ODPS SQL 参考手册

修改记录

修改日期	修改说明	作者
2012/9/8	初始版本	张云远
2012/10/09	更新 ALTER TABLE 及 CHAR_MATCHCOUNT 的说明	张云远
2012/11/19	更新 show tables in project ceil/percent_rank/regexp_extract 自动统计的行为(inf,filter) Mapjoin 支持不等值 join 及 or	张云远

目录

1. 简要说明

ODPS SQL 模块适用于数据量大（TB 级别），实时性要求不高的场合，它的每个作业的准备，提交等阶段要花费数秒时间，因此要求每秒处理几千至数万笔事务的业务是不能用 DE 完成的，这种情况请用 OTS。

ODPS SQL 采用的是类似于 SQL 的语法，可以看作是标准 SQL 的一个子集，但不能因此简单的把 DE 等价成一个数据库，它在很多方面并不具备数据库的特征，如事务、主键约束。

2. 数据存储单元

在 ODPS 中，所有的数据都被存储在表中，表是由行和列组成的二维数据结构，每行代表一条记录，不同的列由列名和列数据类型标识，如下表所示：

列

行

SHOP_NAME	REVENUE
SHOP1	1000
SHOP2	1500

表 2-1

为了提高处理效率，可以对表进行分区(partition)，这时数据可以看成由逻辑上独立的许多块组成，在使用数据时如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描，在数据量大的时候这个特性非常有用。大部分的数据可以采用日期，地区等属性进行分区。关于分区的详细介绍，请参见“DDL 语法”一章

分区键

HZ 分区

SH 分区

SHOP_NAME	REVENUE	REGION
SHOP1	1000	HZ
SHOP2	1500	HZ
SHOP3	1200	SH
SHOP4	200	SH

表 2-2

3. 数据类型

类型	说明
BIGINT	8 字节有符号整型
BOOLEAN	布尔型，包括 TRUE/FALSE
DOUBLE	8 字节双精度浮点数
STRING	字符串，
DATETIME	日期类型

表 3-1

备注：

1. BIGINT 的取值范围是-9223372036854775807 ~ 9223372036854775807。请不要使用

-9223372036854775808，这是系统保留值。

2. DOUBLE 的取值范围是 -1.0 * 10^308 ~ +1.0 * 10^308。
3. DATETIME 类型表达的有效日期范围是 0001-01-01 00:00:00 到 9999-12-31 23:59:59

4. 保留字

以下关键字是 ODPS SQL 的保留字，在对表、列或是分区命名时请不要使用，否则会报错。保留字不区分大小写。

-	FROM	WHERE
%	FULL	
&	GROUP	
&&	IF	
(	IN	
)	INTO	
*	IS	
.	JOIN	
/	LEFT	
;	LIKE	
?	LIMIT	
|	LOAD	
||	NOT	
+	NULL	
<	ON	
<=	OR	
<>	ORDER	
=	OUTER	
>	OVER	
>=	PARTITION	
ALL	PARTITIONED	
AND	RANGE	
AS	READ	
BETWEEN	RENAME	
BY	RESOURCE	
CASE	RIGHT	
CLUSTERED	RLIKE	
CREATE	SELECT	
DESC	STRING	
DISTINCT	TABLE	
DROP	THEN	
ELSE	TO	
END	TRUE	
EXISTS	UNION	
EXPORT	VIEW	
FALSE	WHEN	

5. 运算符

5.1. 关系操作符

操作符	说明
A=B	如果 A 或 B 为 NULL，返回 NULL；如果 A 等于 B，返回 TRUE，否则返回 FALSE
A<>B	如果 A 或 B 为 NULL，返回 NULL；如果 A 不等于 B，返回 TRUE，否则返回 FALSE
A<B	如果 A 或 B 为 NULL，返回 NULL；如果 A 小于 B，返回 TRUE，否则返回 FALSE
A<=B	如果 A 或 B 为 NULL，返回 NULL；如果 A 小于等于 B，返回 TRUE,否则返回 FALSE
A>B	如果 A 或 B 为 NULL，返回 NULL；如果 A 大于 B，返回 TRUE，否则返回 FALSE
A>=B	如果 A 或 B 为 NULL，返回 NULL；如果 A 大于等于 B，返回 TRUE，否则返回 FALSE
A IS NULL	如果 A 为 NULL，返回 TRUE，否则返回 FALSE
A IS NOT NULL	如果 A 不为 NULL，返回 TRUE，否则返回 FALSE
A LIKE B	如果 A 或 B 为 NULL，返回 NULL，A 为字符串，B 为要匹配的模式，如果匹配，返回 TRUE，否则返回 FALSE。’%’匹配任意多个字符，’_’匹配单个字符。要匹配%或_需要用转义符表示\\%, _ ‘aaa’ LIKE ‘a__ ‘→TRUE ‘aaa’ LIKE ‘a%’ →TRUE ‘aaa’ LIKE ‘aab’ →FALSE ‘a%b’ LIKE ‘a\\%b’ →TRUE ‘axb’ LIKE ‘a\\%b’ →FALSE
A RLIKE B	如果 A 或 B 为 NULL，返回 NULL，A 是字符串，B 是字符串常量正则表达式，如果 B 为空串会报错退出。如果匹配成功，返回 TRUE，否则返回 FALSE
A IN B	B 是一个集合，如果 A 为 NULL，返回 NULL，如 A 在 B 中则返回 TRUE，否则返回 FALSE

表 5-1

备注：

1. 两个 double 相等当且仅当 15 位有效数字之前都相等，且第 15 位相差不能大于 3。

5.2. 算术操作符

操作符	说明
A + B	如果 A 或 B 为 NULL，返回 NULL；否则返回 A + B 的结果，A 和 B 为数字类型。如果 A 和 B 的类型不同，则自动转为高一级的类型。

A – B	如果 A 或 B 为 NULL，返回 NULL；否则返回 A – B 的结果。A 和 B 为数字类型。如果 A 和 B 的类型不同，则自动转为高一级的类型。
A * B	如果 A 或 B 为 NULL，返回 NULL；否则返回 A * B 的结果。A 和 B 为数字类型。如果 A 和 B 类型不同，则自动转为高一级的类型。
A / B	如果 A 或 B 为 NULL，返回 NULL；否则返回 A / B 的结果。A 和 B 为数字类型，结果为 DOUBLE 类型。
A % B	如果 A 或 B 为 NULL，返回 NULL；否则返回 A 模 B 的结果。
+A	仍然返回 A
-A	如果 A 为 NULL，返回 NULL，否则返回-A

表 5-2

备注：算术运算符(+, -, *, /, %)会做隐式类型转换，当 double、bigint 或 string 做运算时，会转换成 double 类型。

5.3. 位运算操作符

A & B	返回 A 与 B 进行按位与的结果。例如: 1&2 返回 0，1&3 返回 1，NULL 与任何值按位与都为 NULL。A 和 B 必须为 bigint 类型。
A | B	返回 A 与 B 进行按位或的结果。例如: 1|2 返回 3，1|3 返回 3，NULL 与任何值按位或都为 NULL。A 和 B8必须为 bigint 类型。

表 5-3

备注：位运算符不支持隐式转换，只允许 bigint 类型。

5.4. 逻辑操作符

操作符	说明
A AND B	TRUE AND TRUE=TRUE TRUE AND FALSE=FALSE FALSE AND TRUE=FALSE FALSE AND NULL=FALSE NULL AND FALSE=FALSE TRUE AND NULL=NULL NULL AND TRUE=NULL NULL AND NULL=NULL
A OR B	TRUE OR TRUE=TRUE TRUE OR FALSE=TRUE FALSE OR TRUE=TRUE FALSE OR NULL=NULL NULL OR FALSE=NULL TRUE OR NULL=TRUE NULL OR TRUE=TRUE NULL OR NULL=NULL
NOT A	如果 A 是 NULL，返回 NULL；如果 A 是 TRUE，返回 FALSE；如果 A 是 FALSE，返回 TRUE

表 5-4

注：逻辑操作符只允许 boolean 类型参与运算，不支持隐式类型转换。

6. 使用控制台

可以用控制台直接提交 SQL，或者使用“在云端”集成开发工具来使用 SQL 功能，详细用法请参考 ODPS 命令用户手册。

7. DDL 语法

7.1. 创建表(CREATE TABLE)

命令格式：

```
CREATE TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [COMMENT col_comment], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [AS select_statement]

CREATE TABLE [IF NOT EXISTS] table_name
  LIKE existing_table_ name
```

实例：创建表 shop 用于存储所有的店铺信息：

```
create table shop(
shop_name string comment 'each shop has a unique name',
shop_location string comment ' where shop is located',
revenue double comment ' revenue in RMB')
comment ' store shop information';
```

备注：

1. 表名与列名均对大小写不敏感。
2. 在创建表时，如果不指定 IF NOT EXISTS 选项而表名已存在，则返回出错。
3. PARTITIONED BY 指定表的分区字段，如前所述，当利用 partition 字段对表进行分区时，新增分区、更新分区内数据和读取分区数据均不需要做全表扫描，可以提高处理效率。
4. 表名，列名中不能有特殊字符，只能用英文的 a-z,A-Z 及数字和下划线_，且以字母开头，名称的长度不超过 128 字节。
5. 分区名称不可以有双字节字符（如中文），必须是以英文字母 a-z,A-Z 开始后可跟字母数字，名称的长度不超过 128 字节。 如果名称中出现了\t, \n 等特殊字符会导致分区中的数据无法读出，允许的字符包括：
   空格” “，冒号”:”，下划线”_”，美元符”$”，井号”#”，点”.”，感叹号”!”和@
6. 注释内容是长度不超过 1024 字节的有效字符串。

在下面的例子中，创建表 sale_detail 保存销售记录，该表使用销售时间(sale_date)和销售区域(region)作为分区列：

```
create table sale_detail(
```

```
shop_name string,
customer_id string,
total_price double)
partitioned by (sale_date string,region string);
```

7. 目前分区键只能用 STRING 类型。

也可以通过 CREATE TABLE … AS SELECT ..语句创建表，并在建表的同时将数据复制到新表中，如：

```
create table shop_backup as
select * from shop;
```

如果只想复制已有表结构，而不复制任何数据，可以用 CREATE LIKE，用这种方式创建的新表会复制表的结构，列注释，表注释以及定义在表上的统计项。

```
create table shop_tmp like shop;
```

注：正常情况下，我们规定建的分区层次不超过 5 级。

7.2. 创建视图(CREATE VIEW)

命令格式：

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] view_name
  [(col_name [COMMENT col_comment], ...)]
  [COMMENT view_comment]
  [AS select_statement]
```

注：

- 创建视图时，必须有对基础表的读权限。
- 视图只能包含一个有效的 SELECT 语句。
- 视图可以引用其它视图，但不能引用自己，也不能循环引用。
- 视图不可以写。
- 当视图建好以后，如果视图的基础表发生了变更，有可能导致视图无法访问，如基础表被删除。
- 在 VIEW 已经存在时用 CREATE VIEW 会导致异常，这种情况可以用 CREATE OR REPLACE VIEW 来重建 VIEW，重建后 VIEW 本身的权限保持不变。

7.3. 删除表(DROP TABLE)

命令格式：

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

在交互模式下运行 DROP TABLE 命令时，会有提示是否确认删除。

实例：

```
DROP TABLE shop;
```

7.4. 删除视图(DROP VIEW)

命令格式：

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

在交互模式下运行 DROP VIEW 命令时，会有提示是否确认删除。

7.5. 重命名表(RENAME TABLE)

命令格式：

```
ALTER TABLE table_name RENAME TO new_table_name;
```

Rename 操作仅修改表的名字，不改动表中的数据。：

```
ALTER TABLE shop RENAME to store;
```

7.6. 重命名视图(RENAME VIEW)

命令格式：

```
ALTER VIEW view_name RENAME TO new_view_name;
```

7.7. 添加分区(ADD PARTITIONS)

命令格式：

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec

partition_spec:
  : (partition_col = partition_col_value, partition_col = partiton_col_value, ...)
```

如果未指定 IF NOT EXISTS 而同名的分区已存在，则出错返回。如为 sale_detail 表添加一个新的分区，用来存储 2011 年 10 月 10 日杭州地区的销售记录：

```
ALTER TABLE sale_detail ADD IF NOT EXISTS PARTITION
(sale_date= '20111010', region=' hangzhou');
```

7.8. 删除分区(DROP PARTITION)

```
ALTER TABLE table_name DROP [IF EXISTS] partition_spec’;
```

如从表 sale_detail 中删除一个分区，该分区存储的是 2011 年 10 月 10 日杭州地区的销售记录。这个操作不必扫描全表判断销售日期和销售区域。

```
ALTER TABLE sale_detail DROP PARTITION(sale_date=‘20111010’,region=‘hangzhou’);
```

7.9. 修改列名和列注释(CHANGE COLUMN NAME/COMMENT)

```
ALTER TABLE table_name CHANGE [COLUMN] col_old_name col_new_name column_type [COMMENT col_comment]
```

备注：

1. ODPS SQL 仅仅允许修改列名和列注释，不允许修改列数据类型、列位置，也不允许删除已有列。

实例：

```
ALTER TABLE shop CHANGE shop_location shop_city STRING COMMENT ' in which city the shop';
```

7.10. 修改表的注释

```
ALTER TABLE table_name SET COMMENT ‘tbl comment’
```

7.11. 添加列

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1, col_name2 type2…)
```

7.12. 修改列名

```
ALTER TABLE table_name CHANGE COLUMN old_col_name RENAME TO new_col_name;
```

7.13. 修改列、分区注释

```
ALTER TABLE table_name CHANGE COLUMN col_name COMMENT ‘comment’;
```

7.14. 查看表的信息(SHOW)

查看当前数据库中的所有表，默认显示当前 project 中的表。如果指定了 IN project 选项，则显示的是在该 project 中的表

```
SHOW TABLES [IN project_name];
```

查看一张表的所有 partition

```
SHOW PARTITIONS table_name;
```

7.15. 查看表结构信息(DESCRIBE)

```
DESCRIBE|DESC    table_name [partition_name];
```

该命令可以查看表或是表的 partition 的信息，并且可以返回表的大小、表的创建者、表的创建时间、表的上次 meta 修改时间、上次数据修改时间，输出的格式如：

```
desc xdj_pt partition (pt=2011);
+------------------------------------------------------------------------------------------------------------------+
|      PartitionSize:              184                                                                             |
+------------------------------------------------------------------------------------------------------------------+
| CreateTime:                      2012-08-04 13:14:13_____________________________|
| LastDDLTime:                     2012-08-04 13:27:27_____________________________|
| LastModifiedTime:                2012-08-04 13:27:27_____________________________|
```

7.16. 查看 SQL 语句的执行计划

可以用 explain 功能查看 SQL 语句的执行计划，语法是:

```
explain sql_statement;
```

如:

```
explain create table dual_a as select * from dual;
job0 is root job

In Job job0:
root Tasks: M1_Stg1
```

```
In Task M1_Stg1:
    Data source: odps_de_1.dual
    TS:
        SEL: dual.id
            FS: output: odps_de_1.dual_a
```

注：DDL 建表语句不支持 explain 功能。

1. DML 语法

1.1. 更新表中的数据(INSERT OVERWRITE/INTO)

在用 ODPS SQL 处理数据的过程中，INSERT OVERWRITE/INTO 是最常用到的语句，将计算的结果保存一个表中：

```
INSERT OVERWRITE|INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement
FROM from_statement;
```

如，可以用如下操作计算 sale_detail 表中不同地区的销售额：

```
create table sales_by_region(
region string,
revenue double);

INSERT OVERWRITE TABLE sales_by_region
SELECT region, sum(total_price)
FROM sale_detail
GROUP BY region;
```

INSERT INTO 与 INSERT OVERWRITE 的区别是，INSERT INTO 会向表或表的 partition 中追加数据，而 INSERT OVERWRITE 则会在向表或 partition 插入数据前清空表中的原有数据。

1.2. 多路输出(MULTI INSERT)

ODPS SQL 支持在一个语句中插入不同的结果表或者分区

```
FROM from_statement
INSERT OVERWRITE｜INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1
[INSERT OVERWRITE｜INTO TABLE tablename2 [PARTITION ...] select_statement2]
```

如，从 sale_detail 表中同时统计杭州地区的销售额和 2011 年 11 月 10 日的销售额

```
create table sales_by_date(
date2 string,
revenue double);

FROM sale_detail
```

```
INSERT OVERWRITE TABLE sales_by_region
SELECT region, sum(total_price) as revenue
WHERE region=' hangzhou' group by region
INSERT OVERWRITE TABLE sales_by_date
SELECT sale_date,sum(total_price) as revenue
WHERE sale_date=' 20111110' group by sale_date;
```

一般情况下，我们规定一个 query 里面最多可以写 128 路输出。

在一个 MULTI INSERT 中，对于分区表，同一个目标分区不可以出现多次；对于未分区表，该表不能出现多次。

对于同一张表的不同 PARTITION，不能同时有 INSERT OVERWRITE 和 INSERT INTO 操作，否则会造成所有的 INSERT 操作都会作为添加，而不是覆盖。

1.3. 输出到动态分区(DYNAMIC PARTITION)

```
INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...)
select_statement FROM from_statement;
```

在 insert overwrite 到一张分区表时，可以在语句中指定分区的值，也可以用另外一种更加灵活的方式,在 PARTITION 中指定一个分区列名，但不给出值，相应的，在 SELECT 子句中的对应列来提供分区的值，例如：

```
create table sales_by_region (revenue double)
Partitioned by (region string);

INSERT OVERWRITE TABLE sales_by_region PARTITION(region)
SELECT sum(total_price) as revenue, region
FROM sale_detail where sale_date=' 20111110' group by region;
```

我们规定一个 query 里面单个 Instance 最多只能输出 512 个动态分区，一个 job 不可以生成超过 2000 个动态分区，动态生成的分区值不可以为 NULL，否则会引发异常。

1.4. 分区的动态过滤

ODPS SQL 支持在 WHERE 语句中根据子查询的结果来过滤参与运算的分区数量，从而可以减少读取的数据量。

```
SELECT_STATMENT
From FROM_STATEMENT
WHERE PT1 IN (SUBQUERY) AND PT2 IN (SUBQUERY)…
```

使用时的注意事项：

- 子查询应该只返回一列 STRING 类型，不可以 SELECT*或常数。
- 子查询返回的结果行数不能超过 1000 行。

2. SELECT、UNION 和 JOIN 操作

2.1. SELECT 操作

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY order_condition]
[LIMIT number]
```

其中，

1. [GROUP BY col_list]：必须写列的完整表达式，不可用列的 column-alias。
2. [LIMIT number]的 number 是常数。

SELECT 操作从表中读取数据，要读的列可以用列名指定，或者用*代表所有的列，一个简单的 SELECT 如下。ODPS 可以直接将 select 的结果显示在 console 中，格式与 read table 一致。

```
SELECT * FROM shop;
```

或者只读取 shop 的一列 shop_name

```
SELECT shop_name FROM shop;
```

在 WHERE 中可以指定过滤的条件，如：

```
SELECT * FROM shop
WHERE shop_name LIKE ' hangzhou%';
```

在 select 语句的 where 条件中可以指定 partition 范围，这样可以仅仅扫描表的指定部分，避免全表扫描。如下所示，假设 page_views 使用 date 作为 partition column。

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= ' 2008-03-01' AND page_views.date <= ' 2008-03-31';
```

在 table_reference 中支持使用嵌套子查询。

ALL：如果有重复数据行时，返回所有的行，不指定此选项时默认效果和 ALL 相同

DISTINCT ：如果有重复数据行时，只返回一行记录，如：

```
SELECT DISTINCT shop_name FROM shop;
```

AND &OR 可在 where 语句中把两个或多个条件结合起来。AND 是第一个和第二个条件都成立。OR 是第一个和第二个条件只要有一个成立，如：

```
SELECT *FROM shop
WHERE shop_name LIKE 'hangzhou%' OR shop_name LIKE 'suzhou%';
```

GROUP BY：在 SELECT 中包含聚类函数时，用 GROUP BY 指定分类的列，如：

```
SELECT shop_location FROM shop
Group by shop_location;
```

需要注意的是在使用 order by 排序时，NULL 会被认为比任何值都小，这个行为与 mysql 一致，但是与 oracle 不一致。

LIMIT 限制只输出结果中的几行，当使用 SELECT 语句直接从屏幕输出查看结果时，最多只输出 1000 行。

2.2. UNION ALL

将两个或多个 SELECT 操作返回的数据集联合成一个数据集，如果结果有重复行时，会返回所有符合条件的行，不进行重复行的去重处理。需要注意的是：**ODPS SQL 不支持顶级的两个查询结果合并，要改写为一个子查询的形式**，如：

```
SELECT * FROM A UNION ALL SELECT * FROM B;
```

需要改成：

```
SELECT * FROM (SELECT * FROM A UNION ALL SELECT * FROM B) tmp;
```

备注：

1. UNION ALL 操作对应的各个子查询的列个数、名称和类型必须一致。如果列名不一致时，可以使用列的别名加以解决。
2. 上述 query 中的 tmp 别名不能省略。
3. 一般情况下，我们规定最多允许 128 个表的 UNION ALL。

2.3. 子查询

普通的 SELECT 是从几张表中读数据，如 SELECT column_1, column_2 ... FROM table_name，查询的对象也可以是另外一个 SELECT 操作，如：

```
SELECT * FROM (SELECT shop_name FROM shop) a;
```

注意：**子查询必须要有别名。**

在 FROM 子句中，子查询可以当作一张表来使用，与其它的表或子查询进行 JOIN 操作，如：

```
SELECT a.shop_name, b.customer_id,b.total_price FROM
(SELECT shop_name FROM shop) a
JOIN
(SELECT shop_name,customer_id,total_price FROM sale) b
ON a.shop_name=b.shop_name;
```

2.4. JOIN 操作

```
join_table:
    table_reference JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} OUTER JOIN table_reference join_condition

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [alias]
  | table_subquery alias
  | ( table_references )

join_condition:
```

```
    ON equality_expression ( AND equality_expression )*

equality_expression:
    expression = expression
```

LEFT OUTER JOIN 左连接，返回左表中的所有记录，即使在右表中没有记录与它匹配，例如：

```
SELECT * FROM shop a LEFT OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

RIGHT OUTER JOIN 右连接，返回右表中的所有记录，即使在左表中没有记录与它匹配，例如：

```
SELECT *FROM shop a RIGHT OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

FULL OUTER JOIN 全连接, 返回左右表中的所有记录，例如：

```
SELECT * FROM shop a FULL OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

连接条件，普通 JOIN 中只允许 AND 连接的等值条件，MAPJOIN 时允许有不等值连接和使用 OR 连接的条件。

2.5. MAP JOIN HINT

当一个大表和一个或多个小表做 JOIN 时，可以使用 MAPJOIN，性能比普通的 JOIN 要快很多，下面是一个例子。需要注意，**使用 MAPJOIN 时，LEFT OUTER JOIN 的左表必须是大表，RIGHT OUTER JOIN 的右表必须是大表，INNER JOIN 左右表都可以是大表。FULL OUTER JOIN 不能使用 MAPJOIN。**

对于小表的限制，目前定为在解压后在内存里的数据不超过 512M，如果 mapjoin 中指定多个小表，则小表占用的内存总和不得超过 512M。

```
SELECT /* + MAPJOIN(a) */ a.shop_name, b.customer_id,b.total_price
FROM shop a JOIN sale_detail b
ON a.shop_name = b.shop_name;
```

备注：

1. MAP JOIN 支持小表为子查询，最多允许指定 6 张小表。
2. 使用 map join 时需要引用小表或是子查询时，需要引用**别名**。

2.6. CASE WHEN 表达式

有两种不同的 case when 表达式

```
CASE value
   WHEN (_condition1) THEN result1
   WHEN (_condition2) THEN result2
   …
   ELSE resultn
END
```

```
CASE
    WHEN (_condition1) THEN result1
    WHEN (_condition2) THEN result2
    WHEN (_condition3) THEN result3
```

```
    ……
    ELSE       resultn
END
```

CASE WHEN 表达式可以根据表达式的计算结果灵活返回不同的值， 如以下语句根据 SHOP_NAME 的不同情况得出所属区域：

```
SELECT
CASE
  WHEN SHOP_NAME IS NULL THEN 'DEFAULT_REGION'
  WHEN SHOP_NAME LIKE 'HZ%' THEN 'ZJ_REGION'
END AS REGION
FROM SHOP;
```

3. 可用函数

3.1. 数学运算函数

abs

double abs(double number)

bigint abs(bigint number)

用途：返回绝对值，若输入为 NULL，返回 NULL

参数说明

number: double 或 bigint 类型，输入为 bigint 时返回 bigint，输入为 double 时返回 double 类型。若输入为 string 类型会隐式转换到 double 类型后参与运算，其它类型抛异常。

注：当输入 bigint 类型的值超过 bigint 的最大表示范围时，会返回 double 类型，这种情况下可能会损失精度。

示例

- abs(-1) 返回 1
- abs(-9223372036854775808) 返回 9223372036854776000.0，因为 9223372036854775808 超出了 bigint 能表达的最大值范围。

acos

命令格式：

```
double acos(double number)
```

用途：

反余弦函数。

参数说明：

- number: double 类型， -1≤number≤1。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型，值域在 0 ~ π 之间。若 number 为 NULL，返回 NULL。

asin

命令格式：

```
double asin(double number)
```

用途：

反正弦函数。

参数说明：

- number: double 类型， -1≤number≤1。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型，值域在-π/2 ~π/2 之间。若 number 为 NULL，返回 NULL。

atan

命令格式：

```
double atan(double number)
```

用途：

反正切函数。

参数说明：

- number: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型，值域在-π/2 ~π/2 之间。若 number 为 NULL，返回 NULL。

ceil

命令格式：

```
bigint ceil(double value)
```

用途：

返回不小于输入值的最小整数

参数说明：

- value: double 类型，接受 bigint, double 的隐式转换。

返回值：

bigint 类型。任一输入为 NULL，返回 NULL。

conv

命令格式：

```
string conv(string input, bigint from_base, bigint to_base)
```

用途：

进制转换函数

参数说明：

- input: 以 string 表示的要转换的整数值，接受 bigint, double 的隐式转换。
- from_base,to_base，以十进制表示的进制的值，可接受的的值为 2,8,10,16。接受 string,double 的隐式转换。
- 转换过程以 64 位精度工作，溢出时报异常。
- 输入如果是负值，即以'-'开头，报异常。

返回值：

string 类型。任一输入为 NULL，返回 NULL。

注：如果输入的是小数，则会转为整数值后进行进制转换，小数部分会被舍弃。

cos

命令格式：

```
double cos(double number)
```

用途：

余弦函数，输入为弧度值。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

cosh

命令格式：

```
double cosh(double number)
```

用途：

双曲余弦函数。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

cot

命令格式：

```
double cot(double number)
```

用途：

余弦函数，输入为弧度值。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

exp

命令格式：

```
double exp(double number)
```

用途：

指数函数。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

rand

命令格式：

```
double rand(seed)
```

用途：

- 返回 double 类型的随机数，返回值区间是的 0～1.

参数说明

- seed:bigint 类型， 随机数种子， 决定随机数序列的起始值

round

命令格式：

```
round(number, [decimal_places])
```

用途：

四舍五入到指定小数点位置。

参数说明：

- number: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。
- decimal_place: bigint 类型常量，四舍五入计算到小数点后的位置，其他类型参数会引发异常. 如果省略表示四舍五入到个位数。

返回值：

返回四舍五入的结果, double 类型。若 number 或 decimal_places 为 NULL，返回 NULL。

备注：

- decimal_places 可以是负数。负数会从小数点往左开始 round，并且不保留小数部分；如果 decimal_places 超过了整数部分长度，返回 0.

示例：

- round(125.315) 返回 125
- round(125.315, 0) 返回 125
- round(125.315, 1) 返回 125.3
- round(125.315, 2) 返回 125.32
- round(125.315, 3) 返回 125.315
- round(-125.315, 2) 返回 -125.32
- round(null) 返回 null

floor

命令格式：

```
bigint floor(double number)
```

用途：

向下取整，返回比 number 小的整数值。

参数说明：

- number: double 类型，若输入为 string 类型或 bigint 型会隐式转换到 double 类型后参与运算，其他类型抛异常

返回值：

返回向下取整的结果, bigint 类型。若 number 为 NULL，返回 NULL。

示例：

- floor(1.2)=1

- floor(1.9)=1
- floor(0.1)=0
- floor(-1.2)=-2
- floor(-0.1)=-1
- floor(0.0)=0
- floor(-0.0)=0

sin

double sin(double number)

用途：

正弦函数，输入为弧度值。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

sinh

命令格式：

double sinh(double number)

用途：

双曲正弦函数。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

sqrt

double sqrt(double number)

用途：

计算平方根。

参数说明：

- number: double 类型，必须大于 0。小于 0 时引发异常。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

平方根，double 类型。若 number 为 NULL，返回 NULL。

tan

命令格式：

double tan(double number)

用途：

正切函数，输入为弧度值。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

tanh

命令格式：

```
double tanh(double number)
```

用途：

双曲正切函数。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

返回值：

double 类型。若 number 为 NULL，返回 NULL。

trunc

命令格式：

```
trunc(number[, decimal_places])
```

用途：

将输入值截取到指定小数点位置。

参数说明：

- number:double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。
- decimal_places: bigint 类型常量，要截取到的小数点位置，其他类型参数会引发异常，省略此参数时默认到截取到个位数。

返回值：

返回值类型为 double。若 number 或 decimal_places 为 NULL，返回 NULL。

备注：

- truncate 掉的部分补 0。
- decimal_places 可以是负数，负数会从小数点往左开始 truncate，并且不保留小数部分；如果 decimal_places 超过了整数部分长度，返回 0.

示例：

- trunc(125.815) 返回 125
- trunc(125.815, 0) 返回 125
- trunc(125.815, 1) 返回 125.8
- trunc(125.815, 2) 返回 125.81
- trunc(125.815, 3) 返回 125.815
- trunc(-125.815, 2) 返回 -125.81
- trunc(125.815, -1) 返回 120
- trunc(125.815, -2) 返回 100
- trunc(125.815, -3) 返回 0

ln

命令格式：

```
double ln(double number)
```

用途：

返回 number 的自然对数。

参数说明：

- number: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

说明：

若 number 为 NULL 返回 NULL，若 number 为负数或零，则抛异常。

log

命令格式：

```
double log(double base, double x)
```

用途：

返回以 base 为底的 x 的对数。

参数说明：

- base: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。
- x: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

说明：

若 base 和 x 中存在 NULL，则返回 NULL；若 base 和 x 中某一个值为负数或 0，会引发异常；若 base 为 1（会引发一个除零行为）也会引发异常。

pow

命令格式：

```
double pow(double x, double y)
```

用途：

返回 x 的 y 次方，即 x^y。

参数说明：

X: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

Y: double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。

说明：

若 x 或 y 为 NULL，则返回 NULL

3.2. 字符串处理函数

char_matchcount

命令格式：

```
char_matchcount(str1, str2)
```

参数说明：

Str1, str2: string 类型，必须为有效的 UTF-8 字符串，如果对比中发现有无效字符则函数返回负值。

返回值：

bigint

用途：

用于计算 str1 中的每个字符在 str2 中出现的次数总和，任一输入为 NULL 返回 NULL。

chr

命令格式：

```
chr(ascii)
```

参数说明：

ascii: bigint 类型 ascii 值,若输入为 string 类型或 double 类型会隐式转换到 bigint 类型后参与运算，其它类型抛异常。

返回值：

string

用途：

将给定 ASCII 转换成字符，参数范围是 0~255，超过此范围会引发异常。输入值为 NULL 返回 NULL。

concat

命令格式：

```
concat(string A, string B...)
```

参数说明：

A,B 等为 string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

返回值：

string

用途：

返回值是将参数中的所有字符串连接在一起的结果。

备注：

如果没有参数或者某个参数为 NULL，结果均返回 NULL

concat(), concat(null, 'a'), concat('a', null, 'b')返回值都是 NULL

in

命令格式：

```
key in (value1[, value2, value3, …])
```

用途：

查看 key 是否在给定列表中出现

参数说明：

- key 是待判断值，为任意类型。
- value1, value2, …是指定的列表，必须为常量，至少有一项，所有 value 的类型要一致，否则抛异常。关于 in 的隐式类型转换参考*类型转换*一节。

返回值：

key 在列表中出现返回 TRUE，否则为 FALSE

备注：

- 如果 key 为 NULL，结果返回 NULL

示例：

- 'net' IN ('Tech on the net', 'e') would return FALSE
- 'net' IN ('Tech on the’, 'net', 'e')would return TRUE

is_encoding

命令格式：

```
is_encoding(str, from_encoding,to_encoding)
```

用途：

判断输入字符串是否可以从指定的一个字符集转换到另一个字符集，可以用来是否存在“乱码”，通常情况下是指输入字符串无法转换到 gbk 字符集。

参数说明：

- str:string 类型，输入为 NULL 返回 NULL。空字符串则可以被认为属于任何字符集。
- from_encoding,to_encoding: string 类型，指定要判断的字符集。输入为 NULL 返回 NULL。

返回值：

boolean 类型，如果 str 是指定的字符集，则返回 true，否则返回 false

ip2region

命令格式：

```
ip2region(ip, region_level)
```

用途：

根据 ip 地址取相应的城市地址.

参数说明：

- ip:string 类型，其它类型报异常，ip 格式如 192.168.0.1，每段的值在 0-255 之间，非法的 ip 返回 NULL。
- region_level: string 类型，指定返回地址的级别

  country – 国家

  province – 省

  city – 市

  district – 县/区

  school – 学校，如果是教育网的 IP，则返回学校

返回值：

String 类型，地区名称。当给定 IP 找不到相应记录时，返回 NULL。如果有符合条件的多条记录，只返回一条。

注：

此函数依赖于字典文件 ip_region_dict,字典内容由 PE 维护。当前字典中是用”-“表示内容不存在，因此当字典中的值为”-“时，会返回 NULL.

instr

命令格式：

```
instr(string1, string2[, start_position[, nth_appearance]])
```

用途：

计算一个子串在字符串中的位置.

参数说明：

- string1:string 类型，搜索的字符串，若输入为 bigint, double, datetime 类型会隐式转换为

string 后参与运算，其它类型报异常。

- string2:string 类型， 要搜索的子串，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算, 其它类型报异常。
- start_position:bigint 类型，其它类型会抛异常，表示从 string1 的第几个字符开始搜索，默认起始位置是第一个字符位置 1。开始位置如果小于等于 0 会引发异常。
- nth_appearance:bigint 类型，大于 0，表示子串在字符串中的第 n 次匹配的位置，如果 nth_appearance 为其它类型或小于等于 0 会抛异常。

返回值：

string2 在 string1 中的出现的位置。

备注：

- 如果在 string1 中未找到 string2，返回 0.
- 任一输入参数为 NULL 返回 NULL
- 如果 string2 为空串时总是能匹配成功， 因此 instr('abc','') 会返回 1

示例：

- INSTR ('Tech on the net', 'e') 返回 2
- INSTR ('Tech on the net', 'e', 1, 1)返回 2.
- INSTR ('Tech on the net', 'e', 1, 2)返回 11.
- INSTR ('Tech on the net', 'e', 1, 3)返回 14.

keyvalue

命令格式

```
keyvalue(string srcStr，string split1，string split2， string key)
keyvalue(string srcStr， string key) //split1 = ";"，split2 = ":"
```

keyvalue 的功能：

- 将 srcStr 按 split1 分成 key-value 对，按 split2 将 key-value 对分开，返回 key 所对应的 value
- 只有两个参数时，split1 = ' ;', split2 = ' :'
- Split1 或 split2 为 NULL 时，返回 NULL.
- srcStr,key 为 NULL 或者没有匹配的 key 时，返回 NULL
- 如果有多个 key-value 匹配，返回第一个匹配上的 key 对应的 value

示例：

keyvalue("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;", "\;",":","tf") 返回 "21910"

注：如果从 console 输入时字符串中有分号，应该用\;转义。该函数行为是实现成与 taobao hive 里的 UDF 一致，如果在 taobao hive 里的 UDF 行为有改动，请联系产品经理。

length

命令格式：

```
length(string)
```

参数说明：

- string: string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

用途：

返回一个字符串的长度。返回值为整型。若 string 是 NULL 返回 NULL。如果 string 非 UTF-8 编码格式，返回-1。

lengthb

命令格式：

```
lengthb(string)
```

参数说明：

- string: string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

用途：

返回一个字符串的以字节为单位的长度。返回值为整型。若 string 是 NULL 返回 NULL。

md5

命令格式：

```
md5(value)
```

返回值：

输入字符串的 md5 值

参数说明：

value: string 类型，如果输入类型是 bigint 或 double 会隐式转换成 string 类型参与运算，其它类型报异常。输入为 NULL，返回 NULL。

regexp_extract

命令格式：

```
regexp_extract(source, pattern[,occurrence])
```

用途：

将字符串 source 按照 pattern 正则表达式的规则拆分，返回第 occurrence 个 group 指字的字符。

参数说明：

- source: string 类型，待搜索的字符串。
- pattern: string 类型常量，pattern 为空串时抛异常, pattern 中如果没有指定 group，抛异常。
- occurrence: bigint 类型常量，必须>=0，其它类型或小于 0 时抛异常，不指定时默认为 1，表示返回第一个 group。若 occurrence = 0， 返回满足整个 pattern 的子串。

返回值： string 类型，任一输入为 NULL 返回 NULL。

示例

```
select regexp_extract('foothebar', 'foo(.*?)(bar)', 1) from dual;      the
select regexp_extract('foothebar', 'foo(.*?)(bar)', 2) from dual;      bar
select regexp_extract('foothebar', 'foo(.*?)(bar)', 0) from dual;      foothebar
```

regexp_instr

命令格式：

```
regexp_instr(source, pattern[, start_position[,nth_occurrence[,return_option]])
```

返回值：

视 return_option 指定的类型返回匹配的子串在 source 中的开始或结束位置。

参数说明：

- source: string 类型，待搜索的字符串。

- pattern: string 类型常量，pattern 为空串时抛异常。
- start_position:bigint 类型常量,搜索的开始位置。不指定时默认值为 1，其它类型或小于等于 0 的值会抛异常。
- nth_occurrence: bigint 类型常量，不指定时默认值为 1， 表示搜索第一次出现的位置。小于等于 0 或者其它类型抛异常。
- return_option: bigint 类型常量，值为 0 或 1， 其它类型或不允许的值会抛异常。0 表示返回匹配的开始位置，1 表示返回匹配的结束位置。

用途：

返回字符串 source 从 start_position 开始，和 pattern 第 n 次（nth_occurrence）匹配的子串的 起始/结束 位置。 任一输入参数为 NULL 时返回 NULL。

备注：

示例：

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}",3, 2)，返回 14
```

regexp_replace

命令格式：

```
regexp_replace(source, pattern, replace_string, occurrence)
```

返回值：

将 source 字符串中匹配 pattern 的子串替换成指定字符串后返回，当输入 source, pattern, occurrence 参数为 NULL 时返回 NULL，若 replace_string 为 NULL 且 pattern 有匹配，返回 NULL，replace_string 为 NULL 但 pattern 不匹配，则返回原串。

参数说明：

- source: string 类型，要替换的字符串。
- pattern: string 类型常量，要匹配的模式，pattern 为空串时抛异常。
- replace_string:string，将匹配的 pattern 替换成的字符串。
- occurrence: bigint 类型常量，必须大于等于 0，表示将第几次匹配替换成 replace_string，为 0 时表示替换掉所有的匹配子串。其它类型或小于 0 抛异常。
- 当引用不存在的组时，不进行替换。

```
regexp_replace("123.456.7890","([[:digit:]]{3})\\.([[:digit:]]{3})\\.([[:digit:]]{4})","(\\1)\\2-\\3",0)
结果为(123)456-7890
regexp_replace("abcd","(.)","\\1 ",0) 结果为"a b c d "
regexp_replace("abcd","(.)","\\1 ",1) 结果为"a bcd"
regexp_replace("abcd","(.)","\\2",1) 结果为"abcd"，因为 pattern 中只定义了一个组，引用的第二个组不存在，请避免这样使用，引用不存在的组的结果未定义。
regexp_replace("abcd","(.*)(.)$","\\2",0) 结果为"d"
regexp_replace("abcd","a","\\1",0)，结果为”bcd”，因为在 pattern 中没有组的定义，所以\1 引用了不存在的组，请避免这样使用，引用不存在的组的结果未定义。
```

regexp_substr

命令格式：

```
regexp_substr(source, pattern[, start_position[,nth_occurrence]])
```

返回值：

source 中匹配 pattern 指定模式的子串，任一输入参数为 NULL 返回 NULL。

参数说明：

- source: string 类型，搜索的字符串。
- pattern: string 类型常量，要匹配的模型，pattern 为空串时抛异常。
- start_position: 整型常量，必须大于 0。其它类型或小于等于 0 时抛异常，不指定时默认为 1， 表示从 source 的第一个字符开始匹配。
- nth_occurrence:整型常量，必须大于 0，其它类型或小于等于 0 时抛异常。不指定时默认为 1，表示返回第一次匹配的子串。

用途：

返回字符串 source 从 start_position 开始，和 pattern 第 n 次（nth_occurrence）匹配的子串，没有匹配时返回 NULL。

示例：

```
regexp_substr ("I love aliyun very much", "a[[:alpha:]]{5}")，返回 "aliyun"
regexp_substr('I have 2 apples and 100 bucks!','[[:blank:]][[:alnum:]]*',1,1)，返回” have”
regexp_substr('I have 2 apples and 100 bucks!','[[:blank:]][[:alnum:]]*',1,2),返回” 2”
```

sample

sample_function (*sample_parameter, column_name*)

用途：

对所有读入的 column_name 的值，sample_function 根据 sample_parameter 的要求做 sample，并过滤掉不满足 sample 条件的行。

参数说明：

- sample_parameter 为 x, y，代表哈希为 x 份，取第 y 份。y 可省略，省略时取第一份。x,y 为整型常量，大于 0，其它类型或小于等于 0 时抛异常，若 y>x 也抛异常。x,y 任一输入为 NULL 时返回 NULL。
- column_name 是采样的目标列。column_name 可以省略，省略时根据 x, y 的值随机采样。任意类型，列的值可以为 null。不做隐式类型转换。如果 column_name 为常量 null 会报异常。

备注：

- 为了避免 null 值带来的数据倾斜，因此对于 column_name 中为 null 的值，会在 y 个 bullet 中进行均匀哈希。

示例：

SELECT * FROM *TBLA* where *sample* (4, 1 *, COLA*) = TRUE;

表示数值会根据 COLA hash 为 4 份，取第 1 份。

split_part

命令格式：

```
split_part(string, separator, start[, end])
```

用途：

拆分字符串，返回指定的部分

参数说明：

- String：string 类型，要拆分的字符串。如果是 bigint, double. datetime 类型会隐式转换到 string 类型后参加运算，其它类型报异常。

- Separator：string 类型常量，拆分用的分隔符，可以是一个字符，也可以是一个字符串，其它类型会引发异常。
- start:bigint 类型常量，必须大于 0。非常量或其它类型抛异常。返回段的开始编号（从 1 开始），如果没有指定 end，则返回 start 指定的段。
- end：bigint 类型常量，大于等于 start。返回段的截止编号，非常量或其他类型会引发异常。(start 必须大于 0, end 必须大于或等于 start, 否则抛异常，结果为 start 到 end 的闭区间)

返回值：

separator 连接的字符串片断.若任意参数为 NULL，返回 NULL；若 separator 为空串，返回 string.

备注：

- 如果 separator 不存在于 string 中，且 start 指定为 1，返回整个 string。 若输入为空串，输出为空串。
- 如果 start 越区，比如字符串拆分完有 6 个片段，但 start 大于 6，返回空串（''）。
- 若 end 大于片段个数，按片段个数处理。

to_char

命令格式：

```
to_char(boolean)
to_char(bigint)
to_char(double)
```

用途：

将布尔型、整型或者浮点型数值转为对应的字符串表示

参数类型：

- 单参数的 to_char 可以接受布尔型，整型或者浮点型输入，其它类型抛异常。对 datetime 类型的格式化输出请参考 10.4 节。

返回值：

对应值的字符串表示，如果输入为 NULL，返回 NULL。

示例：

- to_char(123)返回'123'
- to_char(true)返回'TRUE'
- to_char(1.23)返回'1.23'

to_char(cast(null as bigint)) 返回 NULL

substr

命令格式：

```
substr(string1, start_position[, length])
```

用途：

返回字符串 string1 从 start_position 开始长度为 length 的子串。

参数说明：

- string1: string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。
- start_position: bigint 类型，当 start_position 为负数时表示开始位置是从字符串的结尾往前倒数，最后一个字符是-1。其它类型抛异常。
- length: bigint 类型，大于 0，其它类型或小于等于 0 抛异常。子串的长度。

返回值:

在 string1 中的子串，若任一输入为 NULL,返回 NULL.

备注：

当 length 被省略时，返回到 string1 结尾的子串。

示例：

- substr("abc", 2)，返回 bc
- substr("abc", 2, 1)，返回 b

tolower

命令格式：

```
tolower(source)
```

用途：

输入字符串对应的小写字符串。

参数说明：

source: string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

返回值：

小写字符串，输入为 NULL 时返回 NULL.

示例：

- tolower(“aBcd”) 结果”abcd”
- tolower(“哈哈 Cd”) 结果”哈哈 cd”

toupper

命令格式：

```
toupper(source)
```

用途：

输入字符串对应的大写字符串。

参数说明：

source: string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

返回值：

大写字符串，输入为 NULL 时返回 NULL.

示例：

- toupper(“aBcd”) 结果”ABCD”
- toupper(“哈哈 Cd”) 结果”哈哈 CD”

trim

命令格式：

```
trim(string )
```

用途：

将输入字符串去除左右空格。

参数说明：

- string：string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

返回值：

string 类型，输入为 NULL 时返回 NULL。

wm_concat

命令格式：

```
wm_concat(separator, string)
```

用途：

用指定的 spearator 做分隔符，做字符串类型的 SUM 操作。

参数说明：

- separator，string 类型常量，分隔符。其他类型或非常量将引发异常。
- string，string 类型，若输入为 bigint, double, datetime 类型会隐式转换为 string 后参与运算，其它类型报异常。

返回值：

以 separator 分隔的字符串。

备注：

对语句 SELECT wm_concat(‘,’,name) from table;若 table 为空集合，这条语句返回{NULL}。

示例：

```
SELECT deptno, wm_concat(‘阿里巴巴’,ename) AS employees
FROM     emp
GROUP BY deptno;

     DEPTNO EMPLOYEES
---------- --------------------------------------------------
          10 CLARK 阿里巴巴 KING 阿里巴巴 MILLER
          20 SMITH 阿里巴巴 FORD 阿里巴巴 ADAMS 阿里巴巴 SCOTT 阿里巴巴 JONES
          30 ALLEN 阿里巴巴 BLAKE 阿里巴巴 MARTIN 阿里巴巴 TURNER 阿里巴巴 JAMES 阿里巴巴 WARD

3 rows selected.
```

-

3.3. 日期函数

ODPS SQL 中的 DATETIME 模式的函数可以支持 DATETIME 类型和 STRING 类型之间隐式类型转换，转换时使用的格式为 yyyy-mm-dd hh:mi:ss。包括两种情况，一是原接受 DATETIME 类型的函数，如果传入了一个字符串，则 ODPS SQL 会自动将字符串转换为 DATETIME 类型参与运算，二是在将数据插入到表中时，ODPS SQL 会自动的将数据类型转换为目标表中对应的列的类型。

Datetime 类型的表达的有效日期范围是 0001-01-01 00:00:00 到 9999-12-31 23:59:59。

日期字符串格式

格式：

yyyy-mm-dd hh:mi:ss

说明：

部分	字符串
年	yyyy

月	mm(01 ~ 12)
日	dd(01 ~ 31)
时	hh(00 ~ 23)
分	mi(00 ~ 59)
秒	ss(00 ~ 59)

表 10-1

dateadd

命令格式：

```
dateadd(datetime, delta, datepart)
```

用途：

按照指定的单位和幅度修改 datetime 的值

参数说明：

- datetime:datetime 类型，日期值。若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。。
- delta:bigint 类型，修改幅度。若输入为 string 类型或 double 型会隐式转换到 biging 类型后参与运算，其他类型会引发异常。若 delta 大于 0，加；否则减。
- datepart: string 类型常量，修改单位，，支持格式对天的修改 ″dd″，对月的修改 ″mm″，对年的修改 ″yyyy″ ，对小时修改″hh″, 对分钟修改″mi″，对秒修改″ss″ ，此外也支持扩展的日期格式，年-“year”，月-“month”或”mon”，日-“day”，小时-“hour”。 非常量、不支持的格式会或其它类型抛异常。

返回值：

返回修改后的结果，datetime 类型。若任一输入参数为 NULL，返回 NULL。

备注：

按照指定的单位增减 delta 时导致的对更高单位的进位或退位，年、月、时、分、秒分别按照 10 进制、12 进制、24 进制、60 进制、60 进制计算。当 delta 的单位是月时，计算规则如下：若 datetime 的月部分在增加 delta 值之后不造成 day 溢出，则保持 day 值不变，否则把 day 值设置为结果月份的最后一天。

示例：

- 加一天：dateadd(trans_date, 1, 'dd')
- 减一天：dateadd(trans_date, -1, 'dd')
- 加二十个月：dateadd(trans_date, 20, 'mm')
  - 若 trans_date = '2005-02-28 00:00:00', dateadd(transdate, 1, 'mm') = '2005-03-28 00:00:00'
  - 若 trans_date = '2005-01-29 00:00:00', dateadd(transdate, 1, 'mm') = '2005-02-28 00:00:00'
  - 若 trans_date = '2005-03-30 00:00:00', dateadd(transdate, -1, 'mm') = '2005-02-28 00:00:00'

datediff

命令格式：

```
datediff(datetime1, datetime2, datepart)
```

用途：

计算两个时间的差值，并转换成指定的单位，如：秒。

参数说明：

- datetime1 , datetime2: datetime 类型，被减数和减数，若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。
- datepart: string 类型常量，修改单位，yyyy、mm、dd、hh、mi、ss 中的一个，指定时间差值的单位，也支持扩展的日期格式，年-“year”，月-“month”或”mon”，日-“day”，小时-“hour”。。若 datepart 不符合指定的几种 pattern 或者其它类型则会发生异常。

返回值：

返回时间差值，int 类型。任一输入参数是 NULL，返回 NULL。

备注：

计算时会按照 datepart 切掉低单位部分，然后再计算结果。

示例：

若 start = ‘2005-12-31 23:59:59’, end = ‘2006-01-01 00:00:00’:

- datediff(end, start, 'dd') = 1
- datediff(end, start, 'mm') = 1
- datediff(end, start, 'yyyy') = 1
- datediff(end, start, 'hh') = 1
- datediff(end, start, 'mi') = 1
- datediff(end, start, ’ s s ’) = 1

datepart

命令格式：

```
datepart(datetime, part)
```

用途：

提取日期中 part 指定的部分

参数说明：

- datetime: datetime 类型，日期值，若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。
- part：string 类型常量。支持的 pattern 包括 yyyy、mm、dd、hh、mi、ss，此外也支持扩展的日期格式，年-“year”，月-“month”或”mon”，日-“day”，小时-“hour”。。不支持的 pattern 或其它类型会抛异常。

返回值：

返回值类型为 bigint.若任一输入参数为 NULL，返回 NULL。

datetrunc

```
datetrunc (datetime,format)
```

用途：

返回截取后的日期值。

参数说明：

- datetime: datetime 类型,日期值，若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。

- format: string 类型常量，候选值为：yyyy - 年, mm - 月, dd - 日，hh - 小时, mi - 分钟, ss - 秒，也支持扩展的日期格式, 年-“year”, 月-“month”或”mon”, 日-“day”, 小时-“hour”。 非常量、其它类型或不支持的格式会引发异常。

返回值：

返回 datetime.

备注：

任意一个参数为 NULL 的时候返回 NULL。

示例：

- datetrunc("2011-12-07 16:28:46", "yyyy")返回 "2011-01-01 00:00:00"
- datetrunc("2011-12-07 16:28:46", "month")返回"2011-12-01 00:00:00"

datetrunc("2011-12-07 16:28:46", "DD")返回 "2011-12-07 00:00:00"

from_unixtime

命令格式：

```
from_unixtime(unixtime)
```

用途：

将数字型的 unix 时间日期值转为日期值

参数说明：

- unixtime: bigint 类型，秒数，unix 格式的日期时间值，若输入为 string,double 类型会隐式转换为 bigint 后参与运算。

返回值：

DATETIME 类型的日期值, unixtime 为 NULL 时返回 NULL。

getdate

命令格式：

```
getdate()
```

用途：

获取当前系统时间

返回值：

返回当前日期和时间，datetime 类型。

备注：

在一个任务中，即使有多个 INSTANCE，getdate 总是返回一个固定的值。

isdate

命令格式：

```
isdate(datetime,  format)
```

用途：

判断一个日期字符串能否根据对应的格式串转换为一个日期值，如果转换成功返回 TRUE,否则返回 FALSE。

参数说明：

- datetime: string 格式的日期值，若输入为 bigint, double, datetime 类型会隐式转换为 string 类型后参与运算， 其它类型报异常。
- format: string 类型常量，指定日期格式,可选值见上表 10-1 。其它类型或不支持的格式会抛异常。如果 format string 中出现多余的格式串，则只取第一个格式串对应的日期数值,其余的会被视为分隔符。如 isdate('1234-yyyy','yyyy-yyyy'),会返回 TRUE，如果用 to_date('1234-yyyy','yyyy-yyyy')则会返回 1234-01-01 00:00:00。

返回值：

返回 BOOLEAN，如 dt 或 fmt 为 NULL，返回 NULL.

lastday

命令格式：

```
lastday(datetime)
```

用途：

取一个月的最后一天, 截取到天，时分秒部分为 00:00:00。

参数说明：

- datetime: datetime 格式的日期值，若输入为 string 类型会隐式转换为 datetime 类型后参与运算， 其它类型报异常。

返回值：

返回 datetime，如输入为 NULL，返回 NULL

to_date

命令格式：

```
to_date(datetime, format)
```

用途：

将一个字符串按照 format 指定的格式转成日期值。

参数说明：

- datetime: string 类型，要转换的字符串格式的日期值，若输入为 bigint, double, datetime 类型会隐式转换为 string 类型后参与运算，为其它类型抛异常，为空串时抛异常。

- format: string 类型常量，日期格式。非常量或其他类型会引发异常。format 参数中标识的间类型的元素（如年、月、日等），可选值见上表 10-1，其他字符作为无用字符在 parse 时忽略。format 参数至少包含'yyyy'，否则引发异常，如果 format string 中出现多余的格式串，则只取第一个格式串对应的日期数值，其余的会被视为分隔符。如 to_date('1234-2234','yyyy-yyyy')会返回 1234-01-01 00:00:00

返回值：

返回 datetime 类型。若任一输入为 NULL，返回 NULL 值。

备注：

format 参数在 parse 时，出于效率考虑，会要求 yyyy 对应的部分是四个字符，mm、dd、hh、mi、ss 对应的部分是分别是两个字符，否则会引发异常。详见示例。

示例：

- to_date('阿里金融 2010-12*03', '阿里金融 yyyy-mm*dd')返回'2010-12-03 00:00:00'
- to_date('阿里金融 2010-12*3', '阿里金融 yyyy-mm*dd')会引发异常
- to_date('2010-24-01', 'yyyy')会引发异常
- to_date('20080718', 'yyyymmdd')返回'2008-07-18 00:00:00'
- to_date('2008718', 'yyyymmdd')会引发异常

to_char

命令格式：

```
to_char(datetime, format)
```

用途：

将日期类型按照 format 指定的格式转成字符串

参数类型：

- datetime: datetime 类型，要转换的日期值，若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。
- format: string 类型常量。非常量或其他类型会引发异常。Format 中的日期格式部分会被替换成相应的数据，其它字符直接输出。详情见示例

返回值：

返回值为字符串类型。任一输入参数为 NULL，返回 NULL。

示例：

- to_char(trade_date, '阿里金融 yyyyddmm')返回'阿里金融 20102308'
- to_char(trade_date, '从前有座山')返回'从前有座山'
- to_char(trade_date, 'yyyyyy')返回'2010yy'

unix_timestamp

命令格式：

```
unix_timestamp(datetime)
```

用途：

将日期转化为整型的 unix 格式的日期时间值

参数说明：

- datetime: datetime 类型日期值，若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。

返回值：

整型 unix 格式日期值, datetime 为 NULL 时返回 NULL

weekday

命令格式：

```
weekday (datetime)
```

用途：

返回一个日期值是星期几，

参数说明：

- datetime: datetime 类型，日期值，若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。

返回值：

返回 BIGINT，若输入参数为 NULL，返回 NULL.

星期一:0

星期二:1

星期三:2

星期四:3

星期五:4

星期六:5

星期天：6

weekofyear

命令格式：

```
weekofyear (datetime)
```

用途：

返回一个日期位于那一年的第几周。周一作为一周的第一天。

参数说明：

datetime:datetime 类型日期值,若输入为 string 类型会隐式转换为 datetime 类型后参与运算，其它类型抛异常。

返回值：

Bigint 类型， 属于一年的第几周，数字。 若输入为 NULL，返回 NULL

3.4. 窗口函数

ODPS SQL 中可以使用窗口函数进行灵活的分析处理工作，窗口函数只能出现在 SELECT 子句中，窗口函数中不要嵌套使用窗口函数和聚合函数，窗口函数不可以和同级别的聚合函数一起使用。目前在一个 SQL 语句中，可以使用至多 5 个窗口函数。

窗口函数的语法

window_func() over (partition by col1,col2… [order by col1 [asc|desc],col2[asc|desc]…] windowing_clause)

partition by 部分用来指定开窗的列。

order by 用来指定数据在一个窗口内如何排序

windowing_clause 部分可以用 ROWS 指定开窗方式，有两种方式：

1. ROWS BETWEEN x PRECEDING|FOLLOWING AND y PRECEDING|FOLLOWING 表示窗口范围是从前或后 x 行到前或后 y 行。
2. ROWS x PRECEDING|FOLLOWING 窗口范围是从前或后第 x 行到当前行

x, y 必须为>=0 的整数常量， 限定范围<10000， 值为 0 时表示当前行。 必须指定 orderby 才可以用 ROWS 方式指定窗口范围 。

注：

并非所有的窗口函数都可以用 ROWS 指定开窗方式， 支持这种用法的窗口函数有 avg、count、max、min、stddev 和 sum。

avg

命令格式：

```
avg([distinct] expr) over(partition by col_list1 [order by col_list2] [windowing_clause])
```

用途： 计算平均值

参数说明：

- distinct: 当指定 distinct 关键字时表示取唯一值的平均值。
- expr: double 类型，若输入为 string 或 bigint 会隐式转换到 double 类型后参与运算，其它类型抛异常。当 value 值为 NULL 时，该行不参与计算。bool 类型不允许参与计算。
- col_list1:指定开窗口的列
- col_list2:不指定 order by 时返回当前窗口内所有值的平均值，指定 order by 时返回结果以 col_list2 指定的方式排序，并且返回窗口内从开始行到当前行的累计平均值。

返回值：DOUBLE 类型

注: 指明 distinct 关键字时不能写 order by。

count

命令格式：

```
count([distinct] expr) over(partition by col_list1 [order by col_list2] [windowing_clause])
```

用途： 计数值

参数说明：

- distinct: 当指定 distinct 关键字时表示取唯一值的计数值。
- expr: 任意类型，当 value 值为 NULL 时，该行不参与计算。
- col_list1: 指定开窗口的列

- col_list2: 不指定 order by 时，返回当前窗口内 expr 的计数值，指定 order by 时返回结果以 col_list2 指定的顺序排序，并且值为当前窗口内从开始行到当前行的累计计数值。

返回值：BIGINT 类型

注：当指定 distinct 关键字时不能写 order by。

max

命令格式：

```
max( expr) over(partition by col_list1 [order by col_list2] [windowing_clause])
```

用途： 计算最大值

参数说明：

- expr: 除 bool 以外的任意类型，当 value 值为 NULL 时，该行不参与计算。
- col_list1:指定开窗口的列
- col_list2:不指定 order by 时，返回当前窗口内的最大值。指定 order by 时，返回结果以 col_list2 指定的方式排序，并且值为当前窗口内从开始行到当前行的最大值。

返回值：同 expr 类型

median

命令格式：

```
double median(double number) over(partition by col_list)
```

用途：

中位数。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。当输入值为 NULL 时忽略。

返回值：

double 类型。

注：目前限制在一个窗口内不超过 10000 行数据。

min

命令格式：

```
min( expr) over(partition by col_list1 [order by col_list2] [windowing_clause])
```

用途： 计算最小值

参数说明：

- expr: 除 bool 以外的任意类型，当 value 值为 NULL 时，该行不参与计算。
- col_list1:指定开窗口的列
- col_list2:不指定 order by 时，返回当前窗口内的最小值。指定 order by 时，返回结果以 col_list2 指定的方式排序，并且值为当前窗口内从开始行到当前行的最小值。

返回值：同 expr 类型

stddev

命令格式：

```
double stddev(double number) over (partition by col_list1 [order by col_list2]
[windowing_clause])
```

用途：

总体标准差。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。当输入值为 NULL 时忽略该行。

返回值：

double 类型。

stddev_samp

命令格式：

```
double stddev_samp(double number) over(partition by col_list1 [order by col_list2] [windowing_clause])
```

用途：

样本标准差。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。当输入值为 NULL 时忽略该行。

返回值：

double 类型。

sum

命令格式：

```
sum( [distinct] expr) over(partition by col_list1 [order by col_list2] [windowing_clause])
```

用途： 计算汇总值

参数说明：

- distinct:指定 distinct 关键字时表示计算唯一值的汇总值
- expr: double 类型，当输入为 string, bigint 时隐式转换为 double 参与运算，其它类型报异常。当 value 值为 NULL 时，该行不参与计算。
- col_list1: 指定开窗口的列
- col_list2: 不指定 order by 时，返回当前窗口内 expr 的汇总值。指定 order by 时，返回结果以 col_list2 指定的方式排序，并且返回当前窗口从首行至当前行的累计汇总值。

返回值：double 类型

注：当指定 distinct 时不能用 order by。

dense_rank

命令格式：

```
dense_rank() over(partition by col_list1 order by col_list2)
```

用途： 连续排名

参数说明:

- col_list1: 指定开窗口的列
- col_list2: 指定排名依据的值

返回值：BIGINT 类型

lag

```
lag( expr, offset, default) over(partition by col_list1 order by col_list2)
```

用途： 按偏移量取当前行之前第几行的值，如当前行号为 rn，则取行号为 rn-offset 的值

参数说明：

- expr: 任意类型。
- offset: bigint 类型常量，输入为 string, double 到 bigint 的隐式转换, offset>0。
- default: 当 offset 指定的范围越界时的缺省值，常量。
- col_list1: 指定开窗口的列
- col_list2: 指定返回结果的排序方式

返回值：同 expr 类型

lead

```
lead( expr, offset, default) over(partition by col_list1 order by col_list2)
```

用途：按偏移量取当前行之后第几行的值，如当前行号为 rn 则取行号为 rn+offset 的值

参数说明：

- expr: 任意类型。
- offset: bigint 类型常量，输入为 string, double 到 bigint 的隐式转换, offset>0。
- default: 当 offset 指一的范围越界时的缺省值，常量。
- col_list1: 指定开窗口的列
- col_list2: 指定返回结果的排序方式

返回值：同 expr 类型

percent_rank

命令格式：

```
percent_rank() over(partition by col_list1 order by col_list2)
```

用途： 计算一组数据中某行的相对排名。

参数说明:

- col_list1: 指定开窗口的列
- col_list2: 指定排名依据的值

返回值：double 类型，值域为[0,1]，值为(rank-1)/(number of rows -1)。

rank

命令格式：

```
rank() over(partition by col_list1 order by col_list2)
```

用途：计算排名

参数说明:

- col_list1: 指定开窗口的列
- col_list2: 指定排名依据的值

返回值：BIGINT 类型

row_number

命令格式：

```
row_number() over(partition by col_list1 order by col_list2)
```

用途： 返回行号，从 1 开始

参数说明：

col_list1: 指定开窗口的列

col_list2: 指定结果返回时的排序的值

返回值：BIGINT 类型

3.5. 聚合函数

avg

命令格式：

```
avg(value)
```

用途： 计算平均值

参数说明：

- Value: double 类型，若输入为 string 或 bigint 会隐式转换到 double 类型后参与运算，其它类型抛异常。当 value 值为 NULL 时，该行不参与计算。Bool 类型不允许参与计算。

返回值：DOUBLE 类型

如表 TBLA 有一列 VALUE，类型为 BIGINT

VALUE
1
2
NULL

则对该列计算 AVG 结果为(1+2)/2=1.5

count

命令格式：

```
count([distict|all] value)
```

用途： 计算记录数

参数说明：

- distinct|all，指明在计数时是否去除重复记录，默认是 ALL，即计算全部记录，如果指定 DISTINCT，则可以只计算唯一值数量
- value :可以为任意类型，当 value 值为 NULL 时，该行不参与计算, value 可以为*，当 COUNT(*)时，返回所有行数

返回值：BIGINT 类型

如表 TBLA 有列 COL1 类型为 BIGINT

COL1
1
2
NULL

SELECT COUNT(*) FROM TBLA;值为 3,

SELECT COUNT(COL1) FROM TBLA; 值为 2

max

命令格式：

```
max(value)
```

用途： 计算最大值

参数说明：

- value：可以为任意类型，当列中的值为 NULL 时，该行不参与计算。Bool 类型不允许参与运算。

如表 TBLA 有一列 VALUE，类型为 BIGINT

VALUE
1
2
NULL

SELECT MAX(VALUE) FROM TBLA， 返回值为 2

median

命令格式：

```
double median(double number)
```

用途：

中位数。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。当输入值为 NULL 时忽略。

返回值：

double 类型。

注：目前限制在一个组内不超过 100 万行数据

min

命令格式：

```
min(value)
```

用途： 计算最小值

参数说明：

- value:可以为任意类型，当列中的值为 NULL 时，该行不参与计算。Bool 类型不允许参与计算。

如表 TBLA 有一列 VALUE，类型为 BIGINT

VALUE
1
2
NULL

SELECT MIN(VALUE) FROM TBLA， 返回值为 1

stddev

命令格式：

```
double stddev(double number)
```

用途：

总体标准差。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。当输入值为 NULL 时忽略。

返回值：

double 类型。

stddev_samp

命令格式：

```
double stddev_samp(double number)
```

用途：

样本标准差。

参数说明：

- number: double 类型。若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。当输入值为 NULL 时忽略。

返回值：

double 类型。

sum

命令格式：

```
sum(value)
```

用途： 计算汇总值

参数说明：

- value:double 类型，若输入为 string 或 bigint 会隐式转换到 double 类型后参与运算，当列中的值为 NULL 时，该行不参与计算。Bool 类型不允许参与计算。

如表 TBLA 有一列 VALUE，类型为 BIGINT

VALUE
1
2
NULL

SELECT SUM(VALUE) FROM TBLA， 返回值为 3

3.6. 其他函数

cast

命令格式：

```
cast(expr as <type>)
```

用途：

将表达式的结果转换成目标类型，如 cast('1' as BIGINT)将字符串'1'转为整数类型的 1，如果转换不成功或不支持的类型转换会引发异常。

cast(double as bigint)，将 double 值 trunc 成 bigint

cast(string as bigint) 在将字符串转为 BIGINT 时，如果字符串中是以整型表达的数字，会直接转为 BIGINT 类型。如果字符串中是以浮点数或指数形式表达的数字，则会先转为 DOUBLE 类型，再转为 BIGINT 类型。

cast(string as datetime) 或 cast(datetime as string)时，会采用默认的日期格式 yyyy-mm-dd hh:mi:ss。

coalesce

命令格式：

```
coalesce(expr1, expr2,...)
```

用途：

返回列表中第一个非 null 的值，如果列表中所有的值都是 null 则返回 null。

参数说明：

expri 是要测试的值。所有这些值类型必须相同或为 NULL，否则会引发异常。

返回值：

返回值类型和参数类型相同。

备注：

参数至少要有一个，否则引发异常。

decode

命令格式：

```
decode(expression, search, result[, search, result]...[, default])
```

用途：

实现 IF-THEN-ELSE 分支选择的功能。

参数说明：

- Expression: 要比较的表达式.
- Search: 和 expression 进行比较的搜索项。.
- Result:search 和 expression 的值匹配时的返回值.
- Default:可先项，如果所有的搜索项都不匹配，则返回此 default 值，如果未指定，则会返回 null。

返回值：

返回匹配的 search；如果没有匹配，返回 default；如果没有指定 default，返回 NULL。

备注：

- 至少要指定三个参数。
- 所有的 result 类型必须一致，或为 NULL。不一致的数据类型会引发异常。所有的 search 和 expression 类型必须一致，否则报异常。
- 如果 decode 中的 search 选项有重复时且匹配时，会返回第一个值。

示例：

```
 SELECT supplier_name, decode(supplier_id, 10000, 'IBM', 10001, 'Microsoft', 10002, 'Hewlett
ackard', 'Gateway') result FROM suppliers;

上面的 decode 函数实现了下面 IF-THEN-ELSE 语句中的功能
  IF supplier_id = 10000 THEN
    result := 'IBM';
  ELSIF supplier_id = 10001 THEN
    result := 'Microsoft';
  ELSIF supplier_id = 10002 THEN
    result := 'Hewlett Packard';
  ELSE
    result := 'Gateway';
  END IF;
```

get_idcard_age

命令格式：

get_idcard_age(idcardno)

用途：

根据身份证号返回当前的年龄，当前年份减去身份证中标识的出生年份的差值。

参数说明：

- idcardno: string 类型， 15 位或 18 位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性，如果校验不通过会返回 NULL。

返回值：

Bigint 类型，输入为 NULL 返回 NULL。如果当前年份减去出生年份差值大于 100，返回 NULL。

get_idcard_birthday

命令格式：

get_idcard_birthday(idcardno)

用途：

根据身份证号返回出生日期。

参数说明：

- idcardno: string 类型， 15 位或 18 位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性，如果校验不通过会返回 NULL。

返回值：

datetime 类型，输入为 NULL 返回 NULL。

get_idcard_sex

命令格式：

get_idcard_sex(idcardno)

用途：

根据身份证号返回性别，值为 M(男)或 F(女)。

参数说明：

- idcardno: string 类型， 15 位或 18 位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性，如果校验不通过会返回 NULL。

返回值：

string 类型，输入为 NULL 返回 NULL。

greatest

命令格式：

greatest(var1,var2…)

用途：

返回输入参数中最大的一个

参数说明：

- var1,var2 可以为 bigint,double,datetime,string。若所有值都为 NULL 则返回 NULL。
- null 为最小
- 当输入参数类型不同时，double,bigint,string 之间的比较转为 double；string,datetime 的比较转为 datetime。

  不允许其它的隐式转换。

返回值：

输入参数中的最大值，当不存在隐式转换时返回同输入参数类型。

有隐式转换时，double,bigint,string 之间的转换返回 double。
string,datetime 之间的转换返回 datetime

max_pt

命令格式：

```
max_pt(table_name)
```

参数说明：

- Table_name: string 类型，指定表名，用户必须对此表有读权限。

用途：

对于分区的表，此函数返回该分区表的一级分区的最大值，按字母排序，且该分区下有对应的数据文件。

返回值：

最大的一级分区的值。

例：

如 tbl 是分区表，该表对应的分区如下，且都有数据文件

Pt=’20120901’

Pt=’20120902’

则以下语句中 max_pt 返回值为’20120902’，SQL 语句读出 pt=’20120902’分区下的数据。

Select * from tbl where pt=max_pt(‘tbl’)

注：如果只是用 alter table 的方式新加了一个分区，但是此分区中并无任何数据文件，则此分区不会做为返回值。

ordinal

命令格式：

```
ordinal(bigint nth, var1,var2…)
```

用途：

将输入变量按从小到大排序后，返回 nth 指定的位置的值

参数说明

- nth: bigint 类型，指定要返回的位置，为 NULL 时返回 NULL。
- var1,var2，类型可以为 bigint,double,datetime,string。
- null 为最小
- 当输入参数类型不同时，double,bigint,string 之间的比较转为 double；string,datetime 的比较转为 datetime
  不允许其它的隐式转换

返回值：

排在第 nth 位的值，当不存在隐式转换时返回同输入参数类型。
有隐式转换时，double,bigint,string 之间的转换返回 double。
string,datetime 之间的转换返回 datetim

例：

ordinal(3, 1, 2, 2, 3, 4, 5, 6) 返回 2

option_bit

使用命令：

```
option_bit(value, path)
```

参数类型：

Value: string。如果 value 为 NULL，返回“0”

Path: string

返回类型：

string

功能描述：

返回一个 string 类型的“数字”，假设该数字为 returnvalue，returnvalue 由下列原则确定：

1. 如果 value 含有字符串”vip”, returnvalue = 1；
2. 在原则 1 的基础上，如果 value 含有字符串“ppay”:
   a) 如果 returnvalue!= 0， returnvalue 按位或 64，此结果作为 returnvalue 新的值；
   b) 否则 returnvalue=64；
3. 在原则 2 的基础上，如果 value 含有字符串“suitSeller”:
   a) 如果 returnvalue!= 0， returnvalue 按位或 16384，此结果作为 returnvalue 新的值；
   b) 否则 returnvalue=16384;
4. 在原则 3 的基础上，如果 value 含有字符串“suitBuyer”:
   a) 如果 returnvalue!= 0， returnvalue 按位或 32768，此结果作为 returnvalue 新的值；
   b) 否则 returnvalue=32768;
5. 在原则 4 的基础上，如果 value 含有字符串“wap”:
   a) 如果 returnvalue!= 0， returnvalue 按位或 262144，此结果作为 returnvalue 新的值；
   b) 否则 returnvalue=262144;
6. 在原则 5 的基础上，如果 value 含有字符串“fbuy:1”:
   a) 如果 returnvalue!= 0， returnvalue 按位或 524288，此结果作为 returnvalue 新的值；
   b) 否则 returnvalue=524288;
7. 在原则 6 的基础上，如果 path 不为 NULL 并且不是空串：
   a) 如果 returnvalue!= 0， returnvalue 按位或 131072，此结果作为 returnvalue 新的值；
   b) 否则 returnvalue=131072;

备注：此函数是应阿里金融的要求依照淘宝的 hive 的业务逻辑实现的，我们无法保证在淘宝的 hive 中此函数的行为发生变化时 ODPS SQL 中的函数行为会随之发生同样的变化。如果在使用中发现此函数的行为不满足你的要求，可以使用 ODPS 的 SDK 自己开发自定义函数。

least

命令格式：

```
least(var1,var2…)
```

用途：

返回输入参数中最小的一个

参数说明：

- var1,var2 可以为 bigint,double,datetime,string。若所有值都为 NULL 则返回 NULL。
- null 为最小
- 当输入参数类型不同时，double,bigint,string 之间的比较转为 double，string,datetime 的比较转为 datetime

  不允许其它的隐式转换

返回值：

输入参数中的最小值，当不存在隐式转换时返回同输入参数类型。

有隐式转换时，double,bigint,string 之间的转换返回 double。
string,datetime 之间的转换返回 datetime

trans_cols

命令格式：

```
trans_cols (num_keys, key1,key2,…,col1, col2,col3) as (idx, key1,key2,…,col1, col2)
```

用途：

用于将一行数据转为多行的 UDTF，将不同的列转为行。

参数说明：

num_keys: bigint 类型常量，必须>=0。在转为多行时作为转置 key 的列的个数。

Key 是指在将一行转为多行时，在多行中重复的列,如要将 A,B,C,D 转为

A,B,C

A,B,D

则 A,B 列为 key

keys: 转置时作为 key 的列，由 num_keys 决定哪些列作为 key。

cols: 要转为行的列，类型必须相同。

返回：

转置后新的列名由 as 指定。输出的第一列是转置的下标，下标从 1 开始。

作为 key 的列类型保持不变，其余所有的列与原来类型一致。如果 num_keys 指定所有的列都作为 key（即 num_keys 等于所有列的个数），则只返回一行。

注：UDTF 使用上有一些限制

- 所有作为 key 的列必须处在前面，而要转置的列必须放在后面。
- 在一个 select 中只能有一个 udtf，不可以再出现其它的列，如不可以写成
  Select login_id, trans_cols(1, login_id, login_ip1, login_ip2) as(idx, login_id, login_ip)
- 不可以与 roup by/cluster by/distribute by/sort by 一起使用。

例，表中的数据如

Login_id	Login_ip1	Login_ip2
wangwangA	192.168.0.1	192.168.0.2

则 trans_cols(1, login_id, login_ip1, login_ip2) as (idx, login_id, login_ip)的输出为:

idx	Login_id	Login_ip
1	wangwangA	192.168.0.1
2	wangwangA	192.168.0.2

trans_array

```
trans_array (num_keys, separator, key1,key2,…,col1, col2,col3) as (key1,key2,…,col1, col2)
```

用途：

用于将一行数据转为多行的 UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。

参数说明：

num_keys: bigint 类型常量，必须>=0。在转为多行时作为转置 key 的列的个数。

Key 是指在将一行转为多行时，在多行中重复的列。

separator:string 类型常量，用于将字符串拆分成多个元素的分隔符。为空时报异常。

keys:转置时作为 key 的列， 个数由 num_keys 指定。如果 num_keys 指定所有的列都作为 key（即 num_keys 等于所有列的个数），则只返回一行。

cols: 要转为行的数组，keys 之后的所有列视为要转置的数组，必须为 string 类型，存储的内容是字符串格式的数组，如“Hangzhou;Beijing;shanghai”，是以”;”分隔的数组。

返回：

转置后的行，新的列名由 as 指定。作为 key 的列类型保持不变，其余所有的列是 string 类型。拆分成的行数以个数多的数组为准，不足的补 NULL。

注：UDTF 使用上有一些限制

- 所有作为 key 的列必须处在前面，而要转置的列必须放在后面。
- 在一个 select 中只能有一个 udtf，不可以再出现其它的列
- 不可以与 group by/cluster by/distribute by/sort by 一起使用。

例，表中的数据如

Login_id	LOGIN_IP	LOGIN_TIME
wangwangA	192.168.0.1,192.168.0.2	20120101010000,20120102010000

则 trans_array(1, ",", login_id, login_ip, login_time) as (login_id,login_ip,login_time)

产生的数据是

Login_id	Login_ip	Login_time
wangwangA	192.168.0.1	20120101010000
wangwangA	192.168.0.2	20120102010000

如果表中的数据是

Login_id	LOGIN_IP	LOGIN_TIME
wangwangA	192.168.0.1,192.168.0.2	20120101010000

则对数组中不足的数据补 NULL

Login_id	Login_ip	Login_time
wangwangA	192.168.0.1	20120101010000
wangwangA	192.168.0.2	NULL

unique_id

uuid 命令格式：

```
string unique_id()
```

用途：

返回一个随机的唯一 id，形式示例："29347a88-1e57-41ae-bb68-a9edbdd94212_1"，相对于 uuid 运行效率比较高。

uuid

uuid 命令格式：

```
string uuid()
```

用途：

返回一个随机的唯一 id，形式示例："29347a88-1e57-41ae-bb68-a9edbdd94212"。

4. 类型转换和异常处理

本节介绍 ODPS SQL 在不同类型间进行类型转换的规则，以及在运算中（包括运算符计算、读写表、UDF 计算）中遇到异常时的处理逻辑

4.1. 类型转换规则

显式类型转换规则

显式类型转换是指用 CAST 将一种数据类型的值转为以另外一种类型表示的值，在 ODPS SQL 中支持的显式类型转换如下，表中的 cast 表示允许进行转换，X 表示不允许转换。

	to bigint	to double	to string	to datetime	to boolean
from bigint	--	cast	cast	X	X
from double	cast	--	cast	X	X
from string	cast	cast	--	cast	X
from datetime	X	X	cast	--	X
from boolean	X	X	X	X	--

1. 不支持的显式类型转换会导致异常。
2. 对于支持的显式类型转换，如果在执行时转换失败，报错退出。
3. 日期类型转换时采用默认格式 yyyy-mm-dd hh:mi:ss

隐式类型转换规则

隐式类型转换是指在运行时，由系统自动进行的类型转换，系统会判断在合理的情况下进行转换，否则将会抛出异常。推荐的做法是在类型不匹配的时候显式的用 CAST 进行转换而不依赖于隐式类型转换。ODPS SQL 支持的隐式类型转换如下,表中的 Y 表示允许转换，X 表示不允许转换。

	to int	to double	to string	to datetime	to boolean
from int	--	Y	Y	X	X
from double	Y	--	Y	X	X
from string	Y	Y	--	Y	X
from datetime	X	X	Y	--	X
from boolean	X	X	X	X	--

1. 不支持的隐式类型转换会导致异常。
2. 对于支持的隐示类型转换，如果在执行时转换失败，也会导致异常。
3. 隐式类型转换规则是有发生作用域的，在某些作用域中，只有一部分规则可以生效。（参见下面的描述）。

类型转换时的精度与格式

- double 转换为 string

为了性能考虑，最多支持 12 位有效数字，行为是：

```
create table tzn0114p as select cast(1234567890.0987654321 as string) as c1, cast(1234567890.0987654321 as double) as c2, cast(12345678901234567890.123 as string) as c3, cast (0.1234567890123456789 as double) as c4 from tzn0112g;

read tzn0114p;
+----+------------+----+------------+
| c1 | c2         | c3 | c4         |
+----+------------+----+------------+
```

```
| 1234567890.1 | 1234567890.1 | 12345678901200000000.0 | 0.123456789012 |
+----+------------+----+------------+
```

在做 select 屏显和 read table 操作时，因为均需要将 double 转换成 string，因此规则同上。

- 日期类型与字符串类型的隐式转换

日期函数支持隐式类型转换，即原接受 datetime 类型的函数可以接受字符串格式的参数，会自动转换为 datetime 类型后参与运算，转换时使用的格式为 YYYY-MM-DD HH:MI:SS，在隐式将 datetime 转为 string 时，也采用此格式。

隐式类型转换的作用域

INSERT 操作

1. 对于 INSERT 操作，把插入值隐式转换为字段的数据类型。
   假如表 t1 中 id 列的数据类型为 BIGINT
   insert into table t1 select '1' as id from t2;
   系统在将值写入 t1 时会自动将值转换为 BIGINT，因此等价于
   insert into table t1 select cast ('1' as int) as id from t2;

算术运算符(+, -, *, /, %, +, -)

1. 只有字符型、整型和双精度型才能参与算术运算。字符型在参与运算前会进行隐式类型转换到双精度类型。整型和双精度类型共同参与计算时，会将整型隐式转换为双精度类型。
   '1'+2 运算时会将字符串'1'转为数值类型 1.0
2. 日期型和布尔型不允许参与算数运算，也不能进行到允许类型的隐式类型转换。

LIKE、RLIKE

1. LIKE 和 RLIKE 的 source 和 pattern 均仅接受 string 类型
2. 双精度型、字符型、日期型和整型不允许参与 LIKE 和 RLIKE 运算，也不能进行到 string 类型的隐式类型转换。

IN

In 操作的使用方式为 key in (value1, value2, …)

1. 当 key 与右侧所有类型都一致时不进行转换。
2. IN 操作符右侧的类型必须一致，否则报异常。
3. key 和 value 的类型在 bigint, double, string 之间可以比较，统一转 double。
4. key 和 value 的类型在 datetime 和 string 之间可以比较，统一转 datetime
5. 除此之外不允许其它类型之间的转换。

位运算符(&, |)

1. 只有 bigint 类型才能参与位运算。
2. 双精度型、字符型、日期型和布尔型不允许参与位运算，也不能进行到允许类型的隐式类型转换。

逻辑运算符(AND, OR, NOT)

1. 只有布尔型才能参与逻辑运算。
2. 双精度型、字符型、日期型和整型不允许参与逻辑运算，也不能进行到允许类型的隐式类型转换。

关系运算符(=, <>, <, <=, >, >=, IS NULL, IS NOT NULL)

1. 所有类型都可以参与关系运算。
2. 当不同类型的数据共同参与关系运算时，按照下述原则进行隐式类型转换。

	int	**double**	**string**	**datetime**	**boolean**
int	--	double	double	--	--
double	double	--	double	--	--
string	double	double	--	datetime	--
datetime	--	--	datetime	--	--
boolean	--	--	--	--	--

以下是几个例子，

当比较字符型和日期型的数据时，把字符型转换为日期型。

假设 create_date 为字符型

```
select * from t where create_date > getdate();
```

等价于

```
select * from t where to_date(create_date, ‘yyyy-mm-dd hh:mi:ss’) > getdate();
```

假设 create_date 为 date 型

```
select * from t where create_date > '2006-11-11 11:11:11';
```

等价于

```
select * from t where create_date > to_date('2006-11-11 11:11:11', ‘yyyy-mm-dd hh:mi:ss’);
```

当比较整型和布尔型的数据时，将布尔型和整型转换为双精度型。

```
select 2 > true from t;
```

等价于

```
select cast(2 as double) > cast(true as double) from t;
```

3. 如果待比较的两个类型间不能进行隐式类型转换，则该比较不能完成，报错退出。
4. 备注：
   a) 所有类型与 null 判等返回 null。

函数

1. 在调用函数时，如果输入参数的数据类型与函数定义的参数数据类型不一致，系统会把输入参数的数据类型先转换为函数定义的数据类型再参与运算。
   如对于 sqrt(double)函数而言，如果 value 是字符型
   Select sqrt(value) from t; 等价于 select sqrt(cast (value as double)) from t;
2. 每个函数的参数对于允许的隐式类型转换的要求不同，详见每个函数的说明。

CASE WHEN

```
CASE
    WHEN condition_1 THEN result_1
    WHEN condition_2 THEN result_2
    …
    ELSE result_n
END

CASE value
  WHEN (_condition1) THEN result1
  WHEN (_condition2) THEN result2
  …
  ELSE resultn
END
```

1. 当 result_x 中有 string 类型时，所有的返回值统一转为 string，当出现不允许转换为 string 的类型时报异常。
2. 当 result_x 中有类型不一致情况时，如果返回值类型只有 bigint 和 double，则统一转为 double。
3. 除上述 1 和 2 提到的隐式类型转换外，不允许其它类型间的隐式转换。
4. value 与 condition_x 之间类型要一致。

partition column

1. partition column 仅支持字符类型，在生成动态 partition 时不允许隐式转换。

union all

参与 union all 运算的所有列的数据类型、列个数、列名称必须完全一致，否则抛异常

5. 表的统计信息

ODPS SQL 中支持用户自定义的统计项，可以用于数据质量监控等，ODPS 在将数据写到目标表时，会根据统计的内容尽量自动得出统计值从而方便的使用户了解到数据的质量情况。

1. 表的总行数。
2. 表的指定列为 NULL 的行数。
3. 表的指定列的总和。
4. 表的指定列的最大值。
5. 表的指定列的最小值。
6. 表的符合某个表达式的行数。
7. 表的某些列的唯一值个数（此统计项无法自动得出统计值）

注：不可以在 view 上面定义统计项，如果用户使用的是 INSERT INTO 语句或者是动态分区，系统也无法自动得出统计值，这时要使用 analyze 语句进行手工统计。

5.1. 定义统计项

添加特征信息

默认情况下，系统不会统计表的特征信息。用户可以根据需要，指定为某张表添加某项特征信息，命令为：

add statistic <TABLENAME> <STATISTICNAME> [<STATISTICRULE>];

其中，

- <TABLENAME>描述表名。
- <STATISTICNAME>描述需要添加的特征信息的名字。
- <STATISTICRULE>描述某些特定的特征信息所需要的规则；部分特征信息没有规则定义。

需要注意的是：

- 当表被 drop 掉之后重新创建时，表中已添加的特征信息和已经收集的特征信息统计值会丢失。
- **如果规则定义包含列名，当该列名被修改成其他名字后，规则定义不会自动修改，使用 analyze 执行统计流程在语义分析阶段会报无效的列。**
- 使用 CREATE TABLE AS 命令创建的表不会继承原表的特征信息。
- 部分用户定义的统计项并不会在 QUERY 执行的过程中自动收集数据，而是要通过执行 analyze 命令来手动收集。
- **如果自动统计过程中产生了异常，比如 sum 的值溢出，此时不会报异常（否则会导致整个 SQL 的失败），而是将结果置为 inf，在使用时必须注意。**

目前可以统计的特征信息为以下七种：

定义表的总行数：

- 别名：table_count

- 格式：整型
- 设置方法：
  - 添加信息项：add statistic <TABLENAME> table_count;
    如：add statistic tbl_shop table_count;
  - 移除信息项：remove statistic <TABLENAME> table_count;
    如：remove statistic tbl_shop table_count;
- 备注：
  - 该统计项的作用与“count <TABLENAME>;”命令相同。
- 显示示例：

  table_count: 10000

  或（对于 partition 的表）：

  (col1='2011-10-25 00:00:00', col2=1)　　table_count: 2000

定义表的指定列为 NULL 的行数:

- 别名：null_value
- 格式：整型
- 设置方法：
  - 添加信息项：

    add statistic <TABLENAME> null_value <COLUMNNAME>;

    如：add statistic tbl_shop null_value shop_name;
  - 移除信息项：

    remove statistic <TABLENAME> null_value <COLUMNNAME>;

    如：remove statistic tbl_shop null_value shop_name;
- 备注：
  - 当用户需要指定为多列统计 NULL 值时，需要分别为每一列定义。
- 显示示例：

  null_value col1=10000

  或（对于 partition 的表）：

  (col1='2011-10-25 00:00:00', col2=1)　　null_value col1=5000

定义表的指定列总和：

- 别名：column_sum
- 格式：整型或浮点型，取决于作用于的列的类型
- 设置方法：
  - 添加信息项：

add statistic <TABLENAME> column_sum <COLUMNNAME> [<FILTER_EXPR>];

如：add statistic tbl_shop column_sum revenue;

- 移除信息项：

  remove statistic <TABLENAME> column_sum <COLUMNNAME> [<FILTER_EXPR>];

如：remove statistic tbl_shop column_sum revenue;

- 备注：
  - 该特征规则仅能作用于整型和浮点型的列之上，否则在设置时报错。
  - 当列中有部分值是 NULL，NULL 不做统计。当列的全部值都是 NULL，结果是 NULL
  - 利用此统计项，以及所有记录数，可以得到该列的平均值。
  - FILTER_EXPR 是一个有效的表达式，表示只统计符合此表达式的行的值，参见“定义统计符合某个表达式的行数”
- 显示示例：

  column_sum col3=10000

  column_sum col4=111.11

  或（对于 partition 的表）：

  (col1='2011-10-25 00:00:00', col2=1) column_sum col3: 2000

  (col1='2011-10-25 00:00:00', col2=1) column_sum col4: 200.2

定义表的指定列最大值：

- 别名：column_max
- 格式：整型或浮点型或 datetime，取决于作用于的列的类型
- 设置方法：
  - 添加信息项：

    add statistic <TABLENAME> column_max <COLUMNNAME> [<FILTER_EXPR>];

  如：add statistic tbl_shop column_max revenue;

  - 移除信息项：

    remove statistic <TABLENAME> column_max <COLUMNNAME> [<FILTER_EXPR>];

  如：remove statistic tbl_shop column_max revenue;
- 备注：
  - 该特征规则仅能作用于整型和浮点型的列之上，否则在设置时报错。
  - 当列中部分值为 NULL 时，NULL 不做统计。当列全部为 NULL 的话，则结果为 NULL
  - FILTER_EXPR 是一个有效的表达式，表示只统计符合此表达式的行的值，参见“定义统计符合某个表达式的行数”
- 显示示例：

  column_max col3=9979

或（对于 partition 的表）：

(col1='2011-10-25 00:00:00', col2=1)　　column_max col3: 3122

定义表的指定列最小值:

- 别名：column_min
- 格式：整型或浮点型或 datetime，取决于作用于的列的类型
- 设置方法：
  - 添加信息项：

    add statistic <TABLENAME> column_min <COLUMNNAME> [<FILTER_EXPR>];

    如：add statistic tbl_shop column_min revenue;
  - 移除信息项：

    remove statistic <TABLENAME> column_min <COLUMNNAME> [<FILTER_EXPR>];

    如：remove statistic tbl_shop column_min revenue;

- 备注：
  - 该特征规则仅能作用于整型和浮点型的列之上，否则在设置时报错。
  - 当列中部分值为 NULL 时，NULL 不做统计。当列全部为 NULL 的话，则结果为 NULL
  - 利用 column_max 和 column_min 可以统计某列的值域范围。
  - FILTER_EXPR 是一个有效的表达式，表示只统计符合此表达式的行的值，参见“定义统计符合某个表达式的行数”
- 显示示例：

  column_min col3=101

  或（对于 partition 的表）：

  (col1='2011-10-25 00:00:00', col2=1)　　column_min col3: 3

定义统计符合某个表达式的行数：

- 别名：expression_condition
- 格式：整型
- 设置方法：
  - 添加信息项：

    add statistic <TABLENAME> expression_condition <EXPRESSIONDEFINITION>;

    如：add statistic tbl_shop expression_condition tbl_shop='hangzhou';
  - 移除信息项：

    remove statistic <TABLENAME> expression_condition <EXPRESSIONDEFINITION>;

    如：remove statistic tbl_shop expression_condition tbl_shop='hangzhou';
- 备注：

- <EXPRESSIONDEFINITION>定义了一个表达式，该表达式允许以列名或常量作为输入，支持“+”, “-”, “*”, “/”, “=”, “>”, “<”, “AND”, “OR”, “NOT”这些运算符的组合。表达式可以是列名与常量的运算，如“mount>500 AND mount<1000”，也可以是列名与列名的计算，如“start_date<end_date”。
- 表达式的定义应该满足 ODPS SQL 对表达式的限制。
- 该特征信息统计表中满足此表达式的记录数。
- 当表达式的计算结果为 NULL 时，不会被统计。
- 利用表达式定义，用户可以灵活的统计所需的统计值，如：
  - 某列非负的记录数：col_name>=0。
  - 某列为指定值的记录数，如一个字符串类型的字段，值为“BEIJING”的记录数：col_name=”BEIJING”。

- 示例：

添加表达式统计，

```
add statistic table1 expression_condition col3=”BEIJING”;
add statistic table1 expression_condition col3=”SHANGHAI”;
add statistic table1 expression_condition col3=”SHENZHEN”;
```

显示表达式统计，

```
expression_condition col3=‘BEIJING’: 1000
expression_condition col3=‘SHANGHAI’: 500
expression_condition col3=‘SHENZHEN’: 300
```

或（对于 partition 的表）：

(col1=‘2011-10-25 00:00:00’, col2=1)	expression_condition col3=‘BEIJING’: 1000
(col1=‘2011-10-25 00:00:00’, col2=1)	expression_condition col3=‘SHANGHAI’: 500
(col1=‘2011-10-25 00:00:00’, col2=1)	expression_condition col3=‘SHENZHEN’: 300

定义某一列记录的唯一值

- 别名：distinct_value
- 格式：整型
- 设置方法：该信息项在定义时需要指定 STATISTICRULE，格式为某一列或者多列的列名，多个列名之间以逗号分隔。
  - 添加信息项：

    add statistic <TABLENAME> distinct_value [<COLUMNNAME> , …];
  - 移除信息项：

    remove statistic <TABLENAME> distinct_value [<COLUMNNAME> , …];

- 显示示例：

  distinct_value col1: 100

或（对于 partition 的表），

  (col0='2011-10-25 00:00:00') distinct_value col1: 100

5.2. 查看统计项

当用户为某个表添加了特征信息后，可以查看已经为该表添加的特征信息的列表，命令为：

  show statistic_list <TABLENAME>;

如：show statistic_list tbl_shop;

其中，

- <TABLENAME>描述表名。

显示示例，

  TableName: table1

  table_count

  distinct_value col1

  distinct_value col2, col3

  expression_condition col1+col2<10

5.3. 查看统计数据

当用户为某个表添加了特征信息后，可以查看该表特征信息的统计值，要注意的是，查看统计数据的命令为：

  Show statistic <TABLENAME> [<PARTITION SPEC>];

如：Show statistic tbl_shop;

其中，

- <TABLENAME>描述表名。
- < PARTITION SPEC>可选，描述表中已存在的某个 partition 名称。当指定了 partition spec 后，显示该 partition 的统计信息；否则显示全表的统计信息，一般为所有 partition 统计信息的加和，或是集合中满足指定条件的某个值。

5.4. 查看可用统计项

用户可以查询有哪些特征信息可以统计，命令为：

show statistic_list;

5.5. 移除统计数据

用户也可以为某张表移除已经添加的特征信息，命令为：

  remove statistic <TABLENAME> <STATISTICNAME> [<STATISTICRULE>];

如：remove statistic tbl_shop table_count;

需要注意的是：

- 移除未添加的特征信息，或移除未添加的特征信息规则会报错，如当用户未指定为table1 统计 count 时，执行“remove statistic table1 count;”会报错。当用户仅仅指定“add statistic table2 distinct_value col1, col2;”时，执行”remove statistic table2 distinct_value col3;”会报错。
- 移除的特征信息要求跟增加时的特征信息完全一致，区分大小写和空格数，例如：对于“add statistic t expression_condition col1 > 1;”，执行“remove statistic t expression_condition COL1 > 1;”和“remove statistic t expression_condition COL1>1;”均会找不到移除的项。
- 当移除统计项时，相应已经有的统计信息也会被删除掉。

5.6. 执行统计过程

根据事先定义的统计项执行统计过程得到相应的统计值
语法：Analyze table <TABLENAME> <PARTITION SPEC> COMPUTE STATISTICS;
示例：

Analyze table tbl_shop compute statistics;
Analyze table t partition(pt='1') compute statistics

需要注意的是：

- 对于具有分区的表，只能对某个分区执行统计过程
- 对于不具有分区的表，只能对表执行统计过程

5.7. 使用实例

准备测试表

假设已有数据表 stat_pokes(foo int, bar string);

定义统计项

添加行数统计项
add statistic stat_pokes table_count;
添加表达式行数统计项
add statistic stat_pokes expression_condition foo > 100 and foo <= 200;

查看统计项

show statistic_list stat_pokes;
expression_condition foo > 100 and foo <= 200
table_count

计算统计项

analyze table stat_pokes compute statistics;
在计算完成后即可查看统计项的值

查看统计值

show statistic stat_pokes;
expression_condition foo > 100 and foo <= 200: 105
table_count: 500

5.8. 使用表统计信息的注意事项

1.如果修改列名，规则定义包含该列名的统计项会失效
2. 对于 add statistic 操作，要求用户具有 SELECT 权限

6. 字符串与转义

在 ODPS SQL 中的字符串常量可以用单引号或双引号表示，可以在单引号括起的字符串中包含双引号，或在双引号括起的字符串中包含单引号，否则要用转义符来表达，如以下表达方式都是可以的：

```
“I’m a happy manong!”
‘I\’m a happy manong!’
```

在 ODPS SQL 中反斜线\是转义符，用来表达字符串中的特殊字符，或将其后跟的字符解释为其本身，当读入字符串常量时，如果\后跟三位数字的时候，并且是三位有效的 8 进制数字，范围在\001 ~\177 之间，系统会根据 ASCII 值转为相应的字符。对于以下情况，则会将其解释为特殊字符

在 ODPS SQL 中能够用这种方式表达的字符有：

```
\b backspace
\t tab
\n newline
\r carriage-return
\’ 单引号
\”双引号
\\ 反斜线
\; 分号
\Z control-Z
\0 或 \00 结束符
```

select length(‘a\tb’) from dual; 结果是 3，表示字符串里实际有三个字符,\t 被视为一个字符。在转义符后的其它字符被解释为其本身，

select 'a\ab',length('a\ab') from dual;
结果是
‘aab’ 3
\a 被解释成了普通的 a

- LIKE
  在 LIKE 匹配时，%表示匹配任意多个字符，_表示匹配单个字符，如果要匹配%或_本身，则要对其进行转义, \\%匹配字符’%’， _匹配字符’_’。
  ‘abcd’ like ‘ab%’ –true

'abcd' like 'ab\%' – false

'ab%cd' like 'ab\\%%' – true

注：关于字符串的字符集，目前 ODPS 支持 UTF-8 的字符集，如果数据是以其它格式编码，可能计算出的结果不正确。

7. 正则表达式规范

ODPS SQL 中的正则表达式采用的是 PCRE 的规范，匹配时是按字符进行，支持的元字符如下：

元字符	说明
^	行首
$	行尾
.	任意字符
*	匹配零次或多次
+	匹配 1 次或多次
?	匹配零次或 1 次
?	匹配修饰符，当该字符紧跟在任何一个其他限制符（*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。
|	A|B A 或 B
(abc)*	匹配 abc 序列零次或多次
{n}或{m,n}	匹配的次数
[ab]	匹配括号中的任一字符 例中模式匹配 a 或 b 括号里的内容不可以为中文
[a-d]	匹配 a,b,c,d 任一字符, 括号里的内容不可以为中文
[^ab]	^表示非，匹配任一非 a 非 b 的字符 括号里的内容不可以为中文
[::]	见下表 POSIX 字符组
\	转义符
\n	n 为数字 1-9，后向引用
\d	数字
\D	非数字

POSIX 字符组

POSIX 字符组	说明	范围
[[:alnum:]]	字母字符和数字字符	[a-zA-Z0-9]
[[:alpha:]]	字母	[a-zA-Z]
[[:ascii:]]	ASCII 字符	[\x00-\x7F]

[[:blank:]]	空格字符和制表符	[\t]
[[:cntrl:]]	控制字符	[\x00-\x1F\x7F]
[[:digit:]]	数字字符	[0-9]
[[:graph:]]	空白字符之外的字符	[\x21-\x7E]
[[:lower:]]	小写字母字符	[a-z]
[[:print:]]	类似[:graph:]，但包括空白字符	[\x20-\x7E]
[[:punct:]]	标点符号	[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]
[[:space:]]	空白字符	[\t\r\n\v\f]
[[:upper:]]	大写字母字符	[A-Z]
[[:xdigit:]]	十六进制字符	[A-Fa-f0-9]

由于系统采用\作为转义符，因此正则表达式的 pattern 中出现的\都要进行二次转义。如正则表达式要匹配字符串 a+b，其中+是正则中的一个特殊字符，因此要用转义的方式表达，在正则引擎中的表达方式是 a\+b，由于系统还要解释一层转义，因此能够匹配该字符串的表达式是 a\\+b。

```
select a, a rlike 'a\\+b' from dual;

+---+---------+
| a | _c1   |
+---+---------+
| a+b | true |
+---+----------+
```

极端的情况，如果在要匹配字符\， 由于在正则引擎中\是一个特殊字符，因此要表示为\\，而系统还要对表达式进行一次转义，因此写成\\\\

```
select a, a rlike 'a\\\\b' from dual;

+---+------+
| a | _c1   |
+---+------+
| a\b | true |
+---+------+
```

如果字符串中有制表符 TAB，系统在读入'\t'这两个字符的时候，即已经将其存为一个字符，因此在正则的 pattern 中也是表达一个普通的字符。

```
select a, a rlike 'a\tb' from dual;

+---+------+
| a | _c1   |
+---+------+
```

```
| a        b | true |
+---+------+
```

8. 分区的裁剪优化

常量表达式是指在 plan 阶段就可以得出结果的表达式，如 substr('20120101000000',1,8)，而 substr(COLA, 1, 8)引用了一个列所以无法在 plan 阶段计算出固定的值。 系统可以在 plan 阶段计算出此类表达式的结果并对分区进行裁剪优化，如有一张分区表 TBLA, 分区键 pt，pt 是以日期的字符串形式来分区，如 20120101, 20120102 等，以下面的 QUERY 中

```
SELECT * FROM TBLA
WHERE PT>TO_CHAR(DATEADD(GETDATE()-1,'dd),'yyyymmdd)
```

在 plan 阶段系统即可计算出表达式的值并做好分区的过滤从而优化性能， 目前支持常量表达式优化的函数包括：

- SUBSTR
- GETDATE
- DATEADD
- TO_CHAR
- TO_DATE

注：目前裁剪优化中的日期相关的函数支持的格式还仅限于标准的 yyyy, mm, dd, hh, mi, ss, 对应年月日时分秒。

9. 工具

9.1. 血缘关系分析

血缘分析工具包是一个独立的 jar 包，输入是一个 sql 文件，将其解释成所有的输入表和输出表。

```
java -cp .:...:./* com.aliyun.lineage.app.Tool SQL 文件  table|column
```

加上参数 column 表示解析字段级别的血缘关系， table 表示表级别的关系，如果不加默认为 table.

如果要对 depl 脚本中的 SQL 语句进行血缘关系分析，可以加上-x 参数。

在表级别的血缘分析时，工具输出每个 sql 对应的源和目标表的关系，格式如

```
Sql1 = src:tgt
```

如果一个 sql 语句无法解析出源和目标表，则不输出，如 drop 语句，但是 sql 的编号是累加的。

如 sql 文件中的内容是:

```
Drop table a;
Insert overwrite b select * from c;
```

则输出是:

```
Sql2 = c:b
```

如果一个 sql 解析出多个源或目标，则每行只输出一个源和一个目标表的关系, 如 sql 文件内容：

```
from j join h on j.c=h.c join p on c.c=p.c
insert overwrite table a select *
insert overwrite table b select *;
```

则输出是:

```
sql1 = p:b
sql1 = p:a
sql1 = j:b
sql1 = j:a
sql1 = h:b
sql1 = h:a
```

血缘分析工具也支持对创建 VIEW 的语句进行分析，如文件的内容是:

```
Create view v_t as select * from t1;
```

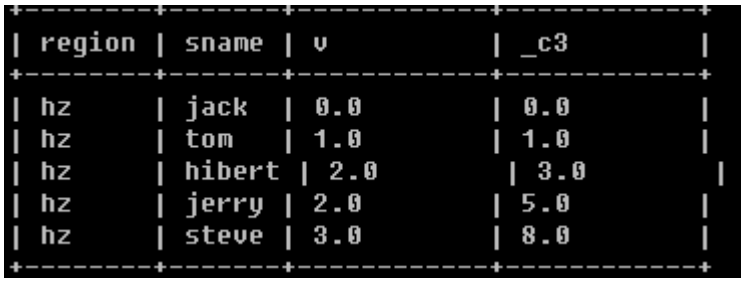
则表级血缘关系分析的输出是：

```
sql1=t1:v_t
```

10.使用注意事项

- 浮点数精度
  由于浮点数内部的存储机制，浮点数运算的结果会有极小差异，当参与计算的浮点数足够多时，这种差异会累积到肉眼可见的程度。有时会导致同一个 SQL 的计算结果中的浮点数在几次计算中不一致。如果需要一致的结果，建议用 BIGINT 类型。
- 由于 double 和 bigint 能表达的有效位数不同，bigint 转为 double 时可能会丢失精度。
- 两个 double 类型判断相等时取决于两个值的精度，更好的做法是用差值的绝对值小于某个范围。
- concat 一个输入为 NULL 则结果为 NULL
- instr 函数，如果要搜索的子串是''，则在任意位置总能匹配成功。
  Instr('abc','') = 1
  Instr('abc','',2) = 2
  Instr('abc',2,2) = 3
- In 操作 左边为 NULL 时，返回 NULL
  0 in (0,1, null) →true
  1 in (0,1,null) →true
  null in (0,1,null) →null
  当列表中有 null，而且左侧值不在右边列表中时，返回 NULL，
  0 in (0,1,null) →true
  2 in (0,1,null) →null
- 在一个 multiinsert 中，如果对于一张表的不同 partition，既有 insert into 又有 insert overwrite，则结果都是 insert into.
- 在 where 条件中，如果条件值为 null 时该行不会返回，对其取 NOT 也不会返回。
  如下面两行都不会有返回数据：
  Select * from dual where 1=null;
  Select * from dual where not (1=null);

- Length 函数，对于输入非 UTF-8 的字符串，返回-1
- 窗口函数中加了 order by 时，在行中有重复值时与 oracle 行为是不同的。
  如:
  select region,sname,v, sum(v) over(partition by region order by v) from zt;
  结果行中有重复的值也是累加的,结果中 hibert 和 jerry 对应的 v 都是 2，sum 值是从上到下累加，而不是相等。



region	sname	v	_c3
hz	jack	0.0	0.0
hz	tom	1.0	1.0
hz	hibert	2.0	3.0
hz	jerry	2.0	5.0
hz	steve	3.0	8.0