YashanDB 开发手册

v23.2.2.0

2024年5月

深圳计算科学研究院

深圳崖山科技有限公司

SQL参考手册

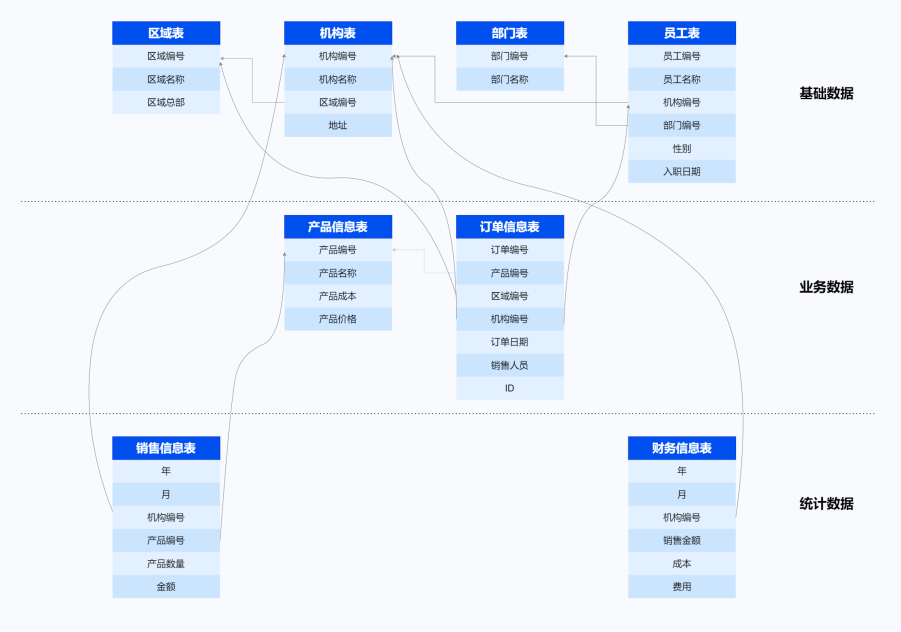
SQL（Structured Query Language）是开发者操作数据库的主要接口，本手册从数据类型、运算符、函数、SQL语句等方面全面介绍YashanDB对于SQL功能的实现。

手册中使用的示例说明

- 示例中的SQL、PL语句均在yasql工具中运行，并以yasql工具的输出结果作为样例进行展示。
- 示例中使用DBMS_OUTPUT.PUT_LINE语句向yasql工具端打印输出，为达到输出效果，需要保证在yasql工具里已运行set serveroutput on来打开控制输出的开关，详见工具手册yasql基本功能使用指导。
- 系统对浮点类型及NUMBER类型数据输出时的显示宽度默认为10，可通过SET NUMWIDTH调节显示宽度。
- 关于日期的显示格式：为尽量模拟实际业务，本手册示例库的日期格式被设置为'YYYY-MM-DD HH24:MI:SS'，如DATE_FORMAT配置参数不是按此格式设置，所看到的日期格式将与本手册不一致。
- 关于示例中使用的通用表：本手册使用sales作为示例用户（参考CREATE USER创建用户），同时在该用户下创建一套样例表（样例表来源于下面所列的一套极简化的业务模型，并依据不同架构/存储特性进行相应调整）。示例中使用的通用表即来自于这套样例表，当执行示例时，切换到sales用户下将可以查询到这些表，产品文档中附：样例表展示了这些表的创建脚本。

> **Note**：
> 为达到特定目的，示例可能需要创建其他表或者修改通用表的结构、数据，这些将不在此处体现，而是查看具体示例。

样例业务模型




```
--区域信息表：area、area1
--机构信息表：branches、branches1
--部门信息表：department
--员工信息表：employees
--产品信息表：product
--订单信息表：orders_info
--销售信息表：sales_info、sales_info_range、sales_info_list、sales_info_hash
--财务信息表：finance_info
```

样例表类型说明

YashanDB支持多种部署形态（单机/分布式/共享集群）和多种表类型（HEAP/TAC/LSC），并提供DEFAULT_TABLE_TYPE参数用于配置创建表对象时的默认表类型，同时还提供ORGANIZATION语法用于创建一个指定表类型的表对象（查看CREATE TABLE了解该语法详细说明）。

本手册示例中所使用的样例表的类型依据系统部署形态，和DEFAULT_TABLE_TYPE参数的值确定。当示例或示例所在前提中未注明语句适用于某一种部署形态或某一种表类型时，表示该示例语句可以在所有部署形态中以任一种表类型运行；否则，该示例语句应在注明的部署形态环境中，并以注明的表类型运行，不合适的部署形态环境或表类型可能导致示例语句运行异常，或产生与预期不一致的输出结果。

内置函数

函数是将零个或多个参数输入，执行特定的计算并将结果返回的操作，其一般形式可以表示为`function_name(argument1,argument2, ...)`。

函数支持数据类型的隐式转换：当输入参数的类型与函数预期的参数类型不同时，YashanDB会先尝试将其转换为可接受的类型，然后再进行计算。因此，在YashanDB中，函数的参数均可以为通用表达式（expr）（除非在函数定义中特殊说明）且表达式的结果不需要为函数要求的参数类型，只需满足系统隐式转换后符合参数类型的要求即可。

本章节介绍YashanDB中的内置函数，PL参考手册中将对用户自定义函数（User-Defined Function）进行介绍，可查看V$FUNCTION视图了解YashanDB提供的所有内置函数信息。

聚集函数（Aggregate Function）

聚集函数是对多行数据运算并返回一行数据的函数，此函数可以出现在SELECT、ORDER BY和HAVING语句中，并通常与GROUP BY语句组合使用。

- 当没有指定GROUP BY语句时，聚集函数对所有行进行计算，并只返回一行计算结果。
- 当指定了GROUP BY语句时，聚集函数按GROUP BY后面的值进行分组计算，并对每个值返回一行计算结果。
- 当指定了GROUP BY语句，并同时指定了HAVING语句时，对上一条返回的多行计算结果按照HAVING后面的条件进行筛选再返回。

使用聚集函数时应注意以下几点：

- 聚集函数都具有确定性，任何时候用一组给定的输入值调用它们时，都返回相同的值。
- 向聚集函数输入的参数必须有且只能有一个。例如`COUNT(c1)`。
- 除了`COUNT(*)`、`COUNT(常量)`，其他聚集函数会忽略NULL；当所有行均为NULL时，这些函数返回NULL。
- 聚集函数不允许嵌套，即聚集函数的参数不可以为聚集函数。
- 聚集函数不允许与FOR UPDATE语句组合使用。
- 部分聚集函数可通过使用OVER关键字作为窗口函数使用，详情见各函数章节。

YashanDB提供如下聚集函数：

AVG

COUNT

GROUPING

GROUPING_ID

GROUP_CONCAT

GROUP_ID

LISTAGG

MAX

MIN

STDDEV

STDDEV_POP

STDDEV_SAMP

STRING_AGG

SUM

VARIANCE

VAR_POP

VAR_SAMP

WM_CONCAT

日期函数（Datetime Function）

日期函数是分别对每行数据运算并对其返回一行数据的函数，此类函数的参数或返回结果中包含日期时间型数据。

YashanDB提供如下日期函数：

ADD_MONTHS

AGE

CURRENT_TIMESTAMP

DATE

DATE_ADD

DATE_FORMAT

DAYOFWEEK

EXTRACT

LAST_DAY

MONTHS_BETWEEN

NOW

SYSDATE

SYSTIMESTAMP

TIME

TIMEDIFF

TIMESTAMP

TIMESTAMPDIFF

UTC_TIMESTAMP

字符函数（Character Function）

字符函数是分别对每行数据运算并对其返回一行数据的函数，此类函数的操作对象主要为字符型数据。

YashanDB提供如下字符函数：

ASCII

BIT_LENGTH

CHAR_LENGTH/CHARACTER_LENGTH

CHR

CONCAT

CONCAT_WS

FIND_IN_SET

INITCAP

INSTR

INSTRB

LEFT

LENGTH/LENGTHB

LENGTH2

LOWER

LPAD

LTRIM

NLSSORT

OCTET_LENGTH

POSITION

REGEXP_COUNT

REGEXP_INSTR

REGEXP_LIKE

REGEXP_REPLACE

REGEXP_SUBSTR

REPLACE

RIGHT

RLIKE_FILTER

RPAD

RTRIM

SPLIT

STRPOS

SUBSTR

SUBSTRB

SUBSTRING

SUBSTRING_INDEX

TO_BASE64

TRANSLATE

TRIM

UNISTR

UPPER

数学函数（Mathematical Function）

数学函数是分别对每行数据运算并对其返回一行数据的函数，此类函数主要对数值型数据进行数学运算，并返回一个数值型数据。

YashanDB提供如下数学函数：

ABS

ACOS

ASIN

ATAN

ATAN2

BITAND/BITOR/BITXOR

CEIL

COS

COT

DIV

EXP

FLOOR

LN

LOG

MOD

PI

POW/POWER

RANDOM

ROUND

SIGN

SIN

SINH

SQRT

TAN

TANH

TRUNC

转换函数（Conversion Function）

转换函数是分别对每行数据运算并对其返回一行数据的函数，此类函数将数据从一种类型转换到另一种类型。

YashanDB提供如下转换函数：

BIN

BIN_TO_NUM

CAST

HEXTORAW

NUMTODSINTERVAL

NUMTOYMINTERVAL

ROWIDTOCHAR

SCN_TO_TIMESTAMP

TIMESTMAP_TO_SCN

TO_CHAR

TO_DATE

TO_DSINTERVAL

TO_NUMBER

TO_TIMESTAMP

TO_YMINTERVAL

窗口函数（Window Function）

窗口为标准的SQL术语，表示一个数据行的集合，窗口函数则是系统提供的用于操作这个数据行集合的各种功能，其与聚集函数的区别是，聚集函数的每个分组返回一个值，而窗口函数的每个分组按行返回多个值。

窗口函数的通用语法可表述为：（[]表示可省略）

```
window_function::= WINDOW_FUNC([func_parameter]) OVER ([query_partition_clause] [order_by_clause [windowing_clause]])
```

```
query_partition_clause::= PARTITION BY ((expr) {"," (expr)}|("(" (expr) {"," (expr)} ")"))
```

query_partition_clause用于对表数据进行分组，可省略，则表示所有数据为一个分组。

expr不能为NULL，但可以为：

- 列字段
- 常量表达式：可以为任意常量值，等价于不分组

```
order_by_clause::= ORDER BY ((expr | position | c_alias) [ASC|DESC] [(NULLS FIRST) | (NULLS LAST)]) {"," ((expr | position |
c_alias) [ASC|DESC] [(NULLS FIRST) | (NULLS LAST)])}
```

order_by_clause用于对分组内的数据行进行排序，可省略。语法同SELECT语句中的order_by_clause描述。

```
windowing_clause::= (ROWS|RANGE) ((BETWEEN (UNBOUNDED PRECEDING|CURRENT ROW|value_expr(PRECEDING|FOLLOWING)) AND (UNBOUNDED
FOLLOWING|CURRENT ROW|value_expr (PRECEDING|FOLLOWING)))|(UNBOUNDED PRECEDING|CURRENT ROW|value_expr PRECEDING))
```

windowing_clause用于对分组内的数据行进行进一步的筛选，每个分组通过BETWEEN ...AND自定义各自的窗口行范围，即滑动窗口，只有部分窗口函数可以使用此功能。基于此特性，此功能需要与order_by_clause结合使用，保证筛选的基础为有序的行集合。

- ROWS：使用相对当前行偏移行数来计算行范围。
- RANGE：使用相对当前行的值差异来计算行范围，此功能要求order_by_clause排序值为数值型或日期时间型。
- UNBOUNDED PRECEDING：以分组的第一行作为开始行。
- UNBOUNDED FOLLOWING：以分组的最后一行作为结束行。
- CURRENT ROW：以当前行作为开始或者结束行。
- value_expr (PRECEDING|FOLLOWING)：通过value_expr来计算开始行和结束行（ROWS），或者开始行的值和结束行的值（RANGE），在列存表上只支持常量。例如，ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING表示以当前行的前一行作为开始行和后一行作为结束行，窗口函数基于此范围执行分析统计。

使用窗口函数时须遵循如下限制：

- 窗口函数不能被嵌套，即不能使用窗口函数作为窗口函数的参数。
- 窗口函数一般应用于SELECT语句中的select_list，或ORDER BY语句中。

YashanDB提供如下专用窗口函数：

DENSE_RANK

FIRST_VALUE

LAG

LAST_VALUE

LEAD

RANK

ROW_NUMBER

MEDIAN

同时，部分其他类型的函数也可作为窗口函数使用，通过OVER关键字识别，例如：

AVG

COUNT

LISTAGG

MAX

MIN

SUM

可查看V$WINDOW_FUNCTION视图了解YashanDB提供的所有窗口函数信息。

数组函数（Varray Function）

数组函数指的是对数组（数组变量，数组对象，数组类型）进行操作或生成数组对象的函数，即其入参或返回值中至少包含一个数组类型值。

ARRAY_APPEND

ARRAY_LENGTH

ARRAY_NDIMS

ARRAY_POSITION

ARRAY_REMOVE

ARRAY_REPLACE

ARRAY_TO_STRING

ARRAY_UPPER

STRING_TO_ARRAY

内置表函数（Table Function）

内置表函数是指以table()语法调用的一类函数，函数内部可以输入不同的标识符，实现不同类型数据的聚集查询。

PX_CHANNEL

PX_OBJ

地理信息处理函数（GIS Function）

地理信息处理函数是指对空间地理信息进行处理的一系列函数，仅支持在单机HEAP表中使用，更多详情请查阅GIS Function。

其他函数（Other Function）

COALESCE

CHECK_SYS_PRIVILEGE

DECODE

DECRYPT_AES128

EMPTY_BLOB

EMPTY_CLOB

ENCRYPT_AES128

GET_TYPE_NAME

GREATEST

IF

IFNULL

ISNULL

JSON

JSON_ARRAY_GET

JSON_ARRAY_LENGTH

JSON_EXISTS

JSON_QUERY

JSON_SERIALIZE

LEAST

LNNVL

LOCALTIME

LOCALTIMESTAMP

MD5

NEXT_DAY

NULLIF

NVL

NVL2

TYPEOF

UNSUPPORT_ERROR

USERENV

SYS_CONNECT_BY_PATH

SYS_CONTEXT

SYS_EXTRACT_UTC

SYS_GUID

SQLCODE

SQLERRM

SOUNDEX

ABS

```
abs::= ABS "(" expr ")"
```

ABS函数计算一个数值的绝对值，其返回值的数据类型为：

参数类型	返回值
TINYINT	SMALLINT
SMALLINT	INT
INT	BIGINT
BIGINT	BIGINT/NUMBER
NUMBER	NUMBER
FLOAT	FLOAT
DOUBLE	NUMBER
CHAR	NUMBER
VARCHAR	NUMBER
NCHAR	NUMBER
NVARCHAR	NUMBER

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT ABS(-2.345) abs1,
ABS(1/10) abs2,
ABS(1-2*3) abs3,
ABS(TO_NUMBER('$3.33','$9.99')) abs4
FROM DUAL;

      ABS1        ABS2                ABS3        ABS4
---------- ----------- -------------------- -----------
     2.345          .1                    5        3.33
```

ACOS

```
acos::= ACOS  "(" expr ")"
```

ACOS函数计算给定参数的反余弦值，参数为弧度表示，大小在区间[-1,1]，函数将返回一个大小在区间[0,pi]的DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT ACOS(-1) res FROM DUAL;
        RES
-----------
 3.142E+000

SELECT ACOS(0) res FROM DUAL;
        RES
-----------
 1.571E+000

SELECT ACOS(1) res FROM DUAL;
        RES
-----------
          0
```

ADD_MONTHS

```
add_months::= ADD_MONTHS "(" date_expr "," integer_expr ")"
```

ADD_MONTHS函数计算date_expr表示的日期加上integer_expr表示的月数（可以为负数），返回一个DATE类型的日期值。

date_expr

YashanDB认可的通用表达式，date_expr的值必须为DATE、TIMESTAMP或者字符型数据。

- 当date_expr为字符型时，必须确保字符串符合当前DATE类型的格式要求，否则返回错误。
- 当date_expr为NULL时，函数返回NULL。

integer_expr

YashanDB认可的通用表达式，integer_expr的值必须为NUMBER类型，或可转换为NUMBER的字符型。

integer_expr的值存在小数时：

- 对于浮点数（FLOAT和DOUBLE），奇进偶舍至整数。
- 对于非浮点数，直接将小数舍去，留下整数部分。

天的特殊处理

由于存在大小月的情况，对天做如下特殊处理：

- 当date_expr的DAY值为所在月最后一天时，函数返回日期的DAY值也为其所在月的最后一天。
- 当date_expr的DAY值大于加上integer_expr后所得月份的最后一天时，函数返回日期的DAY值为该月份的最后一天。

示例

```
SELECT ADD_MONTHS(DATE'2021-5-31',-3) res FROM DUAL;
RES
--------------------------------
2021-02-28


SELECT ADD_MONTHS(DATE'2022-1-30',1) res FROM DUAL;
RES
--------------------------------
2022-02-28
```

AGE

```
age::= AGE "(" expr1 ["," expr2] ")"
```

AGE函数用于计算年龄，可以接受一个或两个参数，并返回一个INTERVAL YEAR TO MONTH类型的数值。

本函数遵循如下规则：

- 一个参数时，函数使用当前时间减去该参数值，获得时间差。
- 两个参数时，函数使用第一个参数值减去第二个参数值，获得时间差。

时间差计算规则：

- 年差：当月差达到12个月时，进位为1年。
- 月差：按两个数的月、日分别对比，月相等时，月差为0；月不等时（假设相差A），对比日/时/分/秒/微秒：
  - A为正数时，被减数值大于等于减数值时，月差为A，否则为A-1。
  - A为负数时，被减数值小于等于减数值时，月差为A，否则为A+1。

expr1、expr2

expr1、expr2为通用表达式，其值须为DATE、TIMESTAMP类型，或可以转换为DATE、TIMESTAMP类型的字符型。

- 对于其他类型，函数返回类型错误信息。
- 当expr1和expr2中任意一个为NULL时，函数返回NULL。

示例

```
SELECT AGE('2000-1-1') res FROM DUAL;
RES
---------------
+22-08

--月差达到12个月时，进位为1年
SELECT AGE('2020-7-31 09:08:00.72','2021-7-31 09:08:00.99') res FROM DUAL;
RES
---------------
-01-00

--月值差为负数时
SELECT AGE('2020-7-31 09:08:00.72','2021-8-31 09:08:00.66') res FROM DUAL;
RES
---------------
-01-00

SELECT AGE('2020-7-31 09:08:00.72','2021-8-31 09:08:00.99') res FROM DUAL;
RES
---------------
-01-01

--月值差为正数时
SELECT AGE('2022-7-31 09:08:00.72','2021-8-31 09:08:00.66') res FROM DUAL;
RES
---------------
+00-11

SELECT AGE('2022-7-31 09:08:00.72','2021-8-31 09:08:00.99') res FROM DUAL;
RES
---------------
+00-10
```

ARRAY_APPEND

```
array_append::= ARRAY_APPEND "(" array_var "," new_member ")"
```

ARRAY_APPEND函数将new_member添加到数组（数组变量，数组对象，数组类型）array_var的末尾，并返回添加后的数组结果。

本函数遵循如下规则：

- 在PL中可以当作特殊的数组初始化函数使用。
- 函数结果不能作为INSERT语句的value使用。
- 不能对包含非标量类型成员的数组使用本函数。
- 执行本函数后，array_var中的成员数量并不会发生变化。
- 该函数不支持列式计算。

array_var

数组变量，其值可以为：

- 一个存在的数组，数组成员必须为普通标量数据类型。
- NULL，此时将会生成一个以new_member的类型作为成员类型的数组（new_member不能为NULL），且新数组仅包含new_member一个成员。

new_member

通用表达式，其类型必须为与array_var数组成员相同，或可进行隐式转换的数据类型。

当array_var不为NULL时，new_member可以为NULL，表示向array_var数组末尾添加一个NULL成员。

示例

```
SET serveroutput ON

DECLARE
  TYPE arr_type IS VARRAY(10) OF CHAR(5);
  a arr_type := arr_type('a', 'b', 'c');
  b arr_type;
BEGIN
  -- 向a的末尾添加一个成员'd'，生成临时数组并将该数组赋值给b
  b := ARRAY_APPEND(a, 'd');
  FOR i IN 1 .. b.COUNT LOOP
    IF b(i) IS NULL THEN
      DBMS_OUTPUT.PUT_LINE(i||' is NULL');
    ELSE
      DBMS_OUTPUT.PUT_LINE(b(i));
    END IF;
  END LOOP;
END;
/
a
b
c
d

-- 首个参数为NULL时生成新数组
DECLARE
  TYPE arr_type IS VARRAY(10) OF CHAR(5);
  b arr_type;
BEGIN
  -- 生成新数组
  b := ARRAY_APPEND(NULL, 'abcd');
  FOR i IN 1 .. b.COUNT LOOP
    IF b(i) IS NULL THEN
      DBMS_OUTPUT.PUT_LINE(i||' is NULL');
    ELSE
      DBMS_OUTPUT.PUT_LINE(i||' is: '||b(i));
    END IF;
```

```
  END LOOP;
END;
/
1 is: abcd
```

ARRAY_LENGTH

```
array_length::= ARRAY_LENGTH "(" array_var "," dimension ")"
```

ARRAY_LENGTH函数对数组（数组变量，数组对象，数组类型）array_var计算按dimension指定维度的数组长度（即成员数量），返回一个INT类型的数值。

本函数遵循如下规则：

该函数不支持列式计算。

array_var

一个已初始化的数组变量，不可为NULL。

dimension

数组维度，即数组嵌套深度，为一个通用表达式，其值类型必须为INT（超过INT值域范围时报错）。

- 当dimension为NULL时，函数返回NULL。
- 数组嵌套深度从1开始，当dimension小于1或者大于最大深度时，函数返回NULL。
- 当在同一dimension存在多个数组时，函数返回最长的数组长度。

示例

```
--单层数组
SELECT ARRAY_LENGTH(STRING_TO_ARRAY('a,b,c,d,e', ','), 1) len FROM DUAL;
         LEN
------------
           5

SELECT ARRAY_LENGTH(STRING_TO_ARRAY('a,b,c,d,e', ','), NULL) len FROM DUAL;
         LEN
------------


SELECT ARRAY_LENGTH(STRING_TO_ARRAY('a,b,c,d,e', ','), -1) len FROM DUAL;
         LEN
------------


--多层数组
SET serveroutput ON
DECLARE
  TYPE arr_type_2 IS VARRAY(10) OF CHAR(5);
  TYPE arr_type_1 IS VARRAY(10) OF arr_type_2;
  b arr_type_1;
BEGIN
    b := arr_type_1(STRING_TO_ARRAY('a1,b1,c1', ',')
                   ,STRING_TO_ARRAY('a2,b2', ',')
                   ,STRING_TO_ARRAY('a3,b3,c3,d3', ','));
    DBMS_OUTPUT.PUT_LINE('Dimension 1 has:'||ARRAY_LENGTH(b,1));
    DBMS_OUTPUT.PUT_LINE('Dimension 2 has:'||ARRAY_LENGTH(b,2));
END;
/
Dimension 1 has:3
Dimension 2 has:4
```

ARRAY_NDIMS

```
array_ndims::= ARRAY_NDIMS "(" array_var ")"
```

ARRAY_NDIMS函数计算数组（数组变量，数组对象，数组类型）array_var的维度数（嵌套深度），返回一个INT类型的数值。

本函数遵循如下规则：

该函数不支持列式计算。

array_var

一个已初始化的数组变量，当array_var为NULL时，函数返回NULL。

示例（HEAP表）

```
SET serveroutput ON

CREATE OR REPLACE TYPE arr_udt_1 IS VARRAY(5) OF INT;
/

CREATE OR REPLACE TYPE arr_udt_2 IS VARRAY(5) OF arr_udt_1;
/

DECLARE
  a arr_udt_1 := arr_udt_1(2);
  b arr_udt_2 := arr_udt_2(a);
  c INT;
BEGIN
  DBMS_OUTPUT PUT_LINE('Dimension of b: '||ARRAY_NDIMS(b));
  DBMS_OUTPUT PUT_LINE('Dimension of b(1): '||ARRAY_NDIMS(b(1)));
END;
/
Dimension of b: 2
Dimension of b(1): 1
```

ARRAY_POSITION

```
array_position::= ARRAY_POSITION "(" array_var "," compare_member [ "," start_locate ] ")"
```

ARRAY_POSITION函数以start_locate为起点查找数组（数组变量，数组对象，数组类型）array_var的成员中第一个出现compare_member的位置，并返回INT类型的该位置值。

该函数不支持列式计算。

array_var

数组变量，其值可以为：

- 一个存在的数组，数组成员必须为普通标量数据类型。
- NULL，此时函数返回NULL。

compare_member

通用表达式，其值类型必须为与array_var数组成员可进行比较的类型，具体参考YashanDB的比较运算规则。

start_locate

通用表达式，不可为NULL，其值类型须为INT，或可隐式转换为INT。

当start_locate为负数，或者超过数组长度时，函数返回NULL。

示例（HEAP表）

```
CREATE OR REPLACE TYPE arr_position_type IS VARRAY(10) OF CHAR(5);
/

SELECT ARRAY_POSITION(arr_position_type('a', 'b', 'c', 'd'), 'c') pos FROM DUAL;
         POS
------------
           3

SELECT ARRAY_POSITION(arr_position_type('a', 'b', NULL, 'd', 'c'), NULL) pos FROM DUAL;
         POS
------------
           3

SELECT ARRAY_POSITION(arr_position_type('a', 'b', 'c', 'd', 'c'), 'c', 4) pos FROM DUAL;
         POS
------------
           5

SELECT ARRAY_POSITION(arr_position_type('a', 'b', 'c', 'd', 'c'), 1, 4) pos FROM DUAL;
         POS
------------
```

ARRAY_REMOVE

```
array_remove::= ARRAY_REMOVE "(" array_var "," compare_member ")"
```

ARRAY_REMOVE函数删除数组（数组变量，数组对象，数组类型）array_var中所有与compare_member相同的成员，并返回删除后的数组。

本函数遵循如下规则：

- 在PL中可以作为数组类型变量的初始化函数。
- 函数结果不能作为INSERT语句的value使用。
- array_var的成员类型或compare_member的类型为非标量类型时报错。
- 数组成员的删除不影响原有的数组变量，仅影响该函数的返回值。
- 该函数不支持列式计算。

array_var

数组变量，其值可以为：

- 一个存在的数组，数组成员必须为普通标量数据类型。
- NULL，此时函数返回NULL。

compare_member

通用表达式，其值类型必须为与array_var数组成员可进行比较的类型，具体参考YashanDB的比较运算规则。

示例

```
SET serveroutput ON

DECLARE
  TYPE arr_type IS VARRAY(10) OF CHAR(5);
  arr  arr_type := arr_type('123', '234', '333', '234', '23');
  --使用ARRAY_REMOVE函数对数组初始化
  res  arr_type := ARRAY_REMOVE(arr, 234);
BEGIN
  FOR i IN 1 .. arr.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('arr'||i||': '||arr(i));
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('---------------');
  FOR i IN 1 .. res.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('res'||i||': '||res(i));
  END LOOP;
END;
/
arr1: 123
arr2: 234
arr3: 333
arr4: 234
arr5: 23
---------------
res1: 123
res2: 333
res3: 23
```

ARRAY_REPLACE

```
array_replace::= ARRAY_REPLACE "(" array_var "," compare_member "," replace_member ")"
```

ARRAY_REPLACE函数将数组（数组变量，数组对象，数组类型）array_var中所有与compare_member相同的成员替换为replace_member，并返回替换后的数组。

本函数遵循如下规则：

- 当array_var的成员类型或compare_member的类型为非标量类型时报错。
- 该函数在PL中可以作为数组类型变量的初始化函数。
- 函数结果不能作为INSERT语句的value使用。
- 数组成员的替换不影响原有的数组变量，仅影响该函数的返回值。
- 该函数不支持列式计算。

array_var

数组变量，其值可以为：

- 一个存在的数组，数组成员必须为普通标量数据类型。
- NULL，此时函数返回NULL。

compare_member

通用表达式，其值类型必须为与array_var数组成员可进行比较的类型，具体参考YashanDB的比较运算规则。

replace_member

通用表达式，其类型必须为与array_var数组成员相同，或可进行隐式转换的数据类型。

示例

```
SET serveroutput ON

DECLARE
  TYPE arr_type IS VARRAY(10) OF CHAR(5);
  arr arr_type := arr_type('a', 'b', 'cc', 'ddd', 'e');
  --使用ARRAY_REPLACE函数对数组初始化
  res arr_type := ARRAY_REPLACE(arr, 'cc', NULL);
BEGIN
  FOR i IN 1 .. res.COUNT LOOP
    IF res(i) IS NULL THEN
      DBMS_OUTPUT.PUT_LINE(i ||' is NULL');
    ELSE
      DBMS_OUTPUT.PUT_LINE(i ||': '|| res(i));
    END IF;
  END LOOP;
END;
/
1: a
2: b
3 is NULL
4: ddd
5: e
```

ARRAY_TO_STRING

```
array_to_string::= ARRAY_TO_STRING "(" array_var "," split_string [ "," replace_string ] ")"
```

ARRAY_TO_STRING函数将数组（数组变量，数组对象，数组类型）array_var的成员以split_string作为分隔符进行连接，若数组成员中存在NULL且replace_string非空，则将NULL成员替换为replace_string后进行连接，函数返回一个VARCHAR类型的字符串。

该函数不支持列式计算。

array_var

一个已初始化的数组变量，不可为NULL。

array_var的数组成员（当array_var为多层嵌套数组时，为最底层的数组成员）须为字符型，或可隐式转换为字符型的其他类型。

split_string

通用表达式，须为字符型，或可隐式转换为字符型的其他类型。split_string为NULL表示无分隔符连接。

replace_string

通用表达式，须为字符型，或可隐式转换为字符型的其他类型。

示例（HEAP表）

```
CREATE OR REPLACE TYPE a2s_type IS VARRAY(20) OF CHAR(20);
/

SELECT ARRAY_TO_STRING(a2s_type('aaa','bbb','ccc'), ' ') res FROM DUAL;
RES
----------------------------------------------------------------
aaa bbb ccc

SELECT ARRAY_TO_STRING(a2s_type('aaa','bbb', NULL, 'ccc'), ',') res FROM DUAL;
RES
----------------------------------------------------------------
aaa bbb ccc

SELECT ARRAY_TO_STRING(a2s_type('aaa','bbb', NULL, 'ccc'), ',', 'NULL') res FROM DUAL;
RES
----------------------------------------------------------------
aaa,bbb,NULL,ccc

SELECT ARRAY_TO_STRING(a2s_type('aaa','bbb', NULL, 'ccc'), NULL) res FROM DUAL;
RES
----------------------------------------------------------------
aaabbbccc

SELECT ARRAY_TO_STRING(a2s_type('aaa','bbb', NULL, 'ccc'), ',', NULL) res FROM DUAL;
RES
----------------------------------------------------------------
aaa,bbb,ccc
```

ARRAY_UPPER

```
array_upper::= ARRAY_UPPER "(" array_var "," dimension ")"
```

ARRAY_UPPER函数对数组（数组变量，数组对象，数组类型）array_var计算按dimension指定维度的成员数量上限，返回一个INT类型的数值。

该函数不支持列式计算。

array_var

一个已初始化的数组变量，不可为NULL。

dimension

数组维度，即数组嵌套深度，为一个通用表达式，其值类型必须为INT（超过INT值域范围时报错）。对该值的计算规则为：

- 当dimension为NULL时，函数返回NULL。
- 数组嵌套深度从1开始，当dimension小于1或者大于最大深度时，函数返回NULL。

示例

```
--单层数组
SELECT ARRAY_UPPER(STRING_TO_ARRAY('a,b,c,d,e', ','), 1) LIMIT FROM DUAL;
       LIMIT
------------
  2147483647

SELECT ARRAY_UPPER(STRING_TO_ARRAY('a,b,c,d,e', ','), NULL) LIMIT FROM DUAL;
       LIMIT
------------


SELECT ARRAY_UPPER(STRING_TO_ARRAY('a,b,c,d,e', ','), -1) LIMIT FROM DUAL;
       LIMIT
------------


--多层数组
SET serveroutput ON
DECLARE
  TYPE arr_type_2 IS VARRAY(10) OF CHAR(5);
  TYPE arr_type_1 IS VARRAY(10) OF arr_type_2;
  b arr_type_1;
BEGIN
    b := arr_type_1(STRING_TO_ARRAY('a1,b1,c1', ',')
                   ,STRING_TO_ARRAY('a2,b2', ',')
                   ,STRING_TO_ARRAY('a3,b3,c3,d3', ','));
    DBMS_OUTPUT.PUT_LINE('Dimension 1 has:'||ARRAY_UPPER(b,1));
    DBMS_OUTPUT.PUT_LINE('Dimension 2 has:'||ARRAY_UPPER(b,2));
END;
/
Dimension 1 has:10
Dimension 2 has:10
```

ASCII

```
ascii::= ASCII "(" expr ")"
```

ASCII函数将expr表示的一个字符转换为ASCII码，返回一个INT类型的数值。本函数遵循如下规则：

- 只支持对有ASCII编码的字符进行转换，其他如中文等非ASCII编码的字符不能转换。
- expr的值必须为数值型，布尔型，日期时间型，或字符型，否则返回类型不支持。
- expr的值如为多字符，不会报错，函数将只对第一个字符进行ASCII转换并返回。
- 当expr的值为NULL时，函数返回NULL。

示例

```
--函数将数值23.35转换为字符'23.35',并返回首字符'2'的ASCII值
SELECT ASCII(23.35) result,
TYPEOF(ASCII(23.35)) result_type
FROM DUAL;
      RESULT RESULT_TYPE
------------ -----------
          50 integer
```

ASIN

```
asin::= ASIN  "(" expr ")"
```

ASIN函数返回给定参数的反正弦值，参数以弧度表示，大小在区间[-1,1]，函数将返回一个大小在区间[-pi/2,pi/2]的DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT ASIN(0) res FROM DUAL;
        RES
-----------
          0

SELECT ASIN(-1) res FROM DUAL;
        RES
-----------
 -1.57E+000

SELECT ASIN(1) res FROM DUAL;
        RES
-----------
 1.571E+000

SELECT ASIN(0.5) res FROM DUAL;
        RES
-----------
 5.236E-001
```

ATAN

```
atan::= ATAN  "(" expr ")"
```

ATAN函数返回给定参数的反正切值，参数为以弧度表示的角度，大小本身无限制（只受限于其所属数据类型所规定范围），函数返回一个大小在区间[-pi/2,pi/2]的DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT ATAN(-1) res FROM DUAL;
        RES
-----------
 -7.85E-001

SELECT ATAN(0) res FROM DUAL;
        RES
-----------
          0

SELECT ATAN(1) res FROM DUAL;
        RES
-----------
 7.854E-001
```

ATAN2

```
atan2::= ATAN2  "(" expr1 "," expr2 ")"
```

ATAN2函数返回给定参数 expr1/expr2 的结果的反正切值，参数以弧度表示，大小本身无限制（只受限于其所属数据类型所规定范围），函数将返回一个大小在区间[-pi,pi]的DOUBLE类型数据。

其中expr1和expr2的值均为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr1或expr2中任一值为NULL时，函数返回NULL。

示例

```
SELECT ATAN2(1,1) res FROM DUAL;
        RES
-----------
 7.854E-001

SELECT ATAN2(1,2) res FROM DUAL;
        RES
-----------
 4.636E-001

SELECT ATAN2(2,2) res FROM DUAL;
        RES
-----------
 7.854E-001
```

AVG

```
avg::= AVG "(" [DISTINCT|ALL] expr ")" [OVER "(" analytic_clause ")"]

analytic_clause::= "(" [query_partition_clause] [order_by_clause [windowing_clause]] ")"
```

AVG函数计算给定参数的平均值，返回值的类型有以下几种情况：

参数类型	返回值
TINYINT、SMALLINT、INT、BIGINT、NUMBER	NUMBER
FLOAT	FLOAT
DOUBLE	DOUBLE
CHAR	NUMBER
VARCHAR	NUMBER
NCHAR	NUMBER
NVARCHAR	NUMBER

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

在单行计算中，当expr的值为NULL时，函数返回NULL。

在多行计算中，函数将忽略expr值为空的行，当所有行均为空时，计算结果为NULL。

聚集函数不可嵌套，因此expr为除聚集函数之外的其他通用表达式。

DISTINCT

表示在计算平均值时，过滤掉重复的行。

ALL

默认值，表示对所有行计算平均值。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME            DHQ             EMPLOYEE_COUNT
------- -------------------- --------------- --------------
01      华东                 Shanghai
02      华西                 Chengdu                   300
03      华南                 Guangzhou                 400
04      华北                 Beijing                   300
05      华中                 Wuhan

--计算平均员工数，员工数为空的行在计算时被忽略
SELECT AVG(employee_count) res FROM area1;
                 RES
--------------------
         333.333333

--计算平均员工数（过滤掉重复值），员工数为空的行在计算时被忽略
SELECT AVG(DISTINCT employee_count) res FROM area1;
                  RES
---------------------
                  350
```

OVER

当指定OVER关键字时，AVG将作为窗口函数，并支持滑动窗口，返回多行的平均值。

analytic_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778

SELECT year,month,
revenue_total curr,
AVG(revenue_total) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info;
YEAR  MONTH        CURR       TONOW
----- ----- ----------- -----------
2001  01           2888        2888
2021  01          28888       28888
2021  01          38888       33888
2021  02          37778  35184.6667

--分年统计每月所有机构的平均收入,及年初至今所有机构的平均收入
SELECT year,month,
AVG(revenue_total) curr,
AVG(AVG(revenue_total)) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info
GROUP BY year,month;
YEAR  MONTH        CURR       TONOW
----- ----- ----------- -----------
2001  01           2888        2888
2021  01          33888       33888
2021  02          37778       35833
```

BIN

```
bin::= BIN "(" expr ")"
```

BIN函数将expr表示的一个数据转换成BIT二进制，并以VARCHAR类型承载该BIT值进行返回。

expr的值为数值型、可以转换为NUMBER类型的其他类型（转换失败返回类型转换错误）或RAW、LOB类型，如输入其他类型，会返回错误。

expr不支持XMLTYPE、JSON类型数据和32000字节以上LOB类型数据。

当expr值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数截断其小数位保留整数位。

上述所涉及的整数数值不能超过int64所能表示的数字范围。

expr在字面上除NULL（此时函数返回NULL）外，应该只包含0和1（除了可以b开头，表示二进制字符串），否则函数返回invalid string错误。当为b开头的二进制字符串时，需满足YashanDB对BIT类型的长度约束。

当expr的值为负数时，函数返回对应数值的64位补码。

示例

```
SELECT BIN('2345.34') res FROM DUAL;
RES
--------------
100100101001

SELECT BIN(b'0100101010') res FROM DUAL;
RES
------------------
100101010

SELECT BIN(-235) res FROM DUAL;
RES
----------------------------------------------------------------
1111111111111111111111111111111111111111111111111111100010101
```

BIN_TO_NUM

```
bin_to_num::= BIN_TO_NUM "(" (expr) {"," (expr)} ")"
```

BIN_TO_NUM函数用于实现二进制向十进制的转换，将一个或多个expr组合表示的二进制位向量串转换成与其等效的数值，返回一个NUMBER类型的数据。该函数不支持列式计算。

每一个expr表示位向量中的一个位，其值可以为任何数值型，或可隐式转换为NUMBER类型的字符型数据。

本函数支持最多输入65535个expr作为参数，但计算的数值超过number表示范围则报错，且所有expr的值都必须为0/1，或进行取整后为0/1，否则报错。取整规则为：

1. 如果expr为FLOAT/DOUBLE类型，函数对其进行奇进偶舍取整（例如：1.5->2，0.5->0）。
2. 对于其他类型，函数进行去掉小数位取整（例如：1.3->1，1.6->1）。

示例（HEAP表）

```
--将二进制位向量串11转换为十进制3
SELECT BIN_TO_NUM(1,1) res FROM DUAL;
RES
-----------
3

--将二进制位向量串1101转换为十进制13
SELECT BIN_TO_NUM(b'1',1,0,1) res FROM DUAL;
RES
-----------
13

--创建bt_table_bin_to_num表，包含多个浮点型字段
CREATE TABLE bt_table_bin_to_num(c1 DOUBLE, c2 DOUBLE, c3 DOUBLE, c4 DOUBLE);
INSERT INTO bt_table_bin_to_num VALUES(0.3,0.8,1.3,1.8);

--对于NUMBER数据，函数按去小数位取整后进行十进制转换，对于浮点型小数，函数按四舍五入取整后进行十进制转换
SELECT BIN_TO_NUM(0.3) "0.3", BIN_TO_NUM(c1) "d0.3",
BIN_TO_NUM(0.8) "0.8", BIN_TO_NUM(c2) "d0.8",
BIN_TO_NUM(1.3) "1.3", BIN_TO_NUM(c3) "d1.3"
FROM bt_table_bin_to_num;
        0.3        d0.3         0.8        d0.8         1.3        d1.3
----------- ----------- ----------- ----------- ----------- -----------
          0           0           0           1           1           1

--对于NUMBER数据，函数按去小数位取整为1后进行十进制转换
SELECT BIN_TO_NUM(1.8) res FROM DUAL;
RES
-----------
1

--c4字段按四舍五入取整为2后进行十进制转换，导致报错
SELECT BIN_TO_NUM(c4) res FROM bt_table_bin_to_num;
[1:19]YAS-06001 the value of parameter expression is invalid, the value of the expression must be 0 or 1
```

BITAND BITOR BITXOR

```
bitand::= BITAND "(" expr1 "," expr2 ")"

bitor::= BITOR "(" expr1 "," expr2 ")"

bitxor::= BITXOR "(" expr1 "," expr2 ")"
```

BITAND/BITOR/BITXOR函数将两个数据进行按位与/按位或/按位异或计算，得到按位的0或1，并将多位0或1由二进制转换为十进制数值返回。

expr1、expr2的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型数据，或可以转换为NUMBER类型的字符型数据（转换失败返回类型转换错误）。对于其他类型，返回类型不支持错误。

函数将先对expr1、expr2执行二进制转换，然后进行位运算，转换规则及约束同BIN函数。

当expr1或expr2中任一值为NULL时，函数返回NULL。

示例

```
SELECT BITAND('',1) b1,
BITAND('2.35',3) b2,
BITOR(5,-5) b3,
BITXOR(3456,1.99) b4
FROM DUAL;
                   B1                    B2                    B3                    B4
-------------------- --------------------- --------------------- ---------------------
                                          2                    -1                  3457
```

BIT_LENGTH

```
BIT_LENGTH::= BIT_LENGTH "(" expr ")"
```

BIT_LENGTH函数按比特位统计expr的长度，返回一个BIGINT的数值。

该函数不支持列式计算。

expr的值须为字符型/BIT类型，或可转化为字符型的其他类型（不包括浮点型）。

当expr的值为NULL时，函数返回NULL。

当expr为字符型或转换后为字符型时，函数统计所有字符所占比特位之和。对于中文字符，其比特位长度与数据库服务端字符集类型相关，例如，在UTF8字符集环境中，一个中文字符占24比特位，而在GBK字符集环境中，一个中文字符占16比特位。

当expr为BIT类型时，函数统计该BIT数字所占的比特位之和。

函数中的expr不能为XMLTYPE类型。

示例

```
--UTF8字符集环境
SELECT CHAR_LENGTH('深圳') r1,OCTET_LENGTH('深圳') r2,BIT_LENGTH('深圳') r3
FROM DUAL;
                    R1                    R2                    R3
---------------------- ---------------------- ----------------------
                     2                      6                     48

--GBK字符集环境
SELECT CHAR_LENGTH('深圳') r1,OCTET_LENGTH('深圳') r2,BIT_LENGTH('深圳') r3
FROM DUAL;
                    R1                    R2                    R3
---------------------- ---------------------- ----------------------
                     2                      4                     32

--BIT数据
SELECT BIT_LENGTH(b'101') rb, BIT_LENGTH('101') rc, BIT_LENGTH(101) rn
FROM DUAL;
                    RB                    RC                    RN
---------------------- ---------------------- ----------------------
                     8                     24                     24
```

CAST

```
cast::= CAST "("expr AS type_name [DEFUALT replace_expr ON CONVERSION ERROR] ")"
```

CAST函数将expr的值转换为指定的数据类型，并按新的类型返回结果。

YashanDB支持如下情况的类型转换：

expr类型	type_name
BOOLEAN	TINYINT、SMALLINT、INT、BIGINT、CHAR、VARCHAR、NCHAR、NVARCHAR
BIT	除FOLAT、DOUBLE外的数值型、字符型数据
CHAR、VARCHAR	除UDT外的所有数据类型
NCHAR、NVARCHAR	除UDT、XMLTYPE外的所有数据类型
DATE	CHAR、VARCHAR、TIMESTAMP
FLOAT、DOUBLE	除BIT外的数值型、字符型数据
NUMBER	所有数值型、字符型
TIMESTAMP	字符型数据、DATE
TINYINT、SMALLINT、INT、BIGINT	所有数值型、字符型
INTERVAL YEAR TO MONTH	字符型数据
INTERVAL DAY TO SECOND	字符型数据
CLOB	字符型数据、JSON、XMLTYPE
NCLOB	字符型数据、JSON
BLOB	字符型数据、JSON、RAW、UROWID
XMLTYPE	CHAR、VARCHAR
ROWID	字符型数据、RAW、UROWID
UROWID	字符型数据、RAW
JSON	字符型数据、CLOB、BLOB、NCLOB、RAW
RAW	字符型数据、CLOB、BLOB、NCLOB、UROWID

其中：

- expr的内容需符合目标类型的格式要求。
- expr的值不能超过目标类型的值域，如10000无法转换为TINYINT类型。
- expr的值为NULL时，函数返回NULL。
- 对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数进行转换。
- 当目标类型为BOOLEAN时，只有'TRUE'、'T'、'YES'、'Y'、'1'可转换为TRUE，'FALSE'、'F'、'NO'、'N'、'0'可转换为FALSE，字符不分大小写。
- 当NUMBER转换BIT时，函数将对NUMBER进行向下取整，且NUMBER数据的大小不能超过64位。
- 当TIME转换DATE跟TIMESTAMPS时，取当天的日期来补齐年月日。

type_name

指定要转换的目标类型，可同时指定size、scale、precision。

若目标类型为字符型数据时，可以指定其size，若expr为字符串类型且长度大于指定的size会截断，为其他类型超过指定的size，则转换失败并返回Out of range错误。

若目标类型为CHAR或NCHAR时，且语句中未指定其size时，不需要关注需要转换的原始数据类型，其预估size长度默认为1。

若目标类型为VARCHAR时，且语句中未指定其size，将根据expr的类型对size进行预估。对于字符串类型，其预估size为原字符串长度；对于其他类型，其预估size为该类型转为字符串的长度。

若目标类型为NVARCHAR时，须在语句中指定其size，否则返回错误。

DEFAULT replace_expr ON CONVERSION ERROR

表示当对expr转换失败时，使用replace_expr值来进行转换。 replace_expr可以为NULL，字符串值和字面量，且必须能够转换为目标类型（转换类型支持，且转换内容符合格式要求）。

此语句可省略，则对expr转换失败时函数返回错误。

当expr的数据类型为BIT、LOB类型，或者type_name为BIT、LOB类型时，不允许指定此语句。

示例

```
SELECT CAST('345' AS FLOAT) cast1,
CAST('345.2345' AS NUMBER(6,2)) cast2,
CAST('345a' AS INT DEFAULT '' ON CONVERSION ERROR) cast3,
CAST(SYSTIMESTAMP AS CHAR(100)) cast4
FROM DUAL;
      CAST1     CAST2    CAST3 CAST4
----------- --------- -------- ----------------------------
  3.45E+002    345.23          2022-01-09 17:52:30.634370

SELECT CAST('abcdef' AS CHAR) cast1,
CAST(1.23456 AS VARCHAR) cast2,
CAST(NULL AS CHAR) cast3,
CAST(NULL AS VARCHAR) cast4
FROM DUAL;
CAST1 CAST2     CAST3 CAST4
----- --------- ----- -----
a     1.23456
```

CEIL

```
ceil::= CEIL "(" expr ")"
```

CEIL函数对expr表示的数据进行向上取整，其返回规则为：

- 当expr的值为数值型时，返回与其相同数据类型的数据。
- 当expr的值为字符型时，返回NUMBER类型的数据。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为浮点类型特殊值时：
  - Nan：函数返回Nan
  - Inf：函数返回Inf
  - -Inf：函数返回-Inf

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

示例

```
SELECT CEIL(7) ceil1,
CEIL('0.97') ceil2,
CEIL(6.22) ceil3
FROM DUAL;
       CEIL1       CEIL2       CEIL3
------------ ----------- -----------
           7           1           7

--创建number_fd表，包含FLOAT、DOUBLE字段
CREATE TABLE number_fd1(numberf FLOAT, numberd DOUBLE);
INSERT INTO number_fd1 VALUES('Nan','Nan');
INSERT INTO number_fd1 VALUES('Inf','Inf');
INSERT INTO number_fd1 VALUES('-inf','-inf');
COMMIT;

SELECT CEIL(numberf) ceilf,
CEIL(numberd) ceild
FROM number_fd1;
      CEILF       CEILD
----------- -----------
        Nan         Nan
        Inf         Inf
       -Inf        -Inf
```

CHAR_LENGTH CHARACTER_LENGTH

```
CHAR_LENGTH::= CHAR_LENGTH "(" expr ")"
CHARACTER_LENGTH::= CHARACTER_LENGTH "(" expr ")"
```

CHAR_LENGTH/CHARACTER_LENGTH函数按字符统计expr的长度，返回一个BIGINT的数值。CHAR_LENGTH、CHARACTER_LENGTH及LENGTH函数相互同义。

expr的值须为字符型，或可转化为字符型的其他类型。

对于列存表中的LOB、XMLTYPE类型字段，若某行数据为行外存储，则无法使用本函数进行长度统计。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT CHAR_LENGTH('深圳') r1,OCTET_LENGTH('深圳') r2,BIT_LENGTH('深圳') r3,
       CHARACTER_LENGTH('aabbccDDee') r4,OCTET_LENGTH('aabbccDDee') r5,BIT_LENGTH('aabbccDDee') r6,
       CHAR_LENGTH(null) r7,OCTET_LENGTH(null) r8,BIT_LENGTH(null) r9
FROM DUAL;
   R1   R2    R3    R4    R5    R6    R7    R8    R9
----- ---- ----- ----- ----- ----- ----- ----- -----
    2    6    48    10    10    80
```

CHECK_SYS_PRIVILEGE

```
check_sys_privilege::= CHECK_SYS_PRIVILEGE "(" user_id "," object_type_id ")"
```

CHECK_SYS_PRIVILEGE函数检测登陆用户是否拥有对某用户下某种类型对象的可访问类系统级权限，返回值为BOOLEAN类型。

该函数不支持列式计算。

当检测表类型对象的权限时，如用户拥有SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE中任一项权限时，函数返回TURE，否则返回FALSE。

user_id

对象所属OWNER用户ID，必须是整数类型的数据，使用USERENV函数或查询DBA_USERS视图获取。

object_type_id

对象的类型ID，必须是整数类型的数据，可参考DBA_OBJECTS视图资料中OBJECT_TYPE字段描述，例如表的类型ID为1。

示例

```
SELECT CHECK_SYS_PRIVILEGE(0,1) pri FROM DUAL;
PRI
--------------------
true
```

CHR

```
chr::= CHR "(" expr ")"
```

CHR函数将expr表示的一个ASCII码数值转换为对应的字符，返回结果为VARCHAR类型。

本函数遵循如下约束：

- expr的值必须为除BIT外的数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误），否则返回类型不支持。
- 如果expr的值为小数，则对其FLOOR取整，取整后参数的范围在 [0, 2^{32}-1]，否则返回Out of range错误。
- ASCII码值范围为[0,255]，当expr的值小于0时，函数返回Out of range错误，当expr的值大于255时，函数将对其按256取模（MOD）后，再进行转换。
- expr的值为NULL时，函数返回NULL。

示例

```
--函数将如下参数转换为小数2344556.44，并取整为2344556，再执行MOD(2344556,256),根据其结果执行转换
SELECT CHR('2344556.44') res FROM DUAL;
RES
-----
l
```

COALESCE

```
coalesce::= COALESCE "(" expr "," (expr)  {"," (expr)} ")"
```

COALESCE函数在多个expr的值中，查找到第一个不为空的expr值，并将其返回，未查找到时返回NULL。expr的数量最少为2个，最多为100个。

expr值的数据类型可以为YashanDB支持的大多数类型，但同一个参数列表里所有expr值的数据类型必须处于下面同一个分类中，否则函数返回Inconsistent datatype错误：

- 数值型：按优先度从高到低为DOUBLE、FLOAT、NUMBER、BIGINT、INT、SMALLINT、TINYINT，但BIT类型只能与同类型在同一个参数列表中。
- 字符型：
  - 按优先度从高到低为VARCHAR、CHAR。
  - 按优先度从高到低为NVARCHAR、NCHAR。
- 日期时间型：按优先度从高到低为TIMESTAMP、DATE、TIME，但INTERVAL YEAR TO MONTH、INTERVAL DAY TO SECOND只能与同类型在同一个参数列表中。
- 布尔型：BOOLEAN
- CLOB
- BLOB
- NCLOB
- XMLTYPE

如所有expr值的数据类型处在同一分类中（字符型数据包含两个子分类），但具体类型不相同，函数将按上面所列优先级顺序进行类型转换，如`(expr1 NUMBER, expr2 SMALLINT,expr3 FLOAT)`时，expr1、expr2的数据类型将被转换成FLOAT，且函数返回一个FLOAT型数据。

当同一个参数列表里的所有expr值均为NULL时，函数返回NULL。

示例

```
--在手册首页中列示的numbers_nobit表
SELECT * FROM numbers_nobit;
     NUMBERA NUMBERB  NUMBERC               NUMBERD     NUMBERE     NUMBERF     NUMBERG
------------ ------- -------- --------------------- ----------- ----------- -----------
          -5      55     5555   5555555555555555555  5.555E+000  5.556E+000         555

--按照三个expr中数据类型优先级最高的进行转换，numbera的值被转换为FLOAT类型
SELECT COALESCE(numbera,numberb,numbere) res FROM numbers_nobit;
        RES
-----------
  -5.0E+000


SELECT COALESCE('','',true) res FROM DUAL;
RES
--------------------
true
```

CONCAT

```
concat::= CONCAT "(" expr "," (expr) {"," (expr)} ")"
```

CONCAT函数将多个expr的值连接成一个字符串，expr的个数最少为2个，最多为65535个。

此函数等同于：expr||expr||...。

函数遵循如下规则：

- 对于非字符型/CLOB类型的expr值，函数先进行到CHAR类型的转换，如类型转换不成功，则返回类型转换错误。
- 参数类型优先级：
  - 当expr值中不存在CLOB、NCLOB类型时：
    - 仅包含CHAR/NCHAR类型数据时，函数返回CHAR/NCHAR类型的字符串。
    - 仅包含CHAR和VARCHAR类型数据时，函数返回VARCHAR类型的字符串。
    - 仅包含CHAR和NCHAR类型数据时，函数返回NCHAR类型的字符串。
    - 存在NVARCHAR类型数据时，函数返回NVARCHAR类型的字符串。
    - 不包含NVARCHAR类型数据，但存在VARCHAR和NCHAR类型时，函数返回NVARCHAR类型的字符串。
  - 当expr值中存在CLOB或NCLOB时，通过如下三个规则判断返回的数据类型：
    - 规则1：当参数列表中只有一个CLOB时，如果前面不包含NCHAR、NVARCHAR数据，则返回CLOB，否则返回NCLOB。
    - 规则2：当参数列表中只有一个NCLOB时，如果前面包含NCHAR、NVARCHAR数据或者没有其他参数，则返回NCLOB，否则返回CLOB。
    - 规则3：当有多个CLOB或这NCLOB时，则以参数列表中从左往右的第一个CLOB或者NCLOB作为唯一的CLOB或者NCLOB代入上述规则1或者规则2。
- 当expr值为NULL时，函数将其转换为长度为0的VARCHAR字符串。

对于列存表中的CLOB类型字段，若某行数据为行外存储，则无法使用本函数。

示例

```
SELECT CONCAT(employee_name,' has joined us for ',CEIL(SYSDATE-entry_date),' days.') res FROM employees;
RES
----------------------------------------------------------------
Mask has joined us for 1001 days.
John has joined us for 2001 days.
Anna has joined us for 301 days.
Jack has joined us for 701 days.
Jim has joined us for 201 days.
```

CONCAT_WS

```
concat_ws::= CONCAT_WS "(" separator "," (expr) {"," (expr)} ")"
```

CONCAT_WS函数将多个expr的值通过separator连接，返回一个拼接后的字符串，函数参数的个数最少为2个，最多为32768个。

函数须遵循如下规则：

- separator必须是字符型或字符串。
- separator不支持32000字节以上的XMLTYPE、LOB类型数据。
- expr中包含NCHAR或NVARCHAR类型数据时，返回值为NVARCHAR类型，否则为VARCHAR类型。
- 分隔符为NULL或者''时，函数返回NULL。
- 当除了分隔符外，只有一个非空参数时，函数只返回这个非空参数（不包含分隔符）。
- 当需要拼接的字符串为NULL或者''时，函数不会对该字符串以及分隔符进行拼接。
- expr支持为数值型、字符型、布尔型、日期时间型。
- 当expr的值为CHAR类型时，函数按其定义长度补齐空格后，再进行拼接。

示例

```
SELECT CONCAT_WS('1','2', '', 'dc') res FROM DUAL;
RES
-----
21dc

SELECT CONCAT_WS(12,null,null,'','a',null) res FROM DUAL;
RES
-----
a
```

COS

```
cos::= COS "(" expr ")"
```

COS函数返回给定参数的余弦值，参数为以弧度表示的角度，大小本身无限制（只受限于其所属数据类型所规定范围），函数将返回一个大小在区间[-1, 1]的DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT COS(30*3.1415926/180) res FROM DUAL;
        RES
-----------
  8.66E-001

SELECT COS(45*3.1415926/180) res FROM DUAL;
        RES
-----------
 7.071E-001

SELECT COS(60*3.1415926/180) res FROM DUAL;
        RES
-----------
   5.0E-001
```

COT

```
cot::= COT  "(" expr ")"
```

COT函数返回给定参数的余切值，参数为以弧度表示的角度，大小本身无限制（只受限于其所属数据类型所规定范围），函数将返回一个DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
--计算以弧度表示的expr计算结果的余切值
SELECT COT(0) res FROM DUAL;
        RES
-----------
        Inf

SELECT COT(45*3.1415926/180) res FROM DUAL;
        RES
-----------
   1.0E+000

SELECT COT(60*3.1415926/180) res FROM DUAL;
        RES
-----------
 5.774E-001
```

COUNT

```
count::= COUNT "(" ("*"|([DISTINCT|ALL] expr)) ")" [OVER "(" analytic_clause ")"]

analytic_clause::= "(" [query_partition_clause] [order_by_clause [windowing_clause]] ")"
```

COUNT函数按给定参数expr统计记录行数，该函数返回一个BIGINT的数值，且永远不会返回NULL。

在单行计算中，当expr的值为NULL时，函数返回0。

在多行计算中，当expr为字面量、变量、常量时，函数统计的是所有行，包括空行。否则，函数将忽略expr值为空的行，当所有行均为空时，计算结果为0。

聚集函数不可嵌套，因此expr为除聚集函数之外的其他通用表达式，其类型可以是UDT以外的任意数据类型。

*****

表示统计所有的行，包括空行。

DISTINCT

表示将表达式结果进行重复过滤，统计的是所有不重复的非空行。

ALL

默认值，表示不对表达式结果进行重复过滤，统计的是所有非空行。

示例

```
--统计机构表的行数
SELECT COUNT(1) res FROM branches;
                  RES
--------------------
                   12

--统计机构表中area_no字段非空的行数,该语句与SELECT COUNT(ALL area_no) FROM branches查询结果一致
SELECT COUNT(area_no) res FROM branches;
                  RES
--------------------
                   10

--统计机构表中area_no字段非空且不值不相同的行数
SELECT COUNT(DISTINCT area_no) res FROM branches;
                  RES
--------------------
                    4
```

OVER

当指定OVER关键字时，COUNT将作为窗口函数，并支持滑动窗口，返回多行的统计行数。

analytic_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778
```

```
--分年统计每月存在收入的机构数
SELECT year,month,
COUNT(revenue_total) OVER (PARTITION BY year ORDER BY month) toall
FROM finance_info;
YEAR  MONTH                 TOALL
----- ----- --------------------
2001  01                       1
2021  01                       2
2021  01                       2
2021  02                       3

--分年统计每月年初至今存在收入的机构数
SELECT year,month,
COUNT(revenue_total) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info;
YEAR  MONTH                 TONOW
----- ----- --------------------
2001  01                       1
2021  01                       1
2021  01                       2
2021  02                       3
```

CURRENT_TIMESTAMP

```
current_timestamp::= CURRENT_TIMESTAMP "("")"
```

CURRENT_TIMESTAMP函数无参数，该函数返回一个TIMESTAMP类型的当前系统时间，精确到毫秒。

示例

```
SELECT CURRENT_TIMESTAMP() res FROM DUAL;
RES
----------------------------------------------------------------
2022-04-12 18:37:26.008000
```

DATE

```
date::= DATE "("expr ["," format] ")"
```

DATE函数有1或2个参数。当只有一个expr参数时，DATE函数将expr的值按照配置参数DATE_FORMAT指定格式进行DATE类型转换；当有两个参数时，DATE函数将expr的值按照第二个参数format进行格式转换。

expr

表达式expr的数据类型须为DATE/TIMESTAMP/TIME类型，内容不论是否符合date_format或format格式，都返回正确结果。

当expr的值为NULL时，函数返回NULL。

format

format支持中文年月日，须用双引号包围中文字符，expr中的中文字符无须用双引号包围。

指定转换的格式，包括如下字符组合：

- 年
  - 年份数字：`YYYY`、`Y`、`YY`、`YYY`
- 月
  - 月份数字：`MM`
  - 月份全称：`MONTH`
  - 月份缩写：`MON`
- 日
  - 日期数字：`DD`
- 日期描述
  - 一周中的一天全称（SUNDAY-SATURDAY）：`DAY`
  - 一周中的一天数字（1-7）：`D`
  - 一年中的一天（1-366）：`DDD`
- 时
  - 24小时制小时：`HH24`
  - 12小时制小时：`HH`、`HH12`
- 分
  - 分钟数：`MI`
- 秒
  - 秒数：`SS`
- 连接字符：`:`、`-`、`/`、`.`、`,`、`;`、`\`、`_`、` `
  - `:`、`-`、`/`、`.`、`,`、`;`、`\`、`_`：八种连接符相互一对一兼容匹配
  - ` `：忽略所有空格，空格不参与匹配

不指定format时，系统按DATE类型指定的默认格式进行转换，如果expr内容不符合默认格式，返回格式匹配错误。

当format的值为NULL时，函数返回NULL。

示例

```
SELECT DATE( 'January 15, 1989, 11:00 A.M.', 'Month dd, YY, HH:MI A.M.' ) res FROM DUAL;
RES
--------------------------------
1989-01-15 11:00:00

SELECT DATE( '1989/2/23', 'YYYY/MM/DD HH24:MI:SS' ) res FROM DUAL;
RES
--------------------------------
1989-02-23 00:00:00

SELECT DATE( '1989/2/23' ) res FROM DUAL;
RES
--------------------------------
```

```
1989-02-23


SELECT DATE( '1989/2/23', '') res FROM DUAL;

RES
--------------------------------


SELECT DATE( '1989/2/23', null) res FROM DUAL;

RES
--------------------------------

SELECT DATE('1989年2月23日','YYYY"年"MM"月"dd"日"') RES FROM DUAL;

RES
--------------------------------
1989-02-23  00:00:00

--不兼容匹配默认DATE格式'YYYY-MM-DD'时，返回错误
SELECT DATE( '1989//2/23' ) res FROM DUAL;
[1:16]YAS-00008 type convert error : literal does not match format string
```

DATE_ADD

```
date_add::= DATE_ADD "(" expr "," INTERVAL interval_value interval_unit ")"
```

DATE_ADD函数用于执行日期运算，对expr的值按给定的区间值将时间前进或者后退。

expr

通用表达式，expr的值须为DATE、TIME、TIMESTAMP类型，或可以转换为DATE、TIMESTAMP类型的字符型。

当expr为字面量时，只能为DATE和TMIESTAMP关键字的输入字符串，不能为TIME关键字的输入字符串。例如，DATE '2012-10-12'、TIMESTAMP '2012-10-12 10:20:24.000006'为函数可接受的expr值，而当expr为TIME '10:20:24'时函数则返回错误。

当expr为NULL时，函数返回NULL。

interval_value

指定时间前进或后退的区间值，该值不可为NULL，且必须为如下形式：

- 0或正整数字面量，如0、1、2等。
- 包含0或正整数内容的字符串字面量，如'0'、'1'、'-2'等。
- 包含负整数内容的字符串字面量，如'-1'、'-2'等。
- 包含INTERVAL内容的字符串字面量，如'10-8'、'-8 8:10:24'等。

interval_unit

指定区间值的单位，该值不可为NULL，且必须为如下形式：

- MONTH、YEAR、YEAR TO MONTH关键字（不区分大小写）：此时interval_value被转换为INTERVAL YEAR TO MONTH类型。
- SECOND、MINUTE、HOUR、DAY、MINUTE TO SECOND、HOUR TO SECOND、HOUR TO MINUTE、DAY TO SECOND、DAY TO MINUTE、DAY TO HOUR关键字（不区分大小写）：此时interval_value被转换为INTERVAL DAY TO SECOND类型。

interval_value与interval_unit必须正确匹配，例如分别为2/YEAR，3/HOUR，'10-8'/YEAR TO MONTH、'-8 8:10:24'/DAY TO SECOND，否则函数返回类型转换错误。

日期运算规则

1.当interval_value为INTERVAL YEAR TO MONTH类型时，运算规则为：

- 先进行month的增减，再看day是否符合month的增减后的day数。
- 如果增减后的month的天数小于增减前的month的天数，那么增减后的day数等于增减后的month的最后一天。

2.当expr和interval_value的数据类型不相同时，函数先执行类型转换，若两个数据类型之间无法按照一定的规则进行转换，则返回类型转换错误。类型转换规则如下：

- expr为DATE/TIME/TIMESTAMP类型时，函数返回expr的数据类型。
- expr为CHAR/VARCHAR类型时，函数先将其能转换为DATE、TIME、TIMESTAMP数据类型中的一种，转换成功则返回转换后的数据类型，否则返回类型转换错误。
- expr为TIME，interval_value为INTERVAL DAY TO SECOND类型时，超过范围将翻转，即超过24小时求余数，例如计算后的小时为25时，将其翻转为1。
- expr为TIME，interval_value为INTERVAL YEAR TO MONTH类型时，函数返回类型不支持错误。

示例

```
SELECT DATE_ADD('2012-10-12',INTERVAL  '-1' YEAR) res FROM DUAL;
RES
-----------------------------------------------------------------
2011-10-12 00:00:00.000000

--interval_value为INTERVAL YEAR TO MONTH类型时
SELECT DATE_ADD('2012-10-31',INTERVAL 1 MONTH) res FROM DUAL;
RES
-----------------------------------------------------------------
2012-11-30 00:00:00.000000
```

```
--interval_value为INTERVAL DAY TO SECOND类型时
SELECT DATE_ADD(CAST('10:20:24' AS TIME),INTERVAL '8:10:24' HOUR TO SECOND) res FROM DUAL;
RES
--------------------
18:30:48.000000

--小时翻转
SELECT DATE_ADD(CAST('10:20:24' AS TIME),INTERVAL '14:10:24' HOUR TO SECOND) res FROM DUAL;
RES
--------------------
00:30:48.000000
```

DATE_FORMAT

```
DATE_FORMAT::= DATE_FORMAT "(" expr "," format ")"
```

DATE_FORMAT函数将给定的参数expr按format定义的格式进行提取，返回VARCHAR类型的字符串。 expr的值为DATE类型或可以转换为DATE类型的其他类型，format必须是字符型或可以转换为字符串类型。

当expr的值为NULL时，函数返回NULL。

format

表示expr对应的format格式，以下为对应列表：

格式定义	返回值
%a	缩写的工作日名称（Sun..Sat）
%b	缩写月份名称（Jan..Dec）
%c	月份（1...12）
%D	带有英文后缀的月份日期（0th、1st、2nd、3rd、…）
%d	每月的第几天（01..31）
%e	每月的第几天（1..31）
%f	微秒（000000..999999）
%H	24进制小时（00...23）
%h	12进制小时（01...12）
%I	12进制小时（01...12）
%i	分钟（00...59）
%j	一年中的第几天（001...366）
%k	24进制小时（0...23）
%l	12进制小时（1...12）
%M	月份名称（January...December）
%m	月份（01...12）
%p	上午或是下午（AM or PM）
%r	取12进制time带AM或PM（精确到秒）
%S	秒数（00...59）
%s	秒数（00...59）
%T	取24进制时间（精确到秒）
%U	一年中的第几周，周日为一周的第一天
%u	一年中的第几周，周一为一周的第一天
%V	一年中的第几周，周日为一周的第一天，一般与%X一起用
%v	一年中的第几周，周一为一周的第一天，与%x一起用
%W	星期名称（Sunday...Saturday）

格式定义	返回值
%w	一周中的第几天（0为周天，6为周六）
%X	一周属于哪一年（四位），周天是一周中的第一天，一般与%V一起用
%x	一周属于哪一年（四位），周一是一周中的第一天，一般与%v一起用
%Y	四位年份
%y	两位年份
%%	输出%

如果输入format为NULL，则函数返回NULL。

如果输入format无法与format列表里的类型相匹配，则输出format对应的字符。

示例

```
SELECT DATE_FORMAT(NOW(), '%a') res FROM DUAL;
RES
------------------------------------------------------------
Tue

SELECT DATE_FORMAT(NOW(), '%b') res FROM DUAL;
RES
------------------------------------------------------------
Apr

SELECT DATE_FORMAT(NOW(), '%c') res FROM DUAL;
RES
------------------------------------------------------------
4

SELECT DATE_FORMAT(NOW(), '%D') res FROM DUAL;
RES
------------------------------------------------------------
12th

SELECT DATE_FORMAT(NOW(), '%%') res FROM DUAL;
RES
------------------------------------------------------------
%
```

DAYOFWEEK

```
DAYOFWEEK::= DAYOFWEEK "(" expr ")"
```

DAYOFWEEK函数用于计算expr位于所在周的第几天（以周日为第一天计算），返回一个INT类型的数值。该函数不支持列式计算。

expr的值须为TIMESTAMP/DATE类型，或可以转换为TIMESTAMP/DATE类型的字符型。

当expr的值为NULL时，函数返回NULL。

> **Note**：
> YashanDB采用外推格历高利历，1582年10月15号（不包括15号）之前与Oracle历法不一致。

示例

```
SELECT DAYOFWEEK(DATE '2022-7-26') res FROM DUAL;
         RES
------------
           3

--微秒数超过6位时按四舍五入进位
SELECT DAYOFWEEK('2022-10-28 23:59:59.999999999') res FROM DUAL;
         RES
------------
           7

SELECT DAYOFWEEK('2022-10-28 23:59:59.99999901') res FROM DUAL;
         RES
------------
           6
```

DECODE

```
decode::= DECODE "(" expr "," (value "," result) {"," (value "," result)} ["," default] ")"
```

DECODE表达式相当于条件表达式，与一系列嵌套的 IF-THEN-ELSE语句类似。

当expr的值等于其后列出中的某一个指定value，则返回紧接着该value后的result；若不等于所列出的所有value，则返回default；当expr与多个value匹配成功时，返回第一个value对应的result。

该函数通常用于做排名等级判断。

当expr与各value之间的数据类型不一致时：

- 同时存在数值型和布尔型时，函数返回invalid datatype。
- 其它场景中，函数先将数据类型进行统一再比较，统一规则参见比较运算符章节描述。

value也为通用表达式，参照expr描述。

空值同样参与比较，其中，当expr与value均为NULL时，执行匹配成功处理。

示例

```
SELECT employee_name Name,
DECODE(sex,'1','Male','2','Female','Unknown') Sex
FROM employees;
NAME          SEX
------------- ---------
Mask          Male
John          Male
Anna          Unknown
Jack          Male
Jim           Male

SELECT DECODE('',1,1,2) res1,
DECODE(1,1,1,'1',2,3) res2,
DECODE(1,'',1,'1',2,3) res3,
DECODE('','',1,3) res4
FROM DUAL;
        RES1         RES2         RES3         RES4
------------ ------------ ------------ ------------
           2            1            2            1
```

DECRYPT_AES128

```
DECRYPT_AES128::= DECRYPT_AES128 "(" expr1 "," expr2 ")"
```

DECRYPT_AES128函数以expr2为key解密expr1的密文，返回一个VARCHAR类型的明文。

本函数遵循如下规则：

- expr1/expr2只支持VARCHAR/CHAR类型。
- expr2不能为null。
- 当expr1为null时返回null。

> **Caution**:
> DECRYPT_AES128函数中的密钥由用户管理，请勿将包含DECRYPT_AES128函数的SQL写入OUTLINE或者SQLMAP视图中，以避免潜在的密钥泄露风险。

示例

```
SELECT encrypt_aes128(area_name, 'admin') FROM area;

ENCRYPT_AES128(AREA_
--------------------------------------------------------------
U��G��
U��K��
U��F��
U��F��
U��G��


SELECT decrypt_aes128(encrypt_aes128(area_name, 'admin'), 'admin') FROM area;

DECRYPT_AES128(ENCRY
--------------------------------------------------------------
华东
华西
华南
华北
华中
```

DENSE_RANK

```
dense_rank::= DENSE_RANK "("")" OVER "(" [query_partition_clause] order_by_clause ")"
```

DENSE_RANK为窗口函数，用于对数据的实时分析。

- DENSE_RANK函数列出相同并列值，并对下一顺序值不跳号，例如1，2，3，3，3，4，5，6.....

该函数不支持列式计算。

查看DENSE_RANK函数与RANK函数的区别。 查看DENSE_RANK函数与ROW_NUMBER函数的区别。

partition by与order by的参数参数支持除LOB、JSON、XMLTYPE、UDT外的其它数据类型。

query_partition_clause|order_by_clause

窗口函数通用语法。

示例（HEAP表）

```
--sales_info_range表中包含如下字段和数据
SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0101   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  11    0101   11001              20         300
2015  03    0102   11001              20         300
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600

--按branch进行分组，并在组内按amount进行排序
SELECT branch,amount,year,month,quantity,
       DENSE_RANK() OVER (PARTITION BY branch ORDER BY amount) denserank_res
FROM sales_info
WHERE product='11001'
ORDER BY branch,quantity;
BRANCH      AMOUNT YEAR  MONTH    QUANTITY         DENSERANK_RES
------ ----------- ----- ----- ---------- --------------------
0101           300 2015  11             20                    1
0101           600 2021  05             40                    2
0102           300 2015  03             20                    1
0102           300 2000  12             20                    1
0201           500 2001  01             30                    1
0402           300 2021  10             20                    1
```

DIV

```
div::= DIV "(" expr1 "," expr2 ")"
```

DIV函数执行除法运算，运算规则为：

- 对于小数（FLOAT/DOUBLE/NUMBER），与算术运算符/算法一致。
- 对于整数，作整除算术运算并返回商数。

在算术运算时，YashanDB通过隐式数据转换，将参与运算的数据类型统一到某个数据类型，并按此数据类型返回运算结果，具体规则请参考算术运算符里的数据类型描述。

expr1、expr2的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回不支持该运算。

当expr1或expr2中任一值为NULL时，函数返回NULL。

当expr2的值为0，且expr1的值为非浮点数值时，函数返回Divided by zero错误。

当expr1或expr2为浮点类型特殊值或0时，函数处理规则见下表：

expr1	expr2	DIV(expr1,expr2)
Nan	任意数	Nan
任意数	Nan	Nan
除Nan/Inf/-Inf外任意数	Inf/-Inf	0
Inf/-Inf	除Nan/Inf/-Inf外任意数	Inf/-Inf
Inf/-Inf	Inf/-Inf	Nan
正浮点数	0	Inf
负浮点数	0	-Inf

示例

```
SELECT * FROM numbers_nobit;
     NUMBERA NUMBERB  NUMBERC               NUMBERD     NUMBERE     NUMBERF     NUMBERG
------------ ------- -------- -------------------- ----------- ----------- -----------
          -5      55     5555   5555555555555555555  5.555E+000  5.556E+000         555

SELECT DIV(numberb,numbera) div1,
DIV(numberd,numbere) div2,
DIV(numberf,numberc) div3,
DIV(numbera,'17') div4,
DIV(numberf,0) div5
FROM numbers_nobit;
                DIV1        DIV2        DIV3        DIV4        DIV5
-------------------- ----------- ----------- ----------- -----------
                 -11    1.0E+018    1.0E-003  -.29411765         Inf

SELECT TYPEOF(DIV(numberb,numbera)) type1,
TYPEOF(DIV(numberd,numbere)) type2,
TYPEOF(DIV(numberf,numberc)) type3,
TYPEOF(DIV(numbera,'17')) type4,
TYPEOF(DIV(numberf,0)) type5
FROM numbers_nobit;
TYPE1      TYPE2    TYPE3     TYPE4    TYPE5
---------- -------- --------- -------- ---------
bigint     float    double    number   double

SELECT DIV(CAST('Nan' AS FLOAT), CAST('Inf' AS DOUBLE)) ndi,
```

```
DIV(CAST('-Inf' AS FLOAT), 0) id0,
DIV(12, CAST('-Inf' AS DOUBLE)) ndi,
DIV(12, CAST('Nan' AS FLOAT)) mdn,
DIV(CAST('-Inf' AS FLOAT), CAST('Inf' AS DOUBLE)) idi
FROM DUAL;
        NDI          ID0          NDI          MDN          IDI
----------- ----------- ----------- ----------- -----------
        Nan         -Inf            0          Nan          Nan
```

EMPTY_BLOB

```
empty_blob::= EMPTY_BLOB "("")"
```

EMPTY_BLOB函数常用来初始化一个BLOB变量，常用在INSERT和UPDATE语句中，返回值为一个空的BLOB。

本函数遵循如下规则：

- 函数参数为空。
- 使用LENGTH()函数查询本函数输出值长度时，返回值为0。
- YashanDB支持直接查询EMPTY_BLOB()函数，返回值为一个空的BLOB。
- 该函数不支持列式计算。

示例（HEAP表）

```
CREATE TABLE LOB_EMPTY_BLOB (clob1 CLOB,blob2 BLOB);
INSERT INTO LOB_EMPTY_BLOB VALUES('1234','234');
SELECT * FROM LOB_EMPTY_BLOB;

CLOB1                                                            BLOB2
---------------------------------------------------------------- ----------------------------------------------------------------
----
1234                                                             0234

UPDATE LOB_EMPTY_BLOB SET clob1 = EMPTY_CLOB(), blob2 = EMPTY_BLOB();
SELECT * FROM LOB_EMPTY_BLOB;

CLOB1                                                            BLOB2
---------------------------------------------------------------- ----------------------------------------------------------------
----
```

示例（HEAP表）

```
--查询LENGTH(EMPTY_BLOB())
SELECT LENGTH(EMPTY_BLOB()) res FROM DUAL;

                 RES
--------------------
                   0
```

EMPTY_CLOB

```
empty_clob::= EMPTY_CLOB "("")"
```

EMPTY_CLOB函数常用来初始化一个CLOB变量，常用在INSERT和UPDATE语句中，返回值为一个空的CLOB。

本函数遵循如下规则：

- 函数参数为空。
- 使用LENGTH()函数查询本函数输出值长度时，返回值为0。
- YashanDB支持直接查询EMPTY_CLOB()函数，返回值为一个空的CLOB。
- 该函数不支持列式计算。

示例（HEAP表）

```
CREATE TABLE LOB_EMPTY_CLOB (clob1 CLOB,blob2 BLOB);
INSERT INTO LOB_EMPTY_CLOB VALUES('1234','234');
SELECT * FROM LOB_EMPTY_CLOB;

CLOB1                                                                BLOB2
-------------------------------------------------------------------- --------------------------------------------------------------------
----
1234                                                                 0234

UPDATE LOB_EMPTY_CLOB SET clob1 = EMPTY_CLOB(), blob2 = EMPTY_BLOB();
SELECT * FROM LOB_EMPTY_CLOB;

CLOB1                                                                BLOB2
-------------------------------------------------------------------- --------------------------------------------------------------------
----
```

示例（HEAP表）

```
--查询LENGTH(EMPTY_CLOB ())
SELECT LENGTH(EMPTY_CLOB()) res FROM DUAL;

                 RES
--------------------
                   0
```

ENCRYPT_AES128

```
ENCRYPT_AES128::= ENCRYPT_AES128 "(" expr1 "," expr2 ")"
```

ENCRYPT_AES128函数以expr2为key加密expr1的明文，返回一个VARCHAR类型的密文。

本函数遵循如下规则：

- expr1支持所有可以隐式转换为字符型的类型；expr2只支持VARCHAR/CHAR类型。
- expr2不能为null。
- 当expr1为null时返回null。

> **Caution**:
> ENCRYPT_AES128函数中的密钥由用户管理，请勿将包含ENCRYPT_AES128函数的SQL写入OUTLINE或者SQLMAP视图中，以避免潜在的密钥泄露风险。

示例

```
SELECT encrypt_aes128(area_name, 'admin') FROM area;

ENCRYPT_AES128(AREA_
--------------------------------------------------------------
U��G��
U��K��
U��F��
U��F��
U��G��
```

EXP

```
exp::= EXP "(" expr ")"
```

EXP函数计算以e=2.71828183... 为底，expr表示的数值为指数的数学结果，返回一个DOUBLE类型的数值。

expr的值须为数值型，可以是数值型字符串，对于其他类型，函数返回类型不支持错误。

当expr为NULL时，返回NULL。

由于显示精度差异，本函数的计算结果与Oracle同函数的计算结果在前15位可以保持一致，之后可能会有差异。

示例

```
SET numwidth 30
SELECT EXP(4.444444) res FROM DUAL;
                              RES
------------------------------
        8.5152519871979379E+001

SELECT EXP(b'0101') res FROM DUAL;
                              RES
------------------------------
         1.484131591025766E+002
```

EXTRACT

```
extract::= EXTRACT "(" (YEAR|MONTH|DAY|HOUR|MINUTE|SECOND) FROM expr ")"
```

EXTRACT函数对给定参数expr进行年、月、日、小时、分、秒等数值的提取，其返回值类型有以下几种情况：

- 当expr值为NULL时，返回NULL。
- 当expr值的数据类型不为DATE、TIMESTAMP、TIME、INTERVAL DAY TO SECOND、INTERVAL YEAR TO MONTH时，返回类型不符合预期。
- expr值的数据类型与年、月、日、小时、分、秒的指定存在如下对应关系，其中出现N/A时返回Illegal format错误，否则按最后一列类型返回：

Field	TIMESTAMP	TIME	DATE	INTERVAL DAY TO SECOND	INTERVAL YEAR TO MONTH	返回类型
DAY	一个月中的日 (1-31)	N/A	一个月中的日 (1-31)	天数	N/A	INT
HOUR	一天中的小时 (0-23)	一天中的小时 (0-23)	N/A	一天中的小时 (0-23)	N/A	INT
MINUTE	一小时中的分 (0-59)	一小时中的分 (0-59)	N/A	一小时中的分 (0-59)	N/A	INT
MONTH	一年中的月 (1-12)	N/A	一年中的月 (1-12)	N/A	月数	INT
SECOND	一分钟的秒 (0-59.999999)	一分钟的秒 (0-59.999999)	N/A	一分钟的秒 (0-59.999999)	N/A	NUMBER
YEAR	年份	N/A	年份	N/A	年数	INT

示例

```
--创建times表，包含DATE、TIMESTAMP、TIME、INTERVAL DAY TO SECOND、INTERVAL YEAR TO MONTH类型的列字段
CREATE TABLE times (timea DATE DEFAULT SYSDATE,
timeb TIMESTAMP DEFAULT SYSDATE,
timec TIME DEFAULT SYSDATE,
timed INTERVAL DAY TO SECOND,
timee INTERVAL YEAR TO MONTH);
INSERT INTO times VALUES (DEFAULT,DEFAULT,DEFAULT,INTERVAL '5' DAY,INTERVAL '2' YEAR);
COMMIT;

SELECT * FROM times;
TIMEA                TIMEB                          TIMEC            TIMED                TIMEE
-------------------- ------------------------------ ---------------- -------------------- ------
2022-01-17 20:47:19   2022-01-17 20:47:19.000000   20:47:19.000000   +05 00:00:00.000000  +02-00

SELECT EXTRACT(YEAR FROM timea) Year,
EXTRACT(SECOND FROM timeb) Second,
EXTRACT(HOUR FROM timec) Hour,
EXTRACT(DAY FROM timed) Day,
EXTRACT(MONTH FROM timee) Month
FROM times;
        YEAR      SECOND         HOUR          DAY        MONTH
------------ ----------- ------------ ------------ ------------
        2022          19           20            5            0
```

FIND_IN_SET

```
find_in_set::= FIND_IN_SET "(" expr "," strlist ")"
```

FIND_IN_SET函数功能是查找expr表示的字符串在字符串列表strlist中第一次出现的位置（以1为基），函数返回值为INT类型。

expr、strlist

expr和strlist的值须为字符型，或除布尔类型、RAW类型外的其他可转换为字符型的类型。

- 本函数在查找过程中对大小写不敏感。
- strlist字符串列表是由','分割的子串组成的字符串。基于此规则，当expr中包含','时，函数将不能保证返回结果的正确性。
- 当expr或strlist中任一值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。

示例

```
SELECT FIND_IN_SET('b', 'a,b,c') a,
FIND_IN_SET('深圳', '广东省,深圳') b,
FIND_IN_SET(3, '1,2,3') c,
FIND_IN_SET('a', 'A,B,C') d
FROM DUAL;

           A            B            C            D
------------ ------------ ------------ ------------
           2            2            3            1
```

FIRST_VALUE

```
first_value::= FIRST_VALUE "(" expr [(RESPECT|IGNORE) NULLS] ")" OVER "(" analytic_clause ")"
analytic_clause::= "(" (query_partition_clause | order_by_clause) [windowing_clause] ")"
```

FIRST_VALUE为窗口函数，并支持滑动窗口。该函数依据给定的窗口条件计算出窗口数据集合，并返回该集合的第一行记录所对应的expr的值。如果第一个值为NULL且未指定IGNORE NULLS，则函数返回 NULL。

窗口函数不可嵌套，因此expr为除窗口函数之外的其他通用表达式，expr的数据类型可以是CLOB、BLOB、NCLOB、XMLTYPE、JSON、UDT以外的其他数据类型。

(RESPECT|IGNORE) NULLS

指定expr的空值是否被包括在函数的计算中，缺省为RESPECT NULLS，即如果集合中的第一个值为NULL，函数将返回NULL。

如果指定IGNORE NULLS，则函数返回集合中的第一个非空值，当集合中所有值都为空时，返回NULL。此设置对于数据密集化很有用。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME            DHQ             EMPLOYEE_COUNT
------- -------------------- --------------- --------------
01      华东                 Shanghai
02      华西                 Chengdu                   300
03      华南                 Guangzhou                 400
04      华北                 Beijing                   300
05      华中                 Wuhan

--返回按AREA_NO排序的每个窗口中的第一个EMPLOYEE_COUNT,windowing_clause省略时的默认窗口为UNBOUNDED PRECEDIN至CURRENT ROW
SELECT FIRST_VALUE(employee_count) OVER(ORDER BY AREA_NO) res FROM area1;
                RES
-------------------




--返回按AREA_NO排序的每个窗口中的第一个非空EMPLOYEE_COUNT,windowing_clause省略时的默认窗口为UNBOUNDED PRECEDIN至CURRENT ROW
SELECT FIRST_VALUE(employee_count IGNORE NULLS) OVER(ORDER BY AREA_NO) res FROM area1;
                RES
-------------------

                300
                300
                300
                300
```

analytic_clause

窗口函数的通用语法。

在windowing_clause语句中无论指定的是ROWS还是RANGE，FIRST_VALUE都将对参数列进行排序，以保证在指定RANGE的情况下本函数返回值的稳定性（当order_by_clause具有相同排名，则函数返回相同排名下任意一行的值都是合理的，由此产生了返回值的不稳定性），但ROWS情况下仍会存在此不稳定性。

进行排序的参数列为：query_partition_clause中的expr 、order_by_clause中的排序列和FIRST_VALUE函数的参数expr。

示例（单机部署）

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
```

```
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778

--按年分组，并在每组记录中排名该年收入最高的机构
SELECT year,revenue_total,
FIRST_VALUE(branch) OVER (PARTITION BY year ORDER BY revenue_total DESC) fr
FROM finance_info;
YEAR  REVENUE_TOTAL FR
----- ------------- --------------------
2001           2888 0201
2021          38888 0101
2021          37778 0101
2021          28888 0101

--每月至今收入最高的机构
SELECT year,month,revenue_total,
FIRST_VALUE(branch) OVER (ORDER BY revenue_total DESC ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) fr
FROM finance_info;
YEAR  REVENUE_TOTAL FR
----- ------------- --------------------
2021  01            38888 0101
2021  02            37778 0101
2021  01            28888 0101
2001  01             2888 0101

--每月在三个月内收入最高的机构
SELECT year,month,revenue_total,
FIRST_VALUE(branch) OVER (ORDER BY revenue_total DESC ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) fr
FROM finance_info;
YEAR  REVENUE_TOTAL FR
----- ------------- --------------------
2021  01            38888 0101
2021  02            37778 0101
2021  01            28888 0101
2001  01             2888 0201
```

FLOOR

```
floor::= FLOOR "(" expr ")"
```

FLOOR函数对给定参数expr的值进行向下取整，其返回类型为：

- 当expr的值为数值型数据时，返回与其相同类型的数据。
- 当expr的值为字符型数据时，返回NUMBER类型的数据。
- 当expr的值为NULL时，返回NULL。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

示例

```
SELECT FLOOR(7) floor1,
FLOOR('0.97') floor2,
FLOOR(6.22) floor3
FROM DUAL;
      FLOOR1      FLOOR2      FLOOR3
------------ ----------- -----------
           7           0           6
```

GET_TYPE_NAME

```
get_type_name::= GET_TYPE_NAME "(" expr ")"
```

GET_TYPE_NAME函数将expr作为数据类型ID，用于查询类型的名称，并将结果作为VARCHAR类型的字符串返回。

本函数遵循如下规则：

该函数不支持列式计算。

expr

expr须为除BIT外数值型数据，或可转换为INTEGER的字符型数据，取值范围[0,254]。对于其他类型，函数返回类型不支持。

expr不能为NULL，当expr的值为NULL时，函数报错。

示例（单机HEAP表）

```
SELECT GET_TYPE_NAME(4) TYPE_NAME FROM DUAL;

TYPE_NAME
---------
INTEGER

SELECT GET_TYPE_NAME(5.21) TYPE_NAME FROM DUAL;

TYPE_NAME
---------
BIGINT

SELECT GET_TYPE_NAME(5.71) TYPE_NAME FROM DUAL;

TYPE_NAME
---------
UTINYINT

SELECT GET_TYPE_NAME('26') TYPE_NAME FROM DUAL;

TYPE_NAME
---------
VARCHAR
```

GIS Function

地理信息处理函数（GIS Function）是指对空间地理信息进行处理的一系列函数，仅支持在单机HEAP表中使用。

GEOMETRYTYPE

ST_AREA

ST_ASBINARY

ST_ASEWKB

ST_ASGEOJSON

ST_ASHEXEWKB

ST_ASLATLONTEXT

ST_ASTEXT

ST_BOUNDARY

ST_BUFFER

ST_BUILDAREA

ST_CLIPBYBOX2D

ST_CLOSESTPOINT

ST_COLLECT

ST_CONCAVEHULL

ST_CONTAINS

ST_CONTAINSPROPERLY

ST_COVEREDBY

ST_COVERS

ST_CROSSES

ST_DIFFERENCE

ST_DISJOINT

ST_DISTANCE

ST_DUMP

ST_DWITHIN

ST_ENVELOPE

ST_EQUALS

ST_EXTENT

ST_GEOMCOLLFROMTEXT

ST_GEOMETRICMEDIAN

ST_GEOMETRYTYPE

ST_GEOMFROMEWKB：需结合ST_GEOMETRY的输出函数进行查询结果输出。

ST_GEOMFROMGEOJSON

ST_GEOMFROMTEXT：需结合ST_GEOMETRY的输出函数进行查询结果输出。

ST_GEOMFROMWKB：需结合ST_GEOMETRY的输出函数进行查询结果输出。

ST_INTERSECTION

ST_INTERSECTS

ST_ISCLOSED

ST_ISEMPTY

ST_ISVALID

ST_ISSIMPLE

ST_LENGTH

ST_LINEFROMTEXT

ST_LINEMERGE

ST_LONGESTLINE

ST_MAKEENVELOPE

ST_MAKELINE

ST_MAKEPOINT

ST_MAXDISTANCE

ST_MULTI

ST_OVERLAPS

ST_PERIMETER

ST_POINT

ST_POINTZ

ST_POLYGON

ST_RELATE

ST_SETSRID

ST_SHORTESTLINE

ST_SIMPLIFY

ST_SRID

ST_SPLIT

ST_TOUCHES

ST_TRANSFORM

ST_UNION

ST_WITHIN

ST_X

ST_Y

ST_Z

GEOMETRYTYPE

```
geometrytype::= GEOMETRYTYPE "(" geometry ")"
```

GEOMETRYTYPE函数用于打印输入的geometry的类型。

该函数会根据输入的geometry返回该geometry的数据类型，返回值为VARCHAR类型，返回值全为大写。

与ST_GEOMETRYTYPE函数相比，GEOMETRYTYPE函数的返回值无ST_前缀且为全大写。

geometry

通用表达式，其值须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT GEOMETRYTYPE(ST_GeomFromText('POINT(-58.2687 29.149)',1356)) res FROM DUAL;

RES
----------------------------------------------------------------
POINT

SELECT GEOMETRYTYPE(ST_GeomFromText('LINESTRING(1 6,3 2,9 7)',1356)) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING

--参数包含NULL
SELECT GEOMETRYTYPE(ST_GeomFromText(null,1356)) res FROM DUAL;

RES
----------------------------------------------------------------
```

ST_AREA

```
st_area::= ST_AREA "(" geometry ")"
```

ST_AREA函数用于计算geometry的面积，或者说计算的是区域的面积，对于不能构成区域的几何图形，则返回0。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry是POLYGON、MULTIPOLYGON或GEOMETRY COLLECTION类型会返回对应的面积，其它的geometry类型会返回0。
- geometry的空间参考系标识号（SRID）必须在spatial_ref_sys系统表中定义或者为0，否则报错。
- geometry的空间参考系标识号（SRID）在spatial_ref_sys中对应的srs_type如果是GEOGRAPHY2D或者GEOGRAPHY3D，会切换到大地坐标算法计算，否则会使用投影坐标算法。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 该函数只会计算2D结果，如果输入的是三维，则会忽略Z坐标进行计算。
- 对于输入的经纬度坐标，如果输入的数值不在有效经纬度范围内，则会转换成有效经纬度进行计算。

示例（单机HEAP表）

```
SELECT ST_Area(ST_GeomFromText('POLYGON ((-71.17 42.39,-72.17 43.39,-72.17 44.39,-71.17 42.39))', 4326)) res FROM dual;

RES
-----------
4.575E+009

SELECT ST_Area(ST_GeomFromText('POLYGON((0 0, 10 0, 10 10, 0 10, 0 0))', 3385)) res FROM dual;

RES
-----------
1.0E+002
```

ST_ASBINARY

```
st_asbinary::= ST_ASBINARY "(" geometry [","  byteorder ]")"
```

```
st_asbinary::= ST_ASBINARY "(" box2d [","  byteorder ]")"
```

ST_ASBINARY函数根据输入的geometry（box2d）和byteorder，返回该geometry（box2d对应geometry）的WKB（Well-Known Binary）表示。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

box2d

通用表达式，其值必须为有效的BOX2D类型的数据。对于该参数，输出规则如下：

- 如果box2d是一个点，则输出对应POINT的WKB。
- 如果box2d是一个线，则输出对应LINESTRING的WKB。
- 如果box2d是一个矩形，则输出对应POLYGON的WKB。
- 输出的geometry的SRID为0。

byteorder

byteorder表示输出结果的字节序，其值为VARCHAR类型，遵循如下规则：

- 支持能够隐式转换成VARCHAR的数据类型。
- "NDR"表示小端序，"XDR"表示大端序，不区分大小写，输入其他字符串则报错。
- 该参数可以省略，省略时默认为编码使用服务器计算机字节序。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
--输出WKB不包含SRID
SELECT ST_AsBinary(ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

RES
--------------------------------------------------------------------
01010000000000000000F03F0000000000000040

SELECT ST_AsBinary(ST_GeomFromText('POINT(1 2)', 4326)) res FROM DUAL;

RES
--------------------------------------------------------------------
01010000000000000000F03F0000000000000040

--大小端序
SELECT ST_AsBinary(ST_GeomFromText('POINT(1 2)'), 'NDR') res FROM DUAL;

RES
--------------------------------------------------------------------
01010000000000000000F03F0000000000000040

SELECT ST_AsBinary(ST_GeomFromText('POINT(1 2)'), 'XDR') res FROM DUAL;

RES
--------------------------------------------------------------------
00000000013FF0000000000000040000000000000000

--box2d为点
SELECT ST_AsBinary(BOX2D(1, 1, 2, 2)) res FROM DUAL;

RES
```

```
------------------------------------------------------------------
0101000000000000000000F03F0000000000000040

--box2d为线
SELECT ST_AsBinary(BOX2D(1, 3, 2, 2)) res FROM DUAL;

RES
------------------------------------------------------------------
0102000000020000000000000000F03F0000000000000040000000000000008400000000000000040

--box2d为矩形
SELECT ST_AsBinary(BOX2D(1, 3, 2, 3)) res FROM DUAL;

RES
------------------------------------------------------------------
0103000000010000000500000000000000000F03F0000000000000040000000000000F03F0000000000000840000000000000084000000000000008400000000000000840000000000000084000000000000008400000000000000840000000000000F03F0000000000000040
```

ST_ASEWKB

```
st_asewkb::= ST_ASEWKB "(" geometry ["," byteorder ]")"
```

```
st_asewkb::= ST_ASEWKB "(" box2d ["," byteorder ]")"
```

ST_ASEWKB函数根据输入的geometry（box2d）和byteorder，返回该geometry（box2d对应geometry）的EWKB（Extended Well-Known Binary）表示。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

box2d

通用表达式，其值必须为有效的BOX2D类型的数据。对于该参数，输出规则如下：

- 如果box2d是一个点，则输出对应POINT的EWKB。
- 如果box2d是一个线，则输出对应LINESTRING的EWKB。
- 如果box2d是一个矩形，则输出对应POLYGON的EWKB。
- 输出的geometry的SRID为0。

byteorder

byteorder表示输出结果的字节序，其值为VARCHAR类型，遵循如下规则：

- 支持能够隐式转换成VARCHAR的数据类型。
- "NDR"表示小端序，"XDR"表示大端序，不区分大小写，输入其他字符串则报错。
- 该参数可以省略，省略时默认为编码使用服务器计算机字节序。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
--输出EWKB不包含SRID
SELECT ST_AsEwkb(ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

RES
----------------------------------------------------------------
01010000000000000000F03F0000000000000040

SELECT ST_AsEwkb(ST_GeomFromText('POINT(1 2)', 4326)) res FROM DUAL;

RES
----------------------------------------------------------------
0101000020E6100000000000000000F03F0000000000000040

--大小端序
SELECT ST_AsEwkb(ST_GeomFromText('POINT(1 2)'), 'NDR') res FROM DUAL;

RES
----------------------------------------------------------------
01010000000000000000F03F0000000000000040

SELECT ST_AsEwkb(ST_GeomFromText('POINT(1 2)'), 'XDR') res FROM DUAL;

RES
----------------------------------------------------------------
00000000013FF0000000000000040000000000000000

--box2d为点
SELECT ST_AsEwkb(BOX2D(1, 1, 2, 2)) res FROM DUAL;
```

```
RES
----------------------------------------------------------------
0101000000000000000000F03F0000000000000040

--box2d为线
SELECT ST_AsEwkb(BOX2D(1, 3, 2, 2)) res FROM DUAL;

RES
----------------------------------------------------------------
0102000000020000000000000000F03F0000000000000040000000000000008400000000000000040

--box2d为矩形
SELECT ST_AsEwkb(BOX2D(1, 3, 2, 3)) res FROM DUAL;

RES
----------------------------------------------------------------
0103000000010000000500000000000000000F03F0000000000000040000000000000F03F0000000000000840000000000000008400000000000000840000000000000084000000000000008400000000000000840000000000000084000000000000F03F0000000000000040
```

ST_ASGEOJSON

```
st_asgeojson::= ST_ASGEOJSON "(" geometry [","  precision ] [","  options ] ")"
```

ST_ASGEOJSON函数根据输入的geometry，返回该geometry的GeoJSON表示，GeoJSON是一种使用JavaScript对象符号(JSON)编码各种地理数据结构的格式。只支持二维几何图形，三维几何图形会丢弃z轴坐标。输出的GeoJSON不包含坐标参考系。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

precision

precision数据类型为INT，该参数用于语法兼容，无实际含义。

- 支持能够隐式转换成INT的类型，如果输入的是小数则进行四舍五入转换。
- 该参数可以省略。

options

options的数据类型是INT，该参数用于语法兼容，无实际含义。

- 支持能够隐式转换成INT的类型，如果输入的是小数则进行四舍五入转换。
- 该参数可以省略，如该函数仅有2个参数时默认省略options。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
-- empty
SELECT ST_AsGeoJSON(ST_GeomFromText('POINT EMPTY')) res FROM DUAL;

RES
----------------------------------------------------------------
{"type":"Point","coordinates":[]}

-- precision
SELECT ST_AsGeoJSON(ST_GeomFromText('POINT(1.1111111 1.1111111)', 4326), -2) res FROM DUAL;

RES
----------------------------------------------------------------
{"type":"Point","coordinates":[1.1111111,1.1111111]}

-- options
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(1 1, 2 2, 3 3, 4 4)'), 0, 1) res FROM DUAL;

RES
----------------------------------------------------------------
{"type":"LineString","coordinates":[[1.0,1.0],[2.0,2.0],[3.0,3.0],[4.0,4.0]]}
```

ST_ASHEXEWKB

```
st_ashexewkb::= ST_ASHEXEWKB "(" geometry "," byteorder ")"
```

ST_ASHEXEWKB函数根据给输入的geometry（box2d）和byteorder，返回该geometry（box2d对应geometry）的hexewkb（Extended Well-Known Binary对应的十六进制形式）表示。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

box2d

通用表达式，其值必须为有效的BOX2D类型的数据。对于该参数，输出规则如下：

- 如果box2d是一个点，则输出对应POINT的HEXEWKB。
- 如果box2d是一个线，则输出对应LINESTRING的HEXEWKB。
- 如果box2d是一个矩形，则输出对应POLYGON的HEXEWKB。
- 输出的geometry的SRID为0。

byteorder

byteorder的数据类型为VARCHAR，表示输出结果的字节序。

- 支持能够隐式转换成VARCHAR的数据类型。
- "NDR"表示小端序，"XDR"表示大端序，不区分大小写，输入其他字符串则报错。
- 该参数可以省略，省略时默认使用服务器计算机字节序。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
--输出wkb包含srid
SELECT ST_AsHexEwkb(ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

RES
----------------------------------------------------------------
0101000000000000000000F03F0000000000000040

SELECT ST_AsHexEwkb(ST_GeomFromText('POINT(1 2)', 4326)) res FROM DUAL;

RES
----------------------------------------------------------------
0101000020E6100000000000000000F03F0000000000000040

--大小端序
SELECT ST_AsHexEwkb(ST_GeomFromText('POINT(1 2)'), 'NDR') res FROM DUAL;

RES
----------------------------------------------------------------
0101000000000000000000F03F0000000000000040

SELECT ST_AsHexEwkb(ST_GeomFromText('POINT(1 2)'), 'XDR') res FROM DUAL;

RES
----------------------------------------------------------------
00000000013FF0000000000000400000000000000

--参数包含NULL
SELECT ST_AsHexEwkb(NULL, 'XDR') res FROM DUAL;

RES
----------------------------------------------------------------

SELECT ST_AsHexEwkb(ST_GeomFromText('POINT(1 2)'), NULL) res FROM DUAL;
```

```
RES
------------------------------------------------------------
```

ST_ASLATLONTEXT

```
st_aslatlontext::= ST_ASLATLONTEXT "(" geometry "," format ")"
```

ST_ASLATLONTEXT函数根据输入的geometry，返回该geometry在经纬度投影中的度、分、秒表示形式。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，且类型必须为POINT。

format

format是一个格式字符串，长度限制到1024，用于表示返回值的格式。

格式字符串有如下四个选项：

- D：代表度数
- M：代表分钟
- S：代表秒
- C：代表基本方向

示例（单机HEAP表）

```
format = 'D°M''S.SSS"C'.
```

该参数需遵守如下规则：

- 必须包含D。
- 如省略M，则度数以十进制显示，且度数精度与指定的位数相同。
- 如省略S，则分钟将显示为十进制，且分钟精度与指定的位数相同。
- 如省略C，则度数在南或西时以“-”符号显示。
- 可以在各选项中通过重复使用D、M、S字符以指定所需的宽度和精度，C字符不可重复。
- D、M、S和C选项在格式字符串中只能出现一次。
- 如省略（或零长度），将使用默认格式 `D°M''S.SSS"C` 。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--省略format
SELECT ST_AsLatLonText(ST_GeomFromText('POINT(-58.2687 29.149)')) res FROM DUAL;

RES
------------------------------------------------------------------
29°8''56.400"N 58°16''7.320"W

--归一化坐标
SELECT ST_AsLatLonText(ST_GeomFromText('POINT (1234 56789)')) res FROM DUAL;

RES
------------------------------------------------------------------
89°0''0.000"S 26°0''0.000"W

--包含format
SELECT ST_AsLatLonText(ST_GeomFromText('POINT(-58.2687 29.149)'), 'D degrees, M minutes, S seconds C') res FROM DUAL;

RES
------------------------------------------------------------------
29 degrees, 8 minutes, 56 seconds N 58 degrees, 16 minutes, 7 seconds W

--选项重复使用时返回错误
SELECT ST_ASLATLONTEXT(ST_GEOMFROMTEXT('POINT(-58.2687 29.149)'), 'D°M.MMMM''S"CD') res FROM DUAL;
```

```
YAS-07202 plugin execution error, bad format, cannot include degrees/minutes/seconds more than once

--参数存在NULL
SELECT ST_AsLatLonText(ST_GeomFromText('POINT(-58.2687 29.149)'), NULL) res FROM DUAL;

RES
------------------------------------------------------------------

SELECT ST_AsLatLonText(NULL, 'D degrees, M minutes, S seconds C') res FROM DUAL;

RES
------------------------------------------------------------------
```

ST_ASTEXT

```
st_astext::= ST_ASTEXT "(" geometry [ "," maxdecimaldigits ] ")"
```

```
st_astext::= ST_ASTEXT "(" box2d [ "," maxdecimaldigits ] ")"
```

ST_ASTEXT函数根据输入的geometry（box2d）和maxdecimaldigits，返回该geometry（box2d对应geometry）的WKT（Well-Known Text）表示。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

box2d

通用表达式，其值必须为有效的BOX2D类型的数据。对于该参数，输出规则如下：

- 如果box2d是一个点，则输出对应POINT的WKT。
- 如果box2d是一个线，则输出对应LINESTRING的WKT。
- 如果box2d是一个矩形，则输出对应POLYGON的WKT。
- 输出的geometry的SRID为0。

maxdecimaldigits

maxdecimaldigits表示输出结果中小数的位数，如果不够则补零，其值为INT类型，遵循如下规则：

- 支持能够隐式转换成INT的类型，如果输入的是小数则进行四舍五入转换。
- 该参数可以省略，省略时默认值是15。
- 如果输入的是负数，则会按照0输出。
- 如果输入的值超过了16，则会给按照16进行处理，但第16位小数不一定有效。
- 结果不会以科学计数法的形式输出。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
--小数位数
SELECT ST_AsText(ST_GeomFromText('POINT(1 2)', 4326), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1 2)

SELECT ST_AsText(ST_GeomFromText('POINT(1 2)', 4326), 1) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1.0 2.0)

SELECT ST_AsText(ST_GeomFromText('POINT(1 2)', 4326), 100) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1.0000000000000000 2.0000000000000000)

SELECT ST_AsText(ST_GeomFromText('POINT(1 2)', 4326), -10) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1 2)

--维度混合
SELECT ST_AsText(ST_GeomFromText('MULTIPOINT(1 2 3, 1 1)', 4326), 0) res FROM DUAL;
```

```
RES
-----------------------------------------------------------------
MULTIPOINT Z (1 2 3, 1 1 0)

--支持inf和nan的输入
SELECT ST_AsText(St_GeomFromText('MULTIPOINT(inf inf,nan nan)')) res FROM DUAL;

RES
-----------------------------------------------------------------
MULTIPOINT (inf inf, EMPTY)

--box2d为点
SELECT ST_AsText(BOX2D(1, 1, 2, 2), 0) res FROM DUAL;

RES
-----------------------------------------------------------------
POINT (1 2)

--box2d为线
SELECT ST_AsText(BOX2D(1, 3, 2, 2), 0) res FROM DUAL;

RES
-----------------------------------------------------------------
LINESTRING (1 2, 3 2)

--box2d为矩形
SELECT ST_AsText(BOX2D(1, 3, 2, 3), 0) res FROM DUAL;

RES
-----------------------------------------------------------------
POLYGON ((1 2, 1 3, 3 3, 3 2, 1 2))
```

ST_BOUNDARY

```
st_boundary::= ST_BOUNDARY "(" geometry ")"
```

ST_BOUNDARY函数用于计算输入的geometry的组合边界，返回值为ST_GEOMETRY类型数据。

该函数根据输入geometry的不同数据类型，返回不同的边界类型：

- POINT和MULTIPOINT类型的边界为空，返回值为GEOMETRYCOLLECTION EMPTY。
- LINESTRING和MULTILINESTRING类型的边界为端点，返回值为MULTIPOINT类型，如输入闭环线则返回MULTIPOINT EMPTY。
- POLYGON和MULTIPOLYGON类型的边界为分割外部和内部的线性环，返回值为LINESTRING类型。

对于LINESTRING和MULTILINESTRING类型的输入，本函数计算时仅计算x、y坐标，z坐标不参与计算，即如输入值的端点x、y坐标相等时，则返回MULTIPOINT EMPTY；输出时间点按照x坐标从负到正的顺序，y坐标从负到正的顺序输出，输出值包含z坐标。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，但不能为GEOMETRY COLLECTION类型。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_BOUNDARY函数根据给定的GEOMETRY返回一个ST_GEOMETRY数据
--输入值为POINT类型
SELECT ST_ASTEXT(ST_BOUNDARY(ST_GeomFromText('POINT(2 6 9 5)'))) res FROM DUAL;

RES
------------------------------------------------------------------
GEOMETRYCOLLECTION EMPTY

--输入值为LINESTRING类型
SELECT ST_ASTEXT(ST_BOUNDARY(ST_GeomFromText('LINESTRING(1 1, 0 0, -1 1)')),0) res FROM DUAL;

RES
------------------------------------------------------------------
MULTIPOINT (1 1, -1 1)

--输入值为POLYGON类型
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((1 1, 0 0, -1 1, 1 1))')),0) res FROM DUAL;

RES
------------------------------------------------------------------
LINESTRING (1 1, 0 0, -1 1, 1 1)


--参数包含NULL
SELECT ST_ASTEXT(ST_BOUNDARY(ST_GeomFromText(null, 4316))) res FROM DUAL;

RES
------------------------------------------------------------------
```

ST_BUFFER

```
st_buffer::= ST_BUFFER "(" geometry "," width [ "," style ] ")"
```

ST_BUFFER函数的功能是返回一个ST_GEOMETRY类型数据，该数据覆盖从输入的geometry到给定的距离width内的所有点，实际上得到的计算结果始终是一个有效的POLYGON数据。

当输入的参数存在NULL时，函数返回NULL。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- 仅支持计算2D结果。如geometry的坐标中有Z轴，则忽略Z坐标进行计算，计算结果仍然是一个2D的ST_GEOMETRY数据。
- 如输入的是一个EMPTY的ST_GEOMETRY数据，则会返回POLYGON EMPTY（如果一个GEOMETRYCOLLECTION只有其中一部分是EMPTY，则计算结果不一定是POLYGON EMPTY）。
- 输入的geometry的坐标中不允许含有非法数字（如inf、nan），否则报错。

width

width用于规定距离，其值为DOUBLE类型，遵循如下规则：

- 支持能够隐式转换成DOUBLE的数据类型。
- 如果输入的width是一个负值，则会缩小该geometry，极端情况下会使结果的POLYGON数据缩小为0，从而返回POLYGON EMPTY；对于POINT和LINESTRING类型而言，如果width是负值，则始终返回POLYGON EMPTY。
- width的单位是输入的geometry的空间参考系的单位。

style

style用于控制结果的精度和样式，其值为VARCHAR类型。该参数可以省略，此时会对精度和样式设置默认值。

- 支持能够隐式转换成VARCHAR的数据类型。
- style有五个参数可以设置，可通过 `key=value` 的方式指定，不同键值对之间使用空格进行分割，键值对的顺序没有限制，且无视大小写，部分键值设有别名，具体规则如下：
  - `quad_segs=#` ：表示四分之一圆有几个片段（线段），默认值为8，小于0时取0，最大值为262144，超过最大值则按最大值计算。如输入的是小数，则会截断成整数；如输入的是无效的数据，则转换成0；如计算结果长度超过32000，则报错。
  - `endcap=round|flat(butt)|square` ：端点样式，默认值为round。
  - `mitre_limit(miter_limit)=#.#` ：用于限制斜接比率，只能影响 `join=mitre(miter)` 的情况，默认值为5.0。
  - `join=round|mitre(miter)|bevel` ：连接样式，默认值为round。
  - `side=both|left|right` ：left和right表示形成一个单边的图形，仅影响LINESTRING类型，不影响POINT或者POLYGON类型（仍形成封闭的ST_GEOMETRY数据），both则会形成一个封闭的ST_GEOMETRY数据，默认值为both。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POLYGON((50 50, 150 150, 150 50, 50 50))'), -2, 'quad_segs=1'), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((55 52, 148 145, 148 52, 55 52))

SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POLYGON((50 50, 150 150, 150 50, 50 50))'), 1, 'quad_segs=1'), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((50 49, 49 50, 49 51, 149 151, 150 151, 151 150, 151 50, 150 49, 50 49))

SELECT ST_AsText(ST_Buffer(ST_GeomFromText('LINESTRING(1 3 5, 2 4 6, 1 3 5)'), 1, 'quad_segs=2 join=bevel'), 0) res FROM
DUAL;

RES
----------------------------------------------------------------
POLYGON ((0 3, 0 4, 1 5, 3 3, 2 2, 1 2, 0 2, 0 3))
```

```
SELECT ST_AsText(ST_Buffer(ST_GeomFromText('LINESTRING(1 3 5, 2 4 6, 1 3 5)'), 1, 'quad_segs=2'), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((0 3, 0 4, 1 5, 2 5, 3 5, 3 4, 3 3, 2 2, 1 2, 0 2, 0 3))

SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POINT(100 90 100)'), 1, 'quad_segs=1'), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((101 90, 100 89, 99 90, 100 91, 101 90))
```

ST_BUILDAREA

```
st_buildarea::= ST_BUILDAREA "(" geometry ")"
```

ST_BUILDAREA函数用于将输入的geometry中的线条组合成一个多边形。

本函数遵守如下规则：

- 本函数针对二维空间对象，当输入的几何对象是三维对象时，输出结果仍为三维对象，但第三维坐标不参与计算。即计算过程中判断线圈是否闭合、是否存在空间包含关系时，只有前二维坐标参与计算，二维空间上闭合但三维空间不闭合的线圈，仍视为闭合。
- 输入参数为NULL时，函数返回NULL。
- 输入参数为空的几何对象时，函数返回一个空的多边形。
- 输入参数为点、多点时，返回NULL。
- 输入参数为一个LineString时，如果LineString为一个首尾闭合的线圈，则返回一个多边形，否则返回NULL。
- 输入参数为一个多边形时，返回多边形。
- 输入参数为集合对象（MultiLineString、MultiPolygon、GeometryCollection）时，函数的表现如下：
  - 如果某个成员单独执行ST_BuildArea返回NULL，则该成员不对结果产生影响。
  - 如果多个成员之间存在包含关系（例如一个线圈包含另外一个线圈，或一个多边形包含另一个多边形），则被包含的对象将被视为多边形的洞。
- 返回的几何对象，其SRID与输入的几何对象一致。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

示例

```
--不闭合的线圈，返回NULL
SELECT st_astext(st_buildarea(st_geomfromtext('linestring(0 0, 4 0, 4 4, 0 4)')), 0) FROM dual;

ST_ASTEXT(ST_BUILDAR
--------------------------------------------------------------------


--闭合线圈返回由线圈构成的多边形
SELECT st_astext(st_buildarea(st_geomfromtext('linestring(0 0, 4 0, 4 4, 0 4, 0 0)')), 0) FROM dual;

ST_ASTEXT(ST_BUILDAR
--------------------------------------------------------------------
POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0))

SELECT st_astext(st_buildarea(st_geomfromtext('polygon((0 0, 4 0, 4 4, 0 4, 0 0))')), 0) FROM dual;

ST_ASTEXT(ST_BUILDAR
--------------------------------------------------------------------
POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0))

SELECT st_astext(st_buildarea(st_geomfromtext('polygon((0 0, 4 0, 4 4, 0 4, 0 0),(2 2, 3 2, 3 3, 2 3, 2 2))')), 0) FROM dual;

ST_ASTEXT(ST_BUILDAR
--------------------------------------------------------------------
POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0), (2 2, 3 2, 3 3, 2 3, 2 2))

--两个线圈存在包含关系，内部线圈围成的区域视为多边形的洞
SELECT st_astext(st_buildarea(st_geomfromtext('multilinestring((0 0, 4 0, 4 4, 0 4, 0 0),(2 2, 3 2, 3 3, 2 3, 2 2))')), 0)
FROM dual;

ST_ASTEXT(ST_BUILDAR
--------------------------------------------------------------------
POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0), (2 2, 3 2, 3 3, 2 3, 2 2))

--二维闭合三维不闭合的线圈，仍视为闭合，输出结果仍包含三维坐标
SELECT st_astext(st_buildarea(st_geomfromtext('linestring(0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 1)')), 0) FROM dual;

ST_ASTEXT(ST_BUILDAR
```

```
------------------------------------------------------------
POLYGON Z ((0 0 1, 0 4 0, 4 4 0, 4 0 0, 0 0 0))
```

ST_CLIPBYBOX2D

```
st_clipbybox2d::= ST_CLIPBYBOX2D "(" geometry1 "," geometry2 ")"
```

ST_CLIPBYBOX2D函数返回geometry1中由geometry2计算出的矩形裁剪框裁剪后的几何图形，返回值为ST_GEOMETRY类型数据。

> **Note**：
> 由geometry2计算出的矩形裁剪框坐标为（ Xmin, Ymin, Xmax, Ymax ），其中Xmin为geometry2中X坐标最小值，Ymin为geometry2中Y坐标最小值，Xmax为geometry2中X坐标最大值，Ymax为geometry2中Y坐标最大值。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

本函数遵守如下规则：

- 当被裁剪图形为EMPTY时，函数返回被裁剪图形对应类型的EMPTY。
- 当被裁剪图形为POINT类型或LINESTRING类型时，函数返回GEOMETRYCOLLECTION EMPTY。
- 当被裁剪图形被包含在裁剪框内时，函数返回被裁剪图形。
- 当被裁剪图形与裁剪框完全不相交时，函数返回被裁剪图形类型的EMPTY。
- 当裁剪框为EMPTY、Nan、POINT类型、垂直线和水平线时，函数返回NULL。
- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POLYGON((2 0, 3 0, 3 2, 2 3, 0 2, 2 0))'), ST_GEOMFROMTEXT('MULTIPOINT(0 0,
0 2, 2 2, 2 0)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POLYGON ((0 2, 2 2, 2 0, 0 2))

--被裁剪图形为EMPTY时返回对应类型EMPTY
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POLYGON EMPTY'), ST_GEOMFROMTEXT('MULTIPOINT(0 0, 0 2, 2 2, 2 0)')), 0) res
FROM DUAL;

RES
------------------------------------------------------------------
POLYGON EMPTY

--被裁剪图形为POINT类型时返回GEOMETRYCOLLECTION EMPTY
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POINT(1 2)'), ST_GEOMFROMTEXT('MULTIPOINT(0 0, 0 2, 2 2, 2 0)')), 0) res
FROM DUAL;

RES
------------------------------------------------------------------
GEOMETRYCOLLECTION EMPTY

--被裁剪图形被包含在裁剪框内时返回被裁剪图形
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POLYGON((0 0,0 1, 1 1, 1 0, 0 0))'), ST_GEOMFROMTEXT('MULTIPOINT(0 0, 0 2, 2
2, 2 0)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POLYGON ((0 0, 0 1, 1 1, 1 0, 0 0))

--被裁剪图形与裁剪框完全不相交时返回被裁剪图形类型的EMPTY
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POLYGON((0 0,0 1, 1 1, 1 0, 0 0))'), ST_GEOMFROMTEXT('MULTIPOINT(6 6, 6 7, 7
7, 7 6)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POLYGON EMPTY
```

```
--裁剪框为EMPTY时返回NULL
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POLYGON((2 0, 3 0, 3 2, 2 3, 0 2, 2 0))'), ST_GEOMFROMTEXT('MULTIPOINT
EMPTY')), 0) res FROM DUAL;

RES
------------------------------------------------------------------

--参数存在NULL时返回NULL
SELECT ST_ASTEXT(ST_CLIPBYBOX2D(ST_GEOMFROMTEXT('POLYGON((2 0, 3 0, 3 2, 2 3, 0 2, 2 0))'), NULL), 0) res FROM DUAL;

RES
------------------------------------------------------------------
```

ST_CLOSESTPOINT

```
st_closestpoint::= ST_CLOSESTPOINT "(" geometry1 "," geometry2 ")"
```

ST_CLOSESTPOINT函数根据输入的geometry1和geometry2，返回geometry1上最接近geometry2的2D点，这个点不一定是输入几何图形的端点。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry1和geometry2的空间参考系标识号（SRID）必须相等，否则报错。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_ClosestPoint(ST_GeomFromText('POLYGON ((7.000 7.000, 6.000 6.000, 5.444 6.169, 7.000 7.000))
'),ST_GeomFromText('POLYGON((6.400 6.600, 6.231 7.156, 63.24 6.983,  6.400 6.600))')),6) res FROM dual;

RES
------------------------------------------------------------------
POINT (7.000000 7.000000)

SELECT ST_AsText(ST_ClosestPoint(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123
42.1546, -72.1244 42.3527, -72.1260 42.45))',4326)), 6) res FROM dual;

RES
------------------------------------------------------------------
POINT (-72.123500 42.352100)
```

ST_COLLECT

```
st_collect::= ST_COLLECT "(" geometry1 "," geometry2 ")"
```

```
st_collect::= ST_COLLECT "(" geomFiled ")"
```

ST_COLLECT函数的功能是对输入的一组geometry进行聚合，根据该组geometry是否具有相同或不同的类型，生成一个GEOMETRYCOLLECTION或MULTI*的geometry。

ST_COLLECT有两种形式，当入参为两个geometry参数时该函数作为普通函数，当入参为一组geometry字段时该函数作为聚合函数。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的一组geometry须具有相同的空间参考系标识号（SRID）和相同的空间维度，否则报错。

本函数遵守如下规则：

- 当输入的参数全部为NULL时，函数返回NULL。
- 当输入的geometry全部为相同原子类型的geometry时，会返回MULTI*的geometry，否则返回GEOMETRYCOLLECTION。
- 本函数支持3D坐标的计算。
- 该函数要求输入的geometry的维度相同，否则报错。
- 该函数要求输入的geometry的SRID相同，否则报错。
- geometry不能做为GROUP BY列。
- 该函数无法使用DISTINCT和ALL。
- 该函数无法指定OVER关键字去作为窗口函数使用。

示例（单机HEAP表）

```
-- 1.通过普通函数对两个GEOMETRY进行聚合
SELECT ST_ASTEXT(ST_COLLECT(ST_GEOMFROMTEXT('POINT(1 2)'), ST_GEOMFROMTEXT('POINT(3 4)')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
MULTIPOINT (1 2, 3 4)

-- 2.创建表
CREATE TABLE geom(id INT, col_geom GEOMETRY);
INSERT INTO geom VALUES(1, ST_GEOMFROMTEXT('POINT(1 2)'));
INSERT INTO geom VALUES(1, ST_GEOMFROMTEXT('LINESTRING(3 4, 5 2)'));

-- 3.通过聚合函数对表中的GEOMETRY进行聚合
SELECT ST_ASTEXT(ST_COLLECT(col_geom), 0) res FROM geom;

RES
----------------------------------------------------------------
GEOMETRYCOLLECTION (POINT (1 2), LINESTRING (3 4, 5 2))
```

ST_CONCAVEHULL

```
st_concavehull::= ST_CONCAVEHULL"(" geometry ", "ratio"[, "allow_holes"])"
```

ST_CONCAVEHULL函数用于计算一个Geometry对象的凹包。凹包指可以覆盖输入的几何对象所有顶点的一个几何对象，该几何对象一般为一个凹多边形。

本函数遵守如下规则：

- 输入为一个点或多个相同的点，返回结果仍为一个点。
- 输入为多个共线的点，返回结果将为一个线段。
- 输入为共线的两个或多个线段，返回结果将为一个线段。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

ratio

通用表达式，用于控制最短与最长边的长度比，从而影响凹包的凹度，类型为DOUBLE，有效范围为[0,1.0]。

- 凹包的构建过程采用Delaunay三角剖分的原理，通过不断地移除最长的外边来达到最优的结果。
- 当ratio为1时，返回结果为凸包；当ratio为0时，返回的结果为凹度最大的凹包；当ratio介于0和1之间时，ratio越大，凹度越大。

allow_holes

通用表达式，该参数类型为bool，缺省值为false，表示是否允许返回的多边形中包含洞。

示例

```
SELECT st_astext(st_concavehull(st_geomfromtext('point(0 0)'), 1, 0), 0) FROM dual;

ST_ASTEXT(ST_CONCAVE
--------------------------------------------------------------------
POINT (0 0)

SELECT st_astext(st_concavehull(st_geomfromtext('multipoint(0 0, 1 1)'), 1, 0), 0) FROM dual;

ST_ASTEXT(ST_CONCAVE
--------------------------------------------------------------------
LINESTRING (0 0, 1 1)

SELECT st_astext(st_concavehull(st_geomfromtext('multipoint(1 1, 1 1)'), 1, 0), 0) FROM dual;

ST_ASTEXT(ST_CONCAVE
--------------------------------------------------------------------
POINT (1 1)

SELECT st_astext(st_concavehull(st_geomfromtext('linestring(0 0, 1 0, 1 1)'), 1, 0), 0) FROM dual;

ST_ASTEXT(ST_CONCAVE
--------------------------------------------------------------------
POLYGON ((0 0, 1 1, 1 0, 0 0))

SELECT st_astext(st_concavehull(st_geomfromtext('geometrycollection(multipoint(0 0, 1 1), multipoint(2 2, 3 3))'), 1, 0), 0)
FROM dual;

ST_ASTEXT(ST_CONCAVE
--------------------------------------------------------------------
LINESTRING (0 0, 3 3)

SELECT st_astext(st_concavehull(st_geomfromtext('geometrycollection(linestring(0 0, 1 1), linestring(2 2, 3 3))'), 1, 0), 0)
FROM dual;
```

```
ST_ASTEXT(ST_CONCAVE
----------------------------------------------------------------
LINESTRING (0 0, 3 3)

SELECT st_astext(st_concavehull(st_geomfromtext('geometrycollection(polygon((0 0, 1 0, 1 1, 0 1, 0 0)), polygon((10 10, 20
10, 20 20, 10 20, 10 10)))'), 1, 0), 0) FROM dual;

ST_ASTEXT(ST_CONCAVE
----------------------------------------------------------------
POLYGON ((1 0, 0 0, 0 1, 10 20, 20 20, 20 10, 1 0))
```

ST_CONTAINS

```
st_contains::= ST_CONTAINS "(" geometry1 "," geometry2 ")"
```

ST_CONTAINS函数的功能是判断geometry1是否包含geometry2，包含时返回TRUE，否则返回FALSE。

geometry1包含geometry2指当且仅当geometry2中没有点位于geometry1的外部，且geometry2的内部至少有一个点位于geometry1的内部的情况。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- ST_CONTAINS是ST_WITHIN的逆。因此，ST_CONTAINS(A,B) = ST_WITHIN(B,A)。
- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Contains(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), ST_GeomFromText('POINT(3 3)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Contains(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_Contains(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_CONTAINSPROPERLY

```
st_containsproperly::= ST_CONTAINSPROPERLY "(" geometry1 "," geometry2 ")"
```

ST_CONTAINSPROPERLY函数的功能是判断geometry1是否完全包含geometry2，完全包含时返回TRUE，否则返回FALSE。与ST_CONTAINS不同的是，ST_CONTAINS(A,A) = TRUE，ST_CONTAINSPROPERLY(A,A) = FALSE。

geometry1完全包含geometry2指geometry2中不存在位于geometry1外部及其边界的点，即geometry2中的所有点都在geometry1内部。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 若输入的任意一个geometry为EMPTY，函数返回FALSE。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 结果精度为小数点后15位，超出部分的精度不作保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。
- 该函数支持使用RTREE索引。

示例

```
--POLYGON完全包含LINESTRING中的点
SELECT ST_ContainsProperly(ST_GeomFromText('POLYGON((0 0, 4 0, 4 4, 0 4, 0 0))'), ST_GeomFromText('LINESTRING(3 3, 2 3)'))
res FROM DUAL;

RES
--------------------
true

--LINESTRING有一个点与POLYGON相交
SELECT ST_ContainsProperly(ST_GeomFromText('POLYGON((0 0, 4 0, 4 4, 0 4, 0 0))'), ST_GeomFromText('LINESTRING(3 3, 4 4)'))
res FROM DUAL;

RES
--------------------
false

--输入存在NULL
SELECT ST_ContainsProperly(ST_GeomFromText('POLYGON((0 0, 4 0, 4 4, 0 4, 0 0))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_COVEREDBY

```
st_coveredby::= ST_COVEREDBY "(" geometry1 "," geometry2 ")"
```

ST_COVEREDBY函数的功能是判断geometry2是否覆盖geometry1，即如果geometry1中没有点位于geometry2之外，则返回TRUE，否则返回FALSE。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_CoveredBy(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), ST_GeomFromText('POINT(3 3)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_CoveredBy(ST_GeomFromText('POINT(3 3)'), ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))')) res FROM DUAL;

RES
--------------------
true

SELECT ST_CoveredBy(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_COVERS

```
st_covers::= ST_COVERS "(" geometry1 "," geometry2 ")"
```

ST_COVERS函数的功能是判断geometry1是否覆盖geometry2，即如果geometry2中没有点位于geometry1之外，则返回TRUE，否则返回FALSE。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Covers(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), ST_GeomFromText('POINT(3 3)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Covers(ST_GeomFromText('POINT(3 3)'), ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))')) res FROM DUAL;

RES
--------------------
false

SELECT ST_Covers(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_CROSSES

```
st_crosses::= ST_CROSSES "(" geometry1 "," geometry2 ")"
```

ST_CROSSES函数的功能是判断两个Geometry是否有部分（非全部）相同的内点。

如它们的交点“在空间上交叉”，即两个Geometry有部分（非全部）内部点共有，交叉则返回TRUE，否则返回FALSE。

交叉需满足如下两个条件：

- 两个Geometry内部的交集必须是非空的，且维度须小于两个输入的Geometry的最大维度。
- 两个Geometry的交集不能等于输入的Geometry中的任何一个。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

如两个Geometry的DE-9IM交叉矩阵匹配以下情况，则此关系成立：

- T*T******：对于Point/Line，Point/Area和Line/Area的场景。
- T*****T**：对于Line/Point，Area/Point和Area/Line的场景。
- 0********：对于Line/Line的场景。

对于Point/Point和Area/Area的场景始终返回FALSE。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Crosses(ST_GeomFromText('LINESTRING(3 5, 1 2, 3 5)'), ST_GeomFromText('LINESTRING(3 5, 4 6 ,3 5)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Crosses(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), ST_GeomFromText('LINESTRING(3 5, 4 6, 3 5)')) res
FROM DUAL;

RES
--------------------
false

SELECT ST_Crosses(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_DIFFERENCE

```
st_difference::= ST_DIFFERENCE "(" geometry1 "," geometry2 [ "," girdsize]")"
```

ST_DIFFERENCE函数返回包含geometry1但不包含geometry2几何图形，返回值为ST_GEOMETRY类型数据。

函数会将geometry对象投射到网格线上进行计算并返回结果。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

gridsize

gridsize的数据类型为DOUBLE，表示函数计算时使用的网格线大小，省略则默认为-1。

本函数遵守如下规则：

- 严格按照参数输入顺序决定输出内容，返回值一定与geometry1相关。
- 当geometry1完全包含在geometry2中时，函数返回geometry1类型的EMPTY。
- 当geometry1和geometry2均为EMPTY或任意一个为EMPTY时，函数均返回geometry1。
- 当输入的参数存在NULL时，函数返回NULL。
- 当输入的参数中包含Nan时，函数返回错误。
- 支持输入3D坐标，但函数会忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING(0 20, 0 80)'),ST_GEOMFROMTEXT('LINESTRING(0 30, 0 60)')), 0) res
FROM DUAL;

RES
----------------------------------------------------------------
MULTILINESTRING ((0 20, 0 30), (0 60, 0 80))

--geometry2完全包含geometry1时返回geometry1类型的EMPTY
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING(0 20, 0 30)'),ST_GEOMFROMTEXT('POLYGON((0 0, 0 60,60 60, 60 0,0
0))')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING EMPTY

--geometry1和geometry2均为空时返回geometry1
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING EMPTY'),ST_GEOMFROMTEXT('POLYGON EMPTY')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING EMPTY

--geometry1和geometry2其中任意一个为空时均返回geometry1
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING EMPTY'),ST_GEOMFROMTEXT('POLYGON((0 0, 0 60,60 60, 60 0,0 0))')),
0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING EMPTY

SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING(0 20, 0 30)'),ST_GEOMFROMTEXT('POLYGON EMPTY')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING (0 20, 0 30)
```

```
--gridsize为NULL时返回NULL
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING(0 20, 0 80)'),ST_GEOMFROMTEXT('LINESTRING(0 30, 0 60)'),NULL), 0)
res FROM DUAL;

RES
-----------------------------------------------------------------


--参数中存在NULL时函数返回NULL
SELECT ST_ASTEXT(ST_DIFFERENCE(NULL,ST_GEOMFROMTEXT('LINESTRING(0 30, 0 60)')), 0) res FROM DUAL;

RES
-----------------------------------------------------------------


--SRID不同时返回错误
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING(0 20, 0 80)',10),ST_GEOMFROMTEXT('LINESTRING(0 30, 0 60)',6)), 0)
res FROM DUAL;

YAS-07202 plugin execution error, Operation on mixed SRID geometries: 10 != 6

--参数中含有Nan时返回错误
SELECT ST_ASTEXT(ST_DIFFERENCE(ST_GEOMFROMTEXT('LINESTRING(0 Nan, 0 80)'),ST_GEOMFROMTEXT('LINESTRING(0 30, 0 60)')), 0) res
FROM DUAL;

YAS-07202 plugin execution error, LINESTRING has invalid coordinate
```

ST_DISJOINT

```
st_disjoint::= ST_DISJOINT "(" geometry1 "," geometry2 ")"
```

ST_DISJOINT函数的功能是判断两个Geometry是否不相交（无共同点），如不相交则返回TRUE，否则返回FALSE。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_DisJoint(ST_GeomFromText('LINESTRING(3 5, 1 2, 3 5)'), ST_GeomFromText('LINESTRING(3 5, 4 6 ,3 5)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_DisJoint(ST_GeomFromText('LINESTRING(3 5, 1 2, 3 5)'), ST_GeomFromText('LINESTRING(1 1, 2 2 ,3 3)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_DisJoint(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_DISTANCE

```
st_distance::= ST_DISTANCE "(" geometry1 "," geometry2 ")"
```

ST_DISTANCE函数根据输入的geometry1和geometry2，返回它们对应的距离数据。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry1和geometry2的空间参考系标识号（SRID）必须相等，否则报错。
- geometry1和geometry2的空间参考系标识号（SRID）必须在spatial_ref_sys系统表中定义或者为0，否则报错。
- geometry1和geometry2的空间参考系标识号（SRID）在spatial_ref_sys中对应的srs_type如果是GEOGRAPHY2D或者GEOGRAPHY3D，会切换到大地坐标算法计算，否则会使用投影坐标算法。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
SELECT ST_Distance(ST_GeomFromText('linestring(-72.1523 42.6343, -72.4524 42.2872)', 4326),
ST_GeomFromText('linestring(-72.4524 42.4526, -72.1235 42.3521)',4326)) res FROM dual;

RES
-------------------------
                        0

SELECT ST_Distance(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123 42.1546, -72.1244
42.3527, -72.1260 42.45))',4326)) res FROM dual;

RES
-------------------------
  7.3801729438882347E+001

SELECT ST_Distance(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123 42.1546, -72.12
42.35, -72.1260 42.45))',4326)) res FROM dual;

RES
-------------------------
                        0

SELECT ST_Distance(ST_GeomFromText('linestring(-72.1235 42.3521, -72.1523 42.6343)', 4326),
ST_GeomFromText('linestring(-72.4524 42.4526, -72.4524 42.2872)',4326)) res FROM dual;

RES
-------------------------
  2.6136292567874534E+004

SELECT ST_Distance(ST_GeomFromText('linestring(-72.1235 42.3521, -72.1523 42.6343)', 4490),
ST_GeomFromText('linestring(-72.4524 42.4526, -72.4524 42.2872)',4490)) res FROM dual;

RES
-------------------------
   2.613629256806742E+004

SELECT ST_Distance(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123 42.1546, -72.12
42.35, -72.1260 42.45))',4326)) res FROM dual;

RES
-------------------------
                        0

SELECT ST_Distance(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('linestring(-72.1260 42.45, -72.123 42.1546, -72.12
42.35, -72.1260 42.45)',4326)) res FROM dual;

RES
```

```
-------------------------
  1.2380207674721363E+002

SELECT ST_Distance(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('multipoint(-72.1260 42.45, -72.123 42.1546, -72.12
42.35, -72.1260 42.45)',4326)) res FROM dual;

RES
-------------------------
  3.7091260963845019E+002
```

ST_DUMP

```
st_dump::= ST_DUMP "(" geometry ")"
```

ST_DUMP函数用于返回输入的Geometry对象的所有原子类型（Point、LineString、Polygon）及访问路径。

本函数返回一组geometry_dump类型的集合，每个geometry_dump包含：

- geom属性：类型为ST_Geometry，表示输入geometry中的原子geometry。
- path属性：integer类型的集合，表示从输入的geometry到该原子geometry的访问路径。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 当输入的Geometry为原子类型时，返回的集合中只有一个元素，其中geom为原始输入的geometry，path为一个空的集合。
- 当输入的Geometry为集合类型时，返回的集合中可能包含多个元素，其中每个元素的path为一个非空的integer集合。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

geometry_dump_set

ST_Dump返回值为geometry_dump_set类型。geometry_dump_set类型是一个集合类型，其声明如下：

```
CREATE OR REPLACE TYPE GEOMETRY_DUMP_SET AS TABLE OF GEOMETRY_DUMP;
/
```

path是一个成员为integer类型的集合类型，其声明如下：

```
CREATE OR REPLACE TYPE GEOMETRY_PATH AS TABLE OF INTEGER;
/
```

geometry_dump_set的成员为geometry_dump类型，geometry_dump类型为一个对象类型，其声明如下：

```
CREATE OR REPLACE TYPE GEOMETRY_DUMP AS OBJECT (
path GEOMETRY_PATH,
geom ST_GEOMETRY
);
/
```

示例

```
SELECT st_astext(t2.geom, 0) FROM (SELECT st_dump(st_GeomFromText('point(0 0)')) DUMP FROM dual) t1, TABLE(t1.DUMP) t2;

ST_ASTEXT(T2.GEOM,0)
--------------------------------------------------------------------
POINT (0 0)

SELECT st_astext(t2.geom, 0) FROM (SELECT st_dump(st_GeomFromText('multipoint(0 0, 1 1)')) DUMP FROM dual) t1, TABLE(t1.DUMP)
t2;

ST_ASTEXT(T2.GEOM,0)
--------------------------------------------------------------------
POINT (0 0)
POINT (1 1)


SELECT st_astext(t2.geom, 0) FROM (SELECT st_dump(st_GeomFromText('geometrycollection(multipoint(0 0, 1 1),
geometrycollection(point(2 2), linestring(3 3, 4 4), multipoint(5 5, 6 6)))')) DUMP FROM dual) t1, TABLE(t1.DUMP) t2;
```

```
ST_ASTEXT(T2.GEOM,0)
----------------------------------------------------------------
POINT (0 0)
POINT (1 1)
POINT (2 2)
LINESTRING (3 3, 4 4)
POINT (5 5)
POINT (6 6)

--创建一个打印path的函数
CREATE OR replace FUNCTION print_path(path MDSYS.GEOMETRY_PATH) RETURN VARCHAR IS
    pathStr VARCHAR(32000);
    i int;
BEGIN
    IF path IS null THEN
        RETURN null;
END IF;

    pathStr := '{';
FOR i IN 1..path.count LOOP
        IF i > 1 THEN
            pathStr := pathStr || ',';
END IF;
        pathStr := pathStr || path(i);
END LOOP;
    pathStr := pathStr || '}';
RETURN pathStr;
END;
/

SELECT print_path(t2.path) path, st_astext(t2.geom, 0) geom FROM (SELECT st_dump(st_GeomFromText('point(0 0)')) DUMP FROM
dual) t1, TABLE(t1.DUMP) t2;

PATH                                                             GEOM
---------------------------------------------------------------- ----------------------------------------------------------------
----
{}                                                               POINT (0 0)

SELECT print_path(t2.path) path, st_astext(t2.geom, 0) geom FROM (SELECT st_dump(st_GeomFromText('multipoint(0 0, 1 1)'))
DUMP FROM dual) t1, TABLE(t1.DUMP) t2;

PATH                                                             GEOM
---------------------------------------------------------------- ----------------------------------------------------------------
----
{1}                                                              POINT (0 0)
{2}                                                              POINT (1 1)

SELECT print_path(t2.path) path, st_astext(t2.geom, 0) geom FROM (SELECT
st_dump(st_GeomFromText('geometrycollection(multipoint(0 0, 1 1), geometrycollection(point(2 2), linestring(3 3, 4 4),
multipoint(5 5, 6 6)))')) DUMP FROM dual) t1, TABLE(t1.DUMP) t2;

PATH                                                             GEOM
---------------------------------------------------------------- ----------------------------------------------------------------
----
{1,1}                                                            POINT (0 0)
{1,2}                                                            POINT (1 1)
{2,1}                                                            POINT (2 2)
{2,2}                                                            LINESTRING (3 3, 4 4)
{2,3,1}                                                          POINT (5 5)
{2,3,2}                                                          POINT (6 6)
```

ST_DWITHIN

```
st_dwithin::= ST_DWITHIN "(" geometry1 "," geometry2 "," distance ")"
```

ST_DWITHIN函数的功能是判断geometry1与geometry2是否在给定的距离distance内，如果在distance内则返回TRUE，否则返回FALSE。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

distance

distance的数据类型是DOUBLE，表示指定的距离。

- 支持能够隐式转换成DOUBLE的类型。
- 如果输入的distance小于0，则报错。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 若输入的任意一个geometry为EMPTY，函数返回FALSE。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- geometry的空间参考系标识号（SRID）必须在spatial_ref_sys系统表中定义或者为0，否则报错。
- geometry的空间参考系标识号（SRID）在spatial_ref_sys中对应的srs_type如果是GEOGRAPHY2D或者GEOGRAPHY3D，会切换到大地坐标算法计算，否则会使用投影坐标算法。
- 该函数支持RTREE索引，但不建议在经纬度坐标系下使用该函数的RTREE索引。

示例

```
--返回两个geometry之间的距离
SELECT ST_Distance(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('POINT(3 4)')) res FROM DUAL;

RES
-----------
5.0E+000

--两个geometry之间的距离在指定的distinct内
SELECT ST_DWithin(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('POINT(3 4)'), 5) res FROM DUAL;

RES
--------------------
true

--两个geometry之间的距离不在指定的distinct内
SELECT ST_DWithin(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('POINT(3 4)'), 4) res FROM DUAL;

RES
--------------------
false
```

ST_ENVELOPE

```
st_envelope::= ST_ENVELOPE "(" geometry ")"
```

ST_ENVELOPE函数用于计算输入的geometry的最小外包矩形，返回值为ST_GEOMETRY类型数据。

该函数根据输入geometry的不同，返回值的数据类型不同：

- 输入值为POINT类型时，返回值为POINT类型。
- 输入值为其他类型时，返回值为POLYGON类型。输入值为垂直线、水平线时，返回值为线形状的POLYGON类型数据。
- 输入值为任意类型EMPTY时，返回值为POINT EMPTY。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，但不能为三维的ST_GEOMETRY类型数据。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_ENVELOPE函数根据给定的GEOMETRY返回一个ST_GEOMETRY数据
--输入值为POINT类型
SELECT ST_ASTEXT(ST_ENVELOPE(ST_GEOMFROMTEXT('POINT(4 9)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT (4 9)

--输入值为MULTIPOINT类型
SELECT ST_ASTEXT(ST_ENVELOPE(ST_GEOMFROMTEXT('MULTIPOINT(4 9,3 12)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POLYGON ((3 9, 4 9, 4 12, 3 12, 3 9))

--输入值为垂直线
SELECT ST_ASTEXT(ST_ENVELOPE(ST_GEOMFROMTEXT('LINESTRING(1 2,1 3,1 6)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POLYGON ((1 2, 1 2, 1 6, 1 6, 1 2))

--输入值为LINESTRING EMPTY
SELECT ST_ASTEXT(ST_ENVELOPE(ST_GEOMFROMTEXT('LINESTRING EMPTY')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT EMPTY

--输入值为GEOMTERY COLLECTION类型
SELECT ST_ASTEXT(ST_ENVELOPE(ST_GeomFromText('GEOMETRYCOLLECTION (LINESTRING(55 75,125 150), POINT(20 80))')), 0) res FROM
DUAL;

RES
------------------------------------------------------------------
POLYGON ((20 75, 125 75, 125 150, 20 150, 20 75))

--参数包含NULL
SELECT ST_ASTEXT(ST_ENVELOPE(ST_GeomFromText(null, 4316))) res FROM DUAL;

RES
------------------------------------------------------------------
```

ST_EQUALS

```
st_equals::= ST_EQUALS "(" geometry1 "," geometry2 ")"
```

ST_EQUALS函数的功能是判断两个Geometry是否包含同一组点，即给定的两个Geometry“空间相等”，相等则返回TRUE，否则返回FALSE。

空间相等指`ST_WITHIN(A,B) = TRUE`和`ST_WITHIN(B,A) = TRUE`，也意味着点的顺序可以不同，但表示相同的几何结构。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Equals(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(1 1, 2 2, 3 3)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Equals(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 4 6 ,3 5)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_Equals(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_EXTENT

```
st_extent::= ST_EXTENT "(" geomFiled ")"
```

ST_EXTENT函数的功能是返回一个包围一组geometry的二维边界框，是一个聚合函数，返回类型是BOX2D。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

本函数遵守如下规则：

- 当输入的一组geometry全部为NULL或全部为EMPTY时，该函数返回NULL。
- 本函数只计算2D结果，如果坐标中有Z轴将直接忽略Z坐标进行计算。
- geometry不能做为GROUP BY列。
- 该函数无法使用DISTINCT和ALL。
- 该函数无法指定OVER关键字去作为窗口函数使用。

示例（单机HEAP表）

```
-- 1.创建表
DROP TABLE IF EXISTS geom;
CREATE TABLE geom(id INT, col_geom GEOMETRY);
INSERT INTO geom VALUES(1, ST_GEOMFROMTEXT('POINT(1 2)'));
INSERT INTO geom VALUES(1, ST_GEOMFROMTEXT('LINESTRING(3 4, 5 2)'));

-- 2.创建BOX2D的输出函数
CREATE OR REPLACE FUNCTION BOX2D_OUT(box BOX2D) RETURN VARCHAR AS
    res VARCHAR(100);
BEGIN
    res := 'BOX(' || box.xmin || ' ' || box.xmax || ',' || box.ymin || ' ' || box.ymax || ')';
RETURN res;
END;
/

-- 3.通过聚合函数对表中的GEOMETRY进行聚合
SELECT BOX2D_OUT(ST_EXTENT(col_geom)) res FROM geom;

RES
-----------------------------------------------------------------
BOX(1.0E+000 5.0E+000,2.0E+000 4.0E+000)
```

ST_GEOMCOLLFROMTEXT

```
st_geomcollfromtext::= ST_GEOMCOLLFROMTEXT "(" wkt [ "," srid ] ")"
```

ST_GEOMCOLLFROMTEXT函数根据给定的wkt（Well-Known Text）和srid返回一个ST_GEOMETRY类型数据，如果传入的WKT不是GEOMETRYCOLLECTION，则返回NULL。

入参wkt和srid的规格与ST_GEOMFROMTEXT函数相同。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_GeomCollFromText('GEOMETRYCOLLECTION EMPTY')) res FROM DUAL;

RES
------------------------------------------------------------------
GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2), LINESTRING(1 3, 4 1))'), 0) res FROM DUAL;

RES
------------------------------------------------------------------
GEOMETRYCOLLECTION (POINT (1 2), LINESTRING (1 3, 4 1))

SELECT ST_AsText(ST_GeomCollFromText('LINESTRING(1 3, 4 1)'), 0) res FROM DUAL;

RES
------------------------------------------------------------------
```

ST_GEOMETRICMEDIAN

```
st_geometricmedian::= ST_GEOMETRICMEDIAN "(" geometry [ "," tolerance ] [ "," maxIter ] [ "," failIfNotConverged ] ")"
```

ST_GEOMETRICMEDIAN函数的功能是使用Weiszfeld算法计算MULTIPOINT数据的几何中位数。最终会返回一个POINT数据，该POINT到MULTIPOINT的所有POINT的距离之和是最小的。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- 当该参数为NULL时，函数返回NULL。
- 输出的POINT数据维度由输入的MULTIPOINT数据的最大维度决定，且不会超过三维。
- 输入geometry的是POINT类型时，则会返回该POINT，输入的是MULTIPOINT EMPTY时则返回POINT EMPTY，输入其他类型则报错。

tolerance

tolerance表示容差，其值为DOUBLE类型。Weiszfeld算法会不断迭代进行计算，直到连续迭代之间的距离变化小于给定的容差（tolerance），该参数遵循如下规则：

- 该参数可以省略，如省略或输入NULL，则会根据输入的geometry的外包框计算一个tolerance。
- 支持能够隐式转换成DOUBLE的数据类型。
- tolerance仅支持为非负数，否则报错。

maxIter

maxIter表示迭代最大次数，其值为INT类型，遵循如下规则：

- 该参数可以省略，默认值为10000。
- 支持能够隐式转换成INT的数据类型。
- maxIter只能是正整数，输入的是负数或者是Null则报错。
- 当failIfNotConverged参数为TRUE时，如进行了maxIter次迭代仍未满足第二个条件，则会报错。
- 当failIfNotConverged参数为FALSE时，则即使进行了maxIter次迭代仍未满足第二个条件，也不会报错。

failIfNotConverged

failIfNotConverged表示迭代计算次数超过maxIter是否报错，其值为BOOLEAN类型，遵循如下规则：

- 该参数可以省略，默认值为FALSE，输入NULL同样视作FALSE。
- 支持能够隐式转换成BOOLEAN的数据类型。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_AsText(ST_GeometricMedian(ST_GeomFromText('MULTIPOINT(1 1, 2 2, 3 3)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT (2 2)

SELECT ST_AsText(ST_GeometricMedian(ST_GeomFromText('MULTIPOINT(1 1, 2 2, 3 3 3)')), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT Z (2 2 0)

SELECT ST_AsText(ST_GeometricMedian(ST_GeomFromText('MULTIPOINT(1 1, 2 2, 3 3 3)'), NULL, 0, TRUE), 0) res FROM DUAL;

YAS-07202 plugin execution error, Median failed to converge within 1e-08 after 0 iterations.
```

ST_GEOMETRYTYPE

```
st_geometrytype::= ST_GEOMETRYTYPE "(" geometry ")"
```

ST_GEOMETRYTYPE函数用于打印输入的geometry的类型。

该函数会根据输入的geometry返回该geometry的数据类型，返回值为VARCHAR类型。

与GEOMETRYTYPE函数相比，ST_GEOMETRYTYPE函数的返回值带ST_前缀且非全大写。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_GEOMETRYTYPE(ST_GeomFromText('POINT(-58.2687 29.149)',1356)) res FROM DUAL;

RES
----------------------------------------------------------------
ST_Point

SELECT ST_GEOMETRYTYPE(ST_GeomFromText('LINESTRING(1 6,3 2,9 7)',1356)) res FROM DUAL;

RES
----------------------------------------------------------------
ST_LineString

--参数包含NULL
SELECT ST_GEOMETRYTYPE(ST_GeomFromText(null,1356)) res FROM DUAL;

RES
----------------------------------------------------------------
```

ST_GEOMFROMEWKB

```
st_geomfromewkb::= ST_GEOMFROMEWKB "(" ewkb ")"
```

ST_GEOMFROMEWKB函数根据给定的ewkb（Extended Well-Known Binary）返回一个ST_GEOMETRY类型的数据。

ewkb

ewkb的数据类型是BLOB，遵循如下规则：

- 支持能够隐式转换成BLOB的类型。
- ewkb需要是一个合法的Extended Well-Known Binary，否则报错。
- 输入的ewkb如果前面的字符已经构成了一个有效的ST_GEOMETRY类型数据，并且后面还有其他有效字符，则只会生成前面的有效的ST_GEOMETRY类型数据。

当输入的ewkb为NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_GeomFromEwkb('0101000020E6100000000000000000F03F0000000000000040')) res FROM DUAL;

RES
------------------------------------------------------------------
POINT (1.000000000000000 2.000000000000000)

SELECT ST_Srid(ST_GeomFromEwkb('0101000020E6100000000000000000F03F0000000000000040')) res FROM DUAL;

         RES
------------
        4326
```

ST_GEOMFROMGEOJSON

```
st_geomfromgeojson::= ST_GEOMFROMGEOJSON "(" geojson ")"
```

ST_GEOMFROMGEOJSON函数根据输入的geojson，返回该geojson对应的ST_GEOMETRY类型数据。只支持二维几何图形，三维几何图形会报错。

geojson

geojson的数据类型是CLOB，必须为有效的GeoJSON数据，遵循如下规则：

- 支持能够隐式转换成CLOB的类型。
- 当输入的参数是NULL时，函数返回NULL。
- 输入的数据可以包含外包框、坐标参考系，对此不检查正确性且不参与结果计算。

示例（单机HEAP表）

```
-- empty
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[]}')) res FROM DUAL;

RES
------------------------------------------------------------------
POINT EMPTY

-- point
SELECT ST_AsText(ST_GeomFromGeoJSON(ST_AsGeoJSON(ST_GeomFromText('POINT(1 1)')))) res FROM DUAL;

RES
------------------------------------------------------------------
POINT (1.000000000000000 1.000000000000000)

-- geometry collection
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"GeometryCollection","geometries":[{"type":"Point","coordinates":[100.0,0.0]},
{"type":"LineString","coordinates":[[101.0,0.0],[102.0,1.0]]}],"crs":{"type":"name","properties":
{"name":"urn:ogc:def:crs:EPSG::3395"}}}')) res FROM DUAL;

RES
------------------------------------------------------------------
GEOMETRYCOLLECTION (POINT (100.000000000000000 0.000000000000000), LINESTRING (101.000000000000000 0.000000000000000,
102.000000000000000 1.000000000000000))
```

ST_GEOMFROMTEXT

```
st_geomfromtext::= ST_GEOMFROMTEXT "(" wkt [ "," srid ] ")"
```

ST_GEOMFROMTEXT（同名函数ST_GEOMETRYFROMTEXT）函数根据给定的wkt（Well-Known Text）和srid返回一个ST_GEOMETRY类型数据。

wkt

wkt的数据类型是CLOB，遵循如下规则：

- wkt需要是一个有效的Well-Known Text，否则报错。
- 支持的坐标的最高维度是三维，如果输入是四维的坐标，或带有'M'字样的坐标，则会忽略第四个坐标轴，生成三维坐标。
- 如果输入的坐标中既有二维的点，也有三维的点，则会把二维的点提升到三维，Z轴的数值补零。
- 支持输入'inf'、'nan'，如果一个坐标全是'nan'，则会存为EMPTY。
- 支持输入空的ST_GEOMETRY数据（如POINT EMPTY）。
- 点坐标不能跟EMPTY同时使用，如MULTIPOINT(1 1，EMPTY)就会报错。
- 输入的wkt如果前面的字符已经构成了一个有效的ST_GEOMETRY数据，并且后面还有其他字符，则会生成前面的有效ST_GEOMETRY数据。
- 支持能够隐式转换成CLOB的数据类型。

srid

srid的数据类型是INT，表示该ST_GEOMETRY类型数据的空间参考系，遵循如下规则：

- 支持能够隐式转换成INT的类型，如果输入的是小数则进行四舍五入转换。
- 该参数可以省略，省略时默认值是0（srid=0表示没有定义空间参考系）。
- 如果输入的是负数，则会按照0进行处理。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
--生成Point
SELECT ST_AsText(ST_GeomFromText('POINT(1 2)', 4326), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1 2)

--维度混合
SELECT ST_AsText(ST_GeomFromText('MULTIPOINT(1 2 3, 1 1)', 4326), 0) res FROM DUAL;

RES
----------------------------------------------------------------
MULTIPOINT Z (1 2 3, 1 1 0)

--支持inf和nan的输入
SELECT ST_AsText(St_GeomFromText('MULTIPOINT(inf inf,nan nan)')) res FROM DUAL;

RES
----------------------------------------------------------------
MULTIPOINT (inf inf, EMPTY)

--Srid
SELECT ST_Srid(st_GeomFromText('POINT(1 2)', 4326)) res FROM DUAL;

         RES
------------
        4326

SELECT ST_Srid(st_GeomFromText('POINT(1 2)')) res FROM DUAL;

         RES
------------
           0
```

```
SELECT ST_Srid(st_GeomFromText('POINT(1 2)', -12)) res FROM DUAL;

         RES
------------
           0

--同名函数ST_GeometryFromText
SELECT ST_AsText(ST_GeometryFromText('POINT(1 2)', 4326), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1 2)
```

ST_GEOMFROMWKB

```
st_geomfromwkb::= ST_GEOMFROMWKB "(" wkb [ "," srid ] ")"
```

ST_GEOMFROMWKB函数根据给定的wkb（Well-Known Binary）和srid返回一个ST_GEOMETRY类型数据。

wkb

wkb的数据类型是BLOB，遵循如下规则：

- 支持能够隐式转换成BLOB的类型。
- wkb需要是一个有效的Well-Known Binary，否则报错。
- 输入的wkb如果前面的字符已经构成了一个有效的ST_GEOMETRY数据，并且后面还有其他有效字符，则只会生成前面的有效ST_GEOMETRY数据。

srid

srid的数据类型是INT，表示该ST_GEOMETRY数据的空间参考系，遵循如下描述：

- 支持能够隐式转换成INT的类型，如果输入的是小数则进行四舍五入转换。
- 该参数可以省略，省略时默认值是0（srid=0表示没有定义空间参考系）。
- 如果输入的是负数，则输出原有SRID。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
--生成Point
SELECT ST_AsText(ST_GeomFromWkb('0101000000000000000000F03F0000000000000040')) res FROM DUAL;

RES
------------------------------------------------------------
POINT (1.000000000000000 2.000000000000000)

--Srid
SELECT ST_Srid(ST_GeomFromWkb('0101000000000000000000F03F0000000000000040', 4322)) res FROM DUAL;

         RES
------------
        4322

SELECT ST_Srid(ST_GeomFromWkb('0101000000000000000000F03F0000000000000040')) res FROM DUAL;

         RES
------------
           0

SELECT ST_Srid(ST_GeomFromWkb('0101000000000000000000F03F0000000000000040', -10)) res FROM DUAL;

         RES
------------
           0
```

ST_INTERSECTION

```
st_intersection::= ST_INTERSECTION "(" geometry1 "," geometry2 [ ","girdsize]")"
```

ST_INTERSECTION函数返回两个geometry对象的交集，返回值为ST_GEOMETRY类型数据。

函数会将geometry对象投射到网格线上进行计算并返回结果。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

gridsize

gridsize的数据类型为DOUBLE，表示函数计算时使用的网格线大小，省略则默认为-1。

本函数遵守如下规则：

- 当geometry1完全包含在geometry2中时，函数返回geometry1。
- 当geometry1和geometry2均为EMPTY时，函数返回geometry2。
- 当geometry1和geometry2其中一个为EMPTY时，函数返回该空对象类型的EMPTY。
- 当输入的参数存在NULL时，函数返回NULL。
- 当输入的参数中包含Nan时，函数返回错误。
- 支持输入3D坐标，但函数会忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POLYGON((2 0, 0 2, 0 0, 2 0))'), ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0, 2
2))')), 0) res FROM DUAL;

RES
-------------------------------------------------------------------
POLYGON ((2 0, 0 0, 1 1, 2 0))

--geometry1完全包含在geometry2中时返回geometry1
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POINT(1 1)'), ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0, 2 2))')), 0) res
FROM DUAL;

RES
-------------------------------------------------------------------
POINT (1 1)

--geometry1和geometry2均为EMPTY时返回geometry2
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POINT EMPTY'), ST_GEOMFROMTEXT('POLYGON EMPTY')), 0) res FROM DUAL;

RES
-------------------------------------------------------------------
POLYGON EMPTY

--geometry1和geometry2其中一个为EMPTY时返回该空对象类型的EMPTY
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POINT EMPTY'), ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0, 2 2))')), 0) res
FROM DUAL;

RES
-------------------------------------------------------------------
POINT EMPTY

--参数中存在NULL时返回NULL
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POLYGON((2 0, 0 2, 0 0, 2 0))'), ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0, 2
2))'),NULL), 0) res FROM DUAL;

RES
-------------------------------------------------------------------
```

```
SELECT ST_ASTEXT(ST_INTERSECTION(NULL, ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0, 2 2))')), 0) res FROM DUAL;

RES
-------------------------------------------------------------------


--SRID不同时返回错误
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POLYGON((2 0, 0 2, 0 0, 2 0))',9), ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0,
2 2))',10)), 0) res FROM DUAL;

YAS-07202 plugin execution error, Operation on mixed SRID geometries: 9 != 10

--参数中含有Nan时返回错误
SELECT ST_ASTEXT(ST_INTERSECTION(ST_GEOMFROMTEXT('POLYGON((2 0, 0 Nan, 0 0, 2 0))'), ST_GEOMFROMTEXT('POLYGON((2 2, 2 0, 0 0,
2 2))')), 0) res FROM DUAL;

YAS-07202 plugin execution error, POLYGON has invalid coordinate
```

ST_INTERSECTS

```
st_intersects::= ST_INTERSECTS "(" geometry1 "," geometry2 ")"
```

ST_INTERSECTS函数的功能是判断两个Geometry是否相交（即至少有一个共同点），如相交则返回TRUE，否则返回FALSE。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

如果两个Geometries的DE-9IM交叉矩阵匹配以下情况，则此关系成立：

- T********
- *T*******
- ***T*****
- ****T****

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Intersects(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 2 2, 3 5)')) res FROM
DUAL;

RES
--------------------
true

SELECT ST_Intersects(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 4 6, 3 5)')) res FROM
DUAL;

RES
--------------------
false

SELECT ST_Intersects(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_ISCLOSED

```
st_isclosed::= ST_IsClosed "(" geometry ")"
```

ST_ISCLOSED函数用于判断线的首、尾两个点是否重合，通常用于判断LineString的首尾是否闭合。

本函数遵守如下规则：

- 当输入对象为NULL，返回NULL。
- 当输入对象为空几何对象（如LineString Empty），返回FALSE。
- 其他情况下对于不同的几何对象类型，其行为如下：

对象类型	ST_IsClosed函数行为
Point	返回true
LineString	判断LineString的首尾是否重合
Polygon	判断Polygon中的每个环是否首尾闭合，所有环均闭合返回true， 否则返回false
Multi及Collection类型	判断集合中每个成员是否闭合，所有成员均闭合返回true，否则返回false

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

示例

```
SELECT st_isclosed(st_geomfromText('point(0 0)')) isclosed FROM dual;

ISCLOSED
--------------------
true

SELECT st_isclosed(st_geomfromText('linestring(0 0, 1 1, 1 0)')) isclosed FROM dual;

ISCLOSED
--------------------
false

SELECT st_isclosed(st_geomfromText('linestring(0 0, 1 1, 1 0, 0 0)')) isclosed FROM dual;

ISCLOSED
--------------------
true

SELECT st_isclosed(st_geomfromText('polygon((0 0, 1 1, 1 0, 0 0))')) isclosed FROM dual;

ISCLOSED
--------------------
true

SELECT st_isclosed(st_geomfromText('multipoint(0 0, 1 1)')) isclosed FROM dual;

ISCLOSED
--------------------
true

SELECT st_isclosed(st_geomfromText('multilinestring((0 0, 1 1, 1 0, 0 0), (0 0, 1 1, 1 0, 0 0))')) isclosed FROM dual;

ISCLOSED
--------------------
true
```

```
SELECT st_isclosed(st_geomfromText('multilinestring((0 0, 1 1, 1 0, 0 0), (0 0, 1 1, 1 0, 0 1))')) isclosed FROM dual;

ISCLOSED
--------------------
false
```

ST_ISEMPTY

```
st_isempty::= ST_ISEMPTY "(" geometry ")"
```

ST_ISEMPTY函数根据输入的geometry，返回该geometry是否为空，即EMPTY，如该geometry均为EMPTY，则返回TRUE，否则返回FALSE。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY')) res FROM DUAL;

RES
--------------------
true

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))')) res FROM DUAL;

RES
--------------------
false
```

ST_ISSIMPLE

```
st_issimple::= ST_ISSIMPLE "(" geometry ")"
```

ST_ISSIMPLE函数根据输入的geometry，返回该geometry是否简单，如该geometry符合简单定义，则返回TRUE，否则返回FALSE。

ST_GEOMETRY类型数据的简单定义遵循OGC SFS规范，按照如下规则判断该数据是否简单：

- POINT：均为简单。
- MULTIPOINT：所有点唯一时，该数据为简单；否则不为简单。
- LINESTRING：除端点之外的所有点唯一时，该数据为简单；否则不为简单。
- MULTILINESTRING：其中所有元素均为简单，且任意两个元素之间的唯一交集出现在两个元素的边界上时，该数据为简单；否则不为简单。
- POLYGON：由简单且闭环的LINESTRING组成，且该POLYGON数据为有效时，该数据为简单；否则不为简单。有效定义可参考ST_ISVALID函数描述。
- MULTIPOLYGON：其中所有元素均为简单时，该数据为简单；否则不为简单。
- GEOMETRYCOLLECTION：其中所有元素均为简单时，该数据为简单；否则不为简单。
- 数据中均为EMPTY时，该数据为简单。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))')) res FROM DUAL;

RES
--------------------
false

SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 3, 5 6, 1 2))')) res FROM DUAL;

RES
--------------------
true
```

ST_ISVALID

```
st_isvalid::= ST_ISVALID "(" geometry ")"
```

ST_ISVALID函数根据输入的geometry，返回该geometry是否有效，如该geometry是有效的，则返回TRUE，否则返回FALSE。

ST_GEOMETRY类型数据是否有效须遵循OGC SFS规范，按照如下规则判断该数据是否有效：

- POINT：均为有效。
- MULTIPOINT：均为有效。
- LINESTRING：均为有效。
- MULTILINESTRING：均为有效。
- POLYGON：有效的POLYGON类型数据须遵循如下限制，否则视为无效：
  - 各环必须闭合。
  - 内环必须处于外环内部。
  - 各环不能自相交。
  - 环之间不能接触，除非在一个点相切。
- MULTIPOLYGON：其中所有元素均为有效，且元素内部不能相交时，该数据为有效；否则不为有效。
- GEOMETRYCOLLECTION：其中所有元素均为有效时，该数据为有效；否则不为有效。
- 数据中均为EMPTY时，该数据为有效。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) res FROM DUAL;

RES
--------------------
false
```

ST_LENGTH

```
st_length::= ST_LENGTH "(" geometry ")"
```

ST_LENGTH函数根据输入的geometry，返回对应的长度数据。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry是LINESTRING、MULTILINESTRING或GEOMETRY COLLECTION类型会返回对应的长度，其它的geometry类型会返回0。
- geometry的空间参考系标识号（SRID）必须在spatial_ref_sys系统表中定义或者为0，否则报错。
- geometry的空间参考系标识号（SRID）在spatial_ref_sys中对应的srs_type如果是GEOGRAPHY2D或者GEOGRAPHY3D，会切换到大地坐标算法计算，否则会使用投影坐标算法。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
SELECT ST_Length(ST_GeomFromText('linestring(-72.1235 42.3521, -72.1523 42.6343)', 4326)) res FROM dual;

        RES
-----------
 3.144E+004

SELECT ST_Length(ST_GeomFromText('linestring(5000 6789, 12345 5789)', 3385)) res FROM dual;

        RES
-----------
 7.413E+003
```

ST_LINEFROMTEXT

```
st_linefromtext::= ST_LINEFROMTEXT "(" wkt [ "," srid ] ")"
```

ST_LINEFROMTEXT函数根据给定的wkt（Well-Known Text）和srid返回一个ST_GEOMETRY类型数据，如果传入的WKT不是LINESTRING，则返回NULL。

入参wkt和srid的规格与ST_GEOMFROMTEXT函数相同。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_LineFromText('LINESTRING EMPTY')) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING EMPTY

SELECT ST_AsText(ST_LineFromText('LINESTRING(1 1, 3 1)'), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING (1 1, 3 1)

SELECT ST_AsText(ST_LineFromText('POINT(3 1)'), 0) res FROM DUAL;

RES
----------------------------------------------------------------
```

ST_LINEMERGE

```
st_linemerge::= ST_LINEMERGE "(" geometry "[,"directed"])"
```

ST_LINEMERGE函数用于将输入的MultiLineString中的线组合成一个LineString或MultiLineString。

本函数遵守如下规则：

- 输入参数为Point、MultiPoint、Polygon或MultiPolygon时，函数返回GeometryCollection Empty。
- 输入参数为NULL时，函数返回NULL。
- 输入参数为空的LineString或MultiLineString时，函数返回LineString Empty或MultiLineString Empty。
- 输入参数为点、多点时，返回NULL。
- 仅当输入的MultiLineString中两条线相交且交点的度为2（即端点处相交）时，两条线才会合并。
- 函数返回的几何对象，其SRID与输入的几何对象一致。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

directed

通用表达式，其类型为bool，缺省值为false。当directed为true时，输入的LineString（或MultiLineString中包含的LineString）被视为有向线，否则视为无向。方向相反的有向线无法组合。

示例

```
--directed = true时，两个方向相反的线，端点处相交，但不会合并
SELECT st_astext(st_lineMerge(st_geomfromtext('multilinestring((0 0, 1 1), (2 2, 1 1))'), true), 0) st_linemerge FROM dual;

ST_LINEMERGE
------------------------------------------------------------
MULTILINESTRING ((0 0, 1 1), (2 2, 1 1))

--directed = false时，两个方向相反的线，端点处相交，会合并
SELECT st_astext(st_lineMerge(st_geomfromtext('multilinestring((0 0, 1 1), (2 2, 1 1))'), false), 0) st_linemerge FROM dual;

ST_LINEMERGE
------------------------------------------------------------
LINESTRING (0 0, 1 1, 2 2)

--两条线的交点（2 0），是第一条线内部的点，相交之后（2 0）的度为3，因此不会合并
SELECT st_astext(st_lineMerge(st_geomfromtext('multilinestring((0 0, 2 0, 4 0), (2 0, 2 1))')), 0) st_linemerge FROM dual;

ST_LINEMERGE
------------------------------------------------------------
MULTILINESTRING ((0 0, 2 0, 4 0), (2 0, 2 1))

--三条线在(1 0)处相交，导致该点的度为3，因此该点不能作为交点，三条线不会合并
SELECT st_astext(st_lineMerge(st_geomfromtext('multilinestring((0 0, 1 0), (1 0, 2 0), (1 0, 1 1))')), 0) st_linemerge FROM
dual;

ST_LINEMERGE
------------------------------------------------------------
MULTILINESTRING ((0 0, 1 0), (1 0, 2 0), (1 0, 1 1))
```

ST_LONGESTLINE

```
st_longestline::= ST_LONGESTLINE "(" geometry1 "," geometry2 ")"
```

ST_LONGESTLINE函数根据输入的geometry1和geometry2，返回geometry1与geometry2之间的二维最长LineString。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry1和geometry2的空间参考系标识号（SRID）必须相等，否则报错。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 返回的LineString从geometry1开始，以geometry2结束。
- 如果geometry1和geometry2相交，则结果是一条以交点为起点和终点的直线。
- 最长的LineString总是出现在两个顶点之间。如果找到多个，则返回第一个最长的LineString。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_LONGESTLINE(ST_GeomFromText('POLYGON ((7.000 7.000, 6.000 6.000, 5.444 6.169, 7.000 7.000))
'),ST_GeomFromText('POLYGON((6.400 6.600, 6.231 7.156, 63.24 6.983,  6.400 6.600))')),6) res FROM dual;

RES
----------------------------------------------------------------
LINESTRING (5.444000 6.169000, 63.240000 6.983000)

SELECT ST_AsText(ST_LONGESTLINE(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123 42.1546,
-72.1244 42.3527, -72.1260 42.45))',4326)), 6) res FROM dual;

RES
----------------------------------------------------------------
LINESTRING (-72.123500 42.352100, -72.123000 42.154600)
```

ST_MAKEENVELOPE

```
st_makeenvelope::= ST_MakeEnvelope "("xmin", "ymin", "xmax", "ymax"["srid"])"
```

ST_MAKEENVELOPE函数用于返回一个根据输入的X、Y的最小、最大值构建的外包矩形。其结果为ST_Geometry类型的Polygon。

本函数遵守如下规则：

- 当xmin、ymin、xmax、ymax中有NULL时，返回结果为NULL。
- 当srid为NULL时，返回结果为NULL。

xmin

通用表达式，类型为double。

ymin

通用表达式，类型为double。

xmax

通用表达式，类型为double。

ymax

通用表达式，类型为double。

srid

通用表达式，类型为integer，缺省值为0。

示例

```
SELECT st_astext(st_MakeEnvelope(1,2,3,4,4326), 0) geom FROM dual;

GEOM
----------------------------------------------------------------
POLYGON ((1 2, 3 2, 3 4, 1 4, 1 2))

SELECT st_srid(st_MakeEnvelope(1,2,3,4,4326))  srid FROM dual;

SRID
------------
4326

SELECT st_srid(st_MakeEnvelope(1,2,3,4)) srid FROM dual;

SRID
------------
0

SELECT st_astext(st_MakeEnvelope(1,2,3,4,null), 0) geom FROM dual;

GEOM
----------------------------------------------------------------


SELECT st_astext(st_MakeEnvelope(1,2,3,null), 0) geom FROM dual;

GEOM
----------------------------------------------------------------
```

ST_MAKELINE

```
st_makeline::= ST_MAKELINE "(" geometry1 "," geometry2  ")"
```

ST_MAKELINE函数根据输入的geometry1和geometry2，返回由组成它们的点按顺序连接的LINESTRING数据。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- 输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。
- geometry1和geometry2须为POINT类型、LINESTRING类型和MULTIPOINT类型的其中一种。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_AsText(ST_MakeLine(ST_GeomFromText('POINT(1 1)'), ST_GeomFromText('POINT(2 2)')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING (1 1, 2 2)

SELECT ST_AsText(ST_MakeLine(ST_GeomFromText('LINESTRING(1 1, 2 2)'), ST_GeomFromText('POINT(3 3)')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING (1 1, 2 2, 3 3)

SELECT ST_AsText(ST_MakeLine(ST_GeomFromText('MULTIPOINT(1 1 1, 2 2 2)'), ST_GeomFromText('POINT(3 3)')), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING Z (1 1 1, 2 2 2, 3 3 0)

--geometry1和geometry2的SRID不同时返回错误
SELECT ST_AsText(ST_MakeLine(ST_GeomFromText('MULTIPOINT(1 1 1, 2 2 2)',3), ST_GeomFromText('POINT(3 3)')), 0) res FROM DUAL;

YAS-07202 plugin execution error, different srid in st_makeline
```

ST_MAKEPOINT

```
st_makepoint::= ST_MAKEPOINT "(" x "," y [ "," z [ "," m ] ] ")"
```

ST_MAKEPOINT函数根据输入的x、y和可选的z和m，返回对应坐标的POINT数据。

x、y、z、m

表示坐标，参数类型为数值型，支持可以转换为DOUBLE类型的CHAR、VARCHAR类型（转换失败返回Invalid Number错误），使用其他数据类型则返回错误。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_MakePoint(1, 2), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT (1 2)

SELECT ST_AsText(ST_MakePoint(1, 2, 3), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT Z (1 2 3)

SELECT ST_AsText(ST_MakePoint(1, 2, 3, 4), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT Z (1 2 3)
```

ST_MAXDISTANCE

```
st_maxdistance::= ST_MAXDISTANCE "(" geometry1 "," geometry2 ")"
```

ST_MAXDISTANCE函数根据输入的geometry1和geometry2，返回它们对应的二维最大距离。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry1和geometry2的空间参考系标识号（SRID）必须相等，否则报错。

本函数遵守如下规则：

- 当输入的参数存在NULL或EMPTY时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_MaxDistance(ST_GeomFromText('linestring(-72.1523 42.6343, -72.4524 42.2872)', 4326),
ST_GeomFromText('linestring(-72.4524 42.4526, -72.1235 42.3521)',4326)) res FROM dual;

RES
-----------
3.508E-001

SELECT ST_MaxDistance(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123 42.1546, -72.1244
42.3527, -72.1260 42.45))',4326)) res FROM dual;

RES
-----------
1.975E-001
```

ST_MULTI

```
st_multi::= ST_Multi "(" geometry ")"
```

ST_MULTI函数用于返回输入的Geometry对象所对应的Geometry Collection类型。若输入的Geometry已是集合类型，则返回输入的Geometry对象。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入Geometry类型与返回Geometry类型的对应关系见下表：

输入类型	输出类型
Point	MultiPoint
LineString	MultiLineString
Polygon	MultiPolygon
MultiPoint	MultiPoint
MultiLineString	MultiLineString
MultiPolygon	MultiPolygon
GeometryCollection	GeometryCollection

示例

```
SELECT st_astext(st_multi(st_geomfromText('point(0 0)', 4326)), 0) multi FROM dual;

MULTI
-----------------------------------------------------------------
MULTIPOINT (0 0)

SELECT st_astext(st_multi(st_geomfromText('linestring(0 0, 1 1)', 4326)), 0) multi FROM dual;

MULTI
-----------------------------------------------------------------
MULTILINESTRING ((0 0, 1 1))

SELECT st_astext(st_multi(st_geomfromText('polygon((0 0, 1 1, 2 0, 0 0))', 4326)), 0) multi FROM dual;

MULTI
-----------------------------------------------------------------
MULTIPOLYGON (((0 0, 1 1, 2 0, 0 0)))

SELECT st_astext(st_multi(st_geomfromText('multipoint(0 0, 1 1)', 4326)), 0) multi FROM dual;

MULTI
-----------------------------------------------------------------
MULTIPOINT (0 0, 1 1)

SELECT st_astext(st_multi(st_geomfromText('multilinestring((0 0, 1 1),(2 2, 4 4))', 4326)), 0) multi FROM dual;

MULTI
-----------------------------------------------------------------
MULTILINESTRING ((0 0, 1 1), (2 2, 4 4))

SELECT st_astext(st_multi(st_geomfromText('multipolygon(((0 0, 1 1, 2 0, 0 0)), ((10 0, 11 1, 12 0, 10 0)))', 4326)), 0)
multi FROM dual;

MULTI
```

```
-------------------------------------------------------------
MULTIPOLYGON (((0 0, 1 1, 2 0, 0 0)), ((10 0, 11 1, 12 0, 10 0)))

SELECT st_astext(st_multi(st_geomfromText('geometrycollection(point(0 0), point(1 1))', 4326)), 0) multi FROM dual;

MULTI
-------------------------------------------------------------
GEOMETRYCOLLECTION (POINT (0 0), POINT (1 1))
```

ST_OVERLAPS

```
st_overlaps::= ST_OVERLAPS "(" geometry1 "," geometry2 ")"
```

ST_OVERLAPS函数的功能是判断两个Geometry相交并具有相同的维度，但彼此之间不完全包含。如果geometry1和geometry2“空间重叠”则返回TRUE，否则返回FALSE。

如果两个Geometry具有相同的维度，每个Geometry至少有一个点不属于另一个Geometry（或等价地说，它们都不覆盖另一个Geometry），并且它们内部的交叉点具有相同的尺寸，那么它们就是重叠的。重叠关系是对称的。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Overlaps(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 2 2, 3 3)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Overlaps(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 2 2, 3 5)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_Overlaps(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_PERIMETER

```
st_perimeter::= ST_PERIMETER "(" geometry ")"
```

ST_PERIMETER函数用于计算geometry的周长，或者说计算的是区域的周长，对于不能构成区域的几何图形，则返回0。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry是POLYGON、MULTIPOLYGON或GEOMETRY COLLECTION类型会返回对应的周长，其它的geometry类型会返回0。
- geometry的空间参考系标识号（SRID）必须在spatial_ref_sys系统表中定义或者为0，否则报错。
- geometry的空间参考系标识号（SRID）在spatial_ref_sys中对应的srs_type如果是GEOGRAPHY2D或者GEOGRAPHY3D，会切换到大地坐标算法计算，否则会使用投影坐标算法。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 该函数只会计算2D结果，如果输入的是三维，则会忽略Z坐标进行计算。
- 对于输入的经纬度坐标，如果输入的数值不在有效经纬度范围内，则会转换成有效经纬度进行计算。

示例（单机HEAP表）

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON ((-71.17 42.39,-72.17 43.39,-72.17 44.39,-71.17 42.39))', 4326)) res FROM dual;

RES
-----------
4.855E+005

SELECT ST_Perimeter(ST_GeomFromText('POLYGON((0 0, 10 0, 10 10, 0 10, 0 0))', 3385)) res FROM dual;

RES
-----------
4.0E+001
```

ST_POINT

```
st_point::= ST_POINT "(" x "," y [ "," srid ] ")"
```

ST_POINT函数根据输入的x、y和可选的srid，返回对应坐标和srid的POINT数据。

x、y

表示坐标，参数类型为数值型，遵循如下规则：

- 支持可以转换为DOUBLE类型的CHAR、VARCHAR类型（转换失败返回Invalid Number错误），使用其他数据类型则返回错误。
- 本函数仅支持二维坐标，使用其他维度的坐标则返回错误。

srid

srid的数据类型是INT，表示输出结果中的空间参考系，遵循如下规则：

- 支持能够隐式转换成INT的类型，如输入小数则进行四舍五入转换。
- 该参数可以省略，省略时默认值是0。
- 如果输入的是负数，则会按照默认的srid输出。

当输入的参数存在NULL时，函数返回NULL。

如需获取三维坐标的POINT数据，请使用ST_POINTZ函数。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_Point(1, 2), 0) res FROM DUAL;
RES
----------------------------------------------------------------
POINT (1 2)

SELECT ST_AsText(ST_Point(1, 2, 4326), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POINT (1 2)

--SRID为负数时按照默认的SRID输出
SELECT ST_SRID(ST_Point(1, 2, -8)) res FROM DUAL;

                RES
--------------------
                  0

SELECT ST_SRID(ST_Point(1, 2)) res FROM DUAL;

                RES
--------------------
                  0

SELECT ST_SRID(ST_Point(1, 2, 4326)) res FROM DUAL;

                RES
--------------------
               4326
```

ST_POINTZ

```
st_pointz::= ST_POINTZ "(" x "," y "," z [ "," srid ] ")"
```

ST_POINTZ函数根据输入的x、y、z和可选的srid，返回对应坐标和srid的POINT数据。

x、y、z

表示坐标，参数类型为数值型，遵循如下规则：

- 支持可以转换为DOUBLE类型的CHAR、VARCHAR类型（转换失败返回Invalid Number错误），使用其他数据类型则返回错误。
- 本函数仅支持三维坐标，使用其他维度的坐标则返回错误。

srid

srid的数据类型是INT，表示输出结果中的空间参考系，遵循如下规则：

- 支持能够隐式转换成INT的类型，如输入小数则进行四舍五入转换。
- 该参数可以省略，省略时默认值是0。
- 如果输入的是负数，则会按照默认的srid输出。

当输入的参数存在NULL时，函数返回NULL。

如需获取二维坐标的POINT数据，请使用ST_POINT函数。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_PointZ(1, 2, 3), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT Z (1 2 3)

SELECT ST_AsText(ST_PointZ(1, 2, 3, 4326), 0) res FROM DUAL;

RES
------------------------------------------------------------------
POINT Z (1 2 3)

--SRID为负数时按照默认的SRID输出
SELECT ST_SRID(ST_PointZ(1, 2, 3, -8)) res FROM DUAL;

                 RES
--------------------
                   0

SELECT ST_SRID(ST_PointZ(1, 2, 3)) res FROM DUAL;

                 RES
--------------------
                   0

SELECT ST_SRID(ST_PointZ(1, 2, 3, 4326)) res FROM DUAL;

                 RES
--------------------
                4326
```

ST_POLYGON

```
st_polygon::= ST_POLYGON "(" geometry "," srid ")"
```

ST_POLYGON函数根据输入的geometry和srid，返回对应srid中由geometry组成的POLYGON数据。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry须是LINESTRING类型。
- geometry包含的POINT数量须大于或等于4。
- geometry的首尾2个POINT必须相同。

srid

srid的数据类型是INT，表示输出结果中的空间参考系，遵循如下规则：

- 支持能够隐式转换成INT的类型，如输入小数则进行四舍五入转换。
- 如果输入的是负数，则会按照geometry本身的空间参考系标识号（SRID）进行计算。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_AsText(ST_Polygon(ST_GeomFromText('LINESTRING(0 0, 1 1, 2 3, 0 0 )'), 0), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((0 0, 1 1, 2 3, 0 0))

SELECT ST_AsText(ST_Polygon(ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 3 4, 0 0 0)'), 4326), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON Z ((0 0 0, 1 1 1, 2 3 4, 0 0 0))

--SRID为负数时按照geometry本身的SRID计算
SELECT ST_SRID(ST_Polygon(ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 3 4, 0 0 0)',5111), -4326)) res FROM DUAL;

         RES
------------
        5111

SELECT ST_SRID(ST_Polygon(ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 3 4, 0 0 0)'), 4326)) res FROM DUAL;

         RES
------------
        4326
```

ST_RELATE

```
st_relate::= ST_RELATE "(" geometry1 "," geometry2 [ "," boundaryNodeRule ] ")"
```

ST_RELATE函数的功能是根据输入的边界值规则(boundaryNodeRule)，计算用于表示输入的两个Geometry之间空间关系的DE-9IM矩阵字符串。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

boundaryNodeRule

boundaryNodeRule表示边界值规则，该参数为INTEGER类型，支持能够隐式转换成INTEGER类型的数据，默认值是1，可选参数有以下四个（输入其他值则报错）：

- 1: OGC/MOD2。
- 2: Endpoint。
- 3: MultivalentEndpoint。
- 4: MonovalentEndpoint。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Relate(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 2 2, 3 3)')) FROM DUAL;

ST_RELATE(ST_GEOMFRO
--------------------------------------------------------------------
1F100F102

SELECT ST_Relate(ST_GeomFromText('POINT(1 2)'), ST_GeomFromText('LINESTRING(3 5, 2 2, 3 3)')) FROM DUAL;

ST_RELATE(ST_GEOMFRO
--------------------------------------------------------------------
FF0FFF102

SELECT ST_Relate(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) FROM DUAL;

ST_RELATE(ST_GEOMFRO
--------------------------------------------------------------------
```

ST_SETSRID

```
st_setsrid::= ST_SETSRID "(" geometry "," srid ")"
```

ST_SETSRID函数用于将输入的geometry的空间参考系标识号（SRID）设置为指定的srid。

该函数根据输入的geometry返回设置好SRID的geometry。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

srid

srid表示要设置的SRID，其值为INT类型。

- 输入的srid如果是负数，则按照0处理。
- 支持能够隐式转换成INT的类型，如果输入的是小数则进行四舍五入转换。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_AsText(ST_SetSrid(ST_GeomFromText('POINT(-58.2687 29.149)',1356), 4326)) res FROM DUAL;

RES
------------------------------------------------------------------
POINT (-58.268700000000003 29.149000000000001)

--ST_GEOMFROMWKB函数会根据给定的WKB和SRID返回一个ST_GEOMETRY数据
SELECT ST_SRID(ST_SetSrid(ST_GeomFromWkb('0101000000000000000000F03F0000000000000040',4322), 4326)) res FROM DUAL;

        RES
------------
        4326


--参数包含NULL
SELECT ST_SRID(ST_SetSrid(ST_GeomFromText(null,1356), 4326)) res FROM DUAL;

         RES
------------
```

ST_SHORTESTLINE

```
st_shortestline::= ST_SHORTESTLINE "(" geometry1 "," geometry2 ")"
```

ST_SHORTESTLINE函数根据输入的geometry1和geometry2，返回geometry1与geometry2之间的二维最短LineString，最短LineString的两个端点不一定是输入的几何图形的端点。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- geometry1和geometry2的空间参考系标识号（SRID）必须相等，否则报错。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 返回的LineString从geometry1开始，以geometry2结束。
- 如果geometry1和geometry2相交，则结果是一条以交点为起点和终点的直线。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_ShortestLine(ST_GeomFromText('POLYGON ((7.000 7.000, 6.000 6.000, 5.444 6.169, 7.000 7.000))
'),ST_GeomFromText('POLYGON((6.400 6.600, 6.231 7.156, 63.24 6.983,  6.400 6.600))')),6) res FROM dual;

RES
----------------------------------------------------------------
LINESTRING (7.000000 7.000000, 7.000000 7.000000)

SELECT ST_AsText(ST_ShortestLine(ST_Point(-72.1235, 42.3521, 4326), ST_GeomFromText('polygon((-72.1260 42.45, -72.123
42.1546, -72.1244 42.3527, -72.1260 42.45))',4326)), 6) res FROM dual;

RES
----------------------------------------------------------------
LINESTRING (-72.123500 42.352100, -72.124396 42.352094)
```

ST_SIMPLIFY

```
st_simplify::= ST_SIMPLIFY "(" geometry "," tolerance ")"
```

ST_SIMPLIFY函数的功能是使用Douglas-Peucker算法来简化输入的geometry。

当输入的参数存在NULL时，函数返回NULL。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据，遵循如下规则：

- 仅计算2D结果，如坐标中有Z轴，则会忽略Z坐标进行计算。
- 如输入的geometry是POINT、MULTIPOINT类型，则直接返回原来的值。
- 如输入的geometry是LINESTRING、MULTILINESTRING类型，除非输入的是LINESTRING EMPTY或者是MULTILINESTRING EMPTY，才会返回EMPTY，否则至少保留两个点。
- 如输入的geometry是POLYGON、MULTIPOLYGON类型，缩小到一定程度之后会返回POLYGON EMPTY或MULTIPOLYGON EMPTY。
- 如输入的geometry是GEOMETRYCOLLECTION类型，内部的各类数据变化与上述一致。

tolerance

tolerance表示容差，其值为DOUBLE类型，容差越大，则简化的程度越大，遵循如下规则：

- 支持能够隐式转换成DOUBLE的数据类型。
- 如输入负数，则转化成对应的正数进行计算。
- tolerance值过大时，输入的geometry可能会消失（变成EMPTY）。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_AsText(ST_Simplify(ST_GeomFromText('LINESTRING (3 5, 4 6, 2 1, 3 5)'), 100), 0) res FROM DUAL;

RES
----------------------------------------------------------------
LINESTRING (3 5, 3 5)

SELECT ST_AsText(ST_Simplify(ST_GeomFromText('POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))'), 0.5), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))

SELECT ST_AsText(ST_Simplify(ST_GeomFromText('POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))'), 1), 0) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON EMPTY

SELECT ST_AsText(ST_Simplify(ST_GeomFromText('POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))'), NULL), 0) res FROM DUAL;

RES
----------------------------------------------------------------
```

ST_SPLIT

```
st_split::= ST_SPLIT "(" input "," blade ")"
```

ST_SPLIT函数用于返回input几何对象被blade几何对象切割之后产生的几何对象。

本函数遵守如下规则：

- 输入参数input和blade须具有相同的空间参考系，否则函数返回错误。
- 输入参数input和blade任意一个为NULL时，函数返回NULL。
- 输入参数input和blade任意一个的坐标中包含NAN和INF时，函数返回错误。
- 输入参数input只能为LineString、MultiLineString、Polygon、MultiPolygon、GeometryCollection。
  - input为LineString时，blade可以为Point、MultiPoint、LineString、MultiLineString、Polygon或MultiPolygon。
  - input为Polygon时，blade只能为MultiLineString和LineString。
  - input为GeometryCollection时，会依次对GeometryCollection中的每个成员做切割，如果其中一个成员不支持切割，返回错误。

input

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

blade

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

示例

```
SELECT st_astext(st_split(st_geomfromtext('linestring(1 1, 3 3)'), st_geomfromtext('point(2 2)')), 0) split FROM dual;

SPLIT
-------------------------------------------------------------
GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2), LINESTRING (2 2, 3 3))

SELECT st_astext(st_split(st_geomfromtext('linestring(1 1, 4 4)'), st_geomfromtext('multipoint(2 2, 3 3)')), 0) split FROM
dual;

SPLIT
-------------------------------------------------------------
GEOMETRYCOLLECTION (LINESTRING (3 3, 4 4), LINESTRING (1 1, 2 2), LINESTRING (2 2, 3 3))

SELECT st_astext(st_split(st_geomfromtext('linestring(1 1, 4 4)'), st_geomfromtext('polygon((0 0, 0 2, 2 2, 2 0, 0 0))')), 0)
split FROM dual;

SPLIT
-------------------------------------------------------------
GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2), LINESTRING (2 2, 4 4))

SELECT st_astext(st_split(st_geomfromtext('polygon((0 0, 0 2, 2 2, 2 0, 0 0))'), st_geomfromtext('linestring(1 -1, 1 3)')),
0) split FROM dual;

SPLIT
-------------------------------------------------------------
GEOMETRYCOLLECTION (POLYGON ((0 0, 0 2, 1 2, 1 0, 0 0)), POLYGON ((1 2, 2 2, 2 0, 1 0, 1 2)))

SELECT st_astext(st_split(st_geomfromtext('geometrycollection(polygon((0 0, 0 2, 2 2, 2 0, 0 0)), linestring(0 0, 2 2))'),
st_geomfromtext('linestring(1 -1, 1 3)')), 0) split FROM dual;

SPLIT
-------------------------------------------------------------
GEOMETRYCOLLECTION (POLYGON ((0 0, 0 2, 1 2, 1 0, 0 0)), POLYGON ((1 2, 2 2, 2 0, 1 0, 1 2)), LINESTRING (0 0, 1 1),
LINESTRING (1 1, 2 2))
```

ST_SRID

```
st_srid::= ST_SRID "(" geometry ")"
```

ST_SRID函数用于查询输入的geometry的空间参考系标识号（SRID）。

该函数会根据输入的geometry返回一个integer类型的SRID。

geometry

geometry是一个合法的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_SRID(ST_GeomFromText('POINT(-58.2687 29.149)',1356)) res FROM DUAL;

         RES
------------
        1356

--ST_GEOMFROMWKB函数会根据给定的WKB和SRID返回一个ST_GEOMETRY数据
SELECT ST_SRID(ST_GeomFromWkb('0101000000000000000000F03F0000000000000040',4322)) res FROM DUAL;

         RES
------------
        4322

--参数包含NULL
SELECT ST_SRID(ST_GeomFromText(null,1356)) res FROM DUAL;

         RES
------------
```

ST_TOUCHES

```
st_touches::= ST_TOUCHES "(" geometry1 "," geometry2 ")"
```

ST_TOUCHES函数的功能是判断两个Geometry是否至少有一个共同点，且它们的内部不相交。如geometry1和geometry2相交，且它们的内部不相交，则返回TRUE，否则返回FALSE。

对于POINT/POINT输入的情况，总是返回FALSE，因为POINT没有边界。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

如果两个Geometry的DE-9IM交叉矩阵匹配以下情况，则此关系成立：

- FT*******
- F**T*****
- F***T****

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Touches(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 2 2, 3 5)')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Touches(ST_GeomFromText('LINESTRING(2 2, 1 1, 3 3)'), ST_GeomFromText('LINESTRING(3 5, 1 2, 3 5)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_Touches(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_TRANSFORM

```
st_transform::= ST_TRANSFORM "(" geometry "," srid ")"
```

ST_TRANSFORM函数根据输入的geometry和srid，返回geometry从原本的空间参考系转换到srid所指定的空间参考系的坐标数据的新geometry。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

- 必须有srid并且srid必须在系统表spatial_ref_sys中定义，否则会报错。

srid

srid的数据类型是INT，表示输出结果中的空间参考系，遵循如下规则：

- 支持能够隐式转换成INT的类型，如输入小数则进行四舍五入转换。
- 必须在系统表spatial_ref_sys中定义，否则会报错。

当输入的参数存在NULL时，函数返回NULL。

示例（单机HEAP表）

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('polygon((73.62 16.7, 74.15 16.45, 73.77 16.32, 73.62 16.7))', 4326), 4491),2)
res FROM dual;

RES
------------------------------------------------------------------
POLYGON ((13352805.80 1847616.32, 13409224.37 1819631.38, 13368550.72 1805451.07, 13352805.80 1847616.32))

SELECT ST_AsText(ST_Transform(ST_GeomFromText('linestring(73.62 16.7, 74.15 16.45)', 4326), 4491),2) res FROM dual;

RES
------------------------------------------------------------------
LINESTRING (13352805.80 1847616.32, 13409224.37 1819631.38)
```

ST_UNION

```
st_union::= ST_UNION "(" geometry1 "," geometry2 [ ","girdsize]")"
```

ST_UNION函数返回两个geometry对象的并集，返回值为ST_GEOMETRY类型数据。

函数会将geometry对象投射到网格线上进行计算并返回结果。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

gridsize

gridsize的数据类型为DOUBLE，表示函数计算时使用的网格线大小，省略则默认为-1。

本函数遵守如下规则：

- 当geometry1和geometry2均为EMPTY时，函数返回geometry1。
- 当geometry1和geometry2其中一个为EMPTY时，函数返回该非空对象。
- 当输入的参数存在NULL时，函数返回NULL。
- 当输入的参数中包含Nan时，函数返回错误。
- 支持输入3D坐标，但函数会忽略Z坐标进行计算。

示例（单机HEAP表）

```
SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON((0 0,0 5,5 5,5 0,0 0))'),ST_GEOMFROMTEXT('POLYGON((1 0,0 8,8 8,8 0,1
0))'))) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON ((0.000000000000000 5.000000000000000, 0.375000000000000 5.000000000000000, 0.000000000000000 8.000000000000000,
8.000000000000000 8.000000000000000, 8.000000000000000 0.000000000000000, 5.000000000000000 0.000000000000000,
1.000000000000000 0.000000000000000, 0.000000000000000 0.000000000000000, 0.000000000000000 5.000000000000000))

--geometry1和geometry2均为EMPTY时返回geometry1
SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON EMPTY'),ST_GEOMFROMTEXT('LINESTRING EMPTY'))) res FROM DUAL;

RES
----------------------------------------------------------------
POLYGON EMPTY

--geometry1和geometry2其中一个为EMPTY时返回非空对象
SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON((0 0,0 5,5 5,5 0,0 0))'),ST_GEOMFROMTEXT('LINESTRING EMPTY'))) res FROM
DUAL;

RES
----------------------------------------------------------------
POLYGON ((0.000000000000000 0.000000000000000, 0.000000000000000 5.000000000000000, 5.000000000000000 5.000000000000000,
5.000000000000000 0.000000000000000, 0.000000000000000 0.000000000000000))

--参数存在NULL时返回NULL
SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON((0 0,0 5,5 5,5 0,0 0))'),NULL)) res FROM DUAL;

RES
----------------------------------------------------------------


SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON((0 0,0 5,5 5,5 0,0 0))'),ST_GEOMFROMTEXT('POLYGON((1 0,0 8,8 8,8 0,1
0))'),NULL)) res FROM DUAL;

RES
----------------------------------------------------------------
```

```
--SRID不同时返回错误
SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON((0 0,0 5,5 5,5 0,0 0))',6),ST_GEOMFROMTEXT('POLYGON((1 0,0 8,8 8,8 0,1
0))',1))) res FROM DUAL;

YAS-07202 plugin execution error, Operation on mixed SRID geometries: 6 != 1

--参数中含有Nan时返回错误
SELECT ST_ASTEXT(ST_UNION(ST_GEOMFROMTEXT('POLYGON((0 0,0 Nan,5 5,5 0,0 0))'),ST_GEOMFROMTEXT('POLYGON((1 0,0 8,8 8,8 0,1
0))'))) res FROM DUAL;

YAS-07202 plugin execution error, POLYGON has invalid coordinate
```

ST_WITHIN

```
st_within::= ST_WITHIN "(" geometry1 "," geometry2 ")"
```

ST_WITHIN函数的功能是判断geometry1是否完全在geometry2的内部，如果是则返回TRUE，否则返回FALSE。

geometry1在geometry2内部指当且仅当geometry1中没有点位于geometry2的外部，且geometry1的内部至少有一个点位于geometry2的内部的情况。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

输入的geometry1和geometry2须具有相同的空间参考系标识号（SRID）。

本函数遵守如下规则：

- ST_WITHIN是ST_CONTAINS的逆。因此，ST_WITHIN(A,B) = ST_CONTAINS(B,A)。
- 当输入的参数存在NULL时，函数返回NULL。
- 仅计算2D结果，若输入参数中存在Z坐标，函数将直接忽略Z坐标进行计算。
- 能够保证的精度是小数点后面15位，小数部分超出15位之后结果不保证。
- 遵循DE-9IM（Dimensionally Extended 9-Intersection Model）规则。

示例（单机HEAP表）

```
--ST_GEOMFROMTEXT函数会根据给定的WKT和SRID返回一个ST_GEOMETRY数据
SELECT ST_Within(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), ST_GeomFromText('POINT(3 3)')) res FROM DUAL;

RES
--------------------
false

SELECT ST_Within(ST_GeomFromText('POINT(3 3)'), ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))')) res FROM DUAL;

RES
--------------------
true

SELECT ST_Within(ST_GeomFromText('POLYGON((2 2, 2 4, 4 4, 4 2, 2 2))'), NULL) res FROM DUAL;

RES
--------------------
```

ST_X

```
st_x::= ST_X "(" geometry ")"
```

ST_X函数根据输入的geometry，返回该点的x轴坐标。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_X(ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

RES
-----------
1.0E+000
```

ST_Y

```
st_y::= ST_Y "(" geometry ")"
```

ST_Y函数根据输入的geometry，返回该点的y轴坐标。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

当输入的参数存在NULL时，函数返回NULL，空串作为NULL处理。

示例（单机HEAP表）

```
SELECT ST_Y(ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

RES
-----------
2.0E+000
```

ST_Z

```
st_z::= ST_Z "(" geometry ")"
```

ST_Z函数用于返回输入的geometry的z轴坐标。

本函数遵守如下规则：

- 当输入的参数存在NULL时，函数返回NULL。
- 当输入的Geometry类型不是Point时，将返回错误。
- 当输入的Geometry坐标只有2维时，将返回NULL。

geometry

通用表达式，其值必须为有效的ST_GEOMETRY类型的数据。

示例

```
SELECT ST_Z(ST_GeomFromText('POINT(1 2 3)')) res FROM DUAL;

RES
-----------
3.0E+000

SELECT ST_Z(ST_GeomFromText('POINT(1 2)')) res FROM DUAL;

        RES
-----------
```

GREATEST

```
greatest::= GREATEST "(" (expr) {"," (expr)} ")"
```

GREATEST从多个参数expr的值中找出最大的一个值，并按第一个参数的数据类型返回结果。其规则为：

- 当expr值全为数值型数据时，按大小进行比较。
- 当第一个expr值为数值型数据，且存在一部分为字符型数据时，将字符型转换为NUMBER类型后执行比较，如转换失败则返回Invalid number错误。
- 当expr值全为字符型数据时，按字典序进行比较。
- 当第一个expr值为字符型数据，且存在一部分为非字符型数据时，会把非字符型转换成字符型，然后按字典序进行比较。
- 当expr值全为日期时间型数据时，按照时间大小进行比较。
- 当expr值全为布尔型数据时，按照true=1，false=0进行比较。
- 当参数列表中的任一个expr值为NULL时，函数返回NULL。
- 当expr为LOB、XMLTYPE、JSON时，函数返回unexpected lob错误。
- 上述之外的其他情形，函数返回Invalid datatype错误。

示例

```
SELECT GREATEST(2, 5, 12, 3, 16, 8, 9) AS RES1,
GREATEST('A', 6, 7, 5000, 'E', 'F','G') AS RES2,
GREATEST(SYSDATE,TO_DATE('2014-08-01','YYYY-MM-DD')) AS RES3,
GREATEST(true,false) RES4
FROM DUAL;
        RES1 RES2  RES3                           RES4
------------ ----- ------------------------------ --------------------
          16 G     2021-12-02 23:09:54            true
```

GROUPING

```
grouping::= GROUPING "(" expr ")"
```

GROUPING函数用于在一个分组聚合结果的每一行中标记字段expr是否为该行所在分组的键值。

该函数的返回值类型为NUMBER，输出结果为0表示该字段为键值，输出结果为1则表示该字段不是键值。

该函数必须与GROUP BY expr/ROLLUP/CUBE/GROUPING SETS语句结合使用，函数内部不能使用DISTINCT。

expr的值可以为除UDT以外的任意数据类型，expr不能为NULL且不能为聚集函数。

该函数仅适用于列存表。

示例（TAC表、LSC表）

```
SELECT GROUPING(year) res FROM finance_info GROUP BY GROUPING SETS(year, month);

        RES
-----------
          1
          1
          0
          0

SELECT GROUPING(year) res FROM finance_info GROUP BY ROLLUP(year, month);

        RES
-----------
          0
          0
          0
          0
          0
          1
```

GROUPING_ID

```
grouping_id::= GROUPING_ID "(" (expr) {"," (expr)} ")"
```

GROUPING_ID函数返回所有参数expr的二进制GROUPING结果所对应的十进制数值。

该函数的返回值类型为NUMBER，当GROUPING_ID的参数expr只有1个时，结果同GROUPING。

该函数支持1 - 126个参数，参数数量不在此范围则报错。

该函数必须与GROUP BY expr/ROLLUP/CUBE/GROUPING SETS语句结合使用，函数内部不能使用DISTINCT。expr可以为除UDT以外的任意数据类型，不能为NULL且不能为聚集函数。

该函数仅适用于列存表。

示例（TAC表、LSC表）

```
SELECT GROUPING_ID(year) res FROM finance_info GROUP BY GROUPING SETS(year, month);

        RES
-----------
          1
          1
          0
          0

SELECT GROUPING_ID(year, month) res FROM finance_info GROUP BY GROUPING SETS(year, month);

        RES
-----------
          2
          2
          1
          1
```

GROUP_CONCAT

```
group_concat::= GROUP_CONCAT "(" [DISTINCT] (string) {"," (string)} [order_by_clause [SEPARATOR sep_character]] ")"
```

GROUP_CONCAT函数在CONCAT函数的功能上增加了聚集功能，即对GROUP BY聚集的每个分组里的多行执行CONCAT操作，函数返回值是CLOB类型。

该函数不支持列式计算。

DISTINCT

过滤在同一组内出现的相同string。

string

string须为字符型，或可转换为字符型的其他类型，但不允许为JSON、NVARCHAR、NCHAR和NCLOB类型。

当string为NULL时，函数返回NULL。

string参数可以为：

- expr
- 查询列为单列且返回行为单行的子查询

order_by_clause

对组内要CONCAT的string排序，其语法与SELECT语句中描述一致。

当ORDER BY后指定的是常量数字时，表示的是string的顺序值。

SEPARATOR sep_character

指定将组内的多行进行CONCAT时，多行之间加上sep_character定义的分隔符，sep_character只能为character类的常量或NULL。

此语句不指定时，默认为','。

示例（HEAP表）

```
--创建exprs_group表，并插入数据
CREATE TABLE exprs_group (expra INT,exprb CHAR(3));
INSERT INTO exprs_group VALUES(1, 'aaa');
INSERT INTO exprs_group VALUES(1, 'bbb');
INSERT INTO exprs_group VALUES(2, 'bbb');
INSERT INTO exprs_group VALUES(2, 'ccc');
INSERT INTO exprs_group VALUES(3, 'ccc');
INSERT INTO exprs_group VALUES(3, 'ddd');
COMMIT;

--未指定GROUP BY时，将所有行CONCAT，得到一行结果
SELECT GROUP_CONCAT(expra,exprb) res FROM exprs_group;
RES
----------------------------------------------------------------
1aaa,1bbb,2bbb,2ccc,3ccc,3ddd

--group by后各组的多行数据分别CONCAT成一行，得到按组的多行结果
SELECT expra, GROUP_CONCAT(expra,exprb ORDER BY 2 SEPARATOR '$') res
FROM exprs_group
GROUP BY expra;
       EXPRA RES
------------ ----------------------------------------------------------------
           1 1aaa$1bbb
           2 2bbb$2ccc
           3 3ccc$3ddd

--子查询
SELECT expra,GROUP_CONCAT((SELECT expra FROM exprs_group WHERE ROWNUM=1),exprb ORDER BY 2 SEPARATOR '$') res
```

```
FROM exprs_group
GROUP BY expra;
       EXPRA RES
------------ ----------------------------------------------------------------
           1 1aaa$1bbb
           2 1bbb$1ccc
           3 1ccc$1ddd
```

GROUP_ID

```
group_id::= GROUP_ID "("")"
```

GROUP_ID函数用于返回重复的分组。不同的分组结果会以0进行标识，相同的分组结果会在0的基础上递增。

该函数无入参，返回类型为INT。

该函数必须与GROUP BY expr/ROLLUP/CUBE/GROUPING SETS语句结合使用。

该函数仅适用于列存表。

示例（LSC表、TAC表）

```
SELECT year, month, GROUP_ID() res FROM finance_info GROUP BY GROUPING SETS(year, year, month);

YEAR  MONTH          RES
----- ----- ------------
      01               0
      02               0
2001                   0
2021                   0
2001                   1
2021                   1

SELECT year, month, GROUP_ID() res FROM finance_info GROUP BY ROLLUP(year, year, month);

YEAR  MONTH          RES
----- ----- ------------
2001  01               0
2021  01               0
2021  02               0
2001                   0
2021                   0
2001                   1
2021                   1
                       0
```

HEXTORAW

```
hextoraw::= HEXTORAW "(" expr ")"
```

HEXTORAW函数将expr表述的十六进制字符串转为二进制数据，返回一个RAW类型的结果。

函数将expr中的每两个十六进制字符子串转成一个字节，如字符串'ab'转成'0xab'；如果expr的字符数量为奇数，函数将先补充4个都是0的连续位，如字符串'abc'转成'0x0abc'。

该函数不支持列式计算。

expr的值须为字符型，或可转换为字符型的其他类型，但不允许为除BLOB外的其它LOB类型数据。当expr的值为NULL时，函数返回NULL。

expr中包含不属于十六进制表述范围（0-9、a-f、A-F）的字符时，函数报错。

当本函数用于插入或更新一个RAW列字段时，如果expr的字节长度超过RAW列宽度的2倍，函数报错。

示例（HEAP表）

```
SELECT HEXTORAW('1234abcd') res FROM DUAL;
RES
--------------------
1234ABCD

SELECT HEXTORAW('') res FROM DUAL;
RES
--------------

--包含不属于十六进制表述范围（0-9、a-f、A-F）的字符时报错
SELECT HEXTORAW('1234abcddfdfh') res FROM DUAL;
YAS-00223 invalid hex number

--奇数位在前面补0
SELECT HEXTORAW('1abcdef') res FROM DUAL;
RES
--------
01ABCDEF

--超过RAW列宽度的2倍时报错
CREATE TABLE tb_raw(c1 RAW(4));
INSERT INTO tb_raw VALUES('abcdabcd');
INSERT INTO tb_raw VALUES('abcdabcda');
YAS-04008 C1 size exceeding limit 4
```

IF

```
if::= IF "(" expr1 "," expr2 "," expr3 ")"
```

IF函数有3个expr参数，若expr1为TRUE，返回expr2，若expr1为FALSE，返回expr3。

expr1的值可以为任意数据类型，为NULL时视为FALSE，若不为NULL：

- 对于数值类型或可隐式转换为数值类型的其他类型，其值为非0时，视为TRUE，其值为0时，视为FALSE；
- 对于不可隐式转换为数值类型的类型，视为FALSE；
- 对于BOOLEAN类型和BOOLEAN表达式，按其真假处理。

expr2，expr3的值可以为任意数据类型。当expr2和expr3数据类型不相同时，函数将先进行隐式类型转换再返回结果，转换规则与IFNULL中规则一致。

需注意，本函数如下规格与MySQL中的IF函数存在差异：

- 当expr2和expr3均为BIT类型时，YashanDB返回BIT类型，MySQL返回INT/BIGINT UNSIGNED类型；
- 当epxr2和epxr3为BOOLEAN、TINYINT或SMALLINT类型时，YashanDB按照BOOLEAN<TINYINT<SMALLINT的优先级进行类型转换，MySQL全部返回INT类型；

示例

```
SELECT IF(1, 2, 3) res FROM DUAL;
         RES
------------
           2

SELECT IF(0, 2, 3) res FROM DUAL;
         RES
------------
           3

SELECT IF(SYSDATE, TRUE, FALSE) res FROM DUAL;
RES
--------------------
false

SELECT IF(TRUE, TRUE, FALSE) res FROM DUAL;
RES
--------------------
true

SELECT IF(FALSE, TRUE, FALSE) res FROM DUAL;
RES
--------------------
false

SELECT IF(2,5.44, CAST('0.232222' AS DOUBLE)) res,
TYPEOF (IF(2,5.44, CAST('0.232222' AS DOUBLE))) res_type
FROM DUAL;
        RES RES_TYPE
----------- -----------
  5.44E+000 double
```

IFNULL

```
ifnull::= IFNULL "(" expr1 "," expr2 ")"
```

IFNULL函数有2个expr参数，当expr1不为NULL时返回expr1，否则返回expr2。

expr1，expr2的值可以为任意数据类型。当expr1和expr2的数据类型相同时，函数返回此数据类型的值。

当expr1，expr2其中之一为RAW、JSON、CLOB、BLOB、NCLOB类型时，不允许与其它类型搭配使用，两个参数必须类型相同，否则函数返回错误，函数返回值类型同expr数据类型。

当expr1，expr2其中之一为XMLTYPE时，expr1和expr2的数据类型必须相同，否则函数返回错误，函数返回值类型为XMLTYPE类型。

若expr1和expr2的数据类型不相同，函数将先进行隐式类型转换后再返回结果，基本规则如下：

- expr1与expr2分属如下不同大类组时，函数返回VARCHAR类型：
- 数值型
- 日期时间型
- 字符型
- ROWID
- expr1与expr2其中之一为布尔型时：
  - 另一方属于非数值型的其他大类时，函数返回VARCHAR类型。
  - 另一方属于数值型（除BIT外）时，函数返回对应的数值类型。
- 另一方为BIT类型时，函数返回BIGINT类型。
- expr1与expr2均属于字符型时，函数返回VARCHAR类型。
- expr1与expr2均属于日期时间型时，函数返回类型如下表所示：

expr1/expr2	DATE	TIME	TIMESTAMP	INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND
DATE	DATE	DATE	TIMESTAMP	VARCHAR	VARCHAR
TIME	DATE	TIME	TIMESTAMP	VARCHAR	VARCHAR
TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP	VARCHAR	VARCHAR
INTERVAL YEAR TO MONTH	VARCHAR	VARCHAR	VARCHAR	INTERVAL YEAR TO MONTH	VARCHAR
INTERVAL DAY TO SECOND	VARCHAR	VARCHAR	VARCHAR	VARCHAR	INTERVAL DAY TO SECOND

其中，在将TIME类型转换为DATE或TIMESTAMP类型时，年月日补充为当前日期。

- expr1与expr2均属于数值型时，函数返回类型如下表所示：

expr1/expr2	BIT	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	NUMBER
BIT	BIT	BIGINT	BIGINT	BIGINT	BIGINT	DOUBLE	DOUBLE	NUMBER
TINYINT	BIGINT	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	NUMBER

expr1/expr2	BIT	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	NUMBER
SMALLINT	BIGINT	SMALLINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	NUMBER
INT	BIGINT	INT	INT	INT	BIGINT	FLOAT	DOUBLE	NUMBER
BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	FLOAT	DOUBLE	NUMBER
FLOAT	DOUBLE	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	DOUBLE	DOUBLE
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE
NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	DOUBLE	DOUBLE	NUMBER(p/s)

- expr1与expr2均为NUMBER类型且精度相同时，函数返回此精度的NUMBER类型。
- expr1与expr2为不同精度的NUMBER类型，或者一方为NUNBER类型，另一方为非NUMBER类型时，函数返回浮动精度的NUMBER类型。

示例

```
SELECT IFNULL(1, 2) res FROM DUAL;
         RES
------------
           1

SELECT IFNULL(NULL, 2) res FROM DUAL;
RES
-----
2

SELECT IFNULL(TRUE, 2) res FROM DUAL;
         RES
------------
           1

SELECT IFNULL(5, CAST('0.232222' AS FLOAT)) res,
TYPEOF (IFNULL(5, CAST('0.232222' AS FLOAT))) res_type
FROM DUAL;
        RES RES_TYPE
----------- ----------
   5.0E+000 float
```

INITCAP

```
initcap::= INITCAP "(" expr ")"
```

INITCAP函数将expr表示的字符串进行单词分隔，并将分隔后的单词转换为首字母大写，非首字母小写样式。

函数将expr中的除英文字母、数字字符外的其他字符认定为分隔符并据此分隔单词，包括标点符号、普通符号、控制字符、希腊字母以及中文字符等。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr为NCLOB/NCHAR/NVARCHAR时返回值为NVARCHAR类型，其余场景返回值为VARCHAR类型。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr的值为NULL时，函数返回NULL。

示例

```
SELECT INITCAP('1wWork who i中文s i1w') res FROM DUAL;
RES
-------------------------
1wwork Who I中文S I1w

SELECT INITCAP(SYSDATE||':today') res FROM DUAL;
RES
-----------------------------------------------------------------
2022-11-08 02:41:17:Today

----对浮点型数据先转为字符型
SELECT INITCAP(CAST('inf' AS FLOAT)||'/'||CAST('1.11' AS FLOAT)) res FROM DUAL;
RES
---------------------
Inf/1.11000001e+000

--对二进制数据先转为字符型
SELECT INITCAP(b'010101') res FROM DUAL;
RES
------------------
21
```

INSTR

```
instr::= INSTR "(" expr "," sub_character ["," position ["," occurrence]] ")"
```

INSTR函数根据子字符串、起始位置、出现次数在expr表示的字符串中查找，并返回第occurrence次出现的BIGINT类型的位置值，未查找到则返回0。

expr、sub_character

expr、sub_character均为通用表达式，两个参数的值须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- 当sub_character的值为NULL时，函数返回NULL。

position

开始查找的起始位置，是与expr相同的通用表达式，该值须为数值型数据，或可转换为NUMBER类型的其他类型数据。起始位置必须为整数，或可被转换为整数，取值范围[-9223372036854775808, 9223372036854775807]。正整数表示从前往后自起始位置开始查找，负整数表示从后向前自起始位置开始查找。

- 当position的值为NULL时，函数返回NULL。
- 当position的值为0时，函数不执行查找并返回0。
- 当position的值为带有小数的NUMBER类型时，函数截断其小数位保留整数位。
- 若position的值为浮点类型时，函数将其奇进偶舍至整数。
- 不指定position时，默认起始位置为1。

occurence

指定按子字符串出现的第几次进行查找，，是与expr相同的通用表达式，该值须为数值型数据，或可转换为NUMBER类型的其他类型数据。occurence的值必须为正整数，或可被转换为正整数，取值范围[1,9223372036854775807]。

- 当occurence的值为NULL时，函数返回NULL。
- 当occurence的值为带有小数的NUMBER类型时，函数截断其小数位保留整数位。
- 若occurence的值为浮点类型时，函数将其奇进偶舍至整数。
- 不指定occurence时，默认为1。

示例

```
SELECT INSTR('This is a playlist', 'is') POS FROM DUAL;
                  POS
---------------------
                    3

SELECT INSTR('This is a playlist', 'is', 1, 2) POS FROM DUAL;
                  POS
---------------------
                    6

SELECT INSTR('This is a playlist', 'is', 1, 3) POS FROM DUAL;
                  POS
---------------------
                   16

SELECT INSTR('This is a playlist', 'is', -2, 3) POS FROM DUAL;
                  POS
---------------------
                    3

SELECT INSTR('This is a playlist', 'is', 5, 3) POS FROM DUAL;
```

```
                   POS
--------------------
                     0
```

INSTRB

```
instrb::= INSTRB "(" expr "," sub_character ["," position ["," occurrence]] ")"
```

INSTRB函数根据子字符串、起始位置、出现次数在expr表示的字符串中按字节查找，并返回第occurrence次出现的BIGINT类型的位置值，未查找到则返回0。

本函数遵循如下规则：

- 该函数不支持列式计算。

expr、sub_character

expr、sub_character，其中sub_character为与expr相同的通用表达式，两个参数的值须为字符型，或可转换为字符型的其他类型，与INSTR函数不同。

- expr、sub_character不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr的值为NULL时，函数返回NULL。
- 当sub_character的值为NULL时，函数返回NULL。

position

开始查找的起始位置，是与expr相同的通用表达式，该值须为数值型数据，或可转换为NUMBER类型的其他类型数据。起始位置必须为整数，或可被转换为整数，取值范围[-4294967295, 4294967295]。正整数表示从前往后自起始位置开始查找，负整数表示从后向前自起始位置开始查找。

- 当position的值为NULL时，函数返回NULL。
- 当position的值为0时，函数不执行查找并返回0。
- 当position的值为带有小数的NUMBER类型时，函数截断其小数位保留整数位。
- 当position的值为浮点类型时，函数将其奇进偶舍至整数。
- 不指定position时，默认起始位置为1。

occurence

指定按子字符串出现的第几次进行查找，是与expr相同的通用表达式，该值须为数值型数据，或可转换为NUMBER类型的其他类型数据。occurence的值必须为正整数，或可被转换为正整数，取值范围[1,4294967295]。

- 当occurence的值为NULL时，函数返回NULL。
- 当occurence的值为带有小数的NUMBER类型时，函数截断其小数位保留整数位。
- 当occurence的值为浮点类型时，函数将其奇进偶舍至整数。
- 不指定occurence时，默认为1。

示例（HEAP表）

```
SELECT INSTRB('abc', 'a') FROM DUAL;

INSTRB('ABC','A')
---------------------
                    1

SELECT INSTRB('abc', 'b') FROM DUAL;

INSTRB('ABC','B')
---------------------
                    2

SELECT INSTRB('一天一夜一天','一夜',7) FROM DUAL;

INSTRB('一天一夜一天','一夜'
```

```
-----------------------------------
                                  7

SELECT INSTRB('一天一夜一天','一夜',7.4) FROM DUAL;

INSTRB('一天一夜一天','一夜'
-----------------------------------
                                  7
```

ISNULL

```
isnull::= ISNULL "(" expr ")"
```

ISNULL函数判断参数expr的值是否为空，返回true或false。

expr的值可以为除ROWID、UDT类型外的其它任意数据类型。

对待空串和NULL均输出true。

示例

```
SELECT ISNULL('') nulla, ISNULL(3-3) nullb, ISNULL('      ') nullc FROM DUAL;
NULLA                NULLB                NULLC
-------------------- -------------------- --------------------
true                 false                false
```

JSON

```
json::= JSON "(" expr [EXTENDED] ")"
```

JSON函数仅包含一个expr参数，返回根据该expr的值所转换而来的二进制json数据。其中，当转换后的json数据为JSON Object类型时，函数返回的是按key值进行排序后的结果。

expr的值必须为CHAR、VARCHAR或CLOB类型，否则函数返回Invalid datatype错误。函数最大支持32M字节的字符串数据作为输入。

当expr的值为NULL时，函数返回NULL。

JSON函数存在别名，为JSON_PARSE，其功能与JSON完全相同。

EXTENDED

EXTENDED关键字用于指定使用扩展模式解析JSON字符串，详见JSON扩展格式解析。

示例

```
SELECT JSON_FORMAT(JSON('[123, "ABC", false]')) yason_res FROM DUAL;
YASON_RES
--------------------------------------------------------------------
[123,"ABC",false]

--JSON Object类型会进行key值排序
SELECT JSON_FORMAT(JSON('{"keyC":123, "keyabc":false, "keyA":234, "keyB":345, "key":true}') pretty) yason_res FROM DUAL;
YASON_RES
--------------------------------------------------------------------
{
  "key" : true,
  "keyA" : 234,
  "keyB" : 345,
  "keyC" : 123,
  "keyabc" : false
}

-- 通过扩展模式解析JSON字符串
SELECT JSON('{"$binary": "SGVsbG8gd29ybGQh"}' EXTENDED) yason_res FROM DUAL;
YASON_RES
--------------------------------------------------------------------
"48656C6C6F20776F726C6421"
```

JSON_ARRAY_GET

```
json_array_get::= JSON_ARRAY_GET "(" json_value ", " index ")"
```

JSON_ARRAY_GET函数从一个JSON数组数据中返回指定位置的元素。

json_value

json_value为一个二进制json数据，可通过JSON函数获取。

当json_value为JSON Array类型时，函数返回指定index位置的元素；当json_value为JSON Object/String/Number/Boolean/Null/扩展格式类型时，函数返回NULL。

当json_value为NULL时，函数返回NULL。

index

index指定返回元素的位置，须为整型数值，即TINYINT/SMALLINT/INT/BIGINT。

当index的值为NULL时，函数返回NULL。

当index的值为正数时，表示按从左往右的顺序取值；当index的值为负数时，表示按从右往左的顺序取值；当index的取值不在数组的索引范围内时，函数返回NULL。

示例

```
SELECT JSON_FORMAT(JSON_ARRAY_GET(JSON('[123, "ABC", false]'), 0)) res FROM DUAL;
RES
----------------------------------------------------------------
123

SELECT JSON_FORMAT(JSON_ARRAY_GET(JSON('{"id": 1}'), 0)) res FROM DUAL;
RES
----------------------------------------------------------------

SELECT JSON_FORMAT(JSON_ARRAY_GET(JSON('[123, "ABC", false]'), -1)) res FROM DUAL;
RES
----------------------------------------------------------------
false
```

JSON_ARRAY_LENGTH

```
json_array_length::= JSON_ARRAY_LENGTH "(" json_value ")"
```

JSON_ARRAY_LENGTH函数返回一个JSON数组数据的长度。

json_value

json_value为一个二进制json数据，可通过JSON函数获取。

当json_value为JSON Array类型时，函数返回数组的长度；当json_value为JSON Object/String/Number/Boolean/Null/扩展格式类型时，函数返回NULL；当json_value为NULL时，函数返回NULL。

示例

```
SELECT JSON_ARRAY_LENGTH(JSON('[123, "ABC", false]')) res FROM DUAL;
                   RES
---------------------
                     3

SELECT JSON_ARRAY_LENGTH(JSON('{"ID": 1}')) res FROM DUAL;
                   RES
---------------------
```

JSON_EXISTS

```
json_exists::= JSON_EXISTS "(" json_value "," "json_path" ")"
```

JSON_EXISTS函数基于json_path所描述的路径对json_value进行查找，若对应查询结果不为空则返回TRUE，否则返回FALSE。

json_value

json_value为一个二进制json数据，可通过JSON函数获取。当json_value为NULL时，函数返回NULL。

json_path

路径表达式，为一个常量字符串，其格式定义请参考json文档中描述。

示例

```
CREATE TABLE IF NOT EXISTS table_json (id INT, c1 VARCHAR(300));
INSERT INTO table_json VALUES(0, '{"key1": 123, "key2": true, "key3": null, "key4": [456, false, null, {"key1": true, "key2":
789, "key3": {"key6": 123}}, [10, false, null]], "key5": {"key1": true, "key2": 789, "key3": null}}');

SELECT JSON_EXISTS(JSON(c1), '$') res FROM table_json ORDER BY id;
RES
--------------------
true

SELECT JSON_EXISTS(JSON(c1), '$.key1') res FROM table_json ORDER BY id;
RES
--------------------
true

SELECT JSON_EXISTS(JSON(c1), '$.key4[10]') res FROM table_json ORDER BY id;
RES
--------------------
false
```

JSON_QUERY

```
json_query::= JSON_QUERY "(" json_value [FORMAT JSON] "," "'" json_path "'" format_clause ")"

format_clause::= [returning_clause] [PRETTY] [wrapper_clause]

returning_clause::= RETURNING "VARCHAR(size)"

wrapper_clause::= (WITH|WITHOUT) [ARRAY] WRAPPER
```

JSON_QUERY函数将基于json_path所描述的路径对json_value进行检索，并将检索到的结果按照format_clause定义的显示选项进行封装并打印。

json_value

json_value为一个二进制json数据，可通过JSON函数获取。当json_value为NULL时，函数返回NULL。

FORMAT JSON

用于语法兼容，无实际含义，可省略。

json_path

路径表达式，为一个常量字符串，其格式定义请参考json文档中描述。

returning_clause

参见json_serialize中returning_clause描述。

PRETTY

参见json_serialize中PRETTY描述。

wrapper_clause

指定是否将检索结果封装为一个 JSON Array类型数据，其中WITH [ARRAY] WRAPPER表示封装，WITHOUT [ARRAY] WRAPPER表示不封装。

本语句省略时，默认为WITHOUT ARRAY WRAPPER不封装。

在如下两种情况中，必须指定WITH [ARRAY] WRAPPER进行封装，否则函数将返回错误。

- 函数根据当前路径表达式查询得出的结果包含多个值。
- 路径表达式末尾指定了json_function_step。

示例

```
SELECT JSON_QUERY(JSON('{"key4":-0.123,"key5":"test"}'),'$') res FROM DUAL;
RES
----------------------------------------------------------------
{"key4":-0.123,"key5":"test"}

SELECT JSON_QUERY(JSON('{"a":{"a":{"a":1}}}'), '$.."a"' WITH ARRAY WRAPPER) res FROM DUAL;
RES
----------------------------------------------------------------
[{"a":{"a":1}},{"a":1},1]

SELECT JSON_QUERY(JSON('{"1":1,"2":2,"3":3}'), '$."1"') res FROM DUAL;
RES
----------------------------------------------------------------
1

--创建用于存储json数据的表table_json
CREATE TABLE IF NOT EXISTS table_json (id INT, c1 VARCHAR(300));
INSERT INTO table_json VALUES(0, '{"key1": 123, "key2": true, "key3": null, "key4": [456, false, null, {"key1": true, "key2":
789, "key3": {"key6": 123}}, [10, false, null]], "key5": {"key1": true, "key2": 789, "key3": null}}');

SELECT JSON_QUERY(JSON(c1), '$.key4[last]') res FROM table_json ORDER BY id;
```

```
RES
----------------------------------------------------------------
[10,false,null]

SELECT JSON_QUERY(JSON(c1), '$.key4[0]' WITH WRAPPER) res FROM table_json ORDER BY id;
RES
----------------------------------------------------------------
[456]

SELECT JSON_QUERY(JSON(c1), '$[0].key4[0][0][0]' WITH WRAPPER) res FROM table_json ORDER BY id;
RES
----------------------------------------------------------------
[456]
```

JSON_SERIALIZE

```
json_serialize::= JSON_SERIALIZE "(" json_value [returning_clause] [PRETTY] [EXTENDED] ")"

returning_clause::= RETURNING "VARCHAR(size)"
```

`JSON_SERIALIZE` 函数将二进制json数据按照pretty格式或compact格式序列化为字符串，且序列化后的字符串长度不超过 `returning_clause` 定义的限制值。

`json_value` 为一个二进制json数据，可通过JSON函数获取，当其值为NULL时，函数返回NULL。

`JSON_SERIALIZE` 函数存在别名，为 `JSON_FORMAT` ，其功能与 `JSON_SERIALIZE` 完全相同。

returning_clause

该子句用于限制输出字符串的长度，可省略，则系统默认输出字符串的长度为32000。

size为一个整形常量，取值范围为[0,32000]。

当实际字符串长度超过size限制值时，函数返回错误。

PRETTY

PRETTY用于指定字符串的打印规则为pretty，不指定PRETTY时，系统默认按compact规则打印字符串。

compact即紧凑型打印规格，此规则在打印字符串时将去掉所有的规格化空格与换行。

pretty规则具体为：

1.对于JSON Array类型的数据：

- Number、String、Bool、Null类型的数组元素出现在单独的行上。
- Object或者Array类型的数组元素占多行。
- `[` 和 `]` 各占一行，位于同一个级别，所有数组元素位于同一级别，与 `[` 和 `]` 相比，缩进了一个额外的级别，每一级别缩进为两个空格。
- 在两个数组元素的换行符前，会添加一个逗号来分割数组元素。
- 空数组仍然需要换行。

2.对于JSON Object类型的数据：

- 如果对象成员中的Value是Number、String、Bool、Null类型，key和value打印在同一行。
- 如果对象成员中的Value是Object、Array类型，key和value不在同一行输出。
- `{` 和 `}` 各占一行，位于同一个级别，所有对象成员位于同一级别，与 `{` 和 `}` 相比，缩进了一个额外的级别，每一级别缩进为两个空格。
- 对象成员的key和value之间用"空格 冒号 空格"分割；
- 在两个对象成员的换行符前，会添加一个逗号来分割对象成员。
- 空对象仍然需要换行。

3.对于JSON String类型的数据：

- 无论是Object中的key还是String类型的标量值，打印时都会在前后使用双引号。
- 字符串本身中的'\b (backspace)'、'\f (form feed)'、'\n (newline, line feed)'、'\r (carriage return)'、'\t (tab, horizontal tab)'、'Double quotation mark (")'、'slash (/)'、'backslash (\)'需要被转义打印；

4.对于JSON Number类型的数据：

- number的指数大于等于0时，如果展开打印后的宽度大于40，则使用科学计数法打印，否则展开打印。例如：
  - 1e39打印结果为：1000000000000000000000000000000000000000
  - 1e40打印结果为：1E+40
- number的指数小于0时，如果展开打印后的小数部分（去掉小数部分后面的0）的宽度大于40，则使用科学计数法打印，否则展开打印。例如：
  - 12300e-43打印结果为：1.23E-39（展开打印结果为0.00000000000000000000000000000000000000123，小数点后面41位，故使用科学计数法打印）
  - 12300e-42打印结果为：0.000000000000000000000000000000000000123（小数点后面正好40位）

EXTENDED

EXTENDED关键字用于设置使用扩展模式输出JSON数据，详见JSON扩展格式输出。

示例

```
SELECT JSON_SERIALIZE(JSON('{"keyC":123, "keyabc":false, "keyA":234, "keyB":345, "key":true}') PRETTY) res FROM DUAL;
RES
----------------------------------------------------------------
{
  "key" : true,
  "keyA" : 234,
  "keyB" : 345,
  "keyC" : 123,
  "keyabc" : false
}

SELECT JSON_SERIALIZE(JSON('{"keyC":123, "keyabc":false, "keyA":234, "keyB":345, "key":true}')) res FROM DUAL;
RES
----------------------------------------------------------------
{"key":true,"keyA":234,"keyB":345,"keyC":123,"keyabc":false}

SELECT JSON_SERIALIZE(JSON('{"keyC":123, "keyabc":false, "keyA":234, "keyB":345, "key":true}') RETURNING VARCHAR(10) PRETTY)
res FROM DUAL;
YAS-00215 length of concat texts exceeds the buffer limit

-- JSON扩展模式的输入输出
SELECT JSON_SERIALIZE(JSON('{"$binary": "SGVsbG8gd29ybGQh"}' EXTENDED)) res FROM DUAL ;
RES
----------------------------------------------------------------
"48656C6C6F20776F726C6421"

SELECT JSON_SERIALIZE(JSON('{"$binary": "SGVsbG8gd29ybGQh"}' EXTENDED) EXTENDED) res FROM DUAL ;
RES
----------------------------------------------------------------
{"$binary": "SGVsbG8gd29ybGQh"}

-- JSON扩展模式输出同样受到PRETTY关键字影响
SELECT JSON_SERIALIZE(JSON('{"$binary": "SGVsbG8gd29ybGQh"}' EXTENDED) PRETTY EXTENDED) res FROM DUAL ;
RES
----------------------------------------------------------------
{
  "$binary": "SGVsbG8gd29ybGQh"
}
```

LAG

```
lag::= LAG "("expr [(RESPECT|IGNORE) NULLS] "," offset ["," default] ")" OVER "(" [query_partition_clause] order_by_clause
")"
```

LAG为窗口函数。该函数根据窗口查询返回的一系列行和游标的位置，提供对位于该位置之前的给定物理偏移量处的行的访问，并返回该行记录对应expr的值。

窗口函数不可嵌套，因此expr为除窗口函数之外的其他通用表达式，其数据类型可以是除大对象型、JSON、XMLTYPE及UDT以外的数据类型。

(RESPECT|IGNORE) NULLS

指定expr的空值是否被包括在函数的计算中，缺省为RESPECT NULLS。

offset

指定物理偏移量的表达式，offset为与expr相同的通用表达式，须为除大对象型、JSON以外可转换为NUMBER的其他类型数据，但不能为NULL。

- offset可省略，则其默认值为1。
- 如果偏移量超出表的范围，则返回default定义的默认值。

default

指定按给定偏移量无法获取到行记录时，函数返回的默认值。default为与expr相同的通用表达式，其数据类型可以是除大对象型、JSON、XMLTYPE及UDT以外的数据类型，可省略，则默认值为NULL。

当default与expr的类型不一致时，则将default向expr类型进行转换，长度取default与expr中最大长度，如果转换失败则报错。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME            DHQ             EMPLOYEE_COUNT
------- -------------------- --------------- --------------
01      华东                 Shanghai
02      华西                 Chengdu                   300
03      华南                 Guangzhou                 400
04      华北                 Beijing                   300
05      华中                 Wuhan

--返回AREA_NO排序上一行的EMPLOYEE_COUNT
SELECT LAG(employee_count) OVER(ORDER BY AREA_NO) lag1 FROM area1;
               LAG1
-------------------


               300
               400
               300

--返回AREA_NO排序上一个非空的的EMPLOYEE_COUNT
SELECT LAG(employee_count IGNORE NULLS) OVER(ORDER BY AREA_NO) lag2 FROM area1;
               LAG2
-------------------


               300
               400
               300

--返回AREA_NO排序上一个非空的的EMPLOYEE_COUNT,若为空则返回0
SELECT LAG(employee_count IGNORE NULLS,1,0) OVER(ORDER BY AREA_NO) lag3 FROM area1;
               LAG3
-------------------
               0
```

```
0
300
400
300
```

query_partition_clause|order_by_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778

--分年统计每一个机构每月和上月的收入对比情况
SELECT year,month,revenue_total curr_month,
LAG(revenue_total,1,0) OVER (PARTITION BY year ORDER BY year,month,branch) last_month
FROM finance_info;
YEAR  MONTH  CURR_MONTH  LAST_MONTH
----- ----- ----------- -----------
2001  01           2888           0
2021  01          38888           0
2021  01          28888       38888
2021  02          37778       28888
```

LAST_DAY

```
last_day::= LAST_DAY "(" expr ")"
```

LAST_DAY函数返回expr表示的日期所在月份的最后一天的日期值，返回类型为DATE，且与DATE_FORMAT参数所指定格式一致。

expr的值须为DATE、TIME、TIMESTAMP类型，或可转换为DATE类型的字符型。其中，当为TIME类型时，函数返回NULL。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT LAST_DAY(TIME(SYSTIMESTAMP)) res FROM DUAL;
RES
--------------------------------

SELECT LAST_DAY(SYSDATE+20) res FROM DUAL;
RES
--------------------------------
2021-12-31 18:14:21
```

LAST_VALUE

```
last_value::= LAST_VALUE "(" expr [(RESPECT|IGNORE) NULLS] ")" OVER "(" analytic_clause ")"
analytic_clause::= "(" [query_partition_clause] [order_by_clause [windowing_clause]] ")"
```

LAST_VALUE是一个作用于HEAP表上的窗口函数，并支持滑动窗口。该函数依据给定的窗口条件计算出窗口数据集合，并返回该集合的最后一行记录所对应的expr的值。如果最后一个值为NULL且未指定IGNORE NULLS，则函数返回 NULL。

窗口函数不可嵌套，因此expr为除窗口函数之外的其他通用表达式，其数据类型可以是除大对象型、JSON、XMLTYPE及UDT以外的数据类型。

(RESPECT|IGNORE) NULLS

指定expr的空值是否被包括在函数的计算中。默认值为RESPECT NULLS，即如果集合中的最后一个值为NULL，函数将返回NULL。

如果指定IGNORE NULLS，则函数返回集合中的最后一个非空值，当集合中所有值都为空时，返回NULL。此设置对于数据密集化很有用。

示例（HEAP表）

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME            DHQ             EMPLOYEE_COUNT
------- -------------------- --------------- --------------
01      华东                 Shanghai
02      华西                 Chengdu                    300
03      华南                 Guangzhou                  400
04      华北                 Beijing                    300
05      华中                 Wuhan

--返回按AREA_NO排序的每个窗口中的最后一个EMPLOYEE_COUNT,windowing_clause省略时的默认窗口为UNBOUNDED PRECEDIN至CURRENT ROW
SELECT LAST_VALUE(employee_count) OVER(ORDER BY AREA_NO) res FROM area1;
                 RES
--------------------

                 300
                 400
                 300


--返回按AREA_NO排序的每个窗口中的最后一个非空EMPLOYEE_COUNT,windowing_clause省略时的默认窗口为UNBOUNDED PRECEDIN至CURRENT ROW
SELECT LAST_VALUE(employee_count IGNORE NULLS) OVER(ORDER BY AREA_NO) res FROM area1;
         RES
------------

         300
         400
         300
         300
```

analytic_clause

窗口函数的通用语法。

在windowing_clause语句中无论指定的是ROWS还是RANGE，LAST_VALUE都将对参数列进行排序，以保证在指定RANGE的情况下本函数返回值的稳定性（当order_by_clause具有相同排名，则函数返回相同排名下任意一行的值都是合理的，由此产生了返回值的不稳定性），但ROWS情况下仍会存在此不稳定性。

进行排序的参数列为：query_partition_clause中的expr 、order_by_clause中的排序列和LAST_VALUE函数的参数expr。

示例（HEAP表）

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
```

```
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778

--按年分组，并在每组记录中排名该年收入最低的机构
SELECT year,revenue_total,
LAST_VALUE(branch) OVER (PARTITION BY year ORDER BY revenue_total DESC) fr
FROM finance_info;
YEAR  REVENUE_TOTAL FR
----- ------------- -----
2001           2888 0201
2021          38888 0101
2021          37778 0101
2021          28888 0201

--每月至今收入最低的机构
SELECT year,month,revenue_total,
LAST_VALUE(branch) OVER (ORDER BY revenue_total DESC ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) fr
FROM finance_info;
YEAR  MONTH REVENUE_TOTAL FR
----- ----- ------------- -----
2021  01            38888 0101
2021  02            37778 0101
2021  01            28888 0201
2001  01             2888 0201

--每月在三个月内收入最低的机构
SELECT year,month,revenue_total,
LAST_VALUE(branch) OVER (ORDER BY revenue_total DESC ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) fr
FROM finance_info;
YEAR  MONTH REVENUE_TOTAL FR
----- ----- ------------- -----
2021  01            38888 0101
2021  02            37778 0201
2021  01            28888 0201
2001  01             2888 0201
```

LEAD

```
lead::= LEAD "(" expr [(RESPECT|IGNORE) NULLS] "," offset "," default ")" OVER "(" [query_partition_clause] order_by_clause
")"
```

LEAD为窗口函数。该函数根据窗口查询返回的一系列行和游标的位置，提供对超出该位置的给定物理偏移量的行的访问，并返回该行记录对应expr的值。

窗口函数不可嵌套，因此expr为除窗口函数之外的其他通用表达式，其数据类型可以是除大对象型、JSON、XMLTYPE及UDT以外的数据类型。

(RESPECT|IGNORE) NULLS

指定expr的空值是否被包括在函数的计算中，缺省为RESPECT NULLS。

offset

指定物理偏移量的表达式，须为除大对象型、JSON以外可转换为NUMBER的其他类型数据。

offset为与expr相同的通用表达式，但不能为NULL。

offset可省略，则其默认值为1。

如果偏移量超出表的范围，则返回default定义的默认值。

default

指定按给定偏移量无法获取到行记录时，函数返回的默认值。default为与expr相同的通用表达式，其数据类型可以是除大对象型、JSON、XMLTYPE及UDT以外的数据类型，可省略，则默认值为NULL。

当default与expr的类型不一致时，则将default向expr类型进行转换，长度取default与expr中最大长度，如果转换失败则报错。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME            DHQ             EMPLOYEE_COUNT
------- -------------------- --------------- --------------
01      华东                 Shanghai
02      华西                 Chengdu                   300
03      华南                 Guangzhou                 400
04      华北                 Beijing                   300
05      华中                 Wuhan

--返回AREA_NO排序下一行的EMPLOYEE_COUNT
SELECT LEAD(employee_count) OVER(ORDER BY AREA_NO) res FROM area1;
         RES
------------
         300
         400
         300



--返回AREA_NO排序下一个非空的的EMPLOYEE_COUNT
SELECT LEAD(employee_count IGNORE NULLS) OVER(ORDER BY AREA_NO) res FROM area1;
         RES
------------
         300
         400
         300



--返回AREA_NO排序下一个非空的的EMPLOYEE_COUNT,若为空则返回0
SELECT LEAD(employee_count IGNORE NULLS,1,0) OVER(ORDER BY AREA_NO) res FROM area1;
         RES
```

```
------------
         300
         400
         300
           0
           0
```

query_partition_clause|order_by_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778

--分年统计每一个机构每月和下月的收入对比情况
SELECT year,month,revenue_total curr_month,
LEAD(revenue_total,1,0) OVER (PARTITION BY year ORDER BY year,month,branch) next_month
FROM finance_info;
YEAR  MONTH  CURR_MONTH  NEXT_MONTH
----- ----- ----------- -----------
2001  01           2888           0
2021  01          38888       28888
2021  01          28888       37778
2021  02          37778           0
```

LEAST

```
least::= LEAST "(" (expr) {"," (expr)} ")"
```

LEAST从多个expr参数值中找出最小的一个值，并按第一个参数的数据类型返回结果。其规则为：

- 当expr值全为数值型数据时，按大小进行比较。
- 当第一个expr值为数值型数据，且存在一部分为字符型数据时，将字符型转换为NUMBER类型后执行比较，如转换失败则返回Invalid number错误。
- 当expr值全为字符型数据时，按字典序进行比较。
- 当第一个expr值为字符型数据，且存在一部分为非字符型数据时，会把非字符型转换成字符型，然后按字典序进行比较。
- 当expr值全为日期时间型数据时，按照时间大小进行比较。
- 当expr值全为布尔型数据时，按照true=1，false=0进行比较。
- 当参数列表中的任一个expr值为NULL时，函数返回NULL。
- 当expr为LOB、XMLTYPE、JSON时，函数返回unexpected lob错误。
- 上述之外的其他情形，函数返回Invalid datatype错误。

示例

```
SELECT LEAST(2, 5, 12, 3, 16, 8, 9) AS RES1,
LEAST('A', 6, 7, 5000, 'E', 'F','G') AS RES2,
LEAST(SYSDATE,TO_DATE('2014-08-01','YYYY-MM-DD')) AS RES3,
LEAST(true,false) RES4
FROM DUAL;
        RES1 RES2  RES3                           RES4
------------ ----- ------------------------------ --------------------
           2 5000  2014-08-01 00:00:00            false
```

LEFT

```
left::= LEFT "(" expr "," length ")"
```

LEFT函数将expr的值从左边截取指定长度，得到一个子字符串并将其返回。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr为NCLOB/NCHAR/NVARCHAR时返回值为NVARCHAR类型。其余场景返回值为VARCHAR类型。

length

指定字符串截取的长度，length为与expr相同的通用表达式，须为除BIT外数值型数据，或可转换为NUMBER类型的其他类型数据，取值范围[-2147483648,2147483647]。

- 当length的值为NULL，0或负数时，函数返回NULL。
- 当length值为小数时，函数将先对其进行取整，规则如下：
  - length为NUMBER型或者浮点型时：四舍五入取整。
  - length为可转换为NUMBER的其他类型时：截取整数部分。
- 当length值大于expr字符串长度时，则将其按expr字符串长度值处理。

示例

```
SELECT LEFT(SYSDATE+1,4.99) res FROM DUAL;
RES
----------------------------------------------------------------
2022-

SELECT LEFT(SYSDATE+1,'4.99') res FROM DUAL;
RES
----------------------------------------------------------------
2022
```

LENGTH LENGTHB

```
length::= LENGTH "(" expr ")"

lengthb::= LENGTHB "(" expr ")"
```

LENGTH/LENGTHB函数统计expr的值的长度，返回一个BIGINT的数值。

LENGTH按字符统计长度，与CHAR_LENGTH和CHARACTER_LENGTH函数同义。

LENGTHB按字节统计长度，与OCTET_LENGTH函数同义。对于中文字符，不同的字符集环境可能返回不同的结果，例如，在UTF8字符集环境中，一个中文字符占3字节，而在GBK字符集环境中，一个中文字符占2字节。

expr的值须为字符型，或可转化为字符型的其他类型。当expr的值为NULL时，函数返回NULL。

函数中的expr不允许为BIT和XMLTYPE类型。

对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。

示例

```
--UTF8字符集环境
SELECT LENGTH('深圳') r1,LENGTHB('深圳') r2,
LENGTH('aabbccDDee') r3,LENGTHB('aabbccDDee') r4,
LENGTH(null) r5,LENGTHB(null) r6
FROM DUAL;
   R1   R2    R3    R4    R5    R6
----- ---- ----- ----- ----- -----
    2    6    10    10

--GBK字符集环境
SELECT LENGTH('深圳') r1,LENGTHB('深圳') r2
FROM DUAL;
                   R1                    R2
--------------------- ----------------------
                    2                     4
```

LENGTH2

```
length2::= LENGTH2 "(" expr ")"
```

LENGTH2函数统计expr的值的长度，expr的值须为字符型，或可转化为字符型的其他类型，但不能为BIT、XMLTYPE、LOB类型。支持以UTF16编码计算长度，返回一个BIGINT的数值。

LENGTH2按字符统计长度，与CHAR_LENGTH和CHARACTER_LENGTH函数同义。

本函数遵循如下规则：

- 当expr的值为NULL时，函数返回NULL。
- 该函数不支持列式计算。

示例（HEAP表）

```
--UTF8字符集环境
SELECT LENGTH2('深圳') r1,LENGTH('深圳') r2,
LENGTH2('aabbccDDee') r3,LENGTH('aabbccDDee') r4,
LENGTH2(null) r5,LENGTH(null) r6
FROM DUAL;
   R1   R2    R3    R4    R5    R6
----- ---- ----- ----- ----- -----
    2    2    10    10

--GBK字符集环境
SELECT LENGTH2('深圳') r1,LENGTH('深圳') r2
FROM DUAL;
                   R1                    R2
--------------------- ----------------------
                    2                     2

--LENGTH与LENGTH2的区别
SELECT LENGTH(TO_CHAR('☕')) FROM DUAL;

LENGTH(TO_CHAR('☕'))
----------------------
                     1

SELECT LENGTH2(TO_CHAR('☕')) FROM DUAL;

LENGTH2(TO_CHAR('☕')
----------------------
                     2
```

LISTAGG

```
listagg::= LISTAGG "(" [ALL|DISTINCT] expr ["," separator] [listagg_overflow_clause] ")"
  [WITHIN GROUP order_by_clause] [OVER query_partition_clause]
```

LISTAGG函数将多行的expr执行拼接操作，并通过分隔符分隔，返回一行VARCHAR/RAW类型的字符串。

expr不能为BIT、BOOLEAN、NCHAR、NVARCHAR、NCLOB和UDT类型。

当expr的类型为RAW时，函数返回RAW类型，否则函数返回VARCHAR类型。

DISTINCT

计算最终拼接结果时，过滤在同一组内出现的重复的行。

ALL

默认值，表示不过滤重复的行，对所有行都进行拼接。

separator

用于定义分隔符。不指定separator时，默认分隔符为NULL。

separator可以为常量（含常量表达式）或绑定参数的变量表达式，其数据类型不能为BOOLEAN、BIT、CLOB、BLOB、JSON、UDT、XMLTYPE和NCLOB。

当expr为RAW类型时，separator必须为RAW（或可以隐式转换为RAW）类型。

当expr为NULL时，该行数据会被忽略。

listagg_overflow_clause

```
listagg_overflow_clause::= (on overflow error | on overflow truncate ["'" text "'"] [(with | without) count])
```

当终止符或分隔符长度超过8000或者（分隔符长度）+ （终止符长度）+ （withcount26个字节）超过8000， 直接报错。

当拼接的结果发生溢出时候（即拼接的expr+分隔符>8000时）进行特殊处理：

- on overflow error：默认选项，发生溢出时直接报错。
- on overflow truncate(text)：text是指定的终止标志（即终止符），在发生溢出后需要预留位置用来放置终止标志，默认终止标志为'...'占3个字节。
  - with count：发生溢出时且使用truncate时，默认输出count。在发生溢出后需要预留26个字节（给count预留24个字符，括号预留2个字符）。将已经拼接的行数往回回滚到可以放置需要给"（count）"和终止标志预留的字节数，将最终溢出的行数赋值给count。使用方法例如on overflow truncate '*' with count。
  - without count：在截断后面不显示截断的行数，发生溢出时需要为终止标志预留位置，不需要为"（count）"预留位置。例如on overflow truncate '*' without count。

指定终止符退化成默认的终止符的情况：

- 如果是without count，当终止符或分隔符长度超过4000或者终止符+分隔符长度超过4000，终止符退化成'...'显示，并且不显示分隔符；如果前面条件不成立，那么会显示可以显示的行数+分隔符+终止符（可显示的行数可能为0）
- 如果是with count或者缺省时，当终止符或分隔符长度超过4000-2-24 = 3974或者终止符+分隔符长度超过3974， 显示count，终止符退化成'...'显示，并且不显示分隔符；如果前面条件不成立，那么会显示可以显示的行数+分隔符+终止符+count（可显示的行数可能为0）

within group order_by_clause

对组内需要拼接的数据进行排序后再拼接。

作为非窗口函数时，本关键字仅适用于HEAP表。

示例（HEAP表）

```
--创建exprs表，并插入数据
DROP TABLE IF EXISTS exprs;
```

```
CREATE TABLE exprs(id INT , name VARCHAR(8000) , class RAW(300));
INSERT INTO exprs VALUES(1,'carrot','012');
INSERT INTO exprs VALUES(3,'rabbit','0ea');
INSERT INTO exprs VALUES(3,'carrot','66734');
INSERT INTO exprs VALUES(2,'clion','012');
INSERT INTO exprs VALUES(3,'apple','12900');
COMMIT;

--普通的数据拼接
SELECT LISTAGG(name) res FROM exprs;
RES
----------------------------------------------------------------
carrotrabbitcarrotclionapple

SELECT LISTAGG(class,'00a') res FROM exprs;
RES
----------------------------------------------------------------
0012000A00EA000A066734000A0012000A012900

--使用DISTINCT关键字去重
SELECT LISTAGG(DISTINCT name ,' ;') res FROM exprs;
RES
----------------------------------------------------------------
apple ;carrot ;clion ;rabbit

--使用within group(order by)进行行内排序
SELECT LISTAGG(name,';') within GROUP(ORDER BY name) FROM exprs;
NAME
----------------------------------------------------------------
apple;carrot;carrot;clion;rabbit

--使用group by进行分组
SELECT id,LISTAGG(name,' ') within GROUP(ORDER BY name) FROM exprs GROUP BY id;
          ID NAME
------------ ----------------------------------------------------------------
           1 carrot
           3 apple carrot rabbit
           2 clion

-- 多插入几行数据使能发生溢出情况
DELETE exprs;
INSERT INTO exprs(name) VALUES(LPAD('as',4000,'cad'));
INSERT INTO exprs(name) VALUES(LPAD('dj',3000,'cad'));
INSERT INTO exprs(name) VALUES(LPAD('sd',500,'cdua'));
INSERT INTO exprs(name) VALUES(LPAD('sss',200,'cpd'));
INSERT INTO exprs(name) VALUES(LPAD('asiufs',100,'cod'));
INSERT INTO exprs(name) VALUES(LPAD('affs',300,'cdd'));

--发生溢出时候的操作1：直接报错
SELECT LISTAGG(name,';' ON overflow error) FROM exprs;
YAS-02511 result of string concatenation value exceeds maximum length of 8000 characters

--发生溢出时候的操作2：截断
SELECT LISTAGG(name,';' ON overflow TRUNCATE) res FROM exprs;
--省略部分输出
......ss;codcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcasiufs;...(1)
```

示例（TAC表、LSC表）

```
-- 创建exprs表，并插入数据
DROP TABLE IF EXISTS exprs;
CREATE TABLE exprs(id INT , name VARCHAR(8000) , class RAW(300));
INSERT INTO exprs VALUES(1,'carrot','012');
INSERT INTO exprs VALUES(3,'rabbit','0ea');
INSERT INTO exprs VALUES(3,'carrot','66734');
INSERT INTO exprs VALUES(2,'clion','012');
INSERT INTO exprs VALUES(3,'apple','12900');
COMMIT;
```

```
-- 普通的数据拼接
SELECT LISTAGG(name) res FROM exprs;
RES
----------------------------------------------------------------
carrotrabbitcarrotclionapple

SELECT LISTAGG(class,'00a') res FROM exprs;
RES
----------------------------------------------------------------
0012000A00EA000A066734000A0012000A012900

-- 使用DISTINCT关键字去重
SELECT LISTAGG(DISTINCT name ,' ;') res FROM exprs;
RES
----------------------------------------------------------------
apple ;carrot ;clion ;rabbit

-- 多插入几行数据使能发生溢出情况
DELETE exprs;
INSERT INTO exprs(name) VALUES(LPAD('as',4000,'cad'));
INSERT INTO exprs(name) VALUES(LPAD('dj',3000,'cad'));
INSERT INTO exprs(name) VALUES(LPAD('sd',500,'cdua'));
INSERT INTO exprs(name) VALUES(LPAD('sss',200,'cpd'));
INSERT INTO exprs(name) VALUES(LPAD('asiufs',100,'cod'));
INSERT INTO exprs(name) VALUES(LPAD('affs',300,'cdd'));

-- 发生溢出时候的操作1：直接报错
SELECT LISTAGG(name,';' ON overflow error) FROM exprs;
YAS-05005 result of string concatenation value exceeds maximum length of 8000 characters

-- 发生溢出时候的操作2：截断
SELECT LISTAGG(name,';' ON overflow TRUNCATE) res FROM exprs;
-- 省略部分输出
......ss;codcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcodcasiufs;...(1)
```

OVER

当指定OVER关键字时，LISTAGG将作为窗口函数，将每个分组窗口内的行进行拼接，分组窗口内每一行的拼接结果都相同。

query_partition_clause

窗口函数通用语法。

示例

```
-- 创建exprs表，并插入数据
DROP TABLE IF EXISTS exprs;
CREATE TABLE exprs(id INT , name VARCHAR(8000) , class RAW(300));
INSERT INTO exprs VALUES(1,'carrot','012');
INSERT INTO exprs VALUES(3,'rabbit','0ea');
INSERT INTO exprs VALUES(3,'carrot','66734');
INSERT INTO exprs VALUES(2,'clion','012');
INSERT INTO exprs VALUES(3,'apple','12900');
COMMIT;

-- 查询并拼接
SELECT id,name,
LISTAGG(name, ',') OVER (PARTITION BY id) c1,
MAX(name) OVER (PARTITION BY id ORDER BY name) c2
FROM exprs;
 ID NAME       C1                    C2
--- ---------- --------------------- ---------
  1 carrot     carrot                carrot
  2 clion      clion                 clion
  3 apple      rabbit,carrot,apple   apple
  3 carrot     rabbit,carrot,apple   carrot
  3 rabbit     rabbit,carrot,apple   rabbit
```

LN

```
ln::= LN "(" expr ")"
```

LN函数用于计算expr的自然对数，返回一个DOUBLE类型的数值。

expr的值为须为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

基于自然对数的数学概念，expr的值应该为一个正数。下表列示函数对非正数和一些特殊值的返回规则：

expr	LN(expr)
负数	Nan
0	-Inf
Inf	Inf
-Inf	Nan
Nan	Nan
-Nan	Nan

示例

```
SELECT LN(2) res FROM DUAL;
RES
-----------
6.931E-001

CREATE TABLE number_fd(numberf FLOAT, numberd DOUBLE);
INSERT INTO number_fd VALUES(0,-5.55);
INSERT INTO number_fd VALUES('Inf','-Inf');
INSERT INTO number_fd VALUES('Nan','-Nan');

SELECT LN(numberf) res1,LN(numberd) res2 FROM number_fd;
       RES1        RES2
----------- -----------
       -Inf         Nan
        Inf         Nan
        Nan         Nan
```

LNNVL

```
lnnvl::= LNNVL "(" condition ")"
```

LNNVL函数用于计算条件，尤其适用于计算存在操作数为NULL的条件。

该函数通常在WHERE语句中使用，也可以作为搜索表达式中的条件，在这类语句中，普通的条件判断对NULL操作数返回false或者未知，从而导致空数据被过滤，使用LNNVL函数则可以解决这个问题，减少用户还需进行IS NULL判断的操作处理。

condition

YashanDB支持的条件语句，具体参考通用SQL语法condition。

当condition的结果为false或者未知时，函数返回true；当condition的结果为true时，函数返回false。

示例

```
--LNNVL函数使用效果
SELECT LNNVL(1 IN (2,3)) res FROM DUAL;
RES
--------------------
true

SELECT LNNVL('' IS NOT NULL) res FROM DUAL;
RES
--------------------
true

--普通条件对NULL判断返回false，从而过滤掉空数据
--area1表共有5条数据，EMPLOYEE_COUNT为空的业务含义为0
SELECT * FROM area1;
AREA_NO AREA_NAME                DHQ                    EMPLOYEE_COUNT
------- ------------------------ ---------------------- --------------
01      华东                     Shanghai
02      华西                     Chengdu                             300
03      华南                     Guangzhou                           400
04      华北                     Beijing                             300
05      华中                     Wuhan

--当统计EMPLOYEE_COUNT值小于400的区域数时，由于空值被过滤导致结果与实际业务不符，除非还进行IS NULL判断
SELECT COUNT(1) res FROM area1 WHERE employee_count<400;
                   RES
----------------------
                     2

SELECT COUNT(1) res FROM area1 WHERE employee_count<400 OR employee_count IS NULL;
                   RES
----------------------
                     4

--使用LNNVL函数可以替代上条语句
SELECT COUNT(1) res FROM area1 WHERE LNNVL(employee_count>=400);
                   RES
----------------------
                     4
```

LOCALTIME

```
localtime::= LOCALTIME ["(" [integer] ")"]
```

LOCALTIME函数返回数据库所在的操作系统设置的当前时间戳，其返回类型为TIMESTAMP，且与TIMESTAMP_FORMAT参数所指定格式一致。

该函数不支持列式计算。

LOCALTIME有以下三种形式：

- `LOCALTIME`
- `LOCALTIME()`
- `LOCALTIME(integer)`，integer必须为一个0~9之间的整数字面量，表示保留的微秒位数，舍去的位按四舍五入。

如果一个SQL语句中出现了多个LOCALTIME函数，在该语句执行过程中将只调用一次函数，即保证多个LOCALTIME函数返回的是相同一个时间戳值。

当LOCALTIME函数参与运算时，其运算规则与TIMESTAMP类型一致，见算术运算符章节描述。

示例

```
SELECT LOCALTIME res1,LOCALTIME() res2,LOCALTIME(9) res3 ,LOCALTIME+1 res4 FROM DUAL;
RES1                                                            RES2
RES3                                                            RES4
--------------------------------------------------------------- ----------------------------------------------------------------
---- ---------------------------------------------------------- ------------------------------
2023-05-03 20:35:30.152249                                      2023-05-03 20:35:30.152249
2023-05-03 20:35:30.152249                                      2023-05-04
```

LOCALTIMESTAMP

```
localtimestamp::= LOCALTIMESTAMP ["(" [integer] ")"]
```

LOCALTIMESTAMP函数返回数据库所在的操作系统设置的当前时间戳，其返回类型为TIMESTAMP，且与TIMESTAMP_FORMAT参数所指定格式一致。

该函数不支持列式计算。

LOCALTIMESTAMP有以下三种形式：

- `LOCALTIMESTAMP`
- `LOCALTIMESTAMP()`
- `LOCALTIMESTAMP(integer)`，integer必须为一个0~9之间的整数字面量，表示保留的微秒位数，舍去的位按四舍五入。

如果一个SQL语句中出现了多个LOCALTIMESTAMP函数，在该语句执行过程中将只调用一次函数，即保证多个LOCALTIMESTAMP函数返回的是相同一个时间戳值。

当LOCALTIMESTAMP函数参与运算时，其运算规则与TIMESTAMP类型一致，见算术运算符章节描述。

示例

```
SELECT LOCALTIMESTAMP res1,LOCALTIMESTAMP() res2,LOCALTIMESTAMP(9) res3,LOCALTIMESTAMP+1 res4 FROM DUAL;
RES1                                                                RES2
RES3                                                                RES4
------------------------------------------------------------------- ------------------------------------------------------------------
---- ---------------------------------------------------------------- -------------------------------
2023-05-03 20:38:06.665584                                          2023-05-03 20:38:06.665584
2023-05-03 20:38:06.665584                                          2023-05-04
```

LOG

```
log::= LOG "(" expr1 "," expr2 ")"
```

LOG函数计算expr2以expr1为底的对数，返回一个DOUBLE类型的数值。

expr1和expr2均为YashanDB认可的通用表达式，其值须为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr1或者expr2的值为NULL时，函数返回NULL。

根据对数的数学概念，expr1应该为除0和1以外的正数，expr2应该为任意一个正数，除此之外的其他情况函数处理规则见下表：

expr2值\expr1值	非0/1正数	-Inf	负数	0	1	Inf	Nan	-Nan
正数	对数	Nan	Nan	0	Inf	0	Nan	Nan
-Inf	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
负数	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
0	-Inf	Nan	Nan	Nan	-Inf	Nan	Nan	Nan
Inf	Inf	Nan	Nan	Nan	Inf	Nan	Nan	Nan
Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
-Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan

示例

```
SELECT LOG(2,4) res FROM DUAL;
RES
-----------
2.0E+000

SELECT LOG(b'10',b'100') res FROM DUAL;
RES
-----------
2.0E+000

DROP TABLE IF EXISTS number_fd;
CREATE TABLE number_fd(numberf FLOAT, numberd DOUBLE);
INSERT INTO number_fd VALUES(0,5.55);
INSERT INTO number_fd VALUES(1,5.55);
INSERT INTO number_fd VALUES(2,-5.55);
INSERT INTO number_fd VALUES(2,0);
INSERT INTO number_fd VALUES('2','Inf');
INSERT INTO number_fd VALUES('2','-Inf');
INSERT INTO number_fd VALUES('2','Nan');
INSERT INTO number_fd VALUES('2','-Nan');

SELECT LOG(numberf,numberd) res1, LOG(numberd,numberf) res2 FROM number_fd;
       RES1        RES2
----------- -----------
          0        -Inf
        Inf           0
        Nan         Nan
       -Inf           0
        Inf           0
        Nan         Nan
        Nan         Nan
        Nan         Nan
```

LOWER

```
lower::= LOWER "(" expr ")"
```

LOWER函数将expr的值中的大写字母转换为小写，返回一个新字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr为CHAR、NCHAR、NVARCHAR类型时，返回值为CHAR、NCHAR、NVARCHAR类型，其余场景返回值为VARCHAR类型。

示例

```
SELECT LOWER('深圳NIHAO') l1, LOWER(NULL) l2 FROM DUAL;
L1            L2
------------- -----
深圳nihao
```

LPAD

```
lpad::= LPAD "(" expr "," pad_length ["," pad_character] ")"
```

LPAD函数从左边对expr的值进行指定字符、指定长度的填充，得到一个新字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr为NCLOB、NCHAR、NVARCHAR类型时，返回值为NVARCHAR类型，其余场景返回值为VARCHAR类型。

pad_length

该值指定了填充之后字符串的长度，pad_length为与expr相同的通用表达式，须为数值型数据，或可转换为NUMBER类型的其他类型数据，取值范围[-9223372036854775808,32000]。

- 当pad_length的值为NULL，0或负数时，函数返回NULL。
- 当pad_length的值为小数时，函数截断其小数位保留整数位。
- 当此长度小于等于expr字符串长度时，函数返回的是从左到右对expr进行此长度截取的子字符串。

pad_character

该值指定了要填充的内容，pad_character为与expr相同的通用表达式，须为字符型数据，或可转换为字符型的其他类型数据。

- 当pad_character的值为NULL时，函数返回NULL。
- 指定pad_character时，本函数将循环从左至右读取pad_character的字符并填充到expr的左边，直到满足pad_length的长度要求为止。
- 未指定pad_character时，默认填充空格。

示例

```
SELECT LPAD('深圳NIHAO',16,'你好') AS res FROM DUAL;
RES
----------------------------------------
你好你好你好你好你深圳NIHAO

SELECT LPAD('深圳NIHAO',1,'你好') AS res FROM DUAL;
RES
-----
深

SELECT LPAD('深圳NIHAO',16) AS res1 FROM DUAL;
RES1
--------------------
         深圳NIHAO

SELECT LPAD('深圳NIHAO',3.5) AS res1 FROM DUAL;
RES1
---------
深圳N

SELECT LPAD('深圳NIHAO',-3.5) AS res1 FROM DUAL;
RES1
-----
```

LTRIM

```
ltrim::= LTRIM "(" expr ["," trim_character] ")"
```

LTRIM函数从左往右删除expr的值里与trim_character匹配的内容，得到一个新的子字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型，不允许为NCLOB类型。

- 当expr为CLOB类型时，返回值为CLOB类型，当expr为NCAHR/NVARCHAR时，返回值为NVARCHAR类型，其余返回值为VARCHAR类型。
- 对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。
- 当expr的值为NULL时，函数返回NULL。

trim_character

指定要匹配的内容，trim_character为与expr相同的通用表达式，须为字符型，或可转换为字符型的其他类型。

- 当trim_character的值为NULL时，函数返回NULL。
- 指定trim_character时，函数将会从左往右对比expr与trim_character中的字符，当expr中的字符在trim_character中则将其删除，直到遇到从左往右的第一个不在trim_character中的字符后停止。
- 未指定trim_character时，默认的匹配内容为一个空格。

示例

```
--从左开始的3个'3'均能在'33'中找到，因此被删除，直到遇到'1'不能在'33'中找到，停止匹配并返回剩余字符串
SELECT LTRIM('33311333','33') res FROM DUAL;
RES
--------------------
11333

--从左开始的所有字符均能在'313'中找到，全部被删除，返回空字符串
SELECT LTRIM('33311333','313') res FROM DUAL;
RES
--------------------

--只有一个参数时按空格执行匹配删除
SELECT LTRIM(' 11333') res FROM DUAL;
RES
--------------
11333
```

MAX

```
max::= MAX "(" [DISTINCT|ALL] expr ")" [OVER "(" analytic_clause ")"]

analytic_clause::= "(" [query_partition_clause] [order_by_clause [windowing_clause]] ")"
```

MAX函数计算给定参数expr的最大值。

当USE_NATIVE_TYPE为TRUE时其返回值类型与参数的类型一致；当USE_NATIVE_TYPE为FALSE时，除FLOAT类型返回为NUMBER外，其余返回值类型与参数的类型一致。

在单行计算中，当expr的值为NULL时，函数返回NULL。

在多行计算中，函数将忽略expr值为空的行，当所有行均为空时，计算结果为NULL。

聚集函数不可嵌套，因此expr为除聚集函数之外的其他通用表达式，其数据类型可以是除布尔型、大对象型、JSON、XMLTYPE及UDT以外的数据类型。

DISTINCT

表示在计算最大值时，过滤掉重复的行。

ALL

默认值，表示对所有行计算最大值。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME              DHQ           EMPLOYEE_COUNT
------- ---------------------- ------------- --------------
01      华东                   Shanghai
02      华西                   Chengdu                 300
03      华南                   Guangzhou               400
04      华北                   Beijing                 300
05      华中                   Wuhan

--计算最大的员工数量，与SELECT MAX(DISTINCT employee_count) FROM area1语句的查询结果一致
SELECT MAX(employee_count) res FROM area1;
         RES
------------
         400

SELECT MAX(true) FROM sys.dual;

[1:12]YAS-08021 invalid data type: BOOLEAN
```

OVER

当指定OVER关键字时，MAX将作为窗口函数，并支持滑动窗口，返回多行的最大值。

analytic_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
```

```
2021  01    0101          38888
2021  02    0101          37778

--分年统计每月的最高收入
SELECT year,month,
MAX(revenue_total) OVER (PARTITION BY year ORDER BY month) toall
FROM finance_info;
YEAR  MONTH       TOALL
----- ----- -----------
2001  01           2888
2021  01          38888
2021  01          38888
2021  02          38888

--分年统计每月年初至今的最高收入
SELECT year,month,
MAX(revenue_total) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info;
YEAR  MONTH       TONOW
----- ----- -----------
2001  01           2888
2021  01          28888
2021  01          38888
2021  02          38888
```

MD5

```
md5::= MD5 "(" expr ")"
```

MD5函数用于计算expr的MD5值。expr的值须为字符型，或可转换为字符型的其他类型。返回一个varchar(32)类型的固定长度的十六进制字符串。

本函数遵循如下规则：

- 当expr的值为NULL或空串时，函数返回值为NULL。
- 当expr为nchar类型或做变量的char类型时，函数会将其末尾的空格进行消除，再计算MD5值。
- 当expr为double或float类型，函数返回其科学计数法的MD5值。
- 当expr为bool类型，函数返回其对应的bool类型数据值（1和0）的MD5值。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 该函数不支持列式计算。

示例（HEAP表）

```
SELECT MD5(1) FROM DUAL;

MD5(1)
----------------------------------
c4ca4238a0b923820dcc509a6f75849b

SELECT MD5(NULL) FROM DUAL;

MD5(NULL)
---------

SELECT MD5('') FROM DUAL;

MD5('')
-------

SELECT MD5(' ') FROM DUAL;

MD5('')
----------------------------------
7215ee9c7d9dc229d2921a40e899ec5f
```

MEDIAN

```
median::= MEDIAN "(" [ALL] expr ")" OVER "(" [query_partition_clause] ")"
```

MEDIAN函数计算给定参数expr的中位数，仅支持expr类型为数值型或者日期时间型，使用其他数据类型会返回错误。

该函数不支持列式计算。

MEDIAN窗口函数不能与DISTINCT以及ORDER BY语句一起使用。

函数返回值类型：

数据类型	返回值类型
TINYINT	NUMBER
SMALLINT	NUMBER
INT	NUMBER
BIGINT	NUMBER
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMBER	NUMBER
DATE	DATE
TIMESTAMP	TIMESTAMP
TIME	TIME
INTERVAL	INTERVAL

单行计算中，当expr的值为空时函数返回NULL；expr为字面NULL时报错。expr使用绑定参数，绑定字面NULL，返回结果NULL。

多行计算中，函数将忽略expr值为空的行。当所有行均为空时，计算结果为NULL。

YashanDB仅支持MEDIAN函数作为窗口函数使用，不支持其作为聚集函数使用。

ALL

表示对所有行计算中位数。

query_partition_clause

窗口函数通用语法。

expr只支持INT、NUMBER与时间类型。partition by的参数支持除LOB、JSON、XMLTYPE、UDT外的其它数据类型。

示例（HEAP表）

```
--创建表exam并插入数据
CREATE TABLE exam(id INT,score INT);
INSERT INTO exam VALUES(1,99);
INSERT INTO exam VALUES(2,80);
INSERT INTO exam VALUES(3,80);

SELECT id, score FROM exam;
          ID        SCORE
------------ ------------
           1           99
           2           80
```

```
           3          80

SELECT MEDIAN(score) OVER () res FROM exam;
               RES
-------------------
                80
                80
                80

SELECT MEDIAN(score) OVER (PARTITION BY id) res FROM exam;
               RES
--------------------
                99
                80
                80

SELECT MEDIAN(null) OVER() FROM sys.dual;
[1:15]YAS-04322 invalid datatype
```

MIN

```
min::= MIN "(" [DISTINCT|ALL] expr ")" [OVER "(" analytic_clause ")"]

analytic_clause::= "(" [query_partition_clause] [order_by_clause [windowing_clause]] ")"
```

MIN函数计算给定参数expr的最小值。

当USE_NATIVE_TYPE为TRUE时其返回值类型与参数的类型一致；当USE_NATIVE_TYPE为FALSE时，除FLOAT类型返回为NUMBER外，其余返回值类型与参数的类型一致。

在单行计算中，当expr的值为NULL时，函数返回NULL。

在多行计算中，函数将忽略expr值为空的行，当所有行均为空时，计算结果为NULL。

聚集函数不可嵌套，因此expr为除聚集函数之外的其他通用表达式，其数据类型可以是除布尔型、大对象型、JSON、XMLTYPE及UDT以外的数据类型。

DISTINCT

表示在计算最小值时，过滤掉重复的行。

ALL

默认值，表示对所有行计算最小值。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME             DHQ               EMPLOYEE_COUNT
------- --------------------- ----------------- --------------
01      华东                  Shanghai
02      华西                  Chengdu                       300
03      华南                  Guangzhou                     400
04      华北                  Beijing                       300
05      华中                  Wuhan

--计算最小的员工数量，与SELECT MIN(DISTINCT employee_count) FROM area1语句的查询结果一致
SELECT MIN(employee_count) res FROM area1;
         RES
------------
         300

SELECT MIN(true) FROM sys.dual;

[1:12]YAS-08021 invalid data type: BOOLEAN
```

OVER

当指定OVER关键字时，MIN将作为窗口函数，并支持滑动窗口，返回多行的最小值。

analytic_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
```

```
2021  01    0101          38888
2021  02    0101          37778

--分年统计每月的最低收入
SELECT year,month,
MIN(revenue_total) OVER (PARTITION BY year ORDER BY month) toall
FROM finance_info;
YEAR  MONTH       TOALL
----- ----- -----------
2001  01           2888
2021  01          28888
2021  01          28888
2021  02          28888

--分年统计每月年初至今的最低收入
SELECT year,month,
MIN(revenue_total) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info;
YEAR  MONTH       TONOW
----- ----- -----------
2001  01           2888
2021  01          28888
2021  01          28888
2021  02          28888
```

MOD

```
mod::= MOD "(" expr1 "," expr2 ")"
```

MOD为取模函数，与算术运算符算法一致。

在算术运算时，YashanDB通过隐式数据转换，将参与运算的数据类型统一到某个数据类型，并按此数据类型返回运算结果，具体规则请参考算术运算符里的数据类型描述。

expr1、expr2的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr1或expr2中任一值为NULL时，函数返回NULL。

示例

```
SELECT * FROM numbers_nobit;
 NUMBERA NUMBERB  NUMBERC               NUMBERD     NUMBERE     NUMBERF   NUMBERG
-------- ------- -------- --------------------- ----------- ----------- ---------
      -5      55     5555   5555555555555555555  5.555E+000  5.556E+000       555


SELECT MOD(numberb,numbera) mod1,
MOD(numberd,numbere) mod2,
MOD(numberf,numberc) mod3,
MOD(numbera,'17') mod4,
MOD(numberf,0) mod5
FROM numbers_nobit;
                MOD1        MOD2        MOD3        MOD4        MOD5
-------------------- ----------- ----------- ----------- -----------
                   0  3.794E+000  5.556E+000          -5  5.556E+000

SELECT TYPEOF(MOD(numberb,numbera)) type1,
TYPEOF(MOD(numberd,numbere)) type2,
TYPEOF(MOD(numberf,numberc)) type3,
TYPEOF(MOD(numbera,'17')) type4,
TYPEOF(MOD(numberf,0)) type5
FROM numbers_nobit;
TYPE1     TYPE2     TYPE3     TYPE4     TYPE5
--------- --------- --------- --------- ---------
bigint    float     double    number    double
```

MONTHS_BETWEEN

```
MONTHS_BETWEEN::= MONTHS_BETWEEN "(" expr1 "," expr2 ")"
```

MONTHS_BETWEEN函数用于计算expr1和expr2之间间隔的满月月份数，和不满月的小数部分（expr1-expr2），返回一个NUMBER类型的数值。

expr1和expr2为YashanDB认可的通用表达式，其值须为DATE/TIMESTAMP类型，或可以转换为DATE/TIMESTAMP的字符型。

expr1或者expr2为NULL时，函数返回NULL。

示例

```
--日期相同，计算结果为整数
SELECT MONTHS_BETWEEN('2021-03-13', '2021-02-13') res FROM DUAL;
        RES
-----------
          1

--均为所在月最后一天，计算结果为整数
SELECT MONTHS_BETWEEN('2021-04-30', '2021-01-31') res FROM DUAL;
        RES
-----------
          3

--间隔不为整月，计算结果带小数
SELECT MONTHS_BETWEEN('2021-03-01', '2021-02-28') res FROM DUAL;
        RES
-----------
 .129032258

--间隔不为整月，计算结果带小数
SELECT MONTHS_BETWEEN('2021-04-01', '2021-02-28') res FROM DUAL;
        RES
-----------
 1.12903226
```

NEXT_DAY

```
NEXT_DAY::= NEXT_DAY "(" expr1 "," expr2 ")"
```

NEXT_DAY函数根据expr2的值，计算expr1表示的日期的下一个星期几，返回一个DATE类型数值。

expr1

YashanDB认可的通用表达式，其值须为DATE/TIMESTAMP类型，或可转换为TIMESTAMP类型的字符型。

当expr1的值为NULL时，函数返回NULL。

函数从expr1表示日期的第二天开始计数，获得下一个星期几的时间，且计算结果的时分秒与expr1的时分秒部分一致。

expr2

YashanDB认可的通用表达式，其值须为数值型或字符型，对于非整数的数值型数据，函数对其进行向下取整。

当expr2的值为NULL时，函数返回NULL。

当为数值型时，expr2的值范围为`[1,8)`，超出范围报错。

当为字符型时，expr2的值只能为如下列示字符串中的某一个（不区分大小写，且函数将忽略字符串右侧空格）：

- 星期缩写mon、tue、wed、thu、fri、sat、sun。
- 星期全称Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sunday。

示例

```
--对空值返回空
SELECT NEXT_DAY(null,null) res FROM DUAL;
RES
--------------------------------

--从2022-09-23 07:00:00的第二天开始计算下一个星期五
SELECT TO_DATE('2022-09-23 07:00:00','yyyy-mm-dd hh24:mi:ss') "cur-year-month-day",
TO_CHAR(TO_DATE('2022-09-23 07:00:00','yyyy-mm-dd hh24:mi:ss'),'yyyy-mm-w-day') "cur-year-month-week-day",
NEXT_DAY(TO_DATE('2022-09-23 07:00:00','yyyy-mm-dd hh24:mi:ss'),6) "year-month-day",
TO_CHAR(NEXT_DAY(TO_DATE('2022-09-23 07:00:00','yyyy-mm-dd hh24:mi:ss'),6),'yyyy-mm-w-day') "year-month-week-day"
FROM DUAL;
cur-year-month-day      cur-year-month-week-day    year-month-day         year-month-week-day
----------------------- -------------------------- ---------------------- --------------------
2022-09-23 07:00:00     2022-09-4-friday           2022-09-30 07:00:00    2022-09-5-friday

--以'fri'作为参数
SELECT NEXT_DAY(TO_DATE('2022-09-23 07:00:00','yyyy-mm-dd hh24:mi:ss'),'fri') "year-month-day",
TO_CHAR(NEXT_DAY(TO_DATE('2022-09-23 07:00:00','yyyy-mm-dd hh24:mi:ss'),'fri'),'yyyy-mm-w-day') "year-month-week-day"
FROM DUAL;
year-month-day                  year-month-week-day
------------------------------- --------------------------------
2022-09-30 07:00:00             2022-09-5-friday

--返回DATA类型结果
SELECT NEXT_DAY(TIMESTAMP '2022-09-23 10:00:00.99','friday') "year-month-day",
TO_CHAR(NEXT_DAY(TIMESTAMP '2022-09-23 10:00:00.99','friday'),'yyyy-mm-w-day') "year-month-week-day"
FROM DUAL;
year-month-day                  year-month-week-day
------------------------------- ---------------------------
2022-09-30 10:00:00             2022-09-5-friday
```

NLSSORT

```
nlssort::= NLSSORT "(" expr ["," "'" NLS_SORT "=" collation "'"] ")"
```

NLSSORT函数根据指定的排序规则对expr的值进行排序，生成一个VARCHAR类型的排序键。本函数只能在UTF-8、GB18030字符集下运行。

此函数一般应用于ORDER BY语句中，此时函数生成的是包含每一行排序键的整体排序键，且在某一行的排序键为NULL时，遵循ASC时NULLS LAST，和DESC时NULLS FIRST原则。

expr的值须为字符型，或可转换为字符型的其他类型（LOB及XMLTYPE类型支持隐式转换）。当expr的值为NULL时，函数返回NULL。

expr不支持32000字节以上的XMLTYPE、LOB类型数据。

NLS_SORT

指定排序规则，可省略，默认为CHINESE_PINYIN。YashanDB支持如下两种排序规则：

- CHINESE_PINYIN：区分大小写的拼音排序
- CHINESE_PINYIN_CI：不区分大小写的拼音排序

在CHINESE_PINYIN排序规则下的排序键构造规则为：

- 对每个字符（空格除外），按其Unicode值（十六进制）生成一个四位的字符串的排序规则键，如对'a'生成'0061'。
- 对一个不含空格的字符串，其排序键由其所有字符的排序规则键拼接，并在最后加上'000001..01'组成，其中'01'的个数对应字符的个数，如对'aA'生成'0061004100000101'。
- 对一个只包含空格的字符串，无论空格有多少个，都生成'00000020'的排序规则键。
- 对一个部分包含空格的字符串，忽略出现在字符串末尾的空格，对其他空格，则在'000001..01'中，按照空格出现的位置，相应增加'20'，如对'a(空格符1)(空格符2)(空格符3)A(空格符4)(空格符5) '生成'00610041000000120202001'。

在CHINESE_PINYIN_CI排序规则下的排序键构造规则为：

- 对每个字符（空格除外），将小写转换为大写后，按其Unicode值（十六进制）生成一个四位的字符串的排序规则键，如对'a'生成'0041'。
- 一个不含空格的字符串，其排序键由其所有字符的排序规则键拼接，如对'aA'生成'00410041'。
- 对一个只包含空格的字符串，无论空格有多少个，都生成'0000'的排序规则键。
- 对一个部分包含空格的字符串，忽略所有空格，如对'a(空格符1)(空格符2)(空格符3)A(空格符4)(空格符5) '生成'00410041'。

示例

```
--更新employees表中John员工名称为join
UPDATE employees SET employee_name='join' WHERE employee_name='John';

--指定不同排序规则对其按姓名排序
SELECT employee_name n,NLSSORT(employee_name) s1,
NLSSORT(employee_name,'NLS_SORT=CHINESE_PINYIN_CI') s2
FROM employees;
N         S1                                 S2
--------- ---------------------------------- --------------------
Mask      004D00610073006B000001010101       004D00410053004B
join      006A006F0069006E000001010101       004A004F0049004E
Anna      0041006E006E0061000001010101       0041004E004E0041
Jack      004A00610063006B000001010101       004A00410043004B
Jim       004A0069006D0000010101             004A0049004D

SELECT employee_name n
FROM employees
ORDER BY NLSSORT(employee_name);
N
-------------
Anna
Jack
Jim
Mask
join
```

```
SELECT employee_name n
FROM employees
ORDER BY NLSSORT(employee_name,'NLS_SORT=CHINESE_PINYIN_CI');
N
-------------
Anna
Jack
Jim
join
Mask
```

NOW

```
now::= NOW ["()"]
```

NOW函数无给定参数，直接返回当前数据库所在的操作系统设置的当前日期，其返回类型为DATE，且与DATE_FORMAT参数所指定格式一致。

如果一个SQL语句中出现了多个NOW函数，在该语句执行过程中每次都会调用NOW函数获取当前日期。

示例

```
SELECT NOW FROM DUAL;
NOW
--------------------------------
2022-08-24 23:55:31
```

NULLIF

```
nullif::= NULLIF "(" expr1 "," expr2 ")"
```

NULLIF函数用于对两个参数的值进行比较，相等时返回NULL，不相等时返回expr1的值。本函数常用于防止程序中因为出现除以零错误而抛出异常。

expr1和expr2均为YashanDB认可的通用表达式，且存在如下约束：

- expr1不能为NULL，否则函数返回错误。
- expr1和expr2均不能为LOB类型及XMLTYPE类型，否则函数返回错误。
- expr2可以为NULL，此时函数返回expr1的值。

本函数根据以下规则确定返回结果的数据类型：

- expr1和expr2均为数值型时，函数将会确定其中具有最高精度的数据类型，并将计算结果转为该类型返回。
- expr1和expr2均为字符型时，函数返回值类型为expr1的类型。
  - CHAR、VARCHAR与NCHAR、NVARCHAR不可以混合运算，否则函数返回错误。
- expr1和expr2中有一个是DATE类型，另一个是TIMESTAMP类型时，函数返回TIMESTAMP类型。
- expr1与expr2类型为除上述外的其他情况，且两者类型相同时，函数返回相同的数据类型；类型不同，且无法按照一定的规则进行转换时，函数返回类型转换错误。

示例

```
--expr1和expr2的值不相等时
SELECT NULLIF('123','456') res FROM DUAL;
RES
-----
123

--expr1和expr2的值相等时
SELECT NULLIF('123','123') res FROM DUAL;
RES
-----

--sales_info表包含了机构的11001产品的销售数量和金额信息，现对其增加一条数量和金额均为0的记录
INSERT INTO sales_info VALUES ('2021','06','0101','11001',0,0,'');
--当需要计算11001产品的在每个月的销售单价时，会出现除以0错误
SELECT year,month,product,amount/quantity
FROM sales_info
WHERE product='11001';
[1:33]YAS-00011 divided by zero
--使用NULLIF函数提前预防此类错误，减少异常捕获处理，避免应用在此中断
SELECT year,month,product,amount/NULLIF(quantity,0) price
FROM sales_info
WHERE product='11001';
YEAR  MONTH PRODUCT         PRICE
----- ----- --------- -----------
2021  10    11001              15
2001  01    11001      16.6666667
2000  12    11001              15
2015  03    11001              15
2021  05    11001              15
2021  06    11001
2015  11    11001              15
```

NUMTODSINTERVAL

```
numtodsinterval::= NUMTODSINTERVAL "(" expr "," "'"(DAY|HOUR|MINUTE|SECOND)"'" ")"
```

NUMTODSINTERVAL函数将expr表示的除BIT外数值型数据转换为以DAY|HOUR|MINUTE|SECOND为单位的INTERVAL DAY TO SECOND类型的数值。

当expr的值为NULL时，返回NULL。

当expr的值为非数值型数据时，将先进行到NUMBER类型的转换，转换失败时返回Invalid number错误。

当指定DAY|HOUR|MINUTE|SECOND单位时，对DAY|HOUR|MINUTE|SECOND的大小写不敏感。

示例

```
SELECT NUMTODSINTERVAL('4','minute') res FROM DUAL;
RES
--------------------------------
+00 00:04:00.000000

--获取在当前时间后四分钟的时间值
SELECT SYSDATE, SYSDATE+NUMTODSINTERVAL('4','minute') sysdate2 FROM DUAL;
SYSDATE                        SYSDATE2
------------------------------ ------------------------------
2021-11-30 17:03:42            2021-11-30 17:07:42
```

NUMTOYMINTERVAL

```
numtoyminterval::= NUMTOYMINTERVAL "(" expr "," "'"(YEAR|MONTH)"'" ")"
```

NUMTOYMINTERVAL函数将expr表示的除BIT外数值型数据转换为以YEAR|MONTH为单位的INTERVAL YEAR TO MONTH类型的数值。

当expr的值为NULL时，返回NULL。

当expr的值为非数值型数据时，将先进行到NUMBER类型的转换，转换失败时返回Invalid number错误。

当指定YEAR|MONTH单位时，对YEAR|MONTH的大小写不敏感。

示例

```
SELECT NUMTOYMINTERVAL('4','year') res FROM DUAL;
RES
--------------------
+04-00
```

NVL

```
nvl::= NVL "(" expr1 "," expr2 ")"
```

NVL函数从左向右判断，返回两个expr参数值中首个非空的表达式的值。

若expr1和expr2都为NULL，则函数返回NULL。

expr1和expr2均支持LOB及XMLTYPE类型隐式转换。

当expr1和expr2的数据类型不相同时，函数先执行类型转换，若两个数据类型之间无法按照一定的规则进行转换则返回类型转换错误。类型转换规则如下：

- 若expr1为字符型，则将expr2转换为VARCHAR类型，且返回值为VARCHAR类型。
- 若expr1与expr2都为数值型，函数将会确定所有数据中具有最高精度的数据类型，将所有expr的执行结果转为该类型，且返回值也为该类型。
- 若expr1为未知类型，将expr1转换为expr2的类型。
- 若expr2为未知类型，将expr2转为expr1的类型。
- 若expr1与expr2皆为未知类型，将expr1与2都转为VARCHAR类型。
- 若expr1与expr2类型已知且类型不相同，同时不满足上述的情况1与情况2时，会强制将expr2转为expr1的数据类型，如果不满足转换条件则返回invalid datatype错误，具体类型转换规则同CAST函数描述。

示例

```
SELECT NVL(NOW(),'2011-05-11 12:28:04') res FROM DUAL;
RES
-------------------------------
2021-12-05 18:02:48

SELECT NVL('a',1) res FROM DUAL;
RES
-----
a
```

NVL2

```
nvl2::= NVL2 "(" expr1 "," expr2 "," expr3 ")"
```

NVL2函数从左向右判断，如果expr1非空，则返回expr2的值，如果expr1为空，则返回expr3的值。

expr1、expr2、expr3均为YashanDB认可的通用表达式。

expr1、expr2、expr3均支持LOB及XMLTYPE类型隐式转换。

当expr2和expr3的数据类型不相同时，函数先执行类型转换，若两个数据类型之间无法按照一定的规则进行转换则返回类型转换错误。 类型转换规则如下：

- expr2为字符型时，函数将expr3转换为expr2的数据类型（除非expr3为NULL，此时不需要进行转换），并返回该数据类型。
- expr2为数值型时，函数根据最小提升规则（ TINYINT->SMALLINT->INT->BIGINT->NUMBER->FLOAT->DOUBLE ），将数值优先级低的参数类型转换为数值优先级高的参数类型，并返回该数据类型。
- expr2和expr3其中一个为空时，函数返回非空值的数据类型。
- expr2和expr3均为空时，函数返回VARCHAR数据类型。

示例

```
--因为第一个参数不为空，所以返回第二个参数，同时第二个参数和第三个参数之间进行隐式类型转换
SELECT NVL2(1,NOW(),'2011-04-01 12:28:03') res FROM DUAL;
RES
--------------------------------
2022-10-09 00:17:09

--因为第一个参数是空的，所以返回第三个参数，同时第二个参数和第三个参数之间进行隐式类型转换
SELECT NVL2('',NOW(),'2011-04-01 12:28:03') res FROM DUAL;
RES
--------------------------------
2011-04-01 12:28:03

--隐式类型转换失败
SELECT NVL2(null,NOW(),'an apple') res FROM DUAL;
[1:13]YAS-00008 type convert error : literal does not match format string

--INT类型转换成VARCHAR类型
SELECT NVL2(null,'an apple',124) res FROM DUAL;
RES
-----
124
```

OCTET_LENGTH

```
OCTET_LENGTH::= OCTET_LENGTH "(" expr ")"
```

OCTET_LENGTH函数按字节统计expr的长度，返回一个BIGINT的数值。本函数与LENGTHB函数同义。

expr支持字符型，或可转化为字符型的其他类型。

对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT CHAR_LENGTH('深圳') r1,OCTET_LENGTH('深圳') r2,BIT_LENGTH('深圳') r3,
       CHARACTER_LENGTH('aabbccDDee') r4,OCTET_LENGTH('aabbccDDee') r5,BIT_LENGTH('aabbccDDee') r6,
       CHAR_LENGTH(null) r7,OCTET_LENGTH(null) r8,BIT_LENGTH(null) r9
FROM DUAL;
   R1   R2    R3    R4    R5    R6    R7    R8    R9
----- ---- ----- ----- ----- ----- ----- ----- -----
    2    6    48    10    10    80
```

PI

```
pi::= PI  "("")"
```

PI函数无给定参数，返回圆周率的值，返回数据类型为DOUBLE。

示例

```
--求圆周率的值
SELECT PI() FROM DUAL;

                PI()
--------------------
 3.1415926535898E+000
```

POSITION

```
position::= POSITION "(" expr1 IN expr2 ")"
```

POSITION函数在一个expr2表示的字符串中从左向右查找expr1表示的字符串，返回expr1第一次出现的位置，该结果为一个BIGINT类型的数字。当没有查找到匹配值时，返回0。

- expr1、expr2的值须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。
- 当expr1或expr2中任一值为NULL时，函数返回NULL。

示例

```
SELECT employee_name n,
POSITION('a' IN employee_name) p
FROM employees;
N                                P
-------------- --------------------
Mask                             2
John                             0
Anna                             4
Jack                             2
Jim                              0
```

POW POWER

```
pow::= POW "(" expr "," exp ")"
power::= POWER "(" expr "," exp ")"
```

POW/POWER函数计算expr参数值的exp次幂，其返回类型为：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER、CHAR、VARCHAR、NCHAR、NVARCHAR类型时，返回NUMBER。
- 当expr的值为FLOAT或DOUBLE类型时，返回DOUBLE。
- 当exp的值为FLOAT或DOUBLE类型时，返回DOUBLE。
- 当expr或exp的值为NULL时，函数返回NULL。
- 对于不能转换为NUMBER类型的expr或exp报错。

exp

指数。

exp为与expr相同的通用表达式，当exp的值为NULL时，函数返回NULL。

下表列示了不同情况下本函数的计算规则：（字符型将被隐式转换为NUMBER类型参与下表规则）

expr底数数据类型	expr底数值	exp指数数据类型	exp指数值	函数计算结果
整数/浮点数/NUMBER	正数/负数/0	整数/浮点数/NUMBER	0	1
整数/浮点数/NUMBER	正数	整数/浮点数/NUMBER	正数/负数	expr的exp次幂
整数/浮点数/NUMBER	0	整数/浮点数/NUMBER	正数	0
整数/浮点数/NUMBER	0	浮点数	负数	Inf
整数/浮点数/NUMBER	0	整数/NUMBER	负数	YAS-00012错误
整数	负数	整数/NUMBER（小数点后无值）	正数/负数	expr的exp次幂
整数	负数	浮点数	正数/负数	Nan
整数	负数	NUMBER（小数点后有值）	正数/负数	YAS-04426错误
浮点数	负数	整数	正数/负数	expr的exp次幂
浮点数	负数	浮点数	正数/负数	Nan
浮点数	负数	NUMBER	正数/负数	Nan
NUMBER	负数	整数/NUMBER（小数点后无值）	正数/负数	expr的exp次幂
NUMBER	负数	浮点数	正数/负数	Nan
NUMBER	负数	NUMBER（小数点后有值）	正数/负数	YAS-04426错误

示例

```
SET NUMWIDTH 20;

SELECT POW(2,2) res FROM DUAL;
                 RES
--------------------
                   4

SELECT POW(2,-2) res FROM DUAL;
                 RES
--------------------
                 .25
```

```
SELECT POW(2,2.1) res FROM DUAL;
                  RES
--------------------
 4.287093850145172657

SELECT POWER(2.2,-12.1) res FROM DUAL;
                  RES
--------------------
 .0000718928270982645

SELECT POW(-1,1.1) res FROM DUAL;
[1:13]YAS-04426 the argument value is out of range

SELECT POW(0, -1) res FROM DUAL;
[1:12]YAS-00012 numeric overflow
```

PX_CHANNEL

```
px_channel::= PX_CHANNEL "(" sessionid ")"
```

PX_CHANNEL表函数根据输入的sessionid，查询返回该连接会话下的tabqueue通道信息。

其中sessionid的值必须为INT类型。

sessionid

在分布式场景下为全局sessionid，可以通过DV$SESSION视图查得。

在单机场景下为当前节点的session编号，可以通过V$SESSION视图查得。

返回值：函数执行返回组成tabqueue通道信息的表。px_channel表函数返回信息如下：

字段	类型	说明
SID	INTEGER	分布式下为全局session id
SERIAL#	INTEGER	
SEQ_NO	INTEGER	sequence number
SQL_ID	VARCHAR	对应SQL
STAGE_ID	SMALLINT	对应stage id
TQ_ID	SMALLINT	table queue id
TYPE	VARCHAR	reader或writer
IS_LOCAL	VARCHAR	是否进程内通道，两端endpoint都是本地endpoint。
WPORT	SMALLINT	写端口号
RPORT	SMALLINT	读端口号
WPORT_ENDPOINT	SMALLINT	写端对应节点endpoint
RPORT_ENDPOINT	SMALLINT	读端对应节点endpoint
LINK_ID	SMALLINT	对应ICS链路ID。reader时对应接收链路ID；writer时对应发送链路ID。
WINDOW_SIZE	INTEGER	滑动窗口总大小
WINDOW_ACK_ID	INTEGER	滑窗可用block id
WINDOW_CURR_ID	INTEGER	滑窗当前使用block id
PENDING_BLOCKS	INTEGER	待处理block数
HOLD_MEMORY	INTEGER	占用内存大小
WRITE_NUM	BIGINT	写次数 writer有效
WRITE_SIZE	BIGINT	写数据大小 writer有效
READ_NUM	BIGINT	读次数 reader有效
READ_SIZE	BIGINT	读数据大小 reader有效
READ_ROWS	BIGINT	读记录数 reader有效
TIMEOUT_TIMES	INTEGER	超时次数 reader时为接收数据的等待超时次数 writer时为等待滑动窗口配额超时次数
LAST_ACTIVE_TIME	TIMESTAMP	最后活动时刻 reader时为最后收到数据的时间 writer时为最后发出数据的时间

字段	类型	说明
MAX_BLOCK_WAIT_TIME	BIGINT	获取block最长等待时间
AVG_BLOCK_WAIT_TIME	BIGINT	获取block平均等待时间
MAX_CHANNEL_WAIT_TIME	BIGINT	数据入列到出列最长耗时
AVG_CHANNEL_WAIT_TIME	BIGINT	数据入列到出列平均耗时

示例（分布式部署）

```
--根据实际场景获取一个全局会话ID
SELECT global_session_id FROM DV$SESSION;

--查询该连接会话下的tabqueue通道信息
SELECT SID, SQL_ID, STAGE_ID, TQ_ID, TYPE, IS_LOCAL FROM TABLE(PX_CHANNEL(131091));

         SID SQL_ID                            STAGE_ID    TQ_ID TYPE      IS_LOCAL
------------ --------------------------------- -------- -------- --------- ---------
      131091 6w7vgtqwc018p                            -1        0 READER    N
      131091 6w7vgtqwc018p                            -1        0 READER    N
      131091 6w7vgtqwc018p                            -1        0 READER    N
      131091 6w7vgtqwc018p                            -1        0 READER    N
      131091 6w7vgtqwc018p                            -1        0 READER    N
      131091 6w7vgtqwc018p                             0        0 WRITER    N
      131091 6w7vgtqwc018p                             0        0 WRITER    N
      131091 6w7vgtqwc018p                             0        0 WRITER    N
      131091 6w7vgtqwc018p                             0        0 WRITER    N
      131091 6w7vgtqwc018p                             0        0 WRITER    N
      131091 6w7vgtqwc018p                             0        0 WRITER    N
```

PX_OBJ

```
px_obj::= PX_OBJ "(" sessionid ")"
```

PX_OBJ表函数根据输入的sessionid，查询返回该连接会话下的并行stage信息。

其中sessionid的值必须为INT类型。

sessionid

在分布式场景下为全局sessionid，可以通过DV$SESSION视图查得。

在单机场景下为当前节点的session编号，可以通过V$SESSION视图查得。

返回值：函数执行返回组成并行stage信息的表。px_obj表函数返回信息如下：

字段	类型	说明
SID	INTEGER	分布式下为全局session id
SERIAL#	INTEGER	
SEQ_NO	INTEGER	sequence number
SQL_ID	VARCHAR	对应SQL
THREAD_ID	BIGINT	对应执行线程ID
STAGE_ID	SMALLINT	对应stage id
BROTHER_ID	SMALLINT	位于同级stage的位置id
TYPE	VARCHAR	sender或者receiver
SENDER_TYPE	INTEGER	对应计划中的sender type RANDOM，HASH，BROADCAST等
TQ_ID	SMALLINT	table queue id
RPORT	SMALLINT	正在接收端口号 receiver有效
WPORT	SMALLINT	正在发送端口号 sender有效
LOCAL_ENDPOINT	SMALLINT	stage所处节点enpoint
REMOTE_ENDPOINT	SMALLINT	远端节点endpoint
REMOTE_GROUP	SMALLINT	远端节点组ID
STATUS	INTEGER	PX状态 INIT：初始状态 BUSY：完成握手 FINISH：正常发送完毕 ABORT：被终止
WORKING_REMOTE_PORTS	SMALLINT	活跃中的远程通道数量
MIN_WINDOW_BLOCKS	INTEGER	最小滑动窗口剩余block数
MIN_WINDOW_PORT	SMALLINT	最小滑动窗口对应端口号
MAX_WINDOW_BLOCKS	INTEGER	最大滑动窗口剩余block数
MAX_WINDOW_PORT	SMALLINT	最大滑动窗口对应端口号
PENDING_BLOCKS	INTEGER	待处理block数
HOLD_MEMORY	INTEGER	占用内存大小
SEND_NUM	BIGINT	发送次数 sender有效
SEND_SIZE	BIGINT	发送数据大小 sender有效

字段	类型	说明
RECV_NUM	BIGINT	接收次数 receiver有效
RECV_SIZE	BIGINT	接收数据大小 receiver有效
RECV_ROWS	BIGINT	接收记录数 receiver有效
TIMEOUT_TIMES	INTEGER	超时次数 receiver时为接收数据的等待超时次数 sender时为等待滑动窗口配额超时次数
LAST_ACTIVE_TIME	TIMESTAMP	最后活动时刻 receiver时为最后收到数据的时间 sender时为最后发出数据的时间

示例（分布式部署）

```
--根据实际场景获取一个全局会话ID
SELECT global_session_id FROM DV$SESSION;

--查询该连接会话下的tabqueue通道信息
SELECT SID, SQL_ID, THREAD_ID, STAGE_ID, TYPE FROM TABLE(PX_OBJ(131094));

         SID SQL_ID                        THREAD_ID STAGE_ID TYPE
------------ ----------------- ---------------------- -------- -----------------
      131094 7x36y59y3gkwu                     7404       -1 receiver
      131094 7x36y59y3gkwu                     7404       -1 receiver
      131094 7x36y59y3gkwu                     7391        0 sender
      131094 7x36y59y3gkwu                     7391        0 sender
      131094 7x36y59y3gkwu                     7391        0 sender
      131094 7x36y59y3gkwu                     8851        1 sender
      131094 7x36y59y3gkwu                     8851        1 sender
      131094 7x36y59y3gkwu                     8851        1 sender
      131094 7x36y59y3gkwu                     7388        0 sender
      131094 7x36y59y3gkwu                     7393        0 sender
      131094 7x36y59y3gkwu                     7393        0 sender
      131094 7x36y59y3gkwu                     7393        1 sender
      131094 7x36y59y3gkwu                     8852        1 sender
      131094 7x36y59y3gkwu                     8852        1 sender
```

RANDOM

```
random::= RANDOM "("")"
```

RANDOM函数返回一个0到1之间的随机数，返回类型为NUMBER型。可以使用 `ORDER BY RANDOM()` 来对一组记录进行随机化排序。

示例

```
SELECT RANDOM() res FROM DUAL;
        RES
-----------
 .936900865

SELECT *
FROM employees
ORDER BY RANDOM();
BRANCH DEPARTMENT EMPLOYEE_NO   EMPLOYEE_NAME SEX   ENTRY_DATE
------ ---------- ------------- ------------- ----- --------------------------------
0101   008        0201008004    Jim           1     2021-06-19 22:55:32
0101   000        0101000001    Mask          1     2019-04-11 22:55:32
0201   010        0201010011    Anna          0     2021-03-11 22:55:32
0201   008        0201008003    Jack          1     2020-02-05 22:55:32
0101   000        0101000002    John          1     2016-07-15 22:55:32
```

RANK

```
rank::= RANK "("")" OVER "(" [query_partition_clause] order_by_clause ")"
```

RANK为窗口函数，用于对数据的实时分析。

- RANK函数列出相同并列值，并对下一顺序值跳号，例如1，2，3，3，3，6，7，8.....

查看RANK函数与DENSE_RANK函数的区别。 查看RANK函数与ROW_NUMBER函数的区别。

query_partition_clause|order_by_clause

partition by与order by的参数参数支持除LOB、JSON、XMLTYPE、UDT外的其它数据类型。

窗口函数通用语法。

示例

```
--sales_info_range表中包含如下字段和数据
SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0101   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  11    0101   11001              20         300
2015  03    0102   11001              20         300
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600

--按branch进行分组，并在组内按amount进行排序
SELECT branch,amount,year,month,quantity,
       RANK() OVER (PARTITION BY branch ORDER BY amount) rank_res
FROM sales_info_range
WHERE product='11001'
ORDER BY branch,quantity;
BRANCH      AMOUNT YEAR  MONTH    QUANTITY              RANK_RES
------ ----------- ----- ----- ----------- ---------------------
0101           300 2021  10             20                     1
0101           300 2015  11             20                     1
0101           500 2001  01             30                     3
0101           600 2021  05             40                     4
0102           300 2015  03             20                     1
0102           300 2000  12             20                     1
```

REGEXP_COUNT

```
regexp_count::= REGEXP_COUNT "(" expr "," regexp ["," position["," match_para]] ")"
```

REGEXP_COUNT为正则表达式函数，在expr表示的源字符串中按正则模式可以匹配到的次数，如果没有找到匹配项，则函数返回0，函数返回值为BIGINT类型。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 函数使用输入字符集定义的字符计算字符串（仅支持UTF-8）。

regexp

RegExp，长度不超过512字节，可为NULL，此时函数返回NULL。

position

指定开始进行匹配的偏移量，系统从expr的第position个字符开始匹配，在找到第一个正则匹配项后，继续从匹配项之后的第一个字符开始，查找第二个匹配项，直到expr的最后一个字符。

position为与expr相同的通用表达式，须为数值型数据，或可转换为NUMBER类型的其他类型数据，且其值应为一个正整数，内部处理时统一转换为BIGINT类型。

- 当position为NULL时，函数返回NULL。
- 当position为0或负数时，函数返回Out of range错误。
- 当position为小数时，截断保留整数作为position值。
- 当position超出expr值的长度时，不执行匹配，函数返回0值。
- 未指定此值时，默认从第一个字符开始匹配。

match_para

正则匹配参数，与REGEXP_LIKE中的含义相同。

示例

```
SELECT REGEXP_COUNT('123123123123123', '123', 1, 'i') REGEXP_COUNT FROM DUAL;
         REGEXP_COUNT
---------------------
                    5

SELECT REGEXP_COUNT('123123123123123', '123', 4, 'i') REGEXP_COUNT FROM DUAL;
         REGEXP_COUNT
---------------------
                    4

SELECT REGEXP_COUNT('A1B2C3', '[A-Z][0-9]') REGEXP_COUNT FROM DUAL;
         REGEXP_COUNT
---------------------
                    3
```

REGEXP_INSTR

```
regexp_instr::= REGEXP_INSTR "(" expr "," regexp ["," position["," occurrence["," return_opt["," match_para["," subexpr]]]]]
")"
```

REGEXP_INSTR为正则表达式函数，其功能与INSTR函数相似，但REGEXP_INSTR函数的regexp参数可使用正则表达式，对expr表示的字符串进行正则匹配，返回匹配到的子字符串的开始或结束位置（取决于return_opt参数），其结果为一个BIGINT类型的数值。如果没有找到匹配项，则函数返回0。

expr的值须为字符型，或可转换为字符型的其他类型，不能为转义字符。当expr的值为NULL时，函数返回NULL。

expr不支持32000字节以上的XMLTYPE、LOB类型数据。

函数使用输入字符集定义的字符计算字符串（仅支持UTF-8）。

regexp

RegExp，长度不超过512字节，支持转义字符，可为NULL，此时函数返回NULL。

position

指定开始进行匹配的偏移量，须为数值型数据，或可转换为NUMBER类型的其他类型数据。内部处理时统一转换为BIGINT类型。

position为与expr相同的通用表达式，其值存在如下规则：

- 应该为一个正整数。
- 为NULL时，函数返回NULL。
- 为0或负数时，函数返回Out of range错误。
- 为小数时，截断保留整数作为position值。
- 超出expr值的长度时，不执行匹配，函数返回0值。

指定此值时，系统从expr的第position个字符开始匹配，在找到第一个正则匹配项后，继续从匹配项之后的第一个字符开始，查找第二个匹配项，直到expr的最后一个字符。

未指定此值时，默认从第一个字符开始匹配。

occurrence

指定返回regexp在expr中第occurrence次匹配到的位置，须为数值型数据，或可转换为NUMBER类型的其他类型数据。内部处理时统一转换为BIGINT类型。

occurrence为与expr相同的通用表达式，其值存在如下规则：

- 应该为一个正整数。
- 为NULL时，函数返回NULL。
- 为0或负数时，函数返回Out of range错误。
- 为小数时，截断保留整数作为occurrence值。
- 当occurrence超出匹配次数时，不执行匹配，函数返回expr值。

未指定此值时，默认为第1次匹配。

return_opt

指定返回的匹配位置选项，其值只能为0或1，可省略，默认为0。

- 如果指定为0，将返回匹配项的第一个字符的位置。
- 如果指定为1，将返回匹配项的最后一个字符的位置。

match_para

正则匹配参数，与REGEXP_LIKE中的含义相同。

subexpr

subexpr参数用于当regexp中含有子表达式时，指定函数返回某一个子表达式所匹配到的位置，subexpr的值只能为0到9之间的某个整数（对于小数将截断为整数）。

子表达式是使用'()'括起来的表达式片段，其中'()'可以嵌套。所有子表达式按其左括号在表达式中出现的顺序编号。例如，考虑下面的表达式：

```
0123(((abc)(de)f)ghi)45(678)
```

该表达式有五个子表达式，按顺序分别为：“abcdefghi”、“abcdef”、“abc”、“de”和“678”。

指定subexpr值时的匹配规则如下：

- 当subexpr等于0时，函数按regexp表达式在expr中查找，返回匹配到的整个子字符串在expr中的位置。
- 当subexpr大于0时：
  - 当regexp中不含子表达式时，函数返回0。
  - 否则，当函数按regexp在expr中未找到匹配项时，函数返回0。
  - 否则，函数返回按正则匹配到的整个子字符串中第subexpr个子表达式在expr中的位置。

示例

```
SELECT REGEXP_INSTR('1234567890', '(123)(4(56)(78))', 1, 1, 0, 'i', 2) "REGEXP_INSTR" FROM DUAL;
         REGEXP_INSTR
--------------------
                    4

SELECT REGEXP_INSTR('1234567890', '[0-9]', 1, 5) "REGEXP_INSTR" FROM DUAL;
         REGEXP_INSTR
--------------------
                    5
```

REGEXP_LIKE

```
regexp_like::= REGEXP_LIKE "(" expr "," regexp ["," match_para] ")"
```

REGEXP_LIKE为正则表达式函数，其功能与LIKE语法相似，但与LIKE只能使用'%'和'_'通配符相比，REGEXP_LIKE函数的regexp参数可使用正则表达式，对expr表示的字符串进行正则匹配，函数返回值为布尔类型，匹配成功时返回TRUE，否则返回FALSE。

expr的值须为字符型，或可转换为字符型的其他类型。当expr的值为NULL时，函数返回NULL。

expr不支持32000字节以上的XMLTYPE、LOB类型数据。

函数使用输入字符集定义的字符计算字符串（仅支持UTF-8）。

regexp

指定一个RegExp，长度不超过512字节，可为NULL，此时函数返回NULL。

match_para

YashanDB支持在进行正则匹配时指定如下匹配参数：

- 'i'：大小写不敏感。
- 'c'：大小写敏感，默认为此值。
- 'n'：允许句点（.）匹配任何字符，包括换行符。如果省略此参数，则句点与换行符不匹配。
- 'm'：将字符串视为多行，将^和$分别解释为字符串中任意行的开始和结束，而不仅仅是整个字符串的开始或结束。如果省略此参数，则将字符串视为一行。
- 'x'：忽略空白和#注释。默认情况下，空白字符与其自身匹配。

match_para参数可以不指定或指定为NULL（均使用默认值），如指定了除上述之外的其他参数，则函数返回YAS-04363错误。

示例

```
SELECT REGEXP_LIKE('aa\naa','^aa$','i') reg1,
REGEXP_LIKE('aa','^aa$','i') reg2 FROM DUAL;
REG1                 REG2
-------------------- --------------------
false                true

SELECT REGEXP_LIKE('AA','A.A') reg1,
REGEXP_LIKE('AA','A.A','n') reg2 FROM DUAL;
REG1                 REG2
-------------------- --------------------
false                false

SELECT REGEXP_LIKE('-654196584','^-[0-9]*[1-9][0-9]*$') reg1,
REGEXP_LIKE('654196584','^[0-9]*[1-9][0-9]*$','m') reg2 FROM DUAL;
REG1                 REG2
-------------------- --------------------
true                 true
```

REGEXP_REPLACE

```
regexp_replace::= REGEXP_REPLACE "(" expr "," regexp ["," replace["," position["," occurrence["," match_para]]]] ")"
```

REGEXP_REPLACE为正则表达式函数，其功能与REPLACE函数相似，但REGEXP_REPLACE函数的regexp参数可使用正则表达式，对expr表示的字符串进行正则匹配，返回匹配成功且匹配部分被replace替代后的VARCHAR类型字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 函数使用输入字符集定义的字符计算字符串（仅支持UTF-8）。
- 函数返回结果字符串长度超过上限（32000）时，返回replace字符串。

regexp

RegExp，长度不超过512字节，可为NULL，此时函数返回expr。

replace

用于替代的字符串，replace为与expr相同的通用表达式，须为字符型，或可转换为字符型的其他类型。

position

指定开始进行匹配的偏移量，系统从expr的第position个字符开始匹配，在找到第一个正则匹配项后，继续从匹配项之后的第一个字符开始，查找第二个匹配项，直到expr的最后一个字符。

position为与expr相同的通用表达式，须为数值型数据，或可转换为NUMBER类型的其他类型数据，且其值应为一个正整数，内部处理时统一转换为BIGINT类型。

- 当position为NULL时，函数返回NULL。
- 当position为0或负数时，函数返回Out of range错误。
- 当position为小数时，截断保留整数作为position值。
- 当position超出expr值的长度时，不执行匹配，函数返回expr值。
- 未指定此值时，默认从第一个字符开始匹配。

occurrence

指定按regexp在expr中第occurrence次匹配到的子字符串执行替代，occurrence为与expr相同的通用表达式，须为数值型数据，或可转换为NUMBER类型的其他类型数据。且其值应为一个负整数，内部处理时统一转换为BIGINT类型。

- 当occurrence为NULL时，函数返回NULL。
- 当occurrence为0时，替换所有匹配项。
- 当occurrence为负数时，函数返回Out of range错误。
- 当occurrence为小数时，截断保留整数作为occurrence值。
- 当occurrence超出匹配次数时，不执行匹配，函数返回expr值。
- 未指定此值时，默认为第1次匹配。

match_para

正则匹配参数，与REGEXP_LIKE中的含义相同。

示例

```
SELECT REGEXP_REPLACE('1234567890', '456','!', 1, 1,'i') "REGEXP_REPLACE" FROM DUAL;
REGEXP_REPLACE
--------------
123!7890
```

```
SELECT REGEXP_REPLACE('1234567890', '[0-9]','!', 1, 5) "REGEXP_REPLACE" FROM DUAL;
REGEXP_REPLACE
--------------
1234!67890

SELECT REGEXP_REPLACE(DHQ,'h','H',1,0,'i') "REGEXP_REPLACE"
FROM area
WHERE area_no='01';
REGEXP_REPLACE
------------------
SHangHai
```

REGEXP_SUBSTR

```
regexp_substr::= REGEXP_SUBSTR "(" expr "," regexp ["," position["," occurrence["," match_para["," subexpr]]]] ")"
```

REGEXP_SUBSTR为正则表达式函数，其功能与参数定义均类似于REGEXP_INSTR，但它不返回子字符串的位置，而是返回子字符串本身。函数返回值为VARCHAR类型。

expr

expr的值须为字符型，或可转换为字符型的其他类型，不能为转义字符。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 函数使用输入字符集定义的字符计算字符串（仅支持UTF-8）。

regexp

RegExp，长度不超过512字节，支持转义字符，可为NULL，此时函数返回NULL。

position

指定开始进行匹配的偏移量，系统从expr的第position个字符开始匹配，在找到第一个正则匹配项后，继续从匹配项之后的第一个字符开始，查找第二个匹配项，直到expr的最后一个字符。

position为与expr相同的通用表达式，须为数值型数据，或可转换为NUMBER类型的其他类型数据。且其值应为一个正整数，内部处理时统一转换为BIGINT类型。

- 当position为NULL时，函数返回NULL。
- 当position为0或负数时，函数返回Out of range错误。
- 当position为小数时，截断保留整数作为position值。
- 当position超出expr值的长度时，不执行匹配，函数返回NULL。
- 未指定此值时，默认从第一个字符开始匹配。

occurrence

指定按regexp在expr中第occurrence次匹配到的子字符串执行替代，occurrence为与expr相同的通用表达式，须为数值型数据，或可转换为NUMBER类型的其他类型数据，且其值应为一个正整数，内部处理时统一转换为BIGINT类型。

- 当occurrence为NULL时，函数返回NULL。
- 当occurrence为0或负数时，函数返回Out of range错误。
- 当occurrence为小数时，截断保留整数作为occurrence值。
- 当occurrence超出匹配次数时，不执行匹配，函数返回expr值。
- 未指定此值时，默认为第1次匹配。

match_para

正则匹配参数，与REGEXP_LIKE中的含义相同。

subexpr

subexpr参数用于当regexp中含有子表达式时，指定函数返回某一个子表达式所匹配到的位置，subexpr的值只能为0到9之间的某个整数（对于小数将截断为整数）。

- 如果有小数点，则截断。
- 如果subexpr是0，则返回匹配整个pattern的位置。
- 如果subexpr > 0，则返回匹配subxpr对应的子表达式的位置。
- 如果subexpr >= 10，则返回NULL。
- 缺省默认值是0。
- 如果subexpr超过pattern子表达式数量，则返回NULL。

示例

```
SELECT REGEXP_SUBSTR('1234567890', '(123)(4(56)(78))', 1, 1,'i', 2) "REGEXP_SUBSTR" FROM DUAL;
REGEXP_SUBSTR
--------------
45678

SELECT REGEXP_SUBSTR('1234567890', '[0-9]', 1, 5) "REGEXP_SUBSTR" FROM DUAL;
REGEXP_SUBSTR
-------------
5
```

REPLACE

```
replace::= REPLACE "(" expr "," search_character ["," replace_character] ")"
```

REPLACE函数将expr表示的字符串里的所有的search_character替换为replace_character，返回一个VARCHAR类型的新字符串。

expr

expr的值须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

当expr的值为NULL时，函数返回NULL。

search_character

要进行替换的字符串，search_character为与expr相同的通用表达式，须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

当search_character的值为NULL时，函数不执行任何替换。

replace_character

按此字符串的值进行替换，replace_character为与expr相同的通用表达式，须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当replace_character的值为NULL时，函数将删除expr中的search_character部分。
- 不指定replace_character时，默认替换值为NULL。

示例

```
SELECT REPLACE('ShenZhen', 'Zhen', 'Yang') REPLACE FROM DUAL;
REPLACE
---------
ShenYang

--不执行任何替换
SELECT REPLACE('ShenZhen', '', 'Yang') REPLACE FROM DUAL;
REPLACE
---------
ShenZhen

--替换值为空时，删除要替换的字符串
SELECT REPLACE('ShenZhen', 'Zhen') REPLACE FROM DUAL;
REPLACE
-------
Shen
```

RIGHT

```
right::= RIGHT "(" expr "," length ")"
```

RIGHT函数将expr表示的字符串从右边截取指定长度，得到一个子字符串并将其返回。

expr

expr的值须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr为NCLOB/NCHAR/NVARCHAR类型时，返回值为NVARCHAR。其余场景返回值为VARCHAR。

length

指定字符串截取的长度，length为与expr相同的通用表达式，须为除BIT外数值型数据，或可转换为NUMBER类型的其他类型数据，取值范围[-2147483648,2147483647]。

- 当length的值为NULL，0或负数时，函数返回NULL。
- 当length值为小数时，函数将先对其进行取整，规则如下：
  - length为NUMBER型或者浮点型时：四舍五入取整。
  - length为可转换为NUMBER的其他类型时：截取整数部分。
- 若length值大于expr字符串长度，则将其按expr字符串长度值处理。

示例

```
SELECT RIGHT(SYSDATE+1,4.99) res FROM DUAL;
RES
-----------------------------------------------------------------
54:23

SELECT RIGHT(SYSDATE+1,'4.99') res FROM DUAL;
RES
-----------------------------------------------------------------
3:24
```

RLIKE_FILTER

```
rlike_filter::= RLIKE_FILTER "(" expr "," regexp ")"
```

RLIKE_FILTER为正则表达式函数，其功能与LIKE语法相似，但与LIKE只能使用'%'和'_'通配符相比，RLIKE_FILTER函数的regexp参数可使用正则表达式，对expr表示的字符串进行正则匹配，函数返回值为布尔类型，匹配成功时返回TRUE，否则返回FALSE。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 函数使用输入字符集定义的字符计算字符串（仅支持UTF-8）。

regexp

指定一个RegExp，长度不超过512字节，可为NULL，此时函数返回NULL。

示例

```
SELECT RLIKE_FILTER('aa\naa','^aa$') reg FROM DUAL;
REG
--------------------
false


SELECT RLIKE_FILTER('-654196584','^-[0-9]*[1-9][0-9]*$') reg FROM DUAL;
REG
--------------------
true
```

ROUND

```
round::= ROUND "(" ((expr ["," fmt])| (expr ["," round_number])) ")"
```

ROUND函数对expr的值按照指定格式四舍五入一个日期值，返回一个DATE类型的日期；或按照round_number指定的位数进行四舍五入，返回类型为：

expr类型	返回值	备注
TINYINT	SMALLINT	
SMALLINT	INT	
INT	BIGINT	
BIGINT	NUMBER	
NUMBER	NUMBER	
FLOAT	FLOAT/NUMBER	未指定round_number时输出FLOAT，否则输出NUMBER。
DOUBLE	DOUBLE/NUMBER	未指定round_number时输出DOUBLE，否则输出NUMBER。

当用于四舍五入日期值时，expr的值必须为DATE、TIMESTAMP类型。

当用于四舍五入数值时，expr的值必须为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。当expr的值为0时，函数返回0。

对于其他类型，函数返回类型转换错误。

当expr的值为NULL时，函数返回NULL。

fmt

指定日期值的截断格式，规则如下：

类型	有效格式	返回值	备注
Century	CC, SCC	大于'XX50'圆整到下一世纪第一天，否则返回当世纪第一天	
Year	SYYYY, YYYY, YEAR, SYEAR, YYY, YY, Y	从七月开始圆整到下一年第一天，否则本年第一天	
ISO Year	IYYY, IY, I	将每年的第一周作为基准，返回该周的第一天，从七月开始圆整到下一年第一天，否则本年第一天	ISO当年第一天可能是公历前一年12月末
Quarter	Q	在每个季度中的第二月的第16天后圆整到下一季度第一天	
Month	MONTH, MON, MM, RM	在每个月的第十六天后圆整到下个月第一天	
Week	WW	每年的1月1日作为当年的第一周的第一天，从每周的第五天圆整到下周的第一天，否则返回当周第一天	
IW	IW	星期一为每周的第一天，从每周的第五天开始圆整到下周的第一天，否则返回当周第一天	
W	W	每月1日作为本月第一周的第一天，从每周的第五天开始圆整到下周的第一天，否则返回当周第一天	
Day	DDD, DD, J	返回天，从一天的中午`12:00`开始圆整到下一天0分0秒，否则返回该天的0分0秒	
Start day of the week	DAY, DY, D	星期日作为每周第一天，从每周的第五天开始圆整到下周的第一天，否则返回当周第一天	

类型	有效格式	返回值	备注
Hour	HH, HH12, HH24	返回小时，从每小时的第三十分钟开始圆整到下一小时0分0秒，否则返回该小时的0分0秒	
Minute	MI	返回分钟，从每分钟的30秒开始圆整到下一分钟0秒，否则返回该分钟0秒	
null	null	null	

当不指定fmt时，默认为'DD'。

对于类型IW、W、 WW、Start day of the week，当输入值的小时数大于等于12时，进位可跨天、跨月、跨年。

round_number

指定截取的小数位数，须为数值型数据，或可转换为NUMBER类型的其他类型数据。

round_number为与expr相同的通用表达式，当round_number的值为NULL时，函数返回NULL。

当round_number的值为正数，且该值大于expr的小数位数时，函数返回expr的原值。

当round_number的值为负数时，表示向小数点左边截取，例如"ROUND(123.456,-2)"，返回的是123四舍五入至小数点左侧第2位的数字，即100。此时当round_number的绝对值大于小数点左边的整数位数时，函数返回0，例如"ROUND(123.456,-4)=0"。

示例

```
SELECT ROUND(SYSDATE-10000) res FROM DUAL;
RES
--------------------------------
1995-02-05

SELECT ROUND(SYSDATE-10000) res FROM DUAL;
RES
--------------------------------
1995-02-05

SELECT * FROM numbers_nobit;
     NUMBERA NUMBERB  NUMBERC               NUMBERD     NUMBERE     NUMBERF     NUMBERG
------------ ------- -------- -------------------- ----------- ----------- -----------
          -5      55     5555   5555555555555555555  5.555E+000  5.556E+000         555

SELECT ROUND(numbere,numbera) round1,
ROUND(numbere,numberb) round2,
ROUND(numberf,15660) round3,
ROUND(numbere,'17') round4,
ROUND(numberf,-1) round5
FROM numbers_nobit;
     ROUND1      ROUND2      ROUND3      ROUND4      ROUND5
---------- ----------- ----------- ----------- -----------
          0  5.55499983  5.55555556  5.55499983          10


SELECT TYPEOF(ROUND(numbere,numbera)) type1,
TYPEOF(ROUND(numbere,numberb)) type2,
TYPEOF(ROUND(numberf,15660)) type3,
TYPEOF(ROUND(numbere,'17')) type4,
TYPEOF(ROUND(numberf,-1)) type5
FROM numbers_nobit;
TYPE1    TYPE2    TYPE3     TYPE4     TYPE5
-------- -------- --------- --------- --------
number   number   number    number    number
```

ROWIDTOCHAR

```
rowidtochar::= ROWIDTOCHAR "(" expr ")"
```

ROWIDTOCHAR函数将expr表示的ROWID类型数据转换为VARCHAR类型的字符串数据。

本函数遵循如下规则：

该函数不支持列式计算。

expr

expr的值须为ROWID类型数据，或者为字符型数据。对于其他类型，函数返回类型不支持。

当expr为NULL时函数返回NULL。

示例（HEAP表）

```
SELECT ROWIDTOCHAR('2368:0:0:3164:0') rowid1 FROM DUAL;
ROWID1
-----------------
2368:0:0:3164:0
```

ROW_NUMBER

```
row_number::= ROW_NUMBER "("")" OVER "(" [query_partition_clause] order_by_clause ")"
```

ROW_NUMBER为窗口函数，其语法描述及约束与RANK，DENSE_RANK函数一致，区别在于排序时对并列值的处理：

- RANK函数列出相同并列值，并对下一顺序值跳号，例如1，2，3，3，3，6，7，8.....
- DENSE_RANK函数列出相同并列值，并对下一顺序值不跳号，例如1，2，3，3，3，4，5，6.....
- ROW_NUMBER函数不列出并列值，而是根据返回的结果递增，不跳号，例如1，2，3，4，5，6，7，8.....

partition by与order by的参数支持除LOB、JSON、XMLTYPE、UDT外的其它数据类型。

示例

```
--sales_info_range表中包含如下字段和数据
SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0101   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  11    0101   11001              20         300
2015  03    0102   11001              20         300
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600

--按branch进行分组，并在组内按amount进行排序
SELECT branch,amount,year,month,quantity,
       ROW_NUMBER() OVER (PARTITION BY branch ORDER BY amount) rownumber_res
FROM sales_info
WHERE product='11001'
ORDER BY branch,quantity;
BRANCH      AMOUNT YEAR  MONTH    QUANTITY         ROWNUMBER_RES
------ ----------- ----- ----- ----------- --------------------
0101           300 2015  11             20                    1
0101           600 2021  05             40                    2
0102           300 2015  03             20                    2
0102           300 2000  12             20                    1
0201           500 2001  01             30                    1
0402           300 2021  10             20                    1
```

RPAD

```
rpad::= RPAD "(" expr "," pad_length ["," pad_character] ")"
```

RPAD从右边对expr表示的字符串进行指定字符、指定长度的填充，得到一个新字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr为NCLOB、NCHAR、NVARCHAR类型时，返回值为NVARCHAR类型，其余场景返回值为VARCHAR类型。

pad_length

指定填充后字符串的长度，pad_length为与expr相同的通用表达式，须为除BIT外数值型数据，或可转换为NUMBER类型的其他类型数据，取值范围[-9223372036854775808,32000]。

- 当pad_length的值为NULL，0或负数时，函数返回NULL。
- 当pad_length的值为小数时，函数截断其小数位保留整数位。
- 当此长度小于等于expr字符串长度时，函数返回的是从左至右对expr进行此长度截取的子字符串。

pad_character

指定要填充的内容，pad_character为与expr相同的通用表达式，须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。

- 当pad_character的值为NULL时，函数返回NULL。
- 指定此值时，函数循环从左至右读取pad_character的字符并填充到expr的左边，直到满足pad_length的长度要求为止。
- 未指定此值时，默认填充空格。

示例

```
SELECT RPAD('深圳',16,'NIHAO') AS res FROM DUAL;
RES
----------------------
深圳NIHAONIHAONIHA

SELECT RPAD('深圳',1,'NIHAO') AS res FROM DUAL;
RES
-----
深

SELECT RPAD('深圳',5,'NIHAO') AS res FROM DUAL;
RES
-------------
深圳NIH

SELECT RPAD('深圳',16) AS res1 FROM DUAL;
RES1
----------------------
深圳

SELECT RPAD('深圳',1.999) AS res1 FROM DUAL;
RES1
-----
深

SELECT RPAD('深圳','1.999') AS res1 FROM DUAL;
RES1
```

```
-----
深
```

RTRIM

```
rtrim::= RTRIM "(" expr ["," trim_character] ")"
```

RTRIM函数从右往左删除expr表示的字符串里与trim_character匹配的内容，得到一个新的子字符串。

expr的值须为字符型，或可转换为字符型的其他类型，但不允许为NCLOB类型。

对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。

本函数返回值类型遵循如下规则：

- 当expr为CLOB类型时，返回CLOB数据类型字符串。
- 当expr为NCHAR或NVARCHAR时，返回NVARCHAR数据类型字符串。
- 当expr的值为NULL时，函数返回NULL。
- 其余情况均返回varchar类型字符串。

trim_character

指定要匹配的内容，须为字符型，或可转换为字符型的其他类型。

trim_character为与expr相同的通用表达式，当trim_character的值为NULL时，函数返回NULL。

指定此值时，函数将从右往左对比expr与trim_character中的字符，当expr中的字符在trim_character中则将其删除，直到遇到从右往左的第一个不在trim_character中的字符后停止。

不指定此值时，默认的匹配内容为一个空格。

示例

```
--从右开始的3个'3'均能在'33'中找到，因此被删除，直到遇到'1'不能在'33'中找到，停止匹配并返回剩余字符串
SELECT RTRIM('33311333','33') res FROM DUAL;
RES
---------
33311

--从右开始的所有字符均能在'313'中找到，全部被删除，返回空字符串
SELECT RTRIM('33311333','313') res FROM DUAL;
RES
---------


--只有一个参数时按空格执行匹配删除
SELECT RTRIM('       11333         ') res FROM DUAL;
RES
-------------
       11333
```

SCN_TO_TIMESTAMP

```
scn_to_timestamp::= SCN_TO_TIMESTAMP "(" expr ")"
```

SCN_TO_TIMESTAMP函数将expr表示的SCN号转换为时间戳数据。

expr的值须为BIGINT类型数据，或者可转换为BIGINT的其他整数类型数据（具体包括TINYINT、SMALLINT、INT、BIGINT四种数据类型），否则返回类型不支持。

函数不支持传递null作为参数。

示例

```
SELECT SCN_TO_TIMESTAMP(247677771776000000) timestamp1,
SCN_TO_TIMESTAMP(555+3) timestamp2 FROM DUAL;
TIMESTAMP1                     TIMESTAMP2
------------------------------ --------------------------
2021-11-30 20:43:26.000000     2020-01-01 00:00:00.000000
```

SIGN

```
sign::= SIGN "(" expr ")"
```

SIGN函数返回expr表示的数值的符号，包括1、-1、0。

expr的值须为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

当expr的值为正数时，函数返回1；当expr的值为负数时，函数返回-1；当expr的值为0时，函数返回0。

对于浮点数据里的Nan、-Inf、Inf特殊值：

- SIGN(Nan) = 1
- SIGN(Inf) = 1
- SIGN(-Inf) = -1

示例

```
DROP TABLE IF EXISTS number_fd;
CREATE TABLE number_fd(numberf FLOAT, numberd DOUBLE);
INSERT INTO number_fd VALUES('NaN','inf');
INSERT INTO number_fd VALUES('',5.5333333323);
INSERT INTO number_fd VALUES('-inf','-4.322323');
COMMIT;

SELECT SIGN(numberf) sign1, SIGN(numberd) sign2 FROM number_fd;
      SIGN1       SIGN2
----------- -----------
          1           1
                      1
         -1          -1
```

SIN

```
sin::= SIN  "(" expr ")"
```

SIN函数返回给定参数的正弦值，参数为以弧度表示的角度，大小本身无限制（只受限于其所属数据类型所规定范围），函数返回一个大小在区间[-1,1]的DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT SIN(30*3.1415926/180) res FROM DUAL;
                 RES
--------------------
           5.0E-001

SELECT SIN(90*3.1415926/180) res FROM DUAL;
                 RES
--------------------
           1.0E+000

SELECT SIN(45*3.1415926/180) res FROM DUAL;
                 RES
--------------------
          7.071E-001
```

SINH

```
sinh::= SINH  "(" expr ")"
```

SINH函数返回其参数的双曲正弦值，参数大小本身无限制（只受限于其所属数据类型所规定范围），函数返回一个DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT SINH(2) res FROM DUAL;
RES
-----------
3.627E+000

SELECT SINH('2') res FROM DUAL;
RES
-----------
3.627E+000

SELECT SINH(b'10') res FROM DUAL;
RES
-----------
3.627E+000
```

SOUNDEX

```
soundex::= SOUNDEX "(" expr ")"
```

SOUNDEX函数返回其参数expr通用表达式的soundex值，expr须为字符型，或可以转换为字符型的其他类型。函数将返回一个VARCHAR/NVARCHAR类型的字符串。 soundex值为英语发音下单词的特殊缩略。

本函数遵循如下规则：

- 该函数不支持列式计算。
- 当expr为NCHAR、NVARCHAR和NCLOB时，返回值为NVARCHAR类型；当expr为其他类型时，返回值为VARCHAR类型。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr隐式转换成字符型后的值中不包含英文字符时，函数返回NULL。
- 当expr为NULL时，函数返回NULL。

示例（HEAP表）

```
SELECT SOUNDEX(NULL) FROM dual;

SOUNDEX(NULL)
-------------

SELECT SOUNDEX('SSHS') FROM dual;

SOUNDEX('SSHS')
---------------
       S200
SELECT SOUNDEX('SMYTHE') FROM dual;

SOUNDEX('SMYTHE')
-----------------
       S530
SELECT SOUNDEX('SMITH') FROM dual;

SOUNDEX('SMITH')
----------------
       S530
```

SPLIT

```
split::= SPLIT "(" expr "," delimiter "," n ")"
```

SPLIT函数将expr表示的字符串按照指定的分割符delimiter分割为若干个子字符串，返回第n个子字符串。

expr

expr的值须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- 当expr为CHAR类型时，系统截断其尾部空格。

delimiter

分割符，delimiter为与expr相同的通用表达式，须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当delimiter的值为NULL时，函数返回NULL。
- 当delimiter为CHAR类型时，若为非空格串，系统截断其尾部空格。
- 当delimiter为CHAR类型时为空格串，系统截断并保留一个空格。

n

指定返回的子字符串序号，n为与expr相同的通用表达式，须为数值型，或可转换为NUMBER的字符型数据，取值范围[-2147483648,2147483647]。

- 对于带有小数的数据，小数部分截断，保留整数位。
- 当n的值为NULL时，函数返回NULL。
- 当n的值为正数时，表示按从左往右的顺序。
- 当n的值为负数时，表示按从右往左的顺序。
- 当n的绝对值大于分割后的子字符串个数，函数将返回NULL。
- 当n为0时，函数将会报错。

示例

```
SELECT SPLIT('a,b,c', ',', 1) a
,SPLIT('a,b,c', ',', 4) b
,SPLIT('广东省深圳', '省', 1) c
,SPLIT('a,b,c', ',' , 2.9) d
FROM DUAL;
A     B     C         D
----- ----- --------- -----
a           广东        b
```

SQLCODE

```
sqlcode::= SQLCODE ["("")"]
```

SQLCODE函数是用来返回当前的错误码。

- SQLCODE是一个函数，没有参数，可以写为`SQLCODE`及`SQLCODE()`。
- SQLCODE只能用在过程体中，不能用于普通SQL语句中，下面为错误用法：

```
INSERT INTO TABLE VALUES (sqlCode)
```

- SQLCODE在过程体的异常句柄中，得到的是当前错误码值；在异常句柄之外，则返回的是0。

示例

```
DECLARE
    past_due EXCEPTION;
    PRAGMA EXCEPTION_INIT (past_due, 30103);
BEGIN
    RAISE past_due;

    EXCEPTION
    WHEN past_due THEN
        DBMS_OUTPUT.PUT_LINE(SQLCODE);
END;
/

--result
30103
```

SQLERRM

```
sqlerrm::= SQLERRM "("[option errCode]")"
```

SQLERRM用来返回错误码对应的错误描述信息。

- SQLERRM是一个函数，其参数个数为0个或者1个。
- SQLERRM只能用在过程体中，不能用于普通SQL语句中，下面为错误用法：

```
INSERT INTO TABLE VALUES (sqlErrm)
```

- SQLERRM在异常句柄之外，当没有参数时，返回的都是 "YAS-00000 normal, successful completion"；有参数时，则根据参数去匹配错误码信息。
- SQLERRM在异常句柄中，如果没有设置参数，则返回当前错误码对应的错误信息，同时会去掉格式符；如果设置了参数，则会返回参数对应的错误信息，并去掉格式符；但是设置的参数不一定是内部有的或者用户自定义的，因此可继续细化。

示例

```
DECLARE
BEGIN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/

--result
YAS-00000 normal, successful completion
```

SQRT

```
sqrt::= SQRT "(" expr ")"
```

SQRT函数计算expr表示的数据的平方根。其返回类型为：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER、CHAR、VARCHAR、NCHAR或NVARCHAR类型时，返回NUMBER。
- 当expr的值为FLOAT类型时，返回FLOAT。
- 当expr的值为DOUBLE类型时，返回DOUBLE。
- 当expr的值为NULL时，返回NULL。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为负数时，函数返回Out or range错误，为负数且数据类型为float或double时，函数返回无效数字Nan。

示例

```
--在手册首页中列示的numbers_nobit表
SELECT * FROM numbers_nobit;
     NUMBERA NUMBERB  NUMBERC               NUMBERD     NUMBERE     NUMBERF     NUMBERG
------------ ------- -------- -------------------- ----------- ----------- -----------
          -5      55     5555   5555555555555555555  5.555E+000  5.556E+000         555

SELECT SQRT(numberb) sqrt1,
SQRT(numberc) sqrt2,
SQRT(numberd) sqrt3,
SQRT(numbere) sqrt4,
SQRT(numberg) sqrt5
FROM numbers_nobit;
      SQRT1       SQRT2       SQRT3       SQRT4       SQRT5
----------- ----------- ----------- ----------- -----------
 7.41619849  74.5318724  2357022604  2.357E+000   23.558438


SELECT TYPEOF(SQRT(numberb)) type1,
TYPEOF(SQRT(numberc)) type2,
TYPEOF(SQRT(numberd)) type3,
TYPEOF(SQRT(numbere)) type4,
TYPEOF(SQRT(numberg)) type5
FROM numbers_nobit;
TYPE1       TYPE2       TYPE3      TYPE4     TYPE5
----------- ----------- ---------- --------- ---------
number      number      number     float     number
```

STDDEV

```
stddev::= STDDEV "(" [DISTINCT|ALL] expr ")"
```

STDDEV函数计算expr的值的样本标准差。

当给定参数只有一行数据时，STDDEV函数的计算结果为0。

STDDEV函数的返回值类型有以下几种情况：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型时，返回NUMBER类型。
- 当expr的值为FLOAT、DOUBLE类型时，返回与参数一致的FLOAT、DOUBLE类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为字面量、变量、常量时，返回0。
- 当expr的值为其他类型且无法转换为NUMBER类型时，不执行计算并返回类型转换错误。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型。对于其他类型，函数返回类型不支持。

在多行计算中，系统忽略数值为空的行，当所有行均为空时，计算结果为NULL。

DISTINCT

表示过滤掉输入的重复数据后，进行样本标准差计算。

ALL

默认值，表示不对表达式输入的重复数据进行过滤，直接进行样本标准差计算。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME           DHQ             EMPLOYEE_COUNT
------- ------------------- --------------- --------------
01      华东                Shanghai
02      华西                Chengdu                    300
03      华南                Guangzhou                  400
04      华北                Beijing                    300
05      华中                Wuhan

--计算员工数量的样本标准差,为空的行将被忽略,该语句等同于SELECT STDDEV(ALL employee_count) FROM area
SELECT STDDEV(employee_count) res FROM area1;
                 RES
--------------------
          57.7350269

--只有一行记录时，样本标准差为0
SELECT STDDEV(employee_count) res FROM area1 WHERE area_no='02';
                 RES
--------------------
                   0

--除去重复的员工数量后计算样本标准差
SELECT STDDEV(DISTINCT employee_count) res FROM area1;
                 RES
--------------------
          70.7106781
```

STDDEV_POP

```
stddev_pop::= STDDEV_POP "(" expr ")"
```

STDDEV_POP函数计算expr的值的总体标准差。

STDDEV_POP函数的返回值类型有以下几种情况：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型时，返回NUMBER类型。
- 当expr的值为FLOAT、DOUBLE类型时，返回与参数一致的FLOAT、DOUBLE类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为字面量、变量、常量时，返回0。
- 当expr的值为其他类型且无法转换为NUMBER类型时，不执行计算并返回类型转换错误。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型。对于其他类型，函数返回类型不支持。

在多行计算中，系统忽略数值为空的行，当所有行均为空时，计算结果为NULL。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME          DHQ            EMPLOYEE_COUNT
------- ------------------ -------------- --------------
01      华东               Shanghai
02      华西               Chengdu                   300
03      华南               Guangzhou                 400
04      华北               Beijing                   300
05      华中               Wuhan

--计算员工数量的总体标准差,为空的行将被忽略
SELECT STDDEV_POP(employee_count) res FROM area1;
                RES
--------------------
         47.1404521
```

STDDEV_SAMP

```
stddev_samp::= STDDEV_SAMP "(" expr ")"
```

STDDEV_SAMP函数计算expr的值的样本标准差。

当给定参数只有一行数据时，STDDEV_SAMP函数的计算结果为NULL。

STDDEV_SAMP函数的返回值类型有以下几种情况：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型时，返回NUMBER类型。
- 当expr的值为FLOAT、DOUBLE类型时，返回与参数一致的FLOAT、DOUBLE类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为字面量、变量、常量时，返回0。
- 当expr的值为其他类型且无法转换为NUMBER类型时，不执行计算并返回类型转换错误。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型。对于其他类型，函数返回类型不支持。

在多行计算中，系统忽略数值为空的行，当所有行均为空时，计算结果为NULL。

示例

```
--使用手册首页列示的area1表
SELECT * FROM area1;
AREA_NO AREA_NAME          DHQ            EMPLOYEE_COUNT
------- ------------------ -------------- --------------
01      华东               Shanghai
02      华西               Chengdu                   300
03      华南               Guangzhou                 400
04      华北               Beijing                   300
05      华中               Wuhan

--计算员工数量的样本标准差,为空的行将被忽略

SELECT STDDEV_SAMP(employee_count) res FROM area1;
                 RES
--------------------
          57.7350269

--只有一行记录时，样本标准差为NULL
SELECT STDDEV_SAMP(employee_count) res FROM area1 WHERE area_no='02';
                 RES
--------------------
```

STRING_AGG

```
string_agg::= STRING_AGG "(" [DISTINCT|ALL] string "," separator [order_by_clause] ")"
```

STRING_AGG函数将多行的数据执行拼接操作，并通过分隔符分隔，返回一行CLOB类型的字符串。该函数与GROUP_CONCAT函数实现功能类似。

该函数不支持列式计算。

DISTINCT

计算最终拼接结果时，过滤在同一组内出现的重复的行。

ALL

默认值，表示不过滤重复的行，对所有行都进行拼接。

string

string可以为：

- 通用表达式expr
- 查询列为单列且返回行为单行的子查询

string的值须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换），但不允许为JSON、NVARCHAR、NCHAR和NCLOB类型。

当string的值为NULL时，函数返回NULL。

separator

指定将组内的多行进行拼接时，多行之间加上separator定义的分隔符，separator为character类的常量或常量表达式。

separator必须指定，但可以指定为NULL。

order_by_clause

对组内要CONCAT的string排序，其语法与SELECT语句中描述一致。

当ORDER BY后指定的是常量数字时，表示的是string的顺序值。

示例（HEAP表）

```
--创建exprs_string_agg表，并插入数据
CREATE TABLE exprs_string_agg (id INT,name VARCHAR(50),money FLOAT);
INSERT INTO exprs_string_agg
VALUES  (1,'小东',10000),(2,'小明',46450),
        (3,'小红',46450 ),(4,'小东',14465),
        (5,'小明',46450),(6,'小东',46450);

--未指定GROUP BY时，将所有行CONCAT，得到一行结果
SELECT STRING_AGG(MONEY,',') AS money FROM exprs_string_agg;
MONEY
-----------------------------------------------------------------
1.0E+004,4.645E+004,4.645E+004,1.4465E+004,4.645E+004,4.645E+004

--group by后各组的多行数据分别CONCAT成一行，得到按组的多行结果
SELECT NAME,STRING_AGG(money,',') AS money FROM exprs_string_agg GROUP BY name;
NAME                      MONEY
------------------------- ----------------------------------
小东                      1.0E+004,1.4465E+004,4.645E+004
小明                      4.645E+004,4.645E+004
小红                      4.645E+004

--使用order by子句排序
SELECT NAME,STRING_AGG(money,',' ORDER BY id DESC) AS money FROM exprs_string_agg GROUP BY name;
```

```
NAME                      MONEY
------------------------- ----------------------------------
小东                      4.645E+004,1.4465E+004,1.0E+004
小明                      4.645E+004,4.645E+004
小红                      4.645E+004

--使用DISTINCT关键字去重
SELECT STRING_AGG(DISTINCT name,',') AS names FROM exprs_string_agg;
NAMES
-------------------------------------------------------------
小东,小明,小红
```

STRING_TO_ARRAY

```
string_to_array::= STRING_TO_ARRAY "(" src_string "," split_string [ "," replace_string ] ")"
```

STRING_TO_ARRAY函数将字符串src_string以split_string作为分隔符进行切分生成一个数组，当被切分的成员与replace_string相同时，则将被切分成员替换为NULL后再生成数组（数组变量，数组对象，数组类型）并返回。

该函数不支持列式计算。

本函数遵循如下规则：

- 最终返回的数组为隐式定义的数组变量，其成员数量上限最大为INT类型上限（2147483647），其成员数据类型为VARCHAR(n)，n的值为生成数组中的最长成员的长度。
- 该函数可以在PL中作为数组的初始化函数进行使用，但存在赋值对象限制，被赋值对象需要能承载该函数生成的数组，即其成员上限需大于等于该函数返回数组的成员上限，成员长度需大于等于该函数返回数组的长度。

src_string

通用表达式，须为字符型，或可隐式转换为字符型的其他类型（LOB类型支持隐式转换）。当src_string为NULL时，函数返回NULL。

split_string

通用表达式，须为字符型，或可隐式转换为字符型的其他类型（LOB类型支持隐式转换）。当split_string为NULL时，函数将src_string按每个单字符逐一切分。

replace_string

通用表达式，须为字符型，或可隐式转换为字符型的其他类型（LOB类型支持隐式转换）。

示例

```
SET serveroutput ON

DECLARE
  TYPE arr_type IS VARRAY(10) OF CHAR(10);
  a arr_type;
BEGIN
  a := STRING_TO_ARRAY('a,b,c,d,e', ',');
  FOR i IN 1 .. a.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(i||' is: '||a(i));
  END LOOP;
END;
/
1 is: a
2 is: b
3 is: c
4 is: d
5 is: e

DECLARE
  TYPE arr_type IS VARRAY(10) OF CHAR(5);
  --使用STRING_TO_ARRAY函数对数组初始化
  a arr_type := STRING_TO_ARRAY(',a,b,c,', ',','b');
BEGIN
  FOR i IN 1 .. a.COUNT LOOP
    IF a(i) IS NULL THEN
      DBMS_OUTPUT.PUT_LINE(i||' is NULL');
    ELSE
      DBMS_OUTPUT.PUT_LINE(i||' is: '||a(i));
    END IF;
  END LOOP;
END;
/
1 is NULL
2 is: a
```

```
3 is NULL
4 is: c
5 is NULL
```

STRPOS

```
strpos::= STRPOS "(" expr "," substring ")"
```

STRPOS函数在expr表示的字符串中从左向右查找另一个给定的字符串substring，返回字符串substring在字符串expr中第一次出现的位置，为一个INT类型的数字。当未找到匹配值时，返回0。

expr

expr的值须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- 当expr的值为CHAR类型时，系统截断其尾部空格后进行匹配。

substring

需查找的字符串表达式，substring为与expr相同的通用表达式，须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当substring的值为NULL时，函数返回NULL。
- 当substring的值为CHAR类型时，系统截断尾部空格后进行匹配，若截断后长度为0，函数将返回1。

示例

```
SELECT STRPOS('abcdef', 'abc') a
,STRPOS('abcdef', 'def') b
,STRPOS('广东深圳', '深圳') c
,STRPOS('  ', '  ') d
,STRPOS(NULL, NULL) e
FROM DUAL;
           A            B            C            D            E
------------ ------------ ------------ ------------ ------------
           1            4            3            1
```

SUBSTR

```
substr::= SUBSTR "(" expr "," pos ["," len] ")"
```

SUBSTR函数提取expr表示的字符串里指定位置和指定长度的子字符串。

本函数遵循如下规则：

expr

expr的值须为字符型，或可转换为字符型的其他类型，但不能为XMLTYPE类型。

- 当expr的值为NULL时，函数返回NULL。
- 当expr的值为CLOB类型时，返回值为CLOB类型；当expr的值为NCLOB类型时，返回值为NCLOB类型；当expr的值为NCHAR/NVARCHAR类型时，返回值为NVARCHAR类型；当expr值为其他类型时，返回值为VARCHAR类型。
- 对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。

pos

表示从pos值指定位置开始提取字符串，pos值须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。

- 当其值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数截断其小数位保留整数位。
- 当其值为浮点类型时，函数将其奇进偶舍至整数。
- pos为与expr相同的通用表达式，当pos的值为NULL时，函数返回NULL。
- 当pos值为正数时，表示从字符串的头部开始确定起始位置；当pos值为负数时，表示从字符串的尾部开始确定起始位置；当pos值为0时，等同于1。
- 当pos绝对值超过字符串的长度时，函数返回NULL。

len

表示提取len值指定长度的字符串，len值须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。

- 当其值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数截断其小数位保留整数位。
- 当其值为浮点类型时，函数将其奇进偶舍至整数。
- len为与expr相同的通用表达式，当len的值为NULL时，函数返回NULL。
- 当不指定len，或len值大于从pos值指定位置开始到字符串结尾的长度时，函数返回从pos值指定位置开始到字符串结尾的子字符串。
- 当len为0或负数时，函数返回NULL。

示例

```
SELECT SUBSTR('abcdefg', 3) a
,SUBSTR('abcdefg', 3, 2) b
,SUBSTR('abcdefg', -3) c
,SUBSTR('abcdefg', 3, -2) d
FROM DUAL;
A         B     C     D
--------- ----- ----- -----
cdefg     cd    efg

SELECT SUBSTR('abcdefg', CAST(2.5 AS NUMBER), CAST(1.5 AS NUMBER)) a
,SUBSTR('abcdefg', '2.5', '1.5') b
,SUBSTR('abcdefg', CAST(2.5 AS FLOAT), CAST(1.5 AS FLOAT)) c
,SUBSTR('abcdefg', CAST(2.5 AS DOUBLE), CAST(1.5 AS DOUBLE)) d
FROM DUAL;
A         B     C         D
--------- ----- --------- ---------
b         b     bc        bc
```

SUBSTRB

```
substrb::= SUBSTRB "(" expr "," pos ["," len] ")"
```

SUBSTRB函数提取expr表示的字符串里指定位置和指定字节长度的子字符串。

本函数遵循如下规则：

- 本函数不支持列式计算。

expr

expr的值须为字符型，或可转换为字符型的其他类型，但不能为XMLTYPE、NCLOB类型。

- 当expr的值为NULL时，函数返回NULL。
- 当expr的值为RAW类型时，返回值为RAW类型。当expr值为NCHAR、NVARCHAR类型时，返回值为NVARCHAR类型。当expr值为其他类型时，返回值为VARCHAR类型。
- expr不支持32000字节以上的LOB类型数据。

pos

表示从pos值指定位置开始提取字符串，pos为与expr相同的通用表达式。pos值须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。当pos值为正数时，表示从前往后确定起始位置，当pos值为负数时，表示从后向前确定起始位置。

- 当其值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数截断其小数位保留整数位。
- 当其值为浮点类型时，函数将其奇进偶舍至整数。
- 当pos的值为NULL时，函数返回NULL。
- 当pos值为0时，等同于1。
- 当pos绝对值超过字符串的长度时，函数返回NULL。

len

表示提取len值指定字节长度的字符串，len为与expr相同的通用表达式，len值须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。

- 当不指定len，或len值大于从pos值指定位置开始到字符串结尾的长度时，函数返回从pos值指定位置开始到字符串结尾的子字符串。
- 当其值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数截断其小数位保留整数位。
- 当其值为浮点类型时，函数将其奇进偶舍至整数。
- 当len的值为NULL时，函数返回NULL。
- 当len为0或负数时，函数返回NULL。

示例（HEAP表）

```
SELECT SUBSTRB('中abc国', 4) a
,SUBSTRB('中abc国', 3, 4) b
,SUBSTRB('中abc国', -3) c
,SUBSTRB('中abc国', 3, -4) d
FROM DUAL;

A             B     C             D
------------- ----- ------------- -----
abc国         abc   国


SELECT SUBSTRB('abcdefg', CAST(2.5 AS NUMBER), CAST(1.5 AS NUMBER)) a
,SUBSTRB('abcdefg', '2.5', '1.5') b
```

```
,SUBSTRB('abcdefg', CAST(2.5 AS FLOAT), CAST(1.5 AS FLOAT)) c
,SUBSTRB('abcdefg', CAST(2.5 AS DOUBLE), CAST(1.5 AS DOUBLE)) d
FROM DUAL;
A         B     C         D
--------- ----- --------- ---------
b         b     bc        bc
```

SUBSTRING

```
substring::= SUBSTRING "(" ((expr "," pos ["," len])|(expr FROM pos [FOR len])) ")"
```

SUBSTRING函数提取expr表示的字符串中指定位置指定长度的子字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- 当expr的值为CLOB类型时，返回值为CLOB类型；当expr的值为NCLOB类型时，返回值为NCLOB类型；当expr值为NCHAR、NVARCHAR类型时，返回值为NVARCHAR类型；当expr值为其他类型时，返回值为VARCHAR类型。
- 对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。
- expr不能为XMLTYPE类型。

pos

表示从pos值指定位置开始提取字符串，pos为与expr相同的通用表达式，须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。

- 当pos的值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数四舍五入进行取整。
- 当pos的值为浮点类型时，函数将其奇进偶舍至整数。
- 当pos的值为NULL时，函数返回NULL。
- 当pos的值为正数时，表示从字符串的头部开始确定起始位置；当pos值为负数时，表示从字符串的尾部开始确定起始位置。
- 当pos的值等于0或其绝对值超过字符串的长度时，函数返回NULL。

len

表示提取len值指定长度的字符串，len为与expr相同的通用表达式，须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。

- 当len的值为带有小数的NUMBER类型（或转换后为NUMBER类型）时，函数四舍五入进行取整。
- 当len的值为浮点类型时，函数将其奇进偶舍至整数。
- 当len的值为NULL时，函数返回NULL。
- 当不指定len，或len值大于从pos值指定位置开始到字符串结尾的长度时，函数返回从pos值指定位置开始到字符串结尾的子字符串。
- 当len的值为0或负数时， 函数返回NULL。

示例

```
SELECT SUBSTRING('abcdefg', 3) a
,SUBSTRING('abcdefg', 3, 2) b
,SUBSTRING('abcdefg', -3) c
,SUBSTRING('abcdefg', 3, -2) d
FROM DUAL;
A         B     C     D
--------- ----- ----- -----
cdefg     cd    efg

SELECT SUBSTRING('abcdefg' FROM 3) a
,SUBSTRING('abcdefg' FROM 3 FOR 2) b
,SUBSTRING('abcdefg' FROM -3) c
,SUBSTRING('abcdefg' FROM 3 FOR -2) d
FROM DUAL;
A         B     C     D
```

```
--------- ----- ----- -----
cdefg     cd    efg

SELECT SUBSTRING('abcdefg', CAST(2.5 AS NUMBER), CAST(1.5 AS NUMBER)) a
,SUBSTRING('abcdefg', '2.5', '1.5') b
,SUBSTRING('abcdefg', CAST(2.5 AS FLOAT), CAST(1.5 AS FLOAT)) c
,SUBSTRING('abcdefg', CAST(2.5 AS DOUBLE), CAST(1.5 AS DOUBLE)) d
FROM DUAL;
A         B     C         D
--------- ----- --------- ---------
cd        b     bc        bc

SELECT SUBSTRING('abcdefg' FROM CAST(2.5 AS NUMBER) FOR CAST(1.5 AS NUMBER)) a
,SUBSTRING('abcdefg' FROM '2.5' FOR '1.5') b
,SUBSTRING('abcdefg' FROM CAST(2.5 AS FLOAT) FOR CAST(1.5 AS FLOAT)) c
,SUBSTRING('abcdefg' FROM CAST(2.5 AS DOUBLE) FOR CAST(1.5 AS DOUBLE)) d
FROM DUAL;
A         B     C         D
--------- ----- --------- ---------
cd        b     bc        bc
```

SUBSTRING_INDEX

```
substring_index::= SUBSTRING_INDEX "(" expr "," delim "," count ")"
```

SUBSTRING_INDEX函数提取expr表示的字符串中在分隔符指定次数出现之前的子字符串。

expr

expr的值须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- 当expr为NCAHR/NVARCHAR类型时，返回值为NVARCHAR类型，其余场景返回值为VARCHAR类型。

delim

表示分隔符，用于分割expr。delim为与expr相同的通用表达式，须为字符型，或除JSON、LOB、XMLTYPE类型外可转化为字符型的其他类型。

- 若使用非字符型的形式输入`(0,1)`之间的小数，转换后的delim将去除小数点前的0进行匹配。
- 当delim的值为NULL时，函数返回NULL。
- 当delim在expr中未匹配到时，函数将返回expr全部内容。

count

表示分隔符出现的次数，用于定位分割的终止位置。count为与expr相同的通用表达式，须为除BIT外数值型数据，或可转换为NUMBER的其他类型数据，取值范围[-2147483648,2147483647]。

- 当count的值为带有小数的NUMBER类型时，函数将其四舍五入至整数。
- 当count的值为浮点类型时，函数将其奇进偶舍至整数。
- 当count的值为NULL或0时，函数返回NULL。
- 当count的值为正数，则返回分隔符第count次出现时左侧的所有内容（从左侧开始计数）。
- 当count的值为负数，则返回分隔符第count次出现时右侧的所有内容（从右侧开始计数）。
- 当count的值超出delim在expr中的出现次数时，函数将返回expr全部内容。

示例

```
SELECT SUBSTRING_INDEX('192.168.0.1',0.1, 1) a
,SUBSTRING_INDEX('192.168.0.1','.', 2) b
,SUBSTRING_INDEX('192.168.0.1','.', 3) c
,SUBSTRING_INDEX('192.168.0.1','.', 4) d
FROM DUAL;
A     B         C             D
----- --------- ------------- -------------
192   192.168   192.168.0     192.168.0.1

SELECT SUBSTRING_INDEX('192.168.0.1','.',-1) a
,SUBSTRING_INDEX('192.168.0.1','.',-2) b
,SUBSTRING_INDEX('192.168.0.1','.',-3) c
,SUBSTRING_INDEX('192.168.0.1','.',-4) d
FROM DUAL;
A     B     C         D
----- ----- --------- -------------
1     0.1   168.0.1   192.168.0.1

SELECT SUBSTRING_INDEX('192.168.0.1','.',CAST(1.5 AS NUMBER)) a
,SUBSTRING_INDEX('192.168.0.1','.',CAST(2.5 AS NUMBER)) b
,SUBSTRING_INDEX('192.168.0.1','.',CAST(1.5 AS FLOAT)) c
,SUBSTRING_INDEX('192.168.0.1','.',CAST(2.5 AS FLOAT)) d
,SUBSTRING_INDEX('192.168.0.1','.',CAST(1.5 AS DOUBLE)) e
```

```
,SUBSTRING_INDEX('192.168.0.1','.',CAST(2.5 AS DOUBLE)) f
FROM DUAL;
A         B             C         D         E         F
--------- ------------- --------- --------- --------- ---------
192.168   192.168.0     192.168   192.168   192.168   192.168

SELECT SUBSTRING_INDEX(NULL,'.',1) a
,SUBSTRING_INDEX('192.168.0.1',NULL,1) b
,SUBSTRING_INDEX('192.168.0.1','.',NULL) c
,SUBSTRING_INDEX('192.168.0.1','.',0) d
FROM DUAL;
A     B     C     D
----- ----- ----- -----
```

SUM

```
sum::= SUM "(" [DISTINCT|ALL] expr ")" [OVER "(" analytic_clause ")"]

analytic_clause::= "(" [query_partition_clause] [order_by_clause [windowing_clause]] ")"
```

SUM函数计算给定参数expr的数值和，其返回值类型如下：

expr类型	返回值
TINYINT、SMALLINT	BIGINT
INT、BIGINT	NUMBER
FLOAT	FLOAT
DOUBLE	DOUBLE

当参数为其他类型且无法转换为上述类型时，不执行计算并返回类型转换错误或是类型不支持。

在单行计算中，当expr的值为NULL时，函数返回NULL。

在多行计算中，函数将忽略expr值为空的行，当所有行均为空时，计算结果为NULL。

聚集函数不可嵌套，因此expr为除聚集函数之外的其他通用表达式，其类型只能是数值型、字符型。

DISTINCT

表示在计算数值和时，过滤掉重复的行。

ALL

默认值，表示对所有行计算数值和。

示例

```
--使用手册首页列示的branches1表

--分区域计算员工总数量，对01、04区域对应员工数量为空的行在计算时被忽略，对02区域对应员工数量只有一行且值为空，此时计算返回NULL结果
SELECT area_no,SUM(employee_count) sum FROM branches1 GROUP BY area_no;
AREA_NO                 SUM
------- -------------------
                         70
01                       70
04                       40
02
05                       40

--分区域计算一个常量的数值和
SELECT area_no,SUM(10) sum FROM branches1 GROUP BY area_no;
AREA_NO         SUM
------- -----------
                 20
01               40
04               40
02               10
05               10
```

OVER

当指定OVER关键字时，SUM将作为窗口函数，并支持滑动窗口，返回多行的数值和。

analytic_clause

窗口函数通用语法。

示例

```
--finance_info表记录了分年、月、机构的收入情况
SELECT year,month,branch,revenue_total FROM finance_info;
YEAR  MONTH BRANCH REVENUE_TOTAL
----- ----- ------ -------------
2001  01    0201            2888
2021  01    0201           28888
2021  01    0101           38888
2021  02    0101           37778

SELECT year,month,
revenue_total curr,
SUM(revenue_total) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info;
YEAR  MONTH        CURR       TONOW
----- ----- ----------- -----------
2001  01           2888        2888
2021  01          28888       28888
2021  01          38888       67776
2021  02          37778      105554

--分年统计每月所有机构的收入和,及年初至今所有机构的收入和
SELECT year,month,
SUM(revenue_total) curr,
SUM(SUM(revenue_total)) OVER (PARTITION BY year ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) tonow
FROM finance_info
GROUP BY year,month;
YEAR  MONTH        CURR       TONOW
----- ----- ----------- -----------
2001  01           2888        2888
2021  01          67776       67776
2021  02          37778      105554
```

SYSDATE

```
sysdate::= SYSDATE ["(" [integer] ")"]
```

SYSDATE函数返回数据库所在的操作系统设置的当前日期，其返回类型为DATE，且与DATE_FORMAT参数所指定格式一致。

以下三种形式的SYSDATE同义，返回结果无差别：

- `SYSDATE`
- `SYSDATE()`
- `SYSDATE(integer)`，integer必须为一个0~6之间的整数字面量。

如果一个SQL语句中出现了多个SYSDATE函数，在该语句执行过程中将只调用一次函数，即保证多个SYSDATE函数返回的是相同一个日期值。

示例

```
SELECT SYSDATE res1,SYSDATE() res2,SYSDATE(5) res3 FROM DUAL;
RES1                    RES2                    RES3
----------------------- ----------------------- -----------------------
2022-11-02 03:07:43     2022-11-02 03:07:43     2022-11-02 03:07:43
```

SYSTIMESTAMP

```
systimestamp::= SYSTIMESTAMP ["(" [integer] ")"]
```

SYSTIMESTAMP函数返回数据库所在的操作系统设置的当前时间戳，其返回类型为TIMESTAMP，且与TIMESTAMP_FORMAT参数所指定格式一致。

SYSTIMESTAMP有以下三种形式：

- `SYSTIMESTAMP`
- `SYSTIMESTAMP()`
- `SYSTIMESTAMP(integer)`，integer必须为一个0~9之间的整数字面量，表示保留的微秒位数，舍去的位按四舍五入。

如果一个SQL语句中出现了多个SYSTIMESTAMP函数，在该语句执行过程中将只调用一次函数，即保证多个SYSTIMESTAMP函数返回的是相同一个时间戳值。

当SYSTIMESTAMP函数参与运算时，其运算规则与TIMESTAMP类型一致，见算术运算符章节描述。

示例

```
SELECT SYSTIMESTAMP res1,SYSTIMESTAMP() res2,SYSTIMESTAMP(9) res3 FROM DUAL;
RES1                          RES2                          RES3
----------------------------- ----------------------------- -----------------------------
2022-11-02 03:11:43.579228    2022-11-02 03:11:43.579228    2022-11-02 03:11:43.579228
```

SYS_CONNECT_BY_PATH

```
SYS_CONNECT_BY_PATH::= SYS_CONNECT_BY_PATH "(" colName "," delimiter ")"
```

SYS_CONNECT_BY_PATH为层次查询场景使用的函数，该函数返回从根节点到当前节点路径上所有节点名为colName的列的值，之间用指定的字符delimiter分隔开。

函数的返回值类型为VARCHAR，返回值最大长度32000，超长就会报错。

该函数不支持列式计算。

colName

该参数为语句中表的列名。

delimiter

字符或字符串分隔符。只能为常量或常量的字符串。

示例（HEAP表）

```
DROP TABLE IF EXISTS area_info;
CREATE TABLE area_info (id INT, area_name VARCHAR(10), father_id INT);
INSERT INTO area_info VALUES(2, '浙江', 0);
INSERT INTO area_info VALUES(571, '杭州', 2);
INSERT INTO area_info VALUES(1, '广东', 0);
INSERT INTO area_info VALUES(755, '深圳', 1);
INSERT INTO area_info VALUES(756, '龙华', 755);
INSERT INTO area_info VALUES(757, '福田', 755);

SELECT  id, father_id, LEVEL,
CONNECT_BY_ROOT area_name AS name,
SYS_CONNECT_BY_PATH(area_name, '/') path
FROM area_info
CONNECT BY PRIOR id = father_id START WITH father_id = 0
ORDER SIBLINGS BY id DESC;
    ID    FATHER_ID    LEVEL NAME         PATH
------ ------------ -------- ------------ ---------------
     2            0        1 浙江         /浙江
   571            2        2 浙江         /浙江/杭州
     1            0        1 广东         /广东
   755            1        2 广东         /广东/深圳
   757          755        3 广东         /广东/深圳/福田
   756          755        3 广东         /广东/深圳/龙华
```

SYS_CONTEXT

```
SYS_CONTEXT::= SYS_CONTEXT "(" namespace "," parameter ["," length] ")"
```

SYS_CONTEXT函数返回当前时刻与namespace关联的parameter值。

函数的返回值类型为VARCHAR，默认长度为256字节。

namespace

该参数须为YashanDB内置的命名空间USERENV，不区分大小写，输入其他值时函数返回NULL。

parameter

指定参数，须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。大小写不敏感。指定的parameter须与namespace相关联，若为非法parameter，将报错。

length

指定返回值的长度。须为数值型或可隐式转换为数值型的其他类型，且必须处于INT类型的值域范围内，否则报错。

系统对length的值按整数处理，对于非整数将进行小数部分截断，保留整数部分。

本函数接受的length合法值在[1,4000]区间，对于不在此范围的其他整数将按默认值256处理。

下表列示USERENV预定义的parameter：

parameter	返回值
AUTHENTICATED_IDENTITY	身份验证中使用的标识 * 经过密码验证的数据库用户：返回数据库用户名，与SCHEMA名称相同 * 用密码文件的系统用户：返回登录名
AUTHENTICATION_METHOD	身份验证方法 * 经过密码或密码文件验证的身份：PASSWORD
CLIENT_INFO	用户会话信息
CLIENT_PROGRAM_NAME	与数据库连接的会话程序的名称
CURRENT_SCHEMA*	当前活动的默认SCHEMA名称
CURRENT_SCHEMAID	当前活动的默认SCHEMAID
CURRENT_USER	当前活动的数据库用户名称
CURRENT_USERID	当前活动的数据库用户ID
DB_NAME	当前数据库名称
HOST	客户端连接的服务器名称
INSTANCE	当前实例的ID
IP_ADDRESS	客户端连接的计算机IP地址
ISDBA	当前用户是否具有DBA权限
LANGUAGE	数据库使用的字符集
YASDB_HOME	YASDB_HOME主目录的完整路径名
OS_USER	启动数据库会话的客户端进程的操作系统用户名
SESSION_USER	登录用户的名称
SESSION_USERID	登录用户的ID

parameter	返回值
SID	会话ID

示例

```
SELECT SYS_CONTEXT('USERENV', 'SESSION_USER') res FROM DUAL;

RES
----------------------------------------------------------------
SYS

SELECT SYS_CONTEXT('USERENV', 'DB_NAME') res FROM DUAL;

RES
----------------------------------------------------------------
yas
```

SYS_EXTRACT_UTC

```
sys_extract_utc::= SYS_EXTRACT_UTC "(" datetime ")"
```

SYS_EXTRACT_UTC函数主要用于将输入的datetime转换成UTC（原格林尼治标准时间）对应的时间并返回返回一个Timestamp类型数据。

本函数遵循如下规则：

- datetime须为Timestamp类型。
- 本函数不支持列式计算。

示例（HEAP表）

```
SELECT SYS_EXTRACT_UTC(TIMESTAMP'2000-02-01 4:00:00') res FROM dual;

RES
----------------------------------------------------------------
2000-01-31 20:00:00.000000
```

SYS_GUID

```
sys_guid::= SYS_GUID "("")"
```

SYS_GUID函数返回一个16字节的随机数，返回类型为RAW类型。

此内置函数中调用的公共函数，linux下是真随机的，Windows下是伪随机的。

示例

```
SELECT SYS_GUID() res FROM DUAL;

RES
--------------------------------
C9FE2ABD165B19CC772A1E914843B994
```

TAN

```
tan::= TAN  "(" expr ")"
```

TAN函数返回给定参数的正切值，参数为以弧度表示的角度，大小本身无限制（只受限于其所属数据类型所规定范围），函数返回一个DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT TAN(0) res FROM DUAL;
        RES
-----------
          0

SELECT TAN(30*3.1415926/180) res FROM DUAL;
                RES
--------------------
          5.774E-001

SELECT TAN(45*3.1415926/180) res FROM DUAL;
                RES
--------------------
           1.0E+000
```

TANH

```
tanh::= TANH  "(" expr ")"
```

TANH函数返回其参数的双曲正切，参数大小本身无限制（只受限于其所属数据类型所规定范围），函数返回一个DOUBLE类型数据。

其中expr的值为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT TANH(1) res FROM DUAL;
RES
-----------
7.616E-001

SELECT TANH('1') res FROM DUAL;
RES
-----------
7.616E-001

SELECT TANH(b'1') res FROM DUAL;
RES
-----------
7.616E-001
```

TIME

```
time::= TIME "(" expr ")"
```

TIME函数对expr的值进行时间部分数值的提取，其返回值类型有以下几种情况：

- 当expr的值为DATE、TIME、TIMESTAMP、字符型时，返回TIME类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为其他类型时，返回类型不支持。

当expr的值为字符型时，其格式必须符合以下规范：

- 字符串类型1：以'yyyy-mm-dd'开头，且至少包含有'yyyy-mm-dd'格式的字符串，需符合年、月、日的一般限制条件，如月份值介于1-12之间、日期值介于1-31之间等。
- 字符串类型2：'hh24:mi:ss.ff'格式的字符串，此字符串可从后向前省略部分，需符合小时、分、秒的一般限制条件，如小时值介于0-23之间、分钟值介于0-59之间等。此时TIME函数会对省略的部分补0。

示例

```
SELECT TIME(MAX(SYSDATE)) res FROM DUAL;
RES
--------------------
22:05:37.000000

SELECT TIME(SYSTIMESTAMP+1) res FROM DUAL;
RES
--------------------
22:05:50.000000

SELECT TIME('2012-3-18') res FROM DUAL;
RES
--------------------
00:00:00.000000

SELECT TIME('2012-3-18 14') res FROM DUAL;
RES
--------------------
14:00:00.000000
```

TIMEDIFF

```
TIMEDIFF::= TIMEDIFF "(" expr1 "," expr2 ")"
```

TIMEDIFF函数用于计算expr1与expr2之间的时间差，返回一个INTERVAL DAY TO SECOND类型的数值。

expr1/expr2

- expr1和expr2为YashanDB认可的通用表达式，并且类型相同，即TIMESTAMP、DATE或TIME类型，或可以转换为TIMESTAMP、DATE、TIME类型的字符型。
- 当expr1和expr2都不为字符型，且expr1和expr2类型不同时，则报错。
- 当其中一个参数为TIMESTAMP、DATE、TIME类型，另一个参数为字符型时，则将字符串类型的参数转换成与另一个参数类型相同的时间日期类型。
- 当expr1和expr2都为字符型时，则将expr1和expr2都转成TIMESTAMP类型。
- 当expr1或者expr2的值为NULL时，函数返回NULL。

示例

```
-- time类型
CREATE TABLE time_time_diff(C1 TIME, C2 TIME);

INSERT INTO time_time_diff VALUES('11:37:10', '10:20:09');

SELECT TIMEDIFF(C1, C2) res FROM time_time_diff;

RES
--------------------------------
+00 01:17:01.000000

-- date类型
CREATE TABLE date_date_diff(C1 DATE, C2 DATE);

INSERT INTO date_date_diff VALUES('2022-11-24', '2020-12-10');

SELECT TIMEDIFF(C1, C2) res FROM date_date_diff;

RES
--------------------------------
+714 00:00:00.000000

-- timestamp类型
CREATE TABLE timestamp_timestamp_diff(C1 TIMESTAMP, C2 TIMESTAMP);

INSERT INTO timestamp_timestamp_diff VALUES('2022-11-24 11:53:10', '2020-12-10 09:12:49');

SELECT TIMEDIFF(C1, C2) res FROM timestamp_timestamp_diff;

RES
--------------------------------
+714 02:40:21.000000
```

TIMESTAMP

```
timestamp::= TIMESTAMP "("timstamp_expr [","  time_expr] ")"
```

如果是1个参数，TIMESTAMP函数计算timestamp_expr表示的日期，返回一个TIMESTAMP类型的日期值。

如果是2个参数，TIMESTAMP函数计算timestamp_expr表示的日期加上time_expr表示的时间，返回一个TIMESTAMP类型的日期值。

timestamp_expr

YashanDB认可的通用表达式，timestamp_expr的值必须为DATE、TIMESTAMP、TIME或者字符型数据，当为字符型时，必须确保字符串符合当前TIMESTAMP类型的格式要求。

当timestamp_expr为NULL时，函数返回NULL。

time_expr

YashanDB认可的通用表达式，time_expr的值必须为DATE、TIMESTAMP、TIME或者字符型数据，当为字符型时，必须确保字符串符合当前TIME类型的格式要求。

当time_expr为NULL时，函数返回NULL。

示例

```
SELECT TIMESTAMP('2021-5-31 10:10:10','01:01:01') AS result FROM DUAL;
RESULT
--------------------------------
2021-5-31 11:11:11.000000


SELECT TIMESTAMP('2022-1-30 10:10:10') AS result FROM DUAL;
RESULT
--------------------------------
2022-1-30 10:10:10.000000
```

TIMESTAMPDIFF

```
TIMESTAMPDIFF::= TIMESTAMPDIFF "(" unit "," expr1 "," expr2 ")"
```

TIMESTAMPDIFF函数根据unit所指定的时间单位，计算expr1与expr2之间的时间差，返回一个BIGINT类型的数值，expr1小于expr2时返回值为正，expr1大于expr2时返回值为负。

unit

表示函数计算结果的单位，unit不可以为NULL，且必须为如下字符字面量中的一项：

- MICROSECOND
- SECOND
- MINUTE
- HOUR
- DAY
- WEEK
- MONTH
- QUARTER
- YEAR

expr1/expr2

expr1和expr2为YashanDB认可的通用表达式，其值须为TIMESTAMP、DATE或TIME类型，或可以转换为TIMESTAMP、DATE类型的字符型。

当expr1或者expr2的值为NULL时，函数返回NULL。

当expr1或者expr2的值为DATE类型时，系统对微秒部分补0。

当expr1或者expr2的值为TIME类型时，系统对缺少的日期部分补充为当天日期值。

示例

```
SELECT TIMESTAMPDIFF(DAY,'1583-10-01 08:00:00','2000-09-30 07:59:59') res FROM DUAL;
                 RES
--------------------
              152305

SELECT TIMESTAMPDIFF(DAY,'2022-3-1','2022-4-1') res FROM DUAL;
                 RES
--------------------
                  31

SELECT TIMESTAMPDIFF(DAY,'2022-4-1','2022-3-1') res FROM DUAL;
                 RES
--------------------
                 -31
```

TIMESTAMP_TO_SCN

```
timestamp_to_scn::= TIMESTAMP_TO_SCN "(" expr ")"
```

TIMESTAMP_TO_SCN函数将expr表示的时间戳数据转换为SCN号。

expr的值须为TIMESTAMP类型，或者符合TIMESTAMP格式的字符型，否则返回类型不支持或是格式转换错误。

函数不支持传递null作为参数。

示例

```
SELECT SYSDATE,TIMESTAMP_TO_SCN(TO_CHAR(SYSDATE)) scn1,SYSTIMESTAMP,TIMESTAMP_TO_SCN(SYSTIMESTAMP) scn2 FROM DUAL;
SYSDATE                              SCN1 SYSTIMESTAMP                                SCN2
-------------------- ------------------ --------------------------- --------------------
2022-01-09 22:07:08    261527961600000000 2022-01-09 22:07:08.594552     261854120323284992
```

TO_BASE64

```
to_base64::= TO_BASE64 "(" expr ")"
```

TO_BASE64函数将给定参数所表示的字符串使用BASE64进行编码，并返回编码后的VARCHAR类型字符串数据。

该函数不支持列式计算。

expr

通用表达式，expr的值须为字符型，或可转化为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。

expr不支持32000字节以上的XMLTYPE、LOB类型数据。

当expr的值为NULL时，函数返回NULL。

示例

```
SELECT TO_BASE64('ABC') RES FROM DUAL;

RES
-----
QUJD
```

TO_CHAR

```
to_char::= TO_CHAR "(" expr ["," format] ")"
```

TO_CHAR函数将expr的值按format格式转换为CHAR/VARCHAR类型字符串数据。

YashanDB支持将所有普通类型的数据转换为CHAR/VARCHAR类型的数据：

根据支持类型可将TO_CHAR函数分为如下三类：

- TO_CHAR（日期时间型）、TO_CHAR（日期时间型，FORMAT）：expr的值为日期时间型时，支持携带格式符；此时函数返回VARCHAR类型字符串数据。
- TO_CHAR（数值型）、TO_CHAR（数值型，FORMAT）：expr的值为数值型时，支持携带格式符；此时函数返回VARCHAR类型字符串数据。
- TO_CHAR（非日期/数值的其他类型）：
  - expr的值为字符型并且携带格式符时，expr尝试往NUMBER类型转换，转换成功则按照TO_CHAR（数值型，FORMAT）执行，转换失败则函数返回转换报错。
  - 不携带格式符时，当expr的值CHAR类型，函数返回CHAR类型字符串数据，否则返回VARCHAR类型字符串数据。

当expr的值为NULL时，函数返回NULL。

expr支持LOB类型隐式转换，但不支持32000字节以上的XMLTYPE、LOB类型数据。

使用本函数时，expr的值超过NUMBER类型的表示范围时，本函数会返回数据溢出错误。

format

指定转换的格式。

format支持中文年月日，须用双引号包围中文字符，expr中的中文字符无须用双引号包围。

日期时间型数据的格式支持以下字符的组合：

- 年
  - 年份：'YYYY'、'YYY'、'YY'、'Y'、'RRRR'、'RR'
  - 由ISO标准定义下的年份： 'IYYY'、'IYY'、'IY'、'I'
  - 一年中的一天（1-366）：'DDD'
  - 一年中的一周（1-53）：'WW'
  - 由ISO标准定义下一年中的一周(1-53)：'IW'
  - 世纪：'CC'
- 月
  - 月份数字：'MM'
  - 月份全称：'MONTH'
  - 月份缩写：'MON'
- 季度
  - 季度（1-4）：'Q'
- 日
  - 日期数字：'DD'
- 周
  - 一个月中的一周（1-5）：'W'
  - 一周中的一天全称（SUNDAY-SATURDAY）：'DAY'
  - 一周中的一天数字（1-7）：'D'
- 时
  - 24小时制小时：'HH24'
  - 12小时制小时：'HH'、'HH12'
  - 12小时制时段：'AM'、'PM'、'A.M.'、'P.M.'
- 分
  - 分钟数：'MI'
- 秒
  - 秒数：'SS'
  - 一天的总秒数：'SSSSS'

- 儒略日（不支持列式计算）
  - 儒略日计数：`J`
  - 儒略日计数英文全拼：`JSP`
- 连接字符：':', '-', '/', '.', ',', ';', '\', '_', ' ', '[', ']'

（以下格式符列存表暂不支持）

- 年份：'RRRR'、'RR'
- 由ISO标准定义下的年份： 'IYYY'、'IYY'、'IY'、'I'
- 由ISO标准定义下一年中的一周(1-53)：'IW'
- 一天的总秒数：'SSSSS'

> **Note**：
> - 有符号年份'SYYYY'格式符暂未实现，目前和无符号年份'YYYY'格式符完全等价。
> - 有符号世纪'SCC'格式符暂未实现，目前和无符号世纪'CC'格式符完全等价。

数值型数据的格式支持以下字符的组合：

- 小数点：`.`，如'99.99'
- 美元符号：`$`，如'$9,999'
- 强迫零显示：`0`，如'00000'
- 指定位置返回数字：`9`，如'9999'
- 指定位置返回小数点：`D`，如'99D99'
- 设首位或末尾为-或+：`S`，如'S9999'
- 千位分隔符：`,`，如'9,999'
- 在指定位置返回千位分隔符：`G`，如'9G9'

（以下为列存表专用格式）

- 当整数部分为零时，返回定点数字整数部分的空格：`B`，如'B99'
- 在指定位置返回货币符号：`C`，如'C99'
- 以科学计数法的形式返回数字：`EEEE`，如'9EEEE'
- 在指定位置显示货币符号：`L`，如'L99''
- 负值末尾填充负号，正值末尾填充空格：`MI`，如'9MI'
- 负值放在<>中，正值首尾填充空格：`PR`，如'9PR'
- 返回大写罗马数字形式：`RN`，如'RN'
- 返回小写罗马数字形式：`rn`，如'rn'
- 以最少的字符数返回十进制数字字符串：`TM`，如'TM'
- 在指定位置返回欧元等货币符号：`U`，如'U99'
- 返回10^n值，其中n是V后面的数字：`V`，如'99V99'
- 返回指定数字的十六进制值：`X`，如'XXX'
- 去掉前后空格和小数后面多余的0：`FM`，如'FM999'

当expr的值为布尔型的数据时，不需要指定format格式，指定会报错。

示例

```
SELECT TO_CHAR('深圳') res1,TO_CHAR(True) res2 FROM DUAL;
RES1      RES2
--------- -----
深圳    true
```

当expr的值为日期时间型的数据时，不指定format表示系统对该日期类型指定的默认格式转换。

示例

```
SELECT TO_CHAR(SYSTIMESTAMP+1) res FROM DUAL;
RES
-----------------------------------------------------------------
2022-01-10 22:09:27

SELECT TO_CHAR(SYSDATE,'YYYYMMDD HH24:MI:SS') res FROM DUAL;
```

```
RES
------------------------------------------------------------
20220109 22:09:27

SELECT TO_CHAR(SYSDATE,'YYYY"年"MM"月"dd"日"') RES FROM DUAL;

RES
--------------------------------
2023年11月09日
```

当expr的值为数值型的数据时，不指定format表示按该数据的字面值转换为字符串。

示例（HEAP表）

```
SELECT TO_CHAR(numbera,'00000') n1,
TO_CHAR(numberb,'99.99') n2,
TO_CHAR(numberc,'$99999999') n3,
TO_CHAR(numberd,'9999999999999999999') n4,
TO_CHAR(numbere,'99D99') n5,
TO_CHAR(numberf) n6,
TO_CHAR(numberg,'9999999999S') n7,
TO_CHAR(numberh) n8
FROM numbers;
N1      N2      N3         N4                   N5        N6                          N7        N8
------- ------- ---------- -------------------- --------- --------------------------- --------- ----
-00005   55.00    $5555     5555555555555555555    5.55    -5.5555555500000002E+000      555+   1
```

示例（TAC、LSC表）

```
SELECT TO_CHAR(numbera,'00000') n1,
TO_CHAR(numberb,'99.99') n2,
TO_CHAR(numberc,'$99999999') n3,
TO_CHAR(numberd,'9999999999999999999') n4,
TO_CHAR(numbere,'99D99') n5,
TO_CHAR(numberf) n6,
TO_CHAR(numberg,'9999999999S') n7
FROM numbers_nobit;
N1        N2        N3            N4                   N5        N6                                            N7
--------- --------- ------------- -------------------- --------- --------------------------------------------- -------------
-00005     55.00         $5555     5555555555555555555    5.55    5.5555555555499998E+000                             555+
```

当expr的值为字符型的数据时，不指定format表示按该数据的字面值转换为字符串。

示例（HEAP表）

```
SELECT TO_CHAR(CAST(numbera AS VARCHAR(30)),'00000') n1,
       TO_CHAR(CAST(numberb AS VARCHAR(30)),'99.99') n2,
       TO_CHAR(CAST(numberc AS VARCHAR(30)),'$99999999') n3,
       TO_CHAR(CAST(numberd AS VARCHAR(30)),'9999999999999999999') n4,
       TO_CHAR(CAST(numbere AS VARCHAR(30)),'99D99') n5,
       TO_CHAR(CAST(numberf AS VARCHAR(30))) n6,
       TO_CHAR(CAST(numberg AS VARCHAR(30)),'9999999999S') n7,
       TO_CHAR(CAST(numberh AS VARCHAR(30))) n8
FROM numbers;

N1        N2        N3            N4                   N5        N6                                  N7            N8
--------- --------- ------------- -------------------- --------- ----------------------------------- ------------- -----------
----------------------
-00005     55.00         $5555     5555555555555555555    5.55    -5.5555555500000002E+000               555+   1
```

示例（TAC、LSC表）

```
SELECT TO_CHAR(CAST(numbera AS VARCHAR(30)),'00000') n1,
       TO_CHAR(CAST(numberb AS VARCHAR(30)),'99.99') n2,
       TO_CHAR(CAST(numberc AS VARCHAR(30)),'$99999999') n3,
```

```
       TO_CHAR(CAST(numberd AS VARCHAR(30)),'9999999999999999999') n4,
       TO_CHAR(CAST(numbere AS VARCHAR(30)),'99D99') n5,
       TO_CHAR(CAST(numberf AS VARCHAR(30))) n6,
       TO_CHAR(CAST(numberg AS VARCHAR(30)),'9999999999S') n7
FROM numbers_nobit;

N1        N2        N3           N4                   N5        N6                                N7
--------- --------- ------------ -------------------- --------- --------------------------------- -------------
-00005     55.00         $5555     5555555555555555555    5.55    5.55555555554999998E+000                555+
```

TO_DATE

```
to_date::= TO_DATE "("expr [DEFUALT replace_expr ON CONVERSION ERROR] [","  format][","nls_calendar] ")"
```

TO_DATE函数将expr的值转换为DATE类型数据，返回值格式为DATE类型数据的默认格式。

expr

expr的值须为字符型，或可以转化为字符型的类型（LOB类型支持隐式转换），且其内容须符合format格式，否则返回格式转换错误。

当expr的值为NULL时，函数返回NULL。

对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。

DEFAULT replace_expr ON CONVERSION ERROR

表示当对expr转换失败时，使用replace_expr值来进行转换，replace_expr为一个字符串字面量，其内容须符合format格式。

示例

```
SELECT TO_DATE('13-21' DEFAULT '2-13' ON CONVERSION ERROR,'mm-dd') res FROM DUAL;
RES
-------------------------------
2022-02-13 00:00:00
```

format

format用于指定expr的时间格式，其值须为字符型，可省略，省略则默认expr格式同DATE类型的默认格式。

format支持中文年月日，须用双引号包围中文字符，expr中的中文字符无须用双引号包围。

如果format中未指定年，则会获取当前年份然后填充；如果不存在月，则会获取当前月份然后填充；如果不存在日，则会用1来填充；未指定时间用0来填充。

YashanDB支持不含连接字符的expr与format进行匹配，该匹配须遵循如下规则：

- expr中数字会按照format指定的格式进行转换，建议输入相同长度的数字和转换格式，否则可能导致结果错误。
- 列存表中，format必须为包含连接字符的时间格式。
- format中不能指定为可变长度的转换格式，如MONTH、DAY。

指定转换的格式，包括如下字符组合：

- 年
  - 年份数字：`YYYY`、`YYY`、`YY`、`Y`、`RRRR`、`RR`
- 月
  - 月份数字：`MM`
  - 月份全称：`MONTH`
  - 月份缩写：`MON`
- 季度
  - 季度（1-4）：`Q`
- 日
  - 日期数字：`DD`
- 日期描述
  - 一周中的一天全称（SUNDAY-SATURDAY）：`DAY`
  - 一周中的一天数字（1-7）：`D`
  - 一年中的一天（1-366）：`DDD`
- 时
  - 24小时制小时：`HH24`
  - 12小时制小时：`HH`、`HH12`
- 分
  - 分钟数：`MI`

- 秒
  - 秒数：`SS`
  - 一天的总秒数：`SSSSS`
- 儒略日
  - 儒略日计数：`J`
- 连接字符：`:`、`-`、`/`、`.`、`,`、`;`、`\`、`_`、` `、`[`、`]`
  - `:`、`-`、`/`、`.`、`,`、`;`、`\`、`_`、` `、`[`、`]`：连接符之间可以相互匹配
  - ` `：空格可不参与匹配

（以下格式符列存表暂不支持）

- 年份：'RRRR'、'RR'
- 一天的总秒数：'SSSSS'
- 儒略日计数：`J`

> **Note**：
> 日期描述类格式须与年月日格式同时指定，且expr中日期描述须与年月日匹配，否则返回错误。 有符号年份'SYYYY'格式符暂未实现，目前和无符号年份'YYYY'格式符完全等价。 当儒略日计数范围在[1,1721057]之内，对应实际年份为负数。

其中对于年格式的指定存在如下转换规则：

expr\format	Y	YY	YYY	YYYY
1	2021-06-01	2001-06-01	2001-06-01	0001-06-01
12	报错	2012-06-01	2012-06-01	0012-06-01
123	报错	0123-06-01	2123-06-01	0123-06-01
1234	报错	1234-06-01	报错	1234-06-01

示例

```
--expr包含连接字符
SELECT TO_DATE( 'January 15, 1989, 11:00 A.M.', 'Month dd, YY, HH:MI A.M.' ) res FROM DUAL;
RES
---------------------------------
1989-01-15 11:00:00

SELECT TO_DATE( '1989/2/23', 'YYYY/MM/DD HH24:MI:SS' ) res FROM DUAL;
RES
---------------------------------
1989-02-23 00:00:00

SELECT TO_DATE( '2020-01-01,1','YYYY-MM-DD,DDD' ) res FROM DUAL;
RES
---------------------------------
2020-01-01 00:00:00

--日期描述与输入的日期不匹配时返回错误
SELECT TO_DATE( '2020-01-01,6','YYYY-MM-DD,DDD' ) res FROM DUAL;
[1:17]YAS-00008 type convert error : not a valid day

SELECT TO_DATE('0120111-','YYYMMDD-') res FROM DUAL;
RES
---------------------------------
2012-01-11 00:00:00

--不兼容匹配默认DATE格式'YYYY-MM-DD'时，返回错误
SELECT TO_DATE( '1989//2/23' ) res FROM DUAL;
[1:16]YAS-00008 type convert error : literal does not match format string

--expr中不含连接字符
--format中包含连接字符
SELECT TO_DATE('20120111','YYYY-MM-DD') res FROM DUAL;
```

```
RES
--------------------------------
2012-01-11 00:00:00

--format中不能为可变长度的转换格式
SELECT TO_DATE('2012JANUARY11','YYYY-MONTH-DD') res FROM DUAL;
[1:16]YAS-00008 type convert error : literal does not match format string

--数字会按照format的转换格式进行转换，本例中12为年份，01为月份，30为日期。由于转换格式为YYY-MM-DD，将120识别为年份，13识别为月份导致错误
SELECT TO_DATE('120130','YYY-MM-DD') res FROM DUAL;
[1:16]YAS-00008 type convert error : not a valid month

--expr和format均不含连接字符
SELECT TO_DATE('20120111','YYYYMMDD') res FROM DUAL;
RES
--------------------------------
2012-01-11 00:00:00

SELECT TO_DATE('2010131','YYYMMDD') res FROM DUAL;
RES
--------------------------------
2201-01-31 00:00:00


SELECT TO_DATE('2016','YYYMM') res FROM DUAL;
RES
--------------------------------
2201-06-01 00:00:00

SELECT TO_DATE('2012年01月01日','YYYY"年"MM"月"dd"日"') RES FROM DUAL;

RES
--------------------------------
2012-01-01 00:00:00

SELECT TO_DATE('1721424','J') res FROM DUAL;
RES
--------------------------------
0001-01-01 00:00:00
```

nls_calendar

指定数据库使用的日历系统，仅支持为字符串类型，默认值为gregorian，否则返回类型错误，当前YashanDB仅支持日历系统为gregorian的情况，其他情况将会报错。

示例

```
SELECT TO_DATE( 'January 15, 1989, 11:00 A.M.', 'Month dd, YYYY, HH:MI A.M.', 'nls_calendar = gregorian' ) res FROM DUAL;
RES
--------------------------------
1989-01-15 11:00:00
```

TO_DSINTERVAL

```
to_dsinterval::= TO_DSINTERVAL "(" "'" sql_format "'" ")"

sql_format::= (["+" | "-"] days hours ":" minutes ":" seconds ["." frac_secs])
```

TO_DSINTERVAL函数将expr的值转换为INTERVAL DAY TO SECOND类型的数值。

当expr的值为NULL时，函数返回NULL。

expr的值应该为字符型数据，否则返回类型不支持。且该字符串需符合sql_format格式要求，否则返回格式转换错误。

sql_format

与SQL标准（ISO/IEC 9075）兼容的SQL间隔格式，需要声明days、hours、minutes、seconds和frac_secs五个元素，days和hours间由一个或多个空格隔开，hours、minutes、seconds间由一个冒号隔开。不同元素之间允许存在额外的空格，例如 `'23 23 : 23:12'` 。

- days：整数，取值范围为[-100000000,100000000]。
- hours：整数，取值范围为[0,23]。
- minutes：整数，取值范围为[0,59]。
- seconds：整数，取值范围为[0,59]。
- frac_secs：小数，取值范围为[.0,.999999]。

示例

```
SELECT TO_DSINTERVAL('23 23 : 23:12') res FROM DUAL;
RES
--------------------------------
+23 23:23:12.000000

--查询在当前日期前两天的时间值
SELECT SYSDATE+TO_DSINTERVAL('-2 00:00:00') res FROM DUAL;
RES
--------------------------------
2021-11-27 22:32:25
```

TO_NUMBER

```
to_number::= TO_NUMBER "("expr [","  format] ")"
```

TO_NUMBER函数将expr的值按format格式转换为NUMBER类型的数据。

expr值的数据类型可以为：

- 数值型：此时不需要指定format，函数直接转换为NUMBER类型。
- 字符型：此时函数将根据format格式转换，且expr内容需符合指定的format格式。

当expr的值为NULL时，函数返回NULL。

format

指定转换的格式，支持以下字符的组合：

- 小数点：`.`，如'99.99'
- 美元符号：`$`，如'$9,999'
- 强迫零显示：`0`，如'00000'
- 指定位置返回数字：`9`，如'9999'
- 指定位置返回小数点：`D`，如'99D99'
- 设首位或末尾为-或+：`S`，如'S9999'
- 千位分隔符：`,`，如'9,999'
- 在指定位置返回千位分隔符：`G`，如'9G9'

（以下为列存表专用格式）

- 当整数部分为零时，返回定点数字整数部分的空格：`B`，如'B99'
- 在指定位置返回货币符号：`C`，如'C99'
- 以科学计数法的形式返回数字：`EEEE`，如'9EEEE'
- 在指定位置显示货币符号：`L`，如'L99'
- 负值末尾填充负号，正值末尾填充空格：`MI`，如'9MI'
- 负值放在<>中，正值首尾填充空格：`PR`，如'9PR'
- 返回大写罗马数字形式：`RN`，如'RN'
- 返回小写罗马数字形式：`rn`，如'RN'
- 以最少的字符数返回十进制数字字符串：`TM`，如'TM'
- 在指定位置返回欧元等货币符号：`U`，如'U99'
- 返回10^n值，其中n是V后面的数字：`V`，如'99V99'
- 返回指定数字的十六进制值：`X`，如'XXX'
- 去掉前后空格和小数后面多余的0：`FM`，如'FM999'

示例

```
--将所有数据转化为NUMBER类型输出，此数据的显示宽度与SET NUMWIDTH有关
SELECT TO_NUMBER(numbera) n1,
TO_NUMBER(numberb) n2,
TO_NUMBER(numberc) n3,
TO_NUMBER(numberd) n4,
TO_NUMBER(numbere) n5,
TO_NUMBER(numberf) n6,
TO_NUMBER(numberg) n7
FROM numbers;
        N1          N2          N3          N4          N5          N6          N7
---------- ----------- ----------- ----------- ----------- ----------- -----------
        -5          55        5555  5.5556E+18  5.55499983  -5.5555556         555
```

TO_TIMESTAMP

```
to_timestamp::= TO_TIMESTAMP "("expr [DEFUALT replace_expr ON CONVERSION ERROR] ["," format] ")"
```

TO_TIMESTAMP函数将expr的值转换为TIMESTAMP类型数据，返回值格式为TIMESTAMP类型的默认格式。

expr

expr的值须为字符型，且其内容须符合format格式，否则返回格式转换错误。

当expr的值为NULL时，函数返回NULL。

DEFAULT replace_expr ON CONVERSION ERROR

表示当对expr转换失败时，使用replace_expr值来进行转换，replace_expr为一个字符串字面量，其内容须符合format格式。

示例

```
SELECT TO_TIMESTAMP('13-21' DEFAULT '2-13' ON CONVERSION ERROR,'mm-dd') res FROM DUAL;
RES
---------------------------------------------------------------
2022-02-13 00:00:00.000000
```

format

format用于指定expr的时间格式，其值须为字符型，可省略，省略则默认expr格式同TIMESTAMP类型的默认格式。

如果format中未指定年，则获取当前年份然后填充；如果不存在月，则获取当前月份然后填充；如果不存在日，则用1来填充；未指定时间则用0来填充。

YashanDB支持不含连接字符的expr与format进行匹配，该匹配须遵循如下规则：

- expr中数字会按照format指定的格式进行转换，建议输入相同长度的数字和转换格式，否则可能导致结果错误。
- format中不能指定为可变长度的转换格式，如MONTH、DAY。

指定转换的格式，包括如下字符组合：

- 年
  - 年份数字：`YYYY`、`Y`、`YY`、`YYY`
- 月
  - 月份数字：`MM`
  - 月份全称：`MONTH`
  - 月份缩写：`MON`
- 日
  - 日期数字：`DD`
- 日期描述
  - 一周中的一天全称（SUNDAY-SATURDAY）：`DAY`
  - 一周中的一天数字（1-7）：`D`
  - 一年中的一天（1-366）：`DDD`
- 时
  - 24小时制小时：`HH24`
  - 12小时制小时：`HH`、`HH12`
- 分
  - 分钟数：`MI`
- 秒
  - 秒数：`SS`
- 儒略日
  - 儒略日计数：`J`
- 连接字符：`:`、`-`、`/`、`.`、`,`、`;`、`\`、`_`、` `、`[`、`]`
  - `:`、`-`、`/`、`.`、`,`、`;`、`\`、`_`、` `、`[`、`]`：连接符之间可以相互匹配
  - ` `：空格可不参与匹配

> **Note**：
> 日期描述类格式须与年月日格式同时指定，且expr中日期描述须与年月日匹配，否则返回错误。

其中对于年格式的指定存在如下转换规则：

expr\format	Y	YY	YYY	YYYY
1	2021-06-01	2001-06-01	2001-06-01	0001-06-01
12	报错	2012-06-01	2012-06-01	0012-06-01
123	报错	0123-06-01	2123-06-01	0123-06-01
1234	报错	1234-06-01	报错	1234-06-01

示例

```
SELECT TO_TIMESTAMP('2','hh') timestamp1,TO_TIMESTAMP(TO_CHAR(SYSDATE)) timestamp2 FROM DUAL;
TIMESTAMP1                  TIMESTAMP2
--------------------------- ---------------------------
2022-01-01 02:00:00.000000   2022-01-09 00:00:00.000000

SELECT TO_TIMESTAMP('2020-01-01-02','YYYY-MM-DD-HH') res FROM DUAL;
RES
------------------------------------------------------------
2020-01-01 02:00:00.000000

SELECT TO_TIMESTAMP( '2020-01-01,1','YYYY-MM-DD,DDD' ) res FROM DUAL;
RES
--------------------------------
2020-01-01 00:00:00.000000

--日期描述与输入的日期不匹配时返回错误
SELECT TO_TIMESTAMP( '2020-01-01,6','YYYY-MM-DD,DDD' ) res FROM DUAL;
[1:22]YAS-00008 type convert error : not a valid day

--不含连接字符的expr
SELECT TO_TIMESTAMP('2020010102','YYYY-MM-DD-HH') res FROM DUAL;
RES
------------------------------------------------------------
2020-01-01 02:00:00.000000

SELECT TO_TIMESTAMP('2020010102','YYYYMMDDHH') res FROM DUAL;
RES
------------------------------------------------------------
2020-01-01 02:00:00.000000

--不含连接字符的expr，数字与转换格式长度不相同可能导致错误。本例中26为年份,06为月份，21为日期，02为小时。
SELECT TO_TIMESTAMP('26062102','YYY-MM-DD-HH') res FROM DUAL;
[1:21]YAS-00008 type convert error : not a valid month

--不含连接字符的expr，format中不能为可变长度的转换格式
SELECT TO_TIMESTAMP('2020010102','YYYY-MONTH-DD-HH') res FROM DUAL;
[1:21]YAS-00008 type convert error : literal does not match format string

SELECT TO_TIMESTAMP('20602','DDD-HH') res FROM DUAL;
```

nls_calendar

指定数据库使用的日历系统，仅支持为字符串类型，默认值为gregorian，否则返回类型错误，当前YashanDB仅支持日历系统为gregorian的情况，其他情况将会报错。

示例

```
SELECT TO_TIMESTAMP('2020010102','YYYYMMDDHH', 'nls_calendar = gregorian') res FROM DUAL;
RES
```

```
------------------------------------------------------------
2020-01-01 02:00:00.000000
```

TO_YMINTERVAL

```
to_yminterval::= TO_YMINTERVAL "(" "'" ym_sql_format "'" ")"

ym_sql_format::= (["+"|"-"] years "-" months)
```

TO_YMINTERVAL函数将expr的值转换为INTERVAL YEAR TO MONTH类型的数值。

当expr的值为NULL时，函数返回NULL。

expr的值应该为字符型，否则返回类型不支持。且该字符串需符合ym_sql_format格式要求，否则返回格式转换错误。

ym_sql_format

与SQL标准（ISO/IEC 9075）兼容的SQL间隔格式，需要声明years和months两个元素，years与months间用减号隔开，且允许存在额外的空格，例如`'5- 2'`。

- years：整数，取值范围为[-178000000,178000000]。
- months：整数，取值范围为[0,11]。

示例

```
SELECT TO_YMINTERVAL('5-2') res FROM DUAL;
RES
--------------------
+05-02

--查询在当前日期前五年的时间值
SELECT SYSDATE+TO_YMINTERVAL('-5-0') res FROM DUAL;
RES
--------------------------------
2016-11-29 22:51:41
```

TRANSLATE

```
translate::= TRANSLATE "(" expr ((USING (CHAR_CS|NCHAR_CS))|("," from_string "," to_string)) ")"
```

TRANSLATE函数将expr表示的字符串里的所有的from_string中的每一个字符替换为相应的to_string中的字符，是多个单字符的一对一替换关系，返回一个VARCHAR类型的新字符串。

在TRANSLATE...USING用法中，TRANSLATE函数将依据CHAR_CS/NCHAR_CS，对expr表示的字符串进行字符集转换。

本函数遵循如下规则：

- 该函数不支持列式计算。
- 当using CHAR_CS时返回值为VARCHAR类型。
- 当using NCHAR_CS时返回值为NVARCHAR类型。
- 当不带using时，expr为NCLOB/NCHAR/NVARCHAR时，返回值为NVARCHAR类型，其余场景返回值为VARCHAR类型。

expr

- expr的值须为字符型，或可转换为字符型的其他类型。当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。

from_string

- 要进行替换的字符串，须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。
- from_string为与expr相同的通用表达式，当from_string的值为NULL时，函数返回NULL。

to_string

- 按此字符串的值进行一对一替换，须为字符型，或可转换为字符型的其他类型（LOB、XMLTYPE类型支持隐式转换）。
- to_string为与expr相同的通用表达式，当to_string的值为NULL时，函数返回NULL。

示例（HEAP表）

```
SELECT TRANSLATE('I am chinese', 'ch', 'ab') TRANSLATE
FROM DUAL;
TRANSLATE
-------------
I am abinese

SELECT TRANSLATE('I am chinese', 'china', 'ab') TRANSLATE
FROM DUAL;
TRANSLATE
-------------
I m abese

SELECT TRANSLATE('我是中国人,我爱中国,China', '中国', 'China') TRANSLATE
FROM DUAL;
TRANSLATE
-----------------------------
我是Ch人,我爱Ch,China
```

translate_using

- CHAR_CS表示将expr转换成用数据库字符集编码的字符串，此时函数返回值的类型是VARCHAR。
- NCHAR_CS表示将expr转换成用国家字符集编码的字符串，此时函数返回值的类型是NVARCHAR。

示例（HEAP表）

```
SELECT TRANSLATE('Yashan Database' USING CHAR_CS) TRANSLATE
FROM DUAL;
TRANSLATE
-----------------
Yashan Database

SELECT TRANSLATE('崖山数据库' USING NCHAR_CS) TRANSLATE
FROM DUAL;
TRANSLATE
------------------------------------------------------------
崖山数据库
```

TRIM

```
trim::= TRIM "(" [[LEADING|TRAILING|BOTH] trim_character from] expr ")"
```

TRIM函数删除expr表示的字符串的前缀或后缀，得到一个新的子字符串。

expr

expr的值须为字符型，或可转换为字符型的其他类型，不允许为NCLOB类型数据。

- 当expr的值为NULL时，函数返回NULL。
- 对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。
- 当expr为CLOB类型时，返回值为CLOB类型，当expr为NCAHR/NVARCHAR时，返回值为NVARCHAR类型，其余返回值为VARCHAR类型。

LEADING|TRAILING|BOTH

指定删除字符串的前缀|后缀|前后缀，不指定时默认为BOTH。

trim_character

指定前后缀的内容，trim_character为与expr相同的通用表达式，须为字符型，或可转换为字符型的其他类型，且其长度只能为1字节。

- 当trim_character的值为NULL时，函数返回NULL。
- 当不指定trim_character时，默认的前后缀内容为一个空格。

示例

```
SELECT TRIM(' bar ') t_default
,TRIM(LEADING 'x' FROM 'xxxbarxxx') t_left
,TRIM(BOTH 'x' FROM 'xxxbarxxx') t_both
,TRIM(TRAILING 'x' FROM 'xxxbarxxx') t_right
FROM DUAL;
T_DEFAULT T_LEFT    T_BOTH T_RIGHT
--------- --------- ------ ---------
bar       barxxx    bar    xxxbar
```

TRUNC

```
trunc::= TRUNC "(" ((expr ["," fmt])|(expr ["," n])) ")"
```

TRUNC函数对expr的值按指定格式截断一个日期值，或按指定位数截断一个数值，返回一个DATE类型的日期值或一个NUMBER类型的数值。

当用于截断日期值时，expr的值必须为DATE、TIMESTAMP类型；当用于截断数值时，expr的值必须为数值型，或可以转换为NUMBER类型的字符型（转换失败返回Invalid number错误）。

对于其他类型，函数返回类型不支持。

当expr的值为NULL时，函数返回NULL。

fmt

指定日期值的截断格式，规则如下：

类型	有效格式	返回值	备注
Century	CC, SCC	返回当世纪第一天	21世纪第一年为2001年
Year	SYYYY, YYYY, YEAR, SYEAR, YYY, YY, Y	返回当年第一天	
ISO Year	IYYY, IY, I	将每年1月4日所处的周作为当年的第一周，返回该周的第一天	ISO 当年第一天可能是公历前一年12月末
Quarter	Q	返回当季度第一天	
Month	MONTH, MON, MM, RM	返回当月第一天	
Week	WW	每年的1月1日作为当年的第一周的第一天，返回当周第一天	星期一到星期日都有可能
IW	IW	星期一为每周的第一天，返回当周第一天	
W	W	每月1日作为本月第一周的第一天，返回当周第一天	
Day	DDD, DD, J	返回当前日期	
Start day of the week	DAY, DY, D	星期日为每周的第一天，返回当周第一天	
Hour	HH, HH12, HH24	返回本小时的开始时间	
Minute	MI	返回本分钟的开始时间	
null	null	null	

当不指定fmt时，默认为'DD'。

n

指定数值的截断位数，规则如下：

n值	作用	示例
0	截断小数部分	trunc(123.456, 0) -> 123
> 0 且未超出number_expr小数位数	截断该小数位数往后的部分	trunc(123.456, 2) -> 123.45
> 0 且超出number_expr小数位数	原数据返回	trunc(123.456, 5) -> 123.456

n值	作用	示例
120; trunc(123.456, -2) -> 100		
0		
小数	正数向下取整，负数向上取整	trunc(123.456, 1.6) = trunc(123.456, 1); trunc(123.456, -1.2) = trunc(123.456, -1)
null	返回 null	

当不指定n时，表示不截断，返回原数据。

示例

```
SELECT TRUNC(SYSDATE-10000,'CC') res FROM DUAL;
RES
-------------------------------
1901-01-01 00:00:00

SET NUMWIDTH 30;

SELECT TRUNC('234.33333343',0) "n0",
TRUNC('234.33333343',3) "n3",
TRUNC('234.33333343',30) "n30",
TRUNC('234.33333343',-2) "n-2",
TRUNC('234.33333343',-20) "n-20",
TRUNC('234.33333343',2.98) "n2.98",
TRUNC('234.33333343',-2.98) "n-2.98"
FROM DUAL;
   n0        n3              n30      n-2      n-20        n2.98     n-2.98
----- --------- ---------------- -------- --------- ------------ ----------
  234   234.333     234.33333343      200         0       234.33        200
```

TYPEOF

```
typeof::= TYPEOF "(" expr1 ["," expr2] ")"
```

TYPEOF函数获取expr1参数的数据类型，将其作为VARCHAR类型的字符串返回。

当expr1的值为NULL时，函数返回varchar。

expr2取值范围为0和1、列存表中不可指定本参数。

- 0表示显示typeof中expr1所有参数类型信息（包括类型精度以及字符串长度）和返回值类型。
- 1表示显示返回值类型以及类型对应的精度或长度信息。

当expr2为空时，typeof函数只显示返回值类型。

示例

```
SELECT TYPEOF('') t1,
TYPEOF('1.2345') t2,
TYPEOF(1.2345) t3,
TYPEOF(SYSDATE) t4
FROM DUAL;
T1        T2    T3        T4
--------- ----- --------- ----------------------------------------------------------------
varchar   char  number    date

SELECT TYPEOF(1.21, 0) RES FROM DUAL;

RES
----------------------------------------
number(3, 2)
[expr_const:number(3, 2)]

SELECT TYPEOF(1.21, 1) RES FROM DUAL;

RES
-------------
number(3, 2)
```

UNISTR

```
unistr::= UNISTR "(" expr ")"
```

UNISTR函数将源字符串中Unicode编码部分转换为对应字符，函数将返回一个NVARCHAR类型的字符串。

本函数遵循如下规则：

- Unicode编码限制为UCS-2编码格式，格式为"后加4位16进制。
- 若要将反斜杠包含在字符串本身中，需在其前面加上另一个反斜杠，不符合的格式将会报错。
- 本函数不支持列式计算。
- 返回值固定为NVARCHAR类型。
- 当expr为NULL时，函数返回NULL。
- 函数返回的最大字符长度：
  - 为常量返回原字符串字符长度。
  - 为列返回最大字符长度。

expr

通用表达式，须为字符型，或可以转换为字符型的其他类型。

- NCHAR/NVARCHAR类型在UNISTR转换下会将字符的各个字节分别转换为UTF16类型。

示例（HEAP表）

```
SELECT UNISTR('\5d16\5c71') FROM DUAL;

UNISTR('\5D16\5C71')
----------------------------------------
崖山

SELECT UNISTR('\\5d16\5c71') FROM DUAL;

UNISTR('\\5D16\5C71')
------------------------------------------------
\5d16山

SELECT UNISTR('\\\5d16\5c71') FROM DUAL;

UNISTR('\\\5D16\5C71')
------------------------------------------------
\崖山
```

UNSUPPORT_ERROR

```
unsupport_error::= UNSUPPORT_ERROR "("")"
```

UNSUPPORT_ERROR函数返回报错信息 This statement is temporarily unable to execute due to an internal error。 以下两种形式的UNSUPPORT_ERROR同义，返回结果无差别，括号内不允许有参数：

- `UNSUPPORT_ERROR`
- `UNSUPPORT_ERROR()`

示例

```
SELECT UNSUPPORT_ERROR() FROM DUAL;

[1:8]YAS-04349 This statement is temporarily unable to execute due to an internal error
```

UPPER

```
upper::= UPPER "(" expr ")"
```

UPPER函数将expr表示的字符串中的小写字母转换为大写，返回一个新字符串。

expr

expr的值须为字符型，或可转化为字符型的其他类型。

- 当expr的值为NULL时，函数返回NULL。
- expr不支持32000字节以上的XMLTYPE、LOB类型数据。
- 当expr为CHAR、NCHAR、NVARCHAR类型时，返回值为CHAR、NCHAR、NVARCHAR类型，其余场景返回值为VARCHAR类型。

示例

```
SELECT UPPER('深圳nihao') res FROM DUAL;
RES
--------------------
深圳NIHAO
```

USERENV

```
userenv::= USERENV "(" "'" parameter "'" ")"
```

USERENV函数根据输入的parameter参数返回当前会话的相关信息。

parameter

为表达参数的字符串字面量，不区分大小写。包括如下值：

- SID：当前会话的SID，一个SID能够唯一标识一个已连接的会话。函数对该参数返回一个NUMBER类型的数值。
- SCHEMAID：当前的USER ID。函数对该参数返回一个NUMBER类型的数值。
- LANGUAGE：当前连接的服务端所使用的字符集。函数对该参数返回一个VARCHAR类型的数值。
- TERMINAL：当前会话所在设备或终端的标识。函数对该参数返回一个VARCHAR类型的数值。
- CLIENT_INFO：当前会话所登录的用户名称与当前会话程序所在路径。函数对该参数返回一个VARCHAR类型的数值。
- GSID：分布式下当前会话的全局会话ID，每一个会话具有不同的GSID，不同CN间亦不相同。非分布式下返回默认值SID。函数对该参数返回一个NUMBER类型的数值。
- CURRENT_SCHEMAID：当前的USER ID。函数对该参数返回一个VARCHAR类型的数值。
- INSTANCE：当前的实例ID。函数对该参数返回一个NUMBER类型的数值。

如parameter指定为上述之外的其他值，函数返回invalid parameter错误。

示例

```
SELECT USERENV('CLIENT_INFO') res FROM DUAL;
RES
----------------------------------------------------------------
user: SYS
program path: /home/yasdb/bin/yasql
```

UTC_TIMESTAMP

```
utc_timestamp::= UTC_TIMESTAMP ["(" [integer] ")"]
```

UTC_TIMESTAMP函数返回数据库所在的操作系统设置的当前协调世界时，其返回类型为TIMESTAMP，且与TIMESTAMP_FORMAT参数所指定格式一致。

UTC_TIMESTAMP有以下三种形式：

- `UTC_TIMESTAMP`
- `UTC_TIMESTAMP()`
- `UTC_TIMESTAMP(integer)`，integer必须为一个0~9之间的整数字面量，表示保留的微秒位数，舍去的位按四舍五入。

如果一个SQL语句中出现了多个UTC_TIMESTAMP函数，在该语句执行过程中将只调用一次函数，即保证多个UTC_TIMESTAMP函数返回的是相同一个时间戳值。

当UTC_TIMESTAMP函数参与运算时，其运算规则与TIMESTAMP类型一致，见算术运算符章节描述。

示例

```
SELECT UTC_TIMESTAMP res1,UTC_TIMESTAMP() res2,UTC_TIMESTAMP(9) res3 FROM DUAL;
RES1                                                            RES2
RES3
--------------------------------------------------------------- ----------------------------------------------------------------
---- ----------------------------------------------------------
2023-05-04 06:24:04.636637                                      2023-05-04 06:24:04.636637
2023-05-04 06:24:04.636637
```

VARIANCE

```
variance::= VARIANCE "(" [DISTINCT|ALL] expr ")"
```

VARIANCE函数计算expr的值的样本方差。

当给定参数只有一行数据时，VARIANCE函数的计算结果为0。

VARIANCE函数的返回值类型有以下几种情况：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型时，返回NUMBER类型。
- 当expr的值为FLOAT、DOUBLE类型时，返回与expr的值一致的FLOAT、DOUBLE类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为字面量、变量、常量时，返回0。
- 当expr的值为其他类型且无法转换为NUMBER类型时，不执行计算并返回类型转换错误。

DISTINCT

表示过滤掉输入的重复数据后，进行样本方差计算。

ALL

默认值，表示不对表达式输入的重复数据进行过滤，直接进行样本方差计算。

示例

```
--计算员工数量的样本方差,为空的行将被忽略,该语句等同于SELECT VARIANCE(ALL employee_count) FROM area1
SELECT VARIANCE(employee_count) res FROM area1;
                RES
--------------------
          3333.33333

--只有一行记录的样本方差为0
SELECT VARIANCE(employee_count) res FROM area1 WHERE area_no='02';
                RES
--------------------
                  0

--除去重复的员工数量后计算样本方差
SELECT VARIANCE(DISTINCT employee_count) res FROM area1;
                RES
--------------------
               5000
```

VAR_POP

```
var_pop::= VAR_POP "(" expr ")"
```

VAR_POP函数计算expr的值的总体方差。

VAR_POP函数的返回值类型有以下几种情况：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型时，返回NUMBER类型。
- 当expr的值为FLOAT、DOUBLE类型时，返回与expr的值一致的FLOAT、DOUBLE类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为字面量、变量、常量时，返回0。
- 当expr的值为其他类型且无法转换为NUMBER类型时，不执行计算并返回类型转换错误。

在多行计算中，系统忽略数值为空的行，当所有行均为空时，计算结果为NULL。

示例

```
--计算员工数量的总体方差,为空的行将被忽略
SELECT VAR_POP(employee_count) res FROM area1;

                  RES
--------------------
          2222.22222
```

VAR_SAMP

```
var_samp::= VAR_SAMP "(" expr ")"
```

VAR_SAMP函数计算expr的值的样本方差。

当给定参数只有一行数据时，VARIANCE函数的计算结果为NULL。

VAR_SAMP函数的返回值类型有以下几种情况：

- 当expr的值为TINYINT、SMALLINT、INT、BIGINT、NUMBER类型时，返回NUMBER类型。
- 当expr的值为FLOAT、DOUBLE类型时，返回与expr的值一致的FLOAT、DOUBLE类型。
- 当expr的值为NULL时，返回NULL。
- 当expr的值为字面量、变量、常量时，返回0。
- 当expr的值为其他类型且无法转换为NUMBER类型时，不执行计算并返回类型转换错误。

在多行计算中，系统忽略数值为空的行，当所有行均为空时，计算结果为NULL。

示例

```
--计算员工数量的样本方差,为空的行将被忽略
SELECT VAR_SAMP(employee_count) res FROM area1;
                RES
--------------------
          3333.33333

--只有一行记录的样本方差为NULL
SELECT VAR_SAMP(employee_count) res FROM area1 WHERE area_no='02';
                RES
--------------------
```

WM_CONCAT

```
wm_concat::= WM_CONCAT "(" [DISTINCT|ALL] string ")"
```

WM_CONCAT函数将多行的数据执行拼接操作，并通过分隔符分隔，返回一行CLOB类型的字符串。该函数与GROUP_CONCAT函数实现功能类似，区别在于WM_CONCAT不能指定SEPARATOR分隔符（固定为`,`），且不能指定ORDER BY排序。

DISTINCT

计算最终拼接结果时，过滤在同一组内出现的重复的行。

ALL

默认值，表示不过滤重复的行，对所有行都进行拼接。

string

string可以为：

- 通用表达式expr
- 查询列为单列且返回行为单行的子查询

string的值为字符型，或可转换为字符型的其他类型，但不允许为JSON、NVARCHAR、NCHAR和NCLOB类型。

对于列存表中的LOB类型字段，若某行数据为行外存储，则无法使用本函数。

在单行计算中，当string的值为NULL时，函数返回NULL。

在多行计算中，函数将忽略string值为空的行，当所有行均为空时，计算结果为NULL。存在多个拼接行时，会将多行的结果使用分隔符`,`分隔开来。

示例

```
--创建exprs_wmconcat表，并插入数据
CREATE TABLE exprs_wmconcat (id INT,name VARCHAR(50),money FLOAT);
INSERT INTO exprs_wmconcat
VALUES  (1,'小东',10000),(2,'小明',46450),
        (3,'小红',46450 ),(4,'小东',14465),
        (5,'小明',46450),(6,'小东',46450);

--未指定GROUP BY时，将所有行CONCAT，得到一行结果
SELECT WM_CONCAT(MONEY) AS money FROM exprs_wmconcat;
MONEY
-------------------------------------------------------------
1.0E+004,4.645E+004,4.645E+004,1.4465E+004,4.645E+004,4.645E+004

--group by后各组的多行数据分别CONCAT成一行，得到按组的多行结果
SELECT NAME,WM_CONCAT(MONEY) AS money FROM exprs_wmconcat GROUP BY name;
NAME                 MONEY
-------------------- ----------------------------------------------
小东                 1.0E+004,1.4465E+004,4.645E+004
小明                 4.645E+004,4.645E+004
小红                 4.645E+004

--使用DISTINCT关键字去重
SELECT WM_CONCAT(DISTINCT name) AS names FROM exprs_wmconcat;
NAMES
------------------------
小东,小明,小红
```