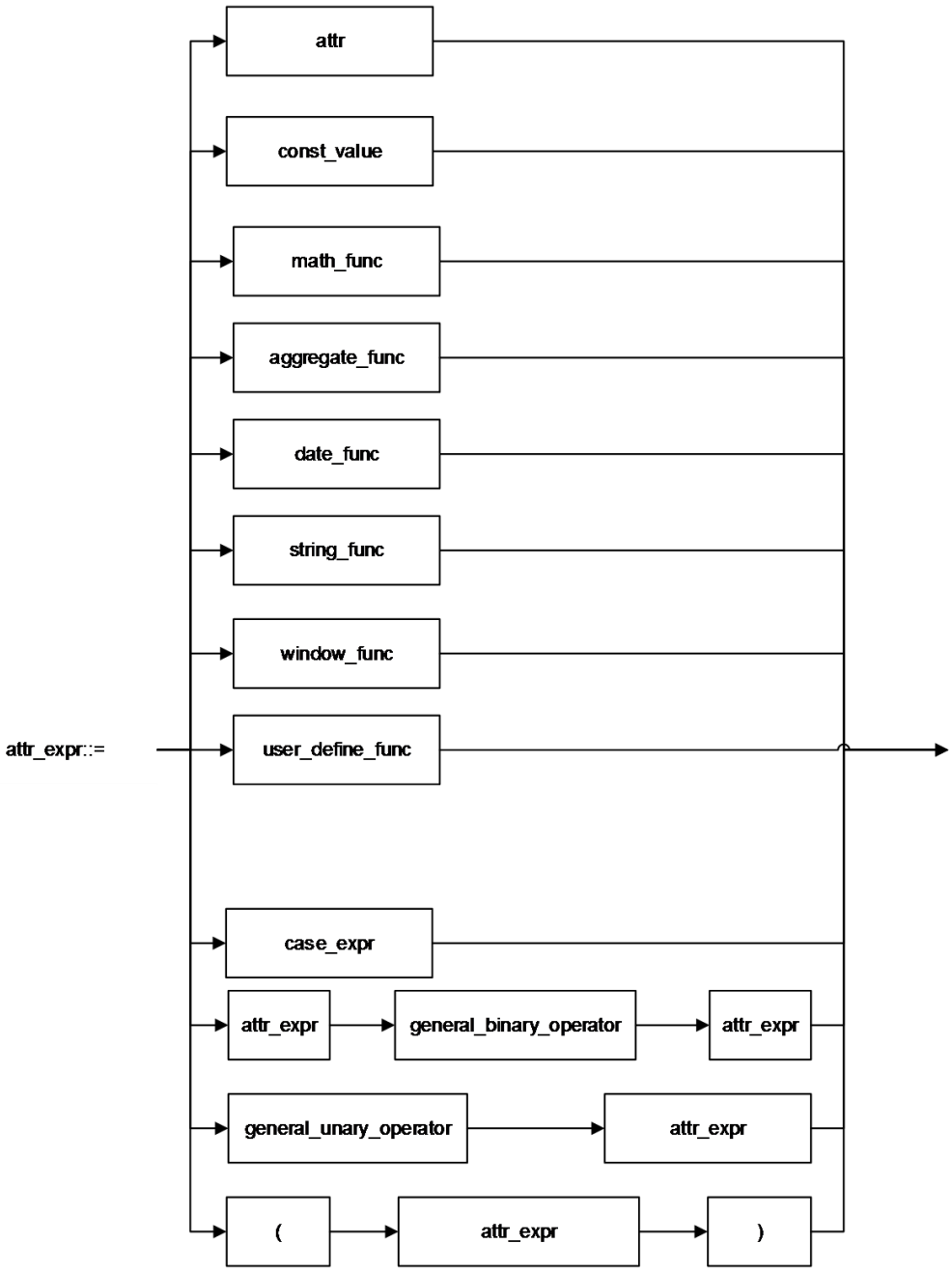
无。

说明

别名，可给字段、表、视图、子查询起表别名，仅支持字符串类型。

1.3 attr_expr

格式



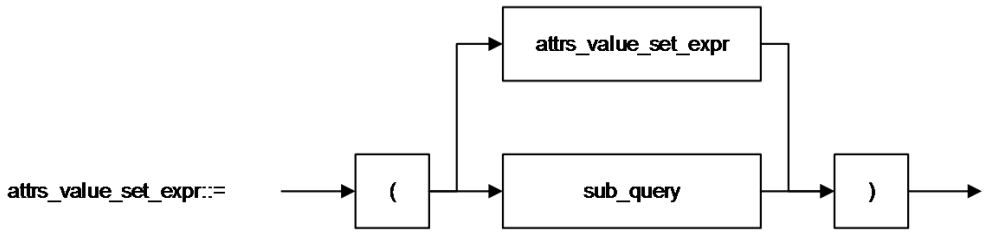
说明

语法	描述
attr_expr	属性表达式
attr	表的字段，与 col_name 相同。

语法	描述
const_value	常量值。
case_expr	case 表达式，CASE…WHEN 用法，详情请参见 8.10 CASE...WHEN。
math_func	数学函数，详情请参见 4.1 数学函数。
date_func	日期函数，详情请参见 4.2 日期函数。
string_func	字符串函数，详情请参见 4.3 字符串函数。
aggregate_func	聚合函数，详情请参见 4.4 聚合函数。
window_func	分析窗口函数，详情请参见 4.5 分析窗口函数。
user_define_func	用户自定义函数，详情请参见 6.12 创建函数。
general_binary_operator	普通二元操作符，详情请参见 3 运算符。
general_unary_operator	普通一元操作符，详情请参见 3 运算符。
(	指定子属性表达式开始。
)	指定子属性表达式结束。

1.4 attrs_value_set_expr

格式



说明

语法	描述
attrs_value_set_expr	属性值集合。

语法	描述
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。

1.5 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的包的完整路径。

1.6 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。

1.7 col_comment

格式

无。

说明

对列（字段）的描述，仅支持字符串类型。

1.8 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。

1.9 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。

1.10 col_value

格式

无。

说明

列值，即字段值。

1.11 col_value_list

格式

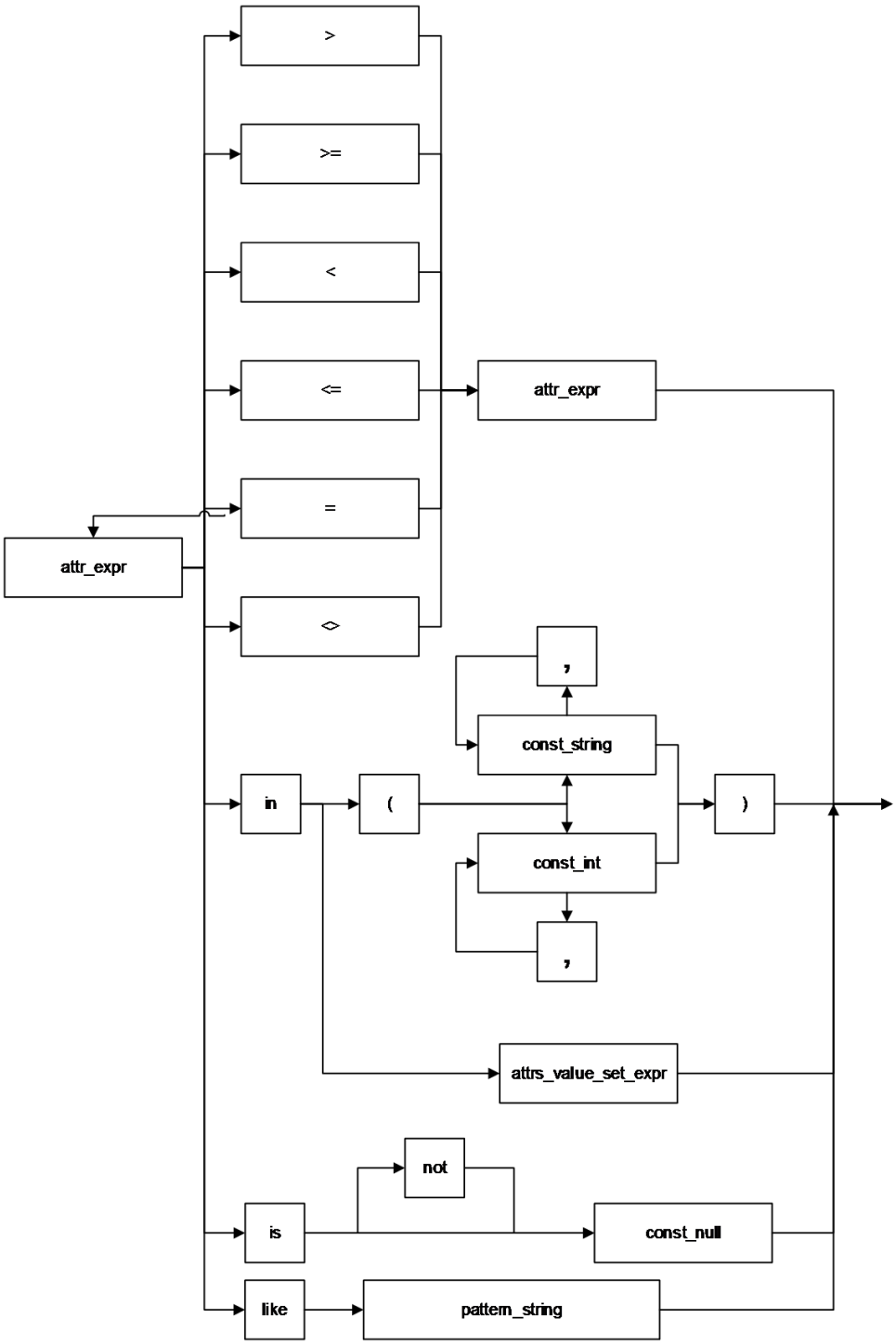
无。

说明

列值构成的列表，由一个或者多个 col_value 构成，多个 col_value 之间用逗号隔开。

1.12 condition

格式



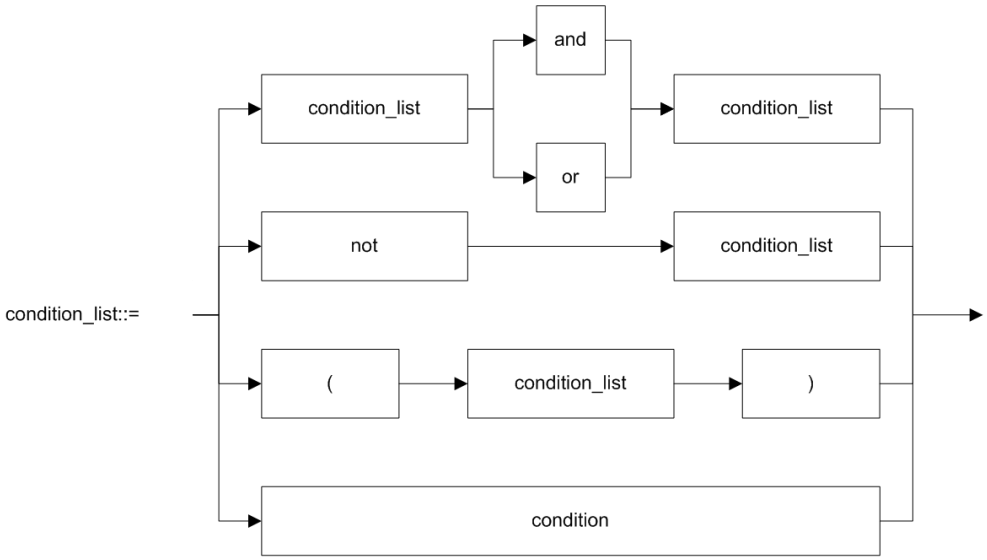
说明

语法	描述

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。CREATE TABLE LIKE 中不支持通配符。
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

1.13 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。

1.14 cte_name

格式

无。

说明

公共表达式的名字。

1.15 data_type

格式

无。

说明

数据类型，包括原生数据类型与复杂数据类型。

1.16 db_comment

格式

无。

说明

对数据库的描述，仅支持字符串类型。

1.17 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 个字节。

1.18 else_result_expression

格式

无。

说明

CASE WHEN 语句中 ELSE 语句后的返回结果。

1.19 file_format

格式

SEQUENCEFILE

| *TEXTFILE*

| *RCFILE*

| *ORC*

| *PARQUET*

| *AVRO*

| *INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname*

说明

- 目前包含以上 7 种格式，以 STORED AS file_format 来规定表的数据格式。
- SEQUENCEFILE 是由 Hadoop API 提供的一种二进制文件格式，数据以(key,value)的形式保存到文件中，具有使用方便、可分割、可压缩的特点。SEQUENCEFILE 支持三种压缩方式：NONE、RECORD、BLOCK。RECORD 压缩率低，一般建议使用 BLOCK 压缩。
- TEXTFILE 默认格式，数据不做压缩，磁盘开销大，数据解析开销大。
- RCFile 是一种专门面向列的数据格式。它遵循“先水平划分，再垂直划分”的设计理念。
- 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。
- AVRO 是一种用于支持密集型数据的二进制文件格式。其文件格式更为紧凑，若要读取大量数据时，AVRO 能够提供更好的序列化和反序列化性能。
- INPUTFORMAT 可指定输入格式，OUTPUTFORMAT 可指定输出格式，如下用法：

  CREATE TABLE student (id INT, name STRING)
  STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
  OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';

1.20 file_path

格式

无。

说明

文件路径，该路径可以是本地路径也可以是 HDFS 路径。

1.21 function_name

格式

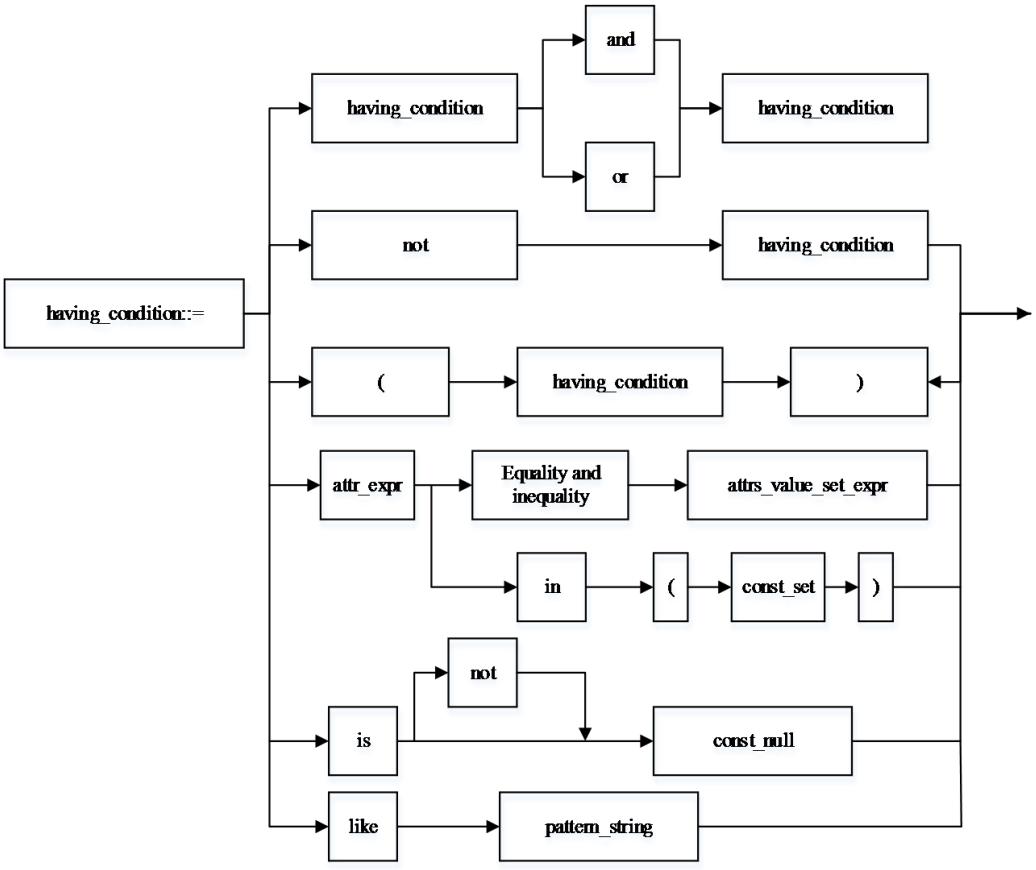
无。

说明

函数名称，仅支持字符串类型。

1.22 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。

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。
attrs_value_set_expr	属性值集合。
attr_expr	属性表达式语法规范。
Equality and inequality	等式与不等式，详情请参见 3.1 关系运算符。
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE 条件过滤时，支持 SQL 通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。CREATE TABLE LIKE 中不支持通配符。
like	关系运算符：用于通配符匹配。

1.23 hdfs_path

格式

无。

说明

HDFS 的路径，如“hdfs:///tmp”。

1.24 input_expression

格式

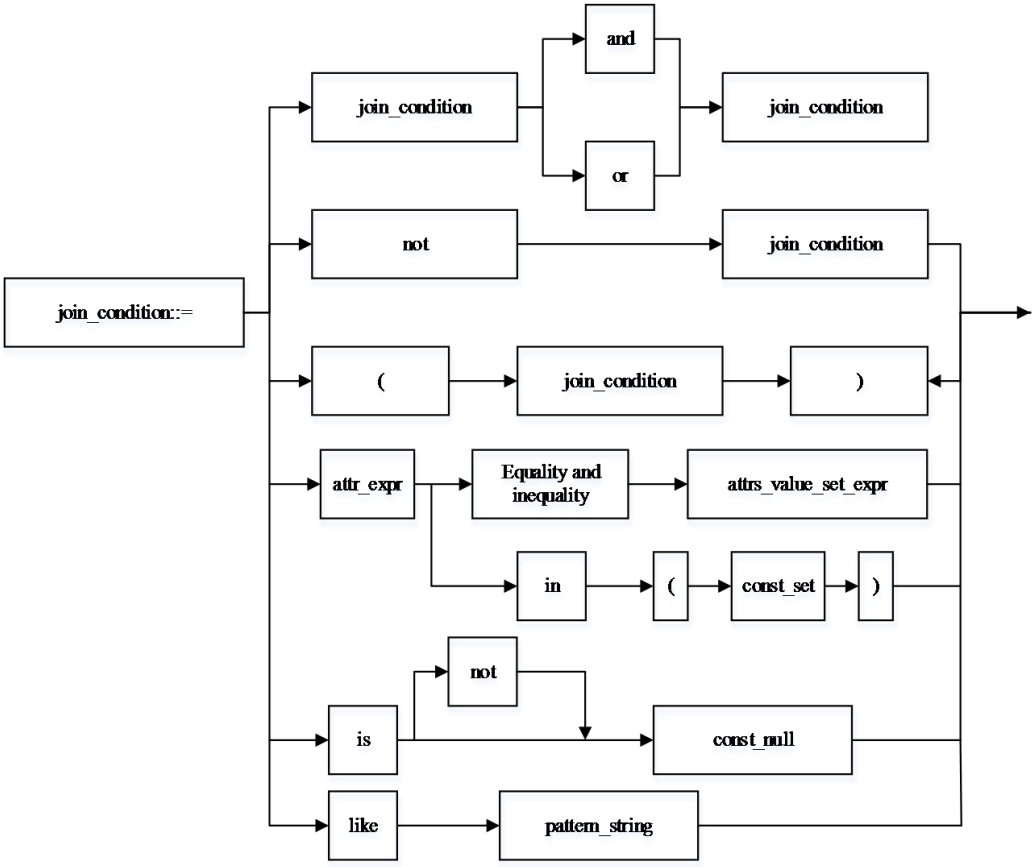
无。

说明

CASE WHEN 的输入表达式。

1.25 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。
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
atrrs_value_set_expr	属性值集合。
attr_exp	属性表达式语法规范。
Equality and inequality	等式与不等式，详情请参见 3.1 关系运算符。
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE 条件过滤时，支持 SQL 通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。CREATE TABLE LIKE 中不支持通配符。

1.26 number

格式

无。

说明

LIMIT 限制输出的行数，只支持 INT 类型。

1.27 num_buckets

格式

无。

说明

分桶的个数，仅支持 INT 类型。

1.28 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。

1.29 partition_col_value

格式

无。

说明

分区列值，即分区字段的值。

1.30 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 ， vaule 为 partition_col_value ， 若存在多个分区字段，每组 key=value 之间用逗号分隔。

1.31 property_name

格式

无。

说明

属性名称，仅支持字符串类型。

1.32 property_value

格式

无。

说明

属性值，仅支持字符串类型。

1.33 regex_expression

格式

无。

说明

模式匹配字符串，支持通配符匹配。在 SHOW FUNCTIONS 中，可利用通配符进行匹配，目前支持“.”与“.*”，其中“.”仅代表一个字符，“.*”代表一个或多个字符。

1.34 result_expression

格式

无。

说明

CASE WHEN 语句中 THEN 语句后的返回结果。

1.35 role_name

格式

无。

说明

角色名称，仅支持字符串类型。

1.36 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，集合类型包含复杂类型中的 ARRAY，MAP，STRUCT 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' = '')

1.37 select_statement

格式

无。

说明

SELECT 基本语句，即查询语句，详见 8.1 SELECT 基本语句。

1.38 separator

格式

无。

说明

分隔符，仅支持 CHAR 类型，可用户自定义，如逗号、分号、冒号等。

1.39 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。

1.40 table_comment

格式

无。

说明

对表的描述，仅支持字符串类型。

1.41 table_name

格式

无。

说明

表名称，仅支持字符串类型，名称长度不能超过 128 个字节。

1.42 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。

1.43 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。

1.44 user_name

格式

无。

说明

用户名称，仅支持字符串类型。

1.45 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。

1.46 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。

1.47 when_expression

格式

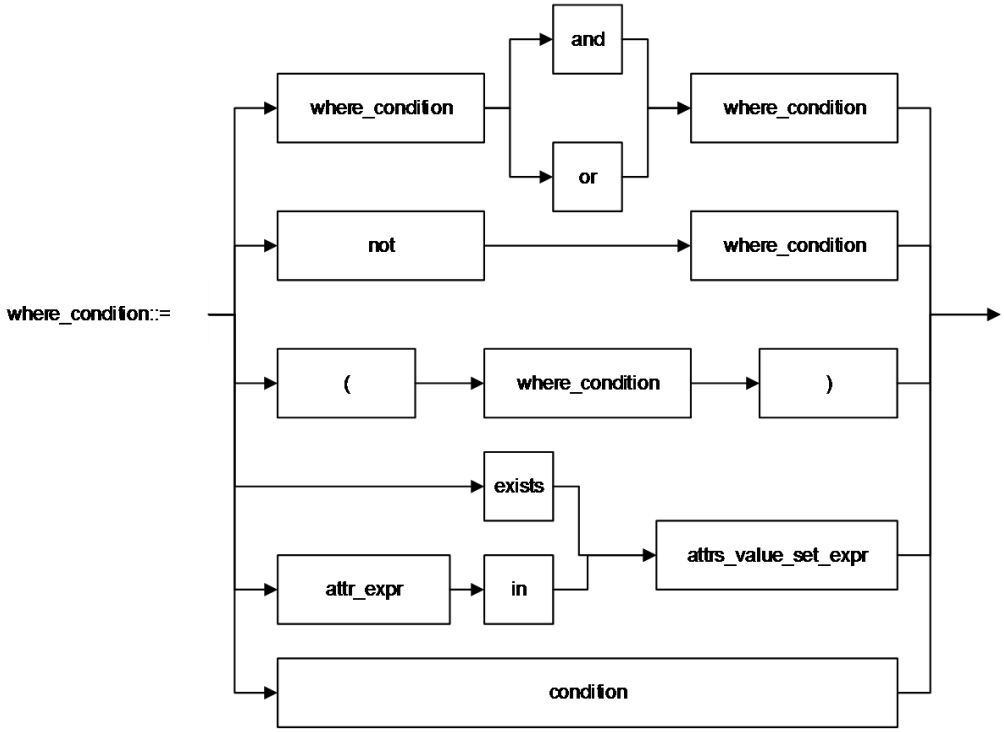
无。

说明

CASE WHEN 语句的 when 表达式，与输入表达式进行匹配。

1.48 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。

语法	描述
)	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。

1.49 window_function

格式

无。

说明

分析窗口函数，详情请参见 4.5 分析窗口函数。

2 数据类型

2.1 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数据表中。数据表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。

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

2.2 原生数据类型

Spark SQL 支持原生数据类型，请参见表 2-1。

表2-1 原生数据类型

数据类型	描述	存储空间	范围
INT	有符号整数	4 字节	-2147483648～2147483647
STRING	字符串	-	-
FLOAT	单精度浮点型	4 字节	-
DOUBLE	双精度浮点型	8 字节	-
DECIMAL	固定有效位数和小数位数的数据类型	-	-

数据类型	描述	存储空间	范围
BOOLEAN	布尔类型	-	TRUE/FALSE
SMALLINT	有符号整数	16 字节	-32768-32767
TINYINT	有符号整数	8 字节	-128-127
BIGINT	有符号整数	64 字节	-9223372036854775808～9223372036854775807
TIMESTAMP	时间戳，表示日期和时间。格式：YYYY-MM-DD HH:MM:SS.ffffff，可达到小数点后 6 位	-	-
BINARY	二进制数	-	-
VARCHAR	可变长度的字符	-	-
DATE	日期类型，描述了特定的年月日，以 YYYY-MM-DD 格式表示，例如 2014-05-29	-	DATE 类型不包含时间，所表示日期的范围为 0000-01-01 to 9999-12-31

INT

有符号整数，存储空间为 4 字节，-2147483648～2147483647，在 NULL 情况下，采用计算值默认为 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。

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。

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

BOOLEAN

布尔类型，包括 TRUE 与 FALSE。

SMALLINT

有符号整数，存储空间为 16 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 8 字节，范围为-128～127。当为 NULL 情况下，采用计算值默认为 0。

BIGINT

有符号整数，存储空间为 64 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：

字符串：必须具有"YYYY-MM-DD HH:MM:SS[.ffffff]"格式。小数点后精度可选。

BINARY

二进制数，如 00000101。

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。

DATE

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

表2-2 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

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

2.3 复杂数据类型

Spark SQL 支持复杂数据类型，如表 2-3 所示。

表2-3 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。

ARRAY 示例

创建数据表“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

创建数据表“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

创建数据表“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
   ```

3 运算符

3.1 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。

Spark SQL 提供的关系运算符，请参见表 3-1。

表3-1 关系运算符

运算符	返回类型	描述
A = B	BOOLEAN	若 A 与 B 相等，返回 TRUE，否则返回 FALSE。用于做赋值操作。
A == B	BOOLEAN	若 A 与 B 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

运算符	返回类型	描述
		或 B 为 NULL，则返回 NULL。
A <= B	BOOLEAN	若 A 小于或者等于 B，则返回 TRUE，否则返回 FALSE。若 A 或 B 为 NULL，则返回 NULL。
A > B	BOOLEAN	若 A 大于 B，则返回 TRUE，否则返回 FALSE。若 A 或 B 为 NULL，则返回 NULL。
A >= B	BOOLEAN	若 A 大于或者等于 B，则返回 TRUE，否则返回 FALSE。若 A 或 B 为 NULL，则返回 NULL。
A [NOT] 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 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。

3.2 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。Spark SQL 所支持的算术运算符如表 3-2 所示。

表3-2 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。

运算符	返回类型	描述
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。
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）。

3.3 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 3-3，表中的 A 和 B 代表逻辑表达式。

表3-3 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	BOOLEAN	若 A 与(val1, val2, ...)中任意值都不相等则返回 TRUE，

运算符	返回类型	描述
(val1, val2, ...)		否则返回 FALSE。
[NOT] 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。

4 内置函数

4.1 数学函数

Spark SQL 所支持的数学函数如表 4-1 所示。

表4-1 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

函数	返回值	描述
		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。
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	返回某数的乘幂。
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) hex(BINARY a)	STRING	将整数或字符转换为十六进制格式。
unhex(STRING a)	BINARY	十六进制字符转换由数字表示的字符。
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，如 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	返回给定角度的正弦值。

函数	返回值	描述
asin(DOUBLE a), asin(DECIMAL a)	DOUBLE	返回 X 的反正弦。
cos(DOUBLE a), cos(DECIMAL a)	DOUBLE	返回给定角度的余弦值。
acos(DOUBLE a), acos(DECIMAL a)	DOUBLE	返回 X 的反余弦。
tan(DOUBLE a), tan(DECIMAL a)	DOUBLE	返回给定角度的正切值。
atan(DOUBLE a), atan(DECIMAL a)	DOUBLE	返回 X 的反正切。
degrees(DOUBLE a), degrees(DECIMAL a)	DOUBLE	返回弧度所对应的角度。
radians(DOUBLE a), radians(DOUBLE 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, INTb)	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, INTb) shiftright(BIGINT a, INTb)	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|INTa, 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。

4.2 日期函数

Spark SQL 所支持的日期函数如表 4-2 所示。

表4-2 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”。
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，范围在 1000 到 9999。
quarter(date/timestamp/string)	INT	返回该 date/timestamp 所在的季度，如 quarter('2015-04-01')=2。
month(string date)	INT	返回指定时间的月份，范围为 1 至 12 月。
day(string date) dayofmonth(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。

函数	返回值	描述
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	返回指定日期所在一年中的星期号，范围为 0 到 53。
datediff(string enddate, string startdate)	INT	两个时间参数的日期之差。
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(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(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 为一周内的星期表示（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(date1, date2)	DOUBLE	返回 date1 与 date2 之间的月份差。
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'。

4.3 字符串函数

Spark SQL 所支持的字符串函数如表 4-3 所示。

表4-3 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。
base64(binary bin)	STRING	将二进制参数进行 base64 编码。
concat(string|binary A, string|binary 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	链接多个字符串，字符串之间以指定的分隔符分开。
concat_ws(string SEP, array<string>)	STRING	链接字符串数组中的字符串，字符串之间以指定的分隔符分开。
decode(binary bin, string charset)	STRING	用 charset 的编码方式对 bin 进行解码。
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。
format_number(number x, int d)	STRING	格式化 x，d 代表小数部分，当 d 为 0 时，表示去除小数部分。
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。
in_file(string str, string filename)	BOOLEAN	如果 str 在 filename 中以正行的方式出现，返回 true。
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 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，给定字符串长度小于指定长度时，由指定字符从左侧填补。
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 URL 指定的部分。parse_url(‘http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1′, ′ HOST′) 返回：′ facebook.com′ 。
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) returns ′ 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，给定字符串长度小于指定长度时，由指定字符从右侧填补。
rtrim(string A)	STRING	删除字符串右边的空格，其他的空格保留。
sentences(string str, string lang, string locale)	ARRAY<ARRAY< STRING >>	将字符串中内容按语句分组，每个单词间以逗号分隔，最后返回数组。 例如 sentences(‘Hello there! How are you?’) 返回：((“Hello”, “there”), (“How”, “are”, “you”))
space(int n)	STRING	返回指定数量的空格。
split(string str, string pat)	ARRAY	将字符串转换为数组。
str_to_map(text[, delimiter1, delimiter2])	MAP< STRING, STRING >	将输入文本按照分割符转化为 key-value 的键值对，默认情况下 delimiter1 为',' delimiter2 为'='。
substr(string|binary A, int start) substring(string|binar	STRING	从文本字符串中指定的起始位置后的字符。

函数	返回值	描述
y A, int start)		
substr(string|binary A, int start, int len) substring(string|binary A, int start, int len)	STRING	从文本字符串中指定的位置指定长度的字符。
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。
trim(string A)	STRING	删除字符串两端的空格，字符之间的空格保留。
unbase64(string str)	BINARY	将 base64 编码的字符串转换为二进制数。
upper(string A) ucase(string A)	STRING	将文本字符串转换成字母全部大写形式。
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。

4.4 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 4-4 所示。

表4-4 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。

函数	返回值类型	描述
sum(col), sum(DISTINCT col)	DOUBLE	求和。
avg(col), avg(DISTINCT col)	DOUBLE	求平均值。
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(BIGINT col, array(p_1[, p_2]...))	ARRAY<DOUBLE>	返回数值区域的一组百分比值分别对应的数值点。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），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。
percentile_approx(DOUBLE col, array(p_1[, p_2]...) [, B])	ARRAY<DOUBLE>	与 percentile_approx(DOUBLE col, p [, B])相同，但接受 array 类型的输入参数，返回 array 类型的输出参数。
histogram_numeric(col, b)	ARRAY<STRUCT {'x','y'}>	使用 b 个非均匀间隔的箱子计算组内数字列的柱状图（直方图），输出的数组大小为 b，double 类型的(x,y)表示直方图的中心和高度。
collect_set(col)	ARRAY	返回无重复记录。
collect_list(col)	ARRAY	返回有重复记录的列表。
ntile(INT x)	INT	将一个有序分区拆分成 x 个桶，并将桶号添加到分区的每一行中。

4.5 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 4-5 中所示的函数。窗口是由一个 8.11 OVER 子句定义的多行记录，窗口函数作用于一个窗口。

表4-5 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。
last_value(col)	参数的数据类型	返回结果集中某列最后条数据的值。
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。
dense_rank()	INT	计算值在分区中的排位。对于并列的值，DENSE_RANK 函数不会在排名序列中留出空位。
ntile(INT x)	INT	将一个有序分区拆分成 x 个桶，并将桶号添加到分区的每一行中。
cume_dist()	DOUBLE	计算某个值在一组行中的相对位置。
percent_rank()	DOUBLE	为窗口的 ORDER BY 子句所指定列中的值返回秩，但以介于 0 和 1 之间的小数形式表示，计算方法为 (RANK - 1)/(- 1)。

5 管理操作语句

5.1 角色管理

5.1.1 查询角色

功能描述

查询角色。

语法格式

SHOW [CURRENT] ROLES;

注意事项

- 当用户拥有 ADMIN 角色且当前处于该角色时，才可以执行 ***SHOW ROLES*** 查看所有用户对应的所有角色，非 ADMIN 角色不能执行该语句。

  通过 ***SET ROLE ADMIN*** 可以切换为 ADMIN 角色。
- 语句中添加 CURRENT 关键字将查看当前用户所处的角色。如果不添加 CURRENT 关键字，则可以查看所有用户所对应的所有角色。

示例

查看当前用户所处的角色。

SHOW CURRENT ROLES;

5.1.2 授予角色

功能描述

授予角色。

语法格式

SET ROLE role_name;

注意事项

- 将用户当前持有的角色设置为 role_name，若用户不拥有 role_name 所指定的角色将出错。
- 任意用户都可执行该语句。

示例

将当前用户设置为 ADMIN 角色。

SET ROLE ADMIN;

6 数据定义语句

6.1 创建数据库

功能描述

创建数据库。

语法格式

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

注意事项

- 若要指定数据库的 LOCATION，创建数据库的用户必须拥有 hive 访问权限，否则会出错。
- DATABASE 与 SCHEMA 两者没有区别，可替换使用，建议使用 DATABASE。

示例

若 testdb 数据库不存在，则创建数据库 testdb。

CREATE DATABASE IF NOT EXISTS testdb;

6.2 修改数据库

功能描述

修改数据库的相关属性及 OWNER。

语法格式

- ***ALTER {DATABASE | SCHEMA} db_name SET DBPROPERTIES (property_name=property_value, ...);***
- ***ALTER {DATABASE | SCHEMA} db_name SET OWNER USER user_name;***

注意事项

- 若要指定数据库的 LOCATION，则创建数据库的用户必须拥有 hive 访问权限，否则会出错。
- DATABASE 与 SCHEMA 两者没有区别，可替换使用，建议使用 DATABASE。
- 通过 ***ALTER {DATABASE | SCHEMA} db_name SET DBPROPERTIES (property_name=property_value, ...)***来设置数据库的属性，通过 ***ALTER {DATABASE | SCHEMA} db_name SET OWNER USER user_name*** 来修改数据库的 OWNER，目前支持修改 USER 用户级别。

示例

将数据库 testdb 的属性 creator 修改为 Jack。

ALTER DATABASE testdb SET DBPROPERTIES("creator"="Jack");

将数据库 testdb 的 OWNER 用户修改为 admin。

ALTER DATABASE testdb SET OWNER USER admin;

6.3 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];

注意事项

- 使用 RESTRICT 关键字时，若数据库非空，删除数据库时将出错，使用 CASCADE 关键字时，不管数据库是否为空，都将删除数据库，默认情况下是使用 RESTRICT。
- DATABASE 与 SCHEMA 两者没有区别，可替换使用，建议使用 DATABASE。

示例

若存在 testdb 数据库，则删除数据库 testdb。

DROP DATABASE IF EXISTS testdb;

6.4 切换数据库

功能描述

切换数据库。

语法格式

- ***USE db_name;***
- ***USE DEFAULT;***

注意事项

可以通过 ***select current_database()***语句校验是否成功切换数据库。

示例

切换到数据库 testdb。

USE testdb;

切换到默认数据库。

USE DEFAULT;

6.5 创建表

6.5.1 CREATE TABLE

功能描述

创建基本表。

语法格式

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name [(col_name data_type [COMMENT col_comment], ...)] [COMMENT table_comment] [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)] [[ROW FORMAT row_format] [STORED AS file_format] | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]] [LOCATION hdfs_path] [TBLPROPERTIES (property_name=property_value, ...)] [AS select_statement];

注意事项

- 本章节为创建表语法整体说明，针对具体建表应用场景的限制和说明将在后续对应章节中详细描述。
- 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。
- 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。
- 在创建表时，可以先通过 USE db_name 命令指定创建表所在的数据库，也可以在创建过程中指定 db.name 参数来创建。
- TBLPROPERTIES 子句允许用户给表添加 key/value 的属性。
- 创建表时要声明列名及对应的数据类型，数据类型可以是原生类型或者复杂类型。
- TEMPORARY 关键字表示创建的表为一张临时表，当 Spark 退出后将自动清除临时表。
- STORED BY 可指定自定义存储类型，并可以通过借助 WITH SERDEPROPERTIES 子句指定 key/value 对将 SerDe 属性与表相关联。例如：

  CREATE TABLE test_stored_by(key INT, value STRING) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val") ;

示例

创建一张名为 student 的表，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);

6.5.2 CREATE TABLE LIKE

功能描述

克隆表，由一张表创建另外一张表。

语法格式

CREATE [TEMPORARY] [EXTERNAL] [db_name.]TABLE [IF NOT EXISTS] [db_name.]table_name LIKE table_reference [LOCATION hdfs_path];

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。
- 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。
- 在创建表时，可以先通过 ***USE db_name*** 命令指定创建表所在的数据库，也可以在创建过程中指定 db.name 参数来创建。
- LIKE 后面所依据的表应该为已经存在的表，否则将会出错。
- CREATE TABLE … LIKE 将创建与原表具有相同 schema 的一张空表，可以利用 DESCRIBE FORMATTED table_name 来查看表的描述，两张表除了表名和创建时间不同外，其他都相同。
- 指定 hdfs_path 可以更改所建表的默认 HDFS 路径。
- 语句中添加 TEMPORARY 关键字表示创建的表为一张临时表，当 Spark 退出后将自动删除。

示例

由已经存在的 student 表创建一张新表，并命名为 empty_student，empty_student 为一张空表，表结果信息与 student 一样。

CREATE TABLE empty_student LIKE student;

6.5.3 CREATE TABLE AS SELECT

功能描述

由查询结果创建另一张表。

语法格式

CREATE [EXTERNAL] [db_name.]TABLE [IF NOT EXISTS] [db_name.]table_name AS select_statement;

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。
- 表名及列名的描述仅支持字符串常量，不区分大小写。
- 在创建表时，可以先通过 ***USE db_name*** 命令指定创建表所在的数据库，也可以在创建过程中指定 db.name 参数来创建。
- CREATE TABLE…AS SELECT 以 select_statement 的执行结果来创建新表，在这张表的内容填写完成之前，其他用户都看不到这张表。所以其他用户只能看到填写完成的表，否则看不到表。
- CREATE TABLE…AS SELECT 语句包含两个部分，CREATE 部分和 SELECT 部分。SELECT 部分可以是任何 SELECT 语句，而 CREATE 部分则将 SELECT 部分得到的查询结果填入新创建的表中。
- 使用 CREATE TABLE…AS SELECT 时可以直接定义表中的列名及列类型，所定义的列的数量与类型都必须与 SELECT 语句的返回结果一致。
- CREATE TABLE…AS SELECT 创建的表不可以分区、分桶，也不能是临时表。

示例

以 student 表中分数为 85.5 的学生查询结果创建新表，并命名为 par_student。

CREATE TABLE par_student AS SELECT * FROM student WHERE score = 85.5;

6.5.4 创建分区表

功能描述

创建分区表。

语法格式

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name [(col_name data_type [COMMENT col_comment], ...)] [COMMENT table_comment] [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)];

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。
- 表名及列名的描述仅支持字符串常量，不区分大小写。
- 在创建表时，可以先通过 USE db_name 命令指定创建表所在的数据库，也可以在创建过程中指定 db.name 参数来创建。
- 在逻辑上，分区表与未分区表没有任何区别，在物理上，表中的数据按照分区存放，一个分区对应一个目录。
- 分区表在创建完成后，可以通过 ALTER TABLE 来添加或者删除分区。一切对普通列适用的操作都是适用于分区列的，因此分区列与普通列不能重名。对于指定分区的查询语句将仅扫描该分区下的数据信息。

- 创建表时要声明列名及对应的数据类型，数据类型可以是原生类型或者复杂类型。

示例

创建一个名为 student 的分区表，表中包含三个字段 id，name，score，对应的类型分别为 INT，STRING，FLOAT，此外 student 表以 city 字段为分区字段。

CREATE TABLE student (id INT, name STRING, score FLOAT) PARTITIONED BY(city STRING);

6.5.5 创建分桶表

功能描述

创建分桶表。

语法格式

CREATE TABLE [IF NOT EXISTS] [db_name.]table_name CLUSTERED BY (col_name_list) INTO n BUCKETS USING parquet AS select_statement;

注意事项

- 若所要创建的表已经存在将报错，可以通过语句中添加 IF NOT EXISTS 避免该错误。
- 表名及列名的描述仅支持字符串常量，不区分大小写。
- 在创建表时，可以先通过 USE db_name 命令指定创建表所在的数据库，也可以在创建过程中指定 db.name 参数来创建。
- 表的分桶是让某些任务（比如取样，map join 等）运行得更快，表的分桶在建表时完成。
- INTO n BUCKETS 中的 n 指定桶的数量。
- 和分区列不同，分桶列必须是表中的列，因此 col_name_list 中的字段必须包含在 select_statement 的查询结果字段中，如示例中 SELECT 语句的查询结果中必须包含分桶字段 id，否则会出错。
- Spark SQL 支持单层和多层分桶。
- 建议在 select_statement 中加入 DISTRIBUTE BY 先进行分桶操作，分桶字段与创建分桶表的字段相同。

示例

根据 student 表查询结果，按照 id 字段进行分桶来创建 testbucket 分桶表。

CREATE TABLE testbucket CLUSTERED BY(id) INTO 2 BUCKETS USING parquet AS SELECT id, name, score FROM student;

6.6 修改表

6.6.1 修改表名

功能描述

修改表名。

语法格式

ALTER TABLE [db_name.]table_name RENAME TO [db_name.]table_name;

注意事项

- 所要修改的表必须是已经存在的表，且新的表名不能与数据库下已有的表名相同，否则会出错。
- 表名不区分大小写。
- 通过该语句可以将一个表从一个数据库转移到另一个数据库。

示例

将 student 表重命名为 new_student。

ALTER TABLE student RENAME TO new_student;

将 db1 中的表 table1 转移到 db2 中的表 table2。

ALTER TABLE db1.table1 RENAME TO db2.table2;

6.6.2 修改属性

功能描述

修改表的属性。

语法格式

ALTER TABLE [db_name.]table_name SET TBLPROPERTIES table_properties;
table_properties:: (property_name = property_value, property_name = property_value, ...);

注意事项

- 所要修改属性的表必须是已经存在的表，否则会出错。
- 可通过 SET TBLPROPERTIES 以 key/value 的形式修改或者添加表的属性，若表中不存在当前所设置的属性则会添加该属性，若已经存在该属性将会覆盖原来的值。
- 可以通过 USE db_name 命令指定数据库，也可在修改过程中指定 db_name 参数。

示例

修改 student 表中的 comment 属性为 this is a student table。

ALTER TABLE student SET TBLPROPERTIES ("comment" = "this is a student table");

6.6.3 修改分区

6.6.3.1 添加分区

功能描述

添加分区。

语法格式

ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_specs [LOCATION 'hdfs_path'] PARTITION partition_specs [LOCATION 'hdfs_path']...;

注意事项

- 所要添加分区的表必须是已经存在的表，否则会出错。
- 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。
- 若所添加的分区已经存在，否则将出错，可通过语句中添加 IF NOT EXISTS 避免该错误。
- 在添加分区时若指定 HDFS 路径，则该 HDFS 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 'hdfs_path']之间用空格隔开。例如：*PARTITION partition_specs [LOCATION 'hdfs_path'] PARTITION partition_specs [LOCATION 'hdfs_path']*。

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 '/user/hive/warehouse/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');

6.6.3.2 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;

注意事项

- 所要重命名分区的表必须是已经存在的表，否则会出错。
- 所要重命名的分区必须是已经存在的分区，否则会出错，所修改的新分区不能与已存在的分区重名，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。

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');

6.6.3.3 交换分区

功能描述

交换分区。

语法格式

ALTER TABLE table_name EXCHANGE PARTITION partition_specs WITH TABLE table_name;

注意事项

- 所参与交换分区的表必须是已经存在的表，否则会出错。
- 若表是按照多字段分区，所指定的分区信息应该比较清晰，即不存在二义性，否则将出错。
- 该语句将实现分区交换，将第二张表中的分区移动第一张表下，因此所交换的分区应是在第二张表中存在，且在第一张表中不存在，否则会出错。
- 交换分区后第二张表中的分区将删除，第一张表中的分区将新增。
- 两张表的 SCHEMA 要相同，否则会报错。

示例

将表 student_2 中的分区 dt='2010-10-10',city='Hangzhou'移动到表 student_1 下面。

ALTER TABLE student_1 EXCHANGE PARTITION (dt='2010-10-10',city='Hangzhou') WITH TABLE student_2;

6.6.3.4 删除分区

功能描述

删除分区。

语法格式

ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_specs, PARTITION partition_specs,...;

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。
- 该语句将删除指定分区，可以指定详细的分区信息，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。

示例

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

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

6.6.3.5 修改表/分区位置

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 hdfs_path;

注意事项

- 所要修改位置的表或分区必须是已经存在的，否则会出错。
- 该语句将修改表或者分区的位置，需要注意的是所指定的新的 HDFS 路径，该路径必须是已经存在的，否则会出错。
- 所指定的 HDFS 路径必须是绝对路径，否则会出错。

示例

将 student 表的分区 dt='2008-08-08',city='Hangzhou'的 HDFS 路径设置为 'hdfs://hacluster/user/hive/warehouse/student/dt=2008-08-08/city=Hangzhou_bk'。

ALTER TABLE student PARTITION(dt='2008-08-08',city='Hangzhou') SET LOCATION 'hdfs://hacluster/user/hive/warehouse/student/dt=2008-08-08/city=Hangzhou_bk';

6.6.4 修改列

功能描述

修改表中的列。

语法格式

ALTER TABLE [db_name.]table_name CHANGE [COLUMN] col_name col_name data_type [COMMENT col_comment] [FIRST | AFTER col_name];

注意事项

- 所要修改的表必须是已经存在的表，否则会出错。
- 第一个 col_name 指旧的列名，第二个 col_name 指新的列名。
- 修改表的列可以修改列名、列的类型以及列的位置，旧的列名必须是已经存在的且新的列名不能和已有列名重名，否则会出错。
- AFTER 关键字可以修改列的位置，使其位于某列之后，FIRST 关键字则将所修改的列的位置置于最前列。
- COLUMN 关键字可选择性添加，对结果无影响。
- 不能采用该种方式对分区表中的分区列进行修改。

示例

将列名为 id 的字段命名为 id1，修改其类型为 STRING，并将其置于列名为 name 的字段后面。

ALTER TABLE student CHANGE id id1 STRING AFTER name;

6.7 删除表

功能描述

删除表。

语法格式

DROP TABLE [IF EXISTS] table_name [PURGE];

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。
- DROP 既能删除托管表也能删除外表，在删除托管表时将删除数据及相应的元数据信息，在删除外表时仅删除外表的元数据信息，不会删除存放在 HDFS 上面的数据。

- 可以通过 ***USE db_name*** 切换到相应的数据库下再删除相应的表，不能在删除表时直接指定该表所在的数据库名。
- DROP TABLE 时加上 PURGE 关键字，表数据将不会放置到相应用户的.Trash/Current 目录下面，这将导致误删除表的数据恢复不了。

示例

将在当前所在数据库下删除名为 student 的表。

DROP TABLE IF EXISTS student;

6.8 清空表

功能描述

清空表。

语法格式

TRUNCATE TABLE table_name [PARTITION partition_specs];

注意事项

- 所要清空的表必须是已经存在的表，否则会出错。
- TRUNCATE TABLE 删除表或者分区中的数据，但不能删除表或者分区中的元数据，不能对外表进行 TRUNCATE 操作，只能用于托管表。
- 可以通过 ***USE db_name*** 命令指定数据库，也可在清空过程中指定 db.name 参数。

示例

将一张名为 student 的表中的数据清空。

TRUNCATE TABLE student;

将清空 student 表中 name="jack "的分区中的数据。

TRUNCATE TABLE student PARTITION (name="jack ");

6.9 创建视图

功能描述

创建视图。

语法格式

CREATE VIEW view_name [(col_name，col_name, ...)] AS select_statement;

注意事项

- 所要创建的视图必须是当前数据库下不存在的，否则会出错。
- 创建视图时，指定列名与列值类型不是必须的，CRETE VIEW 给查询结果创建一个快捷方式，不会将查询结果写入磁盘，支持 CREATE VIEW AS SELECT … UNION SELECT。
- 若在创建视图时，指定字段，那 select_statement 的查询结果的字段数必须与之相同，否则会报错。

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;

6.10 修改视图

6.10.1 修改视图属性

功能描述

修改视图属性。

语法格式

ALTER VIEW [db_name.]view_name SET TBLPROPERTIES view_properties;
view_properties:: (property_name = property_value, property_name = property_value, ...);

注意事项

- 所要修改属性的视图必须是在该数据库下存在，否则会出错。
- 可以通过 ***USE db_name*** 命令指定数据库，也可在修改过程中指定 db_name 参数。
- 可通过 SET TBLPROPERTIES 以 key/value 的形式修改或者添加视图的属性，若视图中不存在当前所设置的属性则会添加该属性，若已经存在该属性将会覆盖原来的值。

示例

将视图 student_view 中的属性 creator 修改为 me。

ALTER VIEW student_view SET TBLPROPERTIES ("creator" = "me");

6.10.2 ALTER VIEW AS SELECT

功能描述

由查询结果修改视图。

语法格式

ALTER VIEW [db_name.]view_name AS select_statement;

注意事项

- 所要修改的视图必须是在该数据库下存在，否则会出错。
- 可以利用 AS SELECT 的方式创建视图，将常用查询的结果保存在视图中。

示例

利用 student 表中 id 等于 4 的查询结果修改 student_view 中的内容。

ALTER VIEW student_view AS SELECT * FROM student WHERE id = 4;

6.11 删除视图

功能描述

删除视图。

语法格式

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

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

示例

删除名为 student_view 的视图。

DROP VIEW student_view;

6.12 创建函数

6.12.1 创建临时函数

功能描述

创建临时函数。

语法格式

CREATE TEMPORARY FUNCTION function_name AS class_name;

注意事项

- 所创建的临时函数只能在当前的 session 中使用。
- 在创建临时函数前，需要添加包含创建相关类的 jar 包，可以在启动 spark-sql 的时候添加 jar 包，如 spark-sql --jars /opt/test/two_udfs.jar，也可在 spark-sql 启动后再添加 jar 包，如 add jar /opt/test/two_udfs.jar，add jar 所指定的路径可以是本地路径也可以是 HDFS 上的路径。
- 创建函数时所指定的类需要指定完整的类所在的包的路径。
- 创建函数时，需要有相应的权限，用户本身需要拥有 ADMIN 权限，并可执行 ***SET ROLE ADMIN*** 赋予权限。
- 允许重复创建同名的临时函数。

示例

创建依赖 org.apache.hadoop.hive.ql.udf.generic.GenericUDFLastDayTest 类的函数 last_day_test 函数。

CREATE TEMPORARY FUNCTION last_day_test AS 'org.apache.hadoop.hive.ql.udf.generic.GenericUDFLastDayTest';

6.12.2 创建永久函数

功能描述

创建永久函数。

语法格式

CREATE FUNCTION [db_name.]function_name AS class_name USING JAR 'hdfs_path';

注意事项

- 所创建的永久函数将在 metastore 登记，不需要在每个 session 重新建临时函数。需要添加的 jar 包，通过 USING 关键字指定。
- 创建函数时所指定的类需要指定完整的类所在的包的路径。
- 该函数可以通过 db_name.function_name 调用，如果函数属于当前数据库，那么可以直接通过 function_name 调用。
- 创建函数时，需要有相应的权限，用户本身需要拥有 ADMIN 权限，可执行 ***SET ROLE ADMIN*** 赋予权限。

示例

基于 two_udfs.jar 中的 org.apache.hadoop.hive.ql.udf.generic.GenericUDFLastDayTest 类创建 last_day_test 函数。

CREATE FUNCTION last_day_test AS 'org.apache.hadoop.hive.ql.udf.generic.GenericUDFLastDayTest' USING JAR 'hdfs:///tmp/two_udfs.jar';

6.13 删除函数

功能描述

删除函数。

语法格式

DROP FUNCTION [IF EXISTS] function_name;

注意事项

- 所要删除的函数必须是已经存在的，否则会出错，可以通过 IF EXISTS 来避免该错误。
- 临时函数与永久函数都可以用上述方式删除，但一般临时函数仅在一个 session 中有效，在另起 spark 后需要重新创建，因此 DROP FUNCTION 一般针对删除永久函数。
- 若所删除的函数是在特定的数据库中创建的，在删除时还需要指定数据库名，若不带数据库名，则删除默认数据库下面的函数。
- 在删除函数时，可以指定该函数所在的数据库，指定数据库名后将删除对应数据库下的函数。

示例

删除默认数据库下的名为 last_day_test 函数。

DROP FUNCTION last_day_test;

6.14 SHOW

6.14.1 SHOW DATABASES

功能描述

查看当前所有的数据库。

语法格式

SHOW {DATABASES | SCHEMAS};

注意事项

DATABASE 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;

6.14.2 SHOW TABLES

功能描述

查看当前数据库下所有的表。

语法格式

SHOW TABLES [IN db_name];

注意事项

SHOW TABLES 将显示当前数据库下的所有表及视图，也可以通过 IN db_name 指定数据库名，显示特定数据库下的表及视图。

示例

查看当前所在数据库中的所有表与视图。

SHOW TABLES;

6.14.3 SHOW COLUMNS

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];

注意事项

- SHOW COLUMNS 查看所有指定表中的列。可以通过 FROM 或者 IN 关键字指定数据库，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。
- 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;

6.14.4 SHOW FUNCTIONS

功能描述

查看所有的函数。

语法格式

SHOW FUNCTIONS [regex_expression];

注意事项

- 通过 ***USE db_name*** 切换到相应的数据库下查看函数，在特定数据库下创建的函数只有切换到对应的数据库才能查看到。
- regex_expression 指模式匹配字符串，支持通配符匹配，该字符串以双引号或单引号扩充起来，目前仅支持.*与.。

示例

查看当前数据库下以 a 开头的所有函数。

SHOW FUNCTIONS "a.*";

6.14.5 SHOW PARTITIONS

功能描述

查看指定表的所有分区。

语法格式

SHOW PARTITIONS table_name [PARTITION partition_specs];

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')。

6.15 DESCRIBE

6.15.1 DESCRIBE DATABASE

功能描述

查看指定数据库的相关信息。

语法格式

DESCRIBE DATABASE [EXTENDED] db_name;

DESCRIBE DATABASE [EXTENDED] DEFAULT;

注意事项

DESCRIBE DATABASE 将返回数据库的相关信息，包括数据库名称、数据库的描述、数据库路径、数据库创建者及其类型等。***DESCRIBE DATABASE EXTENDED*** 除了显示上述信息外，还会额外显示数据库的属性信息。

示例

查看 testdb 数据库的相关信息。

DESCRIBE DATABASE testdb;

查看默认数据库的相关信息。

DESCRIBE DATABASE DEFAULT;

6.15.2 DESCRIBE TABLE

功能描述

查看指定表的相关信息。

语法格式

DESCRIBE [EXTENDED|FORMATTED] table_name;

注意事项

- 若所查看的表不存在，将会出错。
- ***DESCRIBE TABLE*** 返回所有列的列名和列数据类型，***DESCRIBE EXTENDED*** 显示所有表的元数据，通常只在 debug 时用到。***DESCRIBE FORMATTED*** 会用表格形式显示所有表的元数据。

示例

查看 student 表的所有列的列名与列数据类型。

DESCRIBE student;

6.15.3 DESCRIBE COLUMN

功能描述

查看指定表的列的相关信息。

语法格式

DESCRIBE FORMATTED table_name.col_name;

注意事项

所指定的表与列必须是数据库中存在的表与列，否则会出错。

示例

查看 student 表中的 id 字段信息。

DESCRIBE FORMATTED student.id;

6.15.4 DESCRIBE PARTITION

功能描述

查看指定表的分区相关信息。

语法格式

DESCRIBE [EXTENDED|FORMATTED] [db_name.]table_name [col_name] PARTITION partition_specs; partition_specs :: (partition_col_name = partition_col_value, partition_col_name = partition_col_value, ...);

注意事项

- 若所查看的表不存在，将会出错。
- 所查看的表必须是分区表，否则会出错。
- 若表是按照多字段分区的，需要在语句中指定所有的分区字段。
- FORMATTED 关键字可以将结果以表格的形式返回。

示例

查看 student 表中 dt='2008-08-08',city='Shanghai'分区的相关信息。

DESCRIBE FORMATTED student PARTITION(dt='2008-08-08',city='Shanghai');

6.15.5 DESCRIBE FUNCTION

功能描述

查看指定函数的说明及用例。

语法格式

DESCRIBE FUNCTION [EXTENDED] function_name;

注意事项

- DESCRIBE FUNCTION 将返回指定函数的用法，添加 EXTENDED 后返回详细信息。
- 注意在某个数据库下创建的函数，只有该数据库下可见，若要查看函数相关说明，只能通过 ***USE db_name*** 切换到相应的数据库下面才能查看。

示例

查看 upper 的函数说明及用例。

DESCRIBE FUNCTION EXTENDED upper;

7 数据操作语句

7.1 导入数据 LOAD

功能描述

将本地文件或 HDFS 文件数据导入到已有表中。

语法格式

LOAD DATA [LOCAL] INPATH 'file_path' [OVERWRITE] INTO TABLE tablename [PARTITION(partition_col_name = partition_col_value, partition_col_name = partition_col_value ...)];

注意事项

- file_path 是存放数据文件的地址。如果不加上 LOCAL 关键字，file_path 是 HDFS 上的路径。HDFS 上的路径可以是绝对路径也可以是相对路径，file_path 可以指向一个文件也可以指向一个目录。
- file_path 不可以有子目录，若 file_path 不存在，将报错。
- 当 file_path 指向一个文件时，Spark 会将文件移入表内。当 file_path 指向一个目录时，Spark 会将目录下所有的文件移入表内。
- 被导入数据的目标可以是一张表或者一个分区。如果表是分区表，导入数据时必须使用[PARTITION (partition_col_name = partition_col_value, partition_col_name = partition_col_value…)]来指明目标分区。
- 如果加上了 OVERWRITE 选项，那目标表或者分区已有的内容会被导入的文件覆盖；如果不加 OVERWRITE 选项，导入的文件不覆盖已有文件。
- 如果目标表或者分区中存在文件和被导入的文件重名，那么原先的文件会被新文件覆盖。
- 所导入的表应是已经存在的表，否则会报错。

- 为保证数据的正确导入，目标表中的字段分隔符要与被导入文件中的字段分隔符相匹配。

示例

将“本地路径/tmp/student.txt”中的数据导入表 student 中。

LOAD DATA LOCAL INPATH '/tmp/student.txt' INTO TABLE student;

7.2 插入数据 INSERT

功能描述

将查询的结果插入到表或表的分区中。

语法格式

INSERT [OVERWRITE | INTO] TABLE table_name [PARTITION partition_specs] select_statement;

注意事项

- 可以将外表中的数据插入到托管表中，前提是源表需要创建好，同时也可将未分区表的数据插入到分区表中，但需要指定分区信息（partition_specs）。
- INTO 关键字会将查询的结果追加到目标表中，OVERWRITE 则会覆盖原有的数据。
- 目标表必须是已经存在的表，否则会出错。
- select_statement 查询结果的列数必须与目标表的列数相同，否则会出错。
- 当插入分区表时，必须指定分区信息。

示例

将表 student 中的数据插入到表 student_bk 中。

INSERT INTO TABLE student_bk select * from student;

将表 student 中的 id、name 及 dt 插入到分区表 partition_table 中。

INSERT INTO TABLE partition_table PARTITION(dt) SELECT id,name,dt FROM student;

将表 student 中 dt 为"2015-08-22"的 id 与 name 插入到分区表 partition_table 中 dt="2015-08-22"的分区中。

INSERT INTO TABLE partition_table PARTITION(dt="2015-08-22") SELECT id,name FROM student WHERE dt ="2015-08-22";

8 查询语句

8.1 SELECT 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 [TOP num] [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];

注意事项

- 所查询的表必须是已经存在的表，否则会出错。

- TOP num 指定仅输出查询结果的前 num 行。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(所有行会返回)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*（ALL *），否则将会出错。
- WHERE 关键字指定查询的过滤条件，过滤条件中支持算术运算符，关系运算符，逻辑运算符。
- GROUP BY 指定分组的字段，可以单字段分组，也可以多字段分组。
- ORDER BY 用于对查询结果进行排序，col_name_list 为排序依据的字段，ASC 为升序，DESC 为降序，默认情况下为 ASC。
- 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 关键字会对查询结果进行限制，number 参数仅支持 INT 类型。

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;

8.2 过滤

8.2.1 WHERE 过滤子句

功能描述

利用 WHERE 子句过滤查询结果。

语法格式

SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference WHERE where_condition;

注意事项

- 所查询的表必须是已经存在的，否则会出错。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(所有行会返回)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*，否则会出错。
- WHERE 条件过滤，将不满足条件的记录过滤掉，返回满足要求的记录。

示例

将表 student 中，score 在（90，95）之间的记录筛选出来。

SELECT * FROM student WHERE score > 90 AND score < 95;

8.2.2 HAVING 过滤子句

功能描述

利用 HAVING 子句过滤查询结果。

语法格式

SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference [WHERE where_condition] [GROUP BY col_name_list] HAVING having_condition;

注意事项

- 所查询的表必须是已经存在的，否则会出错。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(所有行会返回)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*，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。
- 一般 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

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

SELECT name, max(score) FROM student GROUP BY name HAVING max(score) >95;

8.3 排序

8.3.1 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],...];

注意事项

- 所排序的表必须是已经存在的，否则会出错。
- ASC 关键字指定升序排序，DESC 关键字指定降序排序，默认情况下为 ASC。
- ORDER BY 可以单列排序也可以多列排序。
- ORDER BY 与 SORT BY 的区别是，ORDER BY 作的是全局的排序；SORT BY 一般与 GROUP BY 一起使用，为 PARTITION 级别的局部排序。
- 当 ORDER BY 与 GROUP BY 一起使用时，ORDER BY 后面可以跟聚合函数。

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;

8.3.2 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],...];

注意事项

- 所排序的表必须是已经存在的，否则会出错。
- ASC 关键字指定升序排序，DESC 关键字指定降序排序，默认情况下为 ASC。
- SORT BY 可以单列排序也可以多列排序。
- SORT BY 为 PARTITION 级别的局部排序，ORDER BY 则为全局排序。

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;

8.3.3 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 ,...];

注意事项

- 所排序的表必须是已经存在的，否则会出错。
- CLUSTER BY 会根据指定的字段进行分桶，可以是单字段也可以是多字段，并在桶内进行排序。
- CLSUTER BY 不能指定 ASC（升序）或者 DESC（降序），默认 CLUSTER BY 是降序排序。

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;

8.3.4 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 ,...];

注意事项

- 所排序的表必须是已经存在的，否则会出错。
- DISTRIBUTE BY 会根据指定的字段进行分桶，可以是单字段也可以是多字段，不会在桶内进行排序。
- 可以在 DISTRIBUTE BY 分桶后利用 SORT BY 进行排序。

举例

根据字段 score 对表 student 进行分桶。

SELECT * FROM student DISTRIBUTE BY score;

8.4 分组

8.4.1 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;

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 按列 GROUP BY 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 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会被放在同一个组中，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。

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;

8.4.2 用表达式 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, ...];

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的 GROUP BY 的字段中，表达式支持内置函数，自定义函数等。
- groupby_expression 可以是单字段，多字段，也可以是聚合函数，字符串函数等调用。

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);

8.4.3 GROUP BY 中使用 HAVING 过滤

功能描述

利用 HAVING 子句在表分组后实现过滤。

语法格式

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

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- groupby_expression 可以是单字段，多字段，也可以是聚合函数，字符串函数等调用。
- 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;

8.4.4 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 ROLLUP(col_name_list);

SELECT attr_expr_list FROM table_reference GROUP BY col_name_list WITH ROLLUP;

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 目前支持两种 ROLLUP 语法格式，如上语法格式所示。执行两种格式的语句，结果相同。
- ROLLUP 是 GROUP BY 的扩展，如 ***SELECT a, b, c, SUM(expression) FROM table GROUP BY ROLLUP(a, b, c);***将转换成以下四条查询：
  - ***SELECT a, b, c, sum(expression) FROM table GROUP BY a, b, c;*** //(a, b, c)组合小计
  - ***SELECT a, b, NULL, sum(expression) FROM table GROUP BY a, b;*** //(a, b)组合小计

- ***SELECT a, NULL, NULL, sum(expression) FROM table GROUP BY a;*** //(a)组合小计
- ***SELECT NULL, NULL, NULL, sum(expression) FROM table;*** //总计

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 ROLLUP(group_id, job);

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

8.4.5 CUBE

功能描述

CUBE 生成聚合行、超聚合行、交叉表格行和总计行。可以实现 GROUP BY 字段的所有组合统计，显示某一层次结构的聚合。

语法格式

SELECT attr_expr_list FROM table_reference GROUP BY CUBE(col_name_list);

SELECT attr_expr_list FROM table_reference GROUP BY col_name_list WITH CUBE;

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 目前支持两种 CUBE 语法格式，如上语法格式所示。执行两种格式的语句，结果相同。
- CUBE 是 GROUP BY 的扩展，如 ***SELECT a, b, c, SUM (expression) FROM table GROUP BY CUBE (a,b,c);***将转换成以下八条查询：
  - ***SELECT a, b, c, sum(expression) FROM table GROUP BY a, b, c;*** //(a, b, c)组合小计
  - ***SELECT a, b, NULL, sum(expression) FROM table GROUP BY a, b;*** //(a, b)组合小计
  - ***SELECT a, NULL, c, sum(expression) FROM table GROUP BY a, c;*** //(a, c)组合小计
  - ***SELECT NULL, b, c, sum(expression) FROM table GROUP BY b, c;*** //(b, c)组合小计
  - ***SELECT a, NULL, NULL, sum(expression) FROM table GROUP BY a;*** //(a)组合小计
  - ***SELECT NULL, b, NULL, sum(expression) FROM table GROUP BY b;*** //(b)组合小计

- ***SELECT NULL, NULL, c, sum(expression) FROM table GROUP BY c;*** //(c)组合小计
- ***SELECT NULL, NULL, NULL, sum(expression) FROM table;*** //总计

示例

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

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

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

8.4.6 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 GROUPING SETS(col_name_list);

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 与 CUBE，GROUPING SETS 目前仅支持一种格式。
- GROUPING SETS 是对 GROUP BY 的扩展，如执行 ***SELECT a, b, c, sum(expression) FROM table GROUP BY GROUPING SETS(a,b,c);***将转换为以下三条查询：
  - ***SELECT a, NULL, NULL, sum(expression) FROM table GROUP BY a;*** //(a)组合小计
  - ***SELECT NULL, b, NULL, sum(expression) FROM table GROUP BY b;*** //(b)组合小计
  - ***SELECT NULL, NULL, c, sum(expression) FROM table GROUP BY c;*** //(c)组合小计

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 GROUPING SETS (group_id, job);

8.5 JOIN 连接操作

8.5.1 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;

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。
- 内连接只显示参与连接的表中满足 JOIN 条件的记录。
- JOIN 和 INNER JOIN 在做内连接时用法一样。
- 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);

8.5.2 外连接

8.5.2.1 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;

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。
- 返回左表的所有记录，没有匹配值的记录将返回 NULL。

示例

内连接时利用 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);

8.5.2.2 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;

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。
- 返回右表的所有记录，没有匹配值的记录将返回 NULL。

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);

8.5.2.3 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;

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。
- 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。

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);

8.5.3 隐式连接

功能描述

与内连接功能相同，返回两表中满足 WHERE 条件匹配的结果集，但不用 JOIN 显示的指定连接条件。

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;

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。
- 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。

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;

8.5.4 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 JOIN table_reference ON join_condition;

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。
- join_condition 为连接条件，如果该条件恒成立（比如 1=1），该连接就是笛卡尔连接。连接的结果是由被连接表中所有记录组成的有序数组。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，用法如下：

  SELECT table1.col1, table2.col2, ...FROM table1 CROSS JOIN table2;

  其中，col1 与 col2 分别是 table1 与 table2 中的字段。

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 JOIN course_info ON (1 = 1);

8.5.5 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);

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。
- 左半连接只显示左表中的记录。左半连接可以通过在 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。
- 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);

8.5.6 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;

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

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);

8.6 子查询

8.6.1 WHERE 嵌套子查询

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;

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(返回所有行)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*（ALL *），否则将会出错。
- 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。

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');

8.6.2 FROM 子句嵌套子查询

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;

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(返回所有行)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*，否则将会出错。
- 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;

8.6.3 HAVING 子句嵌套子查询

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);

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(返回所有行)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*，否则将会出错。
- groupby_expression 可以是单字段，多字段，也可以是聚合函数，字符串函数等调用。
- operator 操作符包含等式操作符与不等式操作符，及 IN，NOT IN 操作符。

- 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);

8.6.4 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;

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- ALL 和 DISTINCT 选项，指定重复的行是否返回。如果没有指定，默认是 ALL(返回所有行)。DISTINCT 指定从结果集移除重复的行。ALL 后面只能跟*，否则将会出错。
- 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;

8.7 别名

8.7.1 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;

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。
- table_reference 可以是表，视图或者子查询，可选择 AS 连接 table_reference 和 alias，AS 关键字是否添加不会影响命令执行结果。

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;

8.7.2 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;

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。
- 利用 alias 对 attr_expr 中的字段名称起别名，AS 关键字是否添加不会影响结果。

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";

8.8 集合运算

8.8.1 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;

注意事项

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

8.8.2 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;

注意事项

- INTERSECT 与 UNION 类似，但返回的是多个查询结果的交集，INTERSECT 默认是去重的。
- 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。
- 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 studen _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;

8.8.3 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;

注意事项

- 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。
- 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;

8.9 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;

注意事项

- cte_name 是公共表达式的名字，不要重名，select_statement 是一个完整的 SELECT 语句，sql_containing_cte_name 是包含了刚刚定义的公共表达式的 SQL 语句，注意，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;

8.10 CASE...WHEN

8.10.1 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;

注意事项

- 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。
- 如果没有取值为 TRUE 的 input_expression = when_expression，则当指定 ELSE 子句时，Spark SQL 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;

8.10.2 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;

注意事项

- 如果没有取值为 TRUE 的 Boolean_expression，则当指定 ELSE 子句时，Spark SQL 将返回 else_result_expression；当没有指定 ELSE 子句时，返回 NULL 值。
- boolean_expression 中可以包含子查询，但整个 boolean_expression 表达式返回值只能是布尔类型。

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;

8.11 OVER 子句

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)])

注意事项

- OVER 子句中的三个选项：PARTITION BY 子句，ORDER BY 子句和 WINDOW 子句都为可选项，也可以组合使用。OVER 子句为空则表示窗口为整张表。
- PARTITION BY 子句中可以用一个或多个键分区。和 GROUP BY 子句很像，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。

- ORDER BY 子句决定窗口函数求值的顺序。ORDER BY 子句也可以用一个或多个键排序。排序可以靠 ASC 或者 DESC 决定升序或者降序。当使用 ORDER BY 子句时，窗口可以由 WINDOW 子句指定。如果不指定，默认窗口等同于 ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW，也就是窗口从表或者分区（如果 OVER 子句中用 PARTITION BY 分区）的开头开始到当前行结束。
- WINDOW 子句让用户通过指定一个行区间来定义窗口。
- CURRENT ROW 代表当前行。
- num PRECEDING 定义窗口的下限：窗口从当前行向前数 num 行处开始；UNBOUNDED PRECEDING 代表窗口没有下限。
- num FOLLOWING 定义窗口的上限：窗口从当前行向后数 num 行处结束；UNBOUNDED FOLLOWING 代表窗口没有上限。
- ROWS BETWEEN…和 RANGE BETWEEN…的区别：

  ROW 是物理窗口，即根据 order by 子句排序后，取的前 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 是逻辑窗口，是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 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

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;

9 健康检查语句

9.1 HEALTHCHECK 语句

功能描述

HEALTHCHECK 语句用于检查当前集群的 ThriftServer 的健康状况，该命令不起 Job，且返回为空。

语法格式

HEALTHCHECK

注意事项

- 该命令仅用于健康检查（由外部健康检查程序配合调用），命令能执行成功即意味着 ThriftServer 是健康的。
- 在普通 session 下也可以执行该命令，但该命令不执行任何实际操作，且返回为空。

示例

执行健康检查。

HEALTHCHECK