虚谷数据库 V12.7

系统函数参考指南

文档版本 01

发布日期 2025-01-10

虚谷数据库
XUGU DB

声明

未经本公司正式书面许可，任何企业和个人不得擅自摘抄、复制、使用本文档中的部分或全部内容，且不得以任何形式进行传播。否则，本公司将保留追究其法律责任的权利。

用户承诺在使用本文档时遵守所有适用的法律法规，并保证不以任何方式从事非法活动。不得利用本文档内容进行任何侵犯他人权益的行为。

商标声明

虚谷数据库 XUGU DB

为成都虚谷伟业科技有限公司的注册商标。

本文档提及的其他商标或注册商标均非本公司所有。

注意事项

您购买的产品或服务应受本公司商业合同和条款的约束，本文档中描述的部分产品或服务可能不在您的购买或使用范围之内。由于产品版本升级或其他原因，本文档内容将不定期进行更新。

除非合同另有约定，本文档仅作为使用指导，所有内容均不构成任何声明或保证。

成都虚谷伟业科技有限公司

地址：四川省成都市锦江区锦盛路 138 号佳霖科创大厦 5 楼 3-14 号

邮编：610023

网址：www.xugudb.com

前言

概述

本文档主要介绍了虚谷数据库的系统函数信息，包含功能描述、语法格式、参数说明、返回值和示例。

读者对象

- 数据库管理员
- 软件工程师
- 技术支持工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
⚠ 注意	用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。
🕮 说明	对正文中重点信息的补充说明。“说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2025-01-10	第一次发布

目录

1 数值数据类型函数

1.1 概述

函数形式	功能
ABS	计算数值类型的字段或表达式的绝对值
ACOS	计算参数的反余弦值
ACOSD	计算参数的反余弦值，结果以度为单位
ACOSH	计算参数的反双曲余弦值
ASIN	计算参数的反正弦值
ASIND	计算参数的反正弦值，结果以度为单位
ASINH	计算参数的反双曲正弦值
ATAN	计算参数的反正切值
ATAND	计算参数的反正切值，结果以度为单位
ATANH	计算参数的反双曲正切值
ATAN2	计算参数 1/参数 2 的反正切值
ATAN2D	计算参数 1/参数 2 的反正切值，结果以度为单位
BIN	计算数值数据类型的字段或表达式的二进制
BITAND	返回两个参数进行二进制 AND 运算后对应的十进制数
CBRT	计算参数的立方根
CEILING	计算大于数值类型字段或表达式的最小整数值
接下页	

函数形式	功能
CONV	将数据从一个进制转为另一个进制
COS	计算参数的余弦值
COSD	计算参数的余弦值，结果以度为单位
COSH	计算参数的双曲余弦值
COT	计算参数的余切值
COTD	计算参数的余切值，结果以度为单位
DEGREES	将弧度转换为角度
DIV	计算参数 1/参数 2 的整数商
ERF	误差函数
ERFC	互补误差函数
FACTORIAL	计算参数的阶乘
FLOOR	计算小于数值类型字段或表达式的最大整数值
GCD	计算参数 1，参数 2 的最大公约数
GET_HASH	计算哈希值
HEX	将 DOUBLE 或 FLOAT 类型的字段或表达式转换为十六进制字符串
LCM	计算参数 1，参数 2 的最小公倍数
LOG	返回参数的自然对数
LOG2	返回以 2 为底的对数
NANVL	判断参数 1 是否非数字 (NaN)

接下页

函数形式	功能
PI	返回 π 的值
POW	同 power，返回参数 1 的参数 2 次方
RANDOM	获取一个随机数
RANDOM_NORMAL	获取遵循给定均值和方差的正态分布的随机数
REMAINDER	计算参数 1 除以参数 2 的余数
ROUND	将参数 1 四舍五入到参数 2 指定位置
ROUND_TIES_TO_EVEN	四舍六入 (银行家舍入法)
SETSEED	为后续的 random 和 random_normal 设置调用种子
SIN	计算参数的正弦值
SIND	计算参数的正弦值，结果以度为单位
SINH	计算参数的双曲正弦值
TAN	计算参数的正切值
TAND	计算参数的正切值，结果以度为单位
TANH	计算参数的双曲正切值
TRUNC	功能 1：将日期值参数 1，按照参数 2 日期格式裁剪。 功能 2：将数值类型的参数 1 按照小数点后的保留位数参数 2 进行截取，不进行四舍五入
TRUNCATE	按照小数位数，进行数值截取
TO_CHAR	将数值数据类型的字段或表达式转换为字符串
UNCOMPRESS_FLOAT	将 BINARY 类型数据解压为 FLOAT 类型数据

1.2 ABS

功能描述

计算指定数值表达式的绝对值。

语法格式

```
ABS(expr)
```

参数说明

expr：数值类型的字段或表达式。

函数返回类型

同数值数据类型。

示例

对单个数值求绝对值：

```
SQL> SELECT ABS(-100.4),ABS(100.4) FROM dual;

EXPR1 | EXPR2 |
-----------------------------------------------------------------
100.4| 100.4|
```

对字段中的数值求绝对值：

```
-- 建表并插入值
SQL> CREATE TABLE test (A int);
SQL> INSERT INTO test (A) VALUES(1)(-2);

-- 查询表中的A列值和A列值的绝对值
SQL> SELECT A, ABS(A) AS abs FROM test;

A | ABS |
-----------------------------------------------------------------
1 | 1 |
-2 | 2 |
```

1.3 ACOS

功能描述

计算参数的反余弦值，结果以弧度表示。

输入范围是 [-1.0, 1.0]，输出范围是 [0, π]。

语法格式

```
ACOS(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT acos(1) FROM dual;
EXPR1 |
------------------------------------------------------------
0.000000e+00 |
```

1.4 ACOSD

功能描述

计算参数的反余弦值，结果以角度表示。

输入范围是 [-1.0, 1.0]，输出范围是 [0, 180]。

语法格式

```
ACOSD(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT acosd(0.5) FROM dual;
EXPR1 |
------------------------------------------------------------
6.000000e+01 |
```

1.5 ACOSH

功能描述

计算参数的反双曲余弦值，结果以弧度表示。

输入范围是 [1.0, +∞)。

语法格式

```
ACOSH(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT acosh(1) FROM dual;
EXPR1 |
------------------------------------------------------------
0.000000e+00 |
```

1.6 ASIN

功能描述

计算参数的反正弦值，结果以弧度表示。

输入范围是 [-1.0, 1.0]，输出范围是 [-π/2, π/2]。

语法格式

```
ASIN(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT asin(1) FROM dual;
EXPR1 |
------------------------------------------------------------
1.570796e+00 |
```

1.7 ASIND

功能描述

计算参数的反正弦值，结果以角度表示。

输入范围是 [-1.0, 1.0]，输出范围是 [-90, 90]。

语法格式

```
ASIND(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT asind(0.5) FROM dual;
EXPR1 |
------------------------------------------------------------
3.000000e+01 |
```

1.8 ASINH

功能描述

计算参数的反双曲正弦值，结果以弧度表示。

语法格式

```
ASINH(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT asinh(1) FROM dual;
EXPR1 |
------------------------------------------------------------
8.813736e-01 |
```

1.9 ATAN

功能描述

计算参数的反正切值，结果以弧度表示。当函数传入两个参数时，功能同 ATAN2。

输出范围是 (-π/2, π/2)。

语法格式

```
ATAN(expr1[, expr2])
```

参数说明

expr1：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT atan(1) FROM dual;
EXPR1 |
------------------------------------------------------------
7.853982e-01 |
```

1.10 ATAND

功能描述

计算参数的反正切值，结果以角度表示。

输出范围是 (-90, 90)。

语法格式

```
ATAND(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT atand(1) FROM dual;
EXPR1 |
------------------------------------------------------------
4.500000e+01 |
```

1.11 ATANH

功能描述

计算参数的反双曲正切值，结果以弧度表示。

输入范围是 [-1.0, 1.0]。

语法格式

```
ATANH(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT atanh(0.5) FROM dual;
EXPR1 |
------------------------------------------------------------
5.493081e-01 |
```

1.12 ATAN2

功能描述

计算 expr1/expr2 的反正切值，结果以弧度表示。

输出范围是 (-π, π]。

语法格式

```
ATAN2(expr1,expr2)
```

参数说明

expr1、expr2：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT atan2(1,0) FROM dual;
EXPR1 |
------------------------------------------------------------
1.570796e+00 |
```

1.13 ATAN2D

功能描述

计算 expr1/expr2 的反正切值，结果以角度表示。

输出范围是 (-180, 180]。

语法格式

```
ATAN2D(expr1, expr2)
```

参数说明

expr1、expr2：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT atan2d(1,0) FROM dual;
EXPR1 |
------------------------------------------------------------
9.000000e+01 |
```

1.14 BIN

功能描述

计算数值数据类型的字段或表达式的二进制。

语法格式

```
BIN(expr)
```

参数说明

expr：DOUBLE 或 FLOAT 类型的字段或表达式。

函数返回类型

CHAR 类型。

示例

对单个数值求二进制表示：

```
SQL> SELECT BIN(12) FROM dual;

EXPR1 |
------------------------------------------------------------
1100|

SQL> SELECT BIN(12.1) FROM dual;

EXPR1 |
------------------------------------------------------------
1100|
```

对列中的数值求二进制表示：

```
-- 创 建 表 并 插 入 数 据
SQL> CREATE TABLE numbers (id INT AUTO_INCREMENT, value INT);

SQL> INSERT INTO numbers (value) VALUES (0), (1), (2), (7), (10);
```

```
-- 查询并显示二进制表示
SQL> SELECT id, value, BIN(value) AS binary_value FROM numbers;

ID | VALUE | BINARY_VALUE |
-------------------------------------------------------------
1 | 0 | 0|
2 | 1 | 1|
3 | 2 | 10|
4 | 7 | 111|
5 | 10 | 1010|
```

1.15 BITAND

功能描述

BITAND 是用于执行按位与操作的函数。

它对两个参数的二进制表示进行逐位比较，只有当两个对应的位都为 1 时，结果的相应位才为 1，否则为 0。最终的结果再转为十进制。

语法格式

```
BITAND(expr1, expr2)
```

参数说明

expr1、expr2：十进制整数类型 INTEGER、BIGINT 或 NUMERIC；函数针对参数对应的二进制值进行运算。

函数返回类型

INTEGER 或 BIGINT 数值类型。

示例

两个参数 12 和 6 对应二进制分别为 1100 和 0110，按位与计算后的结果为 0100，其对应的十进制值为 4。

```
SELECT BITAND(12, 6);

EXPR1 |
-------------------------------------------
4 |
```

1.16 CBRT

功能描述

计算参数的立方根。

语法格式

```
CBRT(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT CBRT(8) FROM dual;
EXPR1 |
------------------------------------------------------------
2.000000e+00 |
```

1.17 CEILING

功能描述

计算大于数值类型字段或表达式的最小整数值。

语法格式

```
CEILING(expr)
```

参数说明

expr：数值类型的字段或表达式。

函数返回类型

BIGINT 或 NUMERIC 数值类型。

示例

```
SQL> SELECT CEILING(123.6) FROM dual;

EXPR1 |
-----------------------------------------------
124|
```

1.18 CONV

功能描述

将数据从一个进制转为另一个进制。支持从 2 到 36 之间的任意进制之间的转换。

与 MySQL 无差异。

语法格式

```
CONV(expr1,expr2,expr3)
```

参数说明

- expr1：数据。
- expr2：原进制。
- expr3：要转换的进制，输入的范围是 [2, 36]，其正负号指示 expr1 是有符号数还是无符号数。

函数返回类型

VARCHAR 类型。

示例

```
SQL> SELECT CONV('20',10,16);

EXPR1 |
------------------------------------------------------------------
14|

SQL> SELECT CONV('101',2,10);

EXPR1 |
------------------------------------------------------------------
5|
```

1.19 COS

功能描述

计算参数的余弦值，输入值以弧度为单位。

语法格式

```
COS(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT cos(0) FROM dual;
EXPR1 |
------------------------------------------------------------
1.000000e+00 |
```

1.20 COSD

功能描述

计算参数的余弦值，输入值以角度为单位。

语法格式

```
COSD(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT cosd(60) FROM dual;
EXPR1 |
------------------------------------------------------------
5.000000e-01 |
```

1.21 COSH

功能描述

计算参数的双曲余弦值。

语法格式

```
COSH(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT cosh(0) FROM dual;
EXPR1 |
------------------------------------------------------------
1.000000e+00 |
```

1.22 COT

功能描述

计算参数的余切值，输入值以弧度为单位。

语法格式

```
COT(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT cot(0.5) FROM dual;
EXPR1 |
------------------------------------------------------------
1.830488e+00 |
```

1.23 COTD

功能描述

计算参数的余切值，输入值以角度为单位。

语法格式

```
COTD(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT cotd(45) FROM dual;
EXPR1 |
------------------------------------------------------------
1.000000e+00 |
```

1.24 DEGREES

功能描述

将弧度转换为角度。

语法格式

```
DEGREES(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT degrees((pi/180)*90) FROM dual;
EXPR1 |
------------------------------------------------------------
9.000000e+01 |
```

1.25 DIV

功能描述

计算 expr1/expr2 的整数商。

语法格式

```
DIV(expr1,expr2)
```

参数说明

- expr1：被除数，NUMERIC 类型，或能隐式转换为 NUMERIC 类型的其他类型。
- expr2：除数，NUMERIC 类型，或能隐式转换为 NUMERIC 类型的其他类型。

函数返回类型

NUMERIC 类型。

示例

```
SQL> SELECT div(8,3) FROM dual;
EXPR1 |
------------------------------------------------------------
2|
```

1.26 ERF

功能描述

误差函数。

语法格式

```
ERF(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT erf(1.0) FROM dual;
EXPR1 |
------------------------------------------------------------
8.427008e-01 |
```

1.27 ERFC

功能描述

互补误差函数。定义为 1 减去误差函数的值。

语法格式

```
ERFC(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT ERFC(1.0) FROM dual;

EXPR1 |
------------------------------------------------------------
1.572992e-01 |
```

1.28 FACTORIAL

功能描述

计算参数的阶乘。

输入范围是 [0, 34)，受限于返回值类型的取值范围。

语法格式

```
FACTORIAL(expr)
```

参数说明

expr：BIGINT 类型，或者能隐式转换为 BIGINT 类型的其他类型。

函数返回类型

NUMERIC 类型。

示例

```
SQL> SELECT factorial(3) FROM dual;

EXPR1 |
------------------------------------------------------------
6|
```

1.29 FLOOR

功能描述

计算小于数值类型字段或表达式的最大整数值。

语法格式

```
FLOOR(expr)
```

参数说明

expr：数值类型的字段或表达式。

函数返回类型

DOUBLE、FLOAT、NUMERIC 数值类型。

示例

```
SQL> SELECT FLOOR(-123.6) FROM dual;

EXPR1 |
------------------------------------------------------------
-124|
```

1.30 GCD

功能描述

计算 expr1，expr2 的最大公约数。

语法格式

```
GCD(expr1,expr2)
```

参数说明

expr1、expr2：NUMERIC 类型，或能隐式转换为 NUMERIC 类型的其他类型。

函数返回类型

NUMERIC 类型。

示例

```
SQL> SELECT gcd(15,35) FROM dual;
EXPR1 |
------------------------------------------------------------
5|
```

1.31 GET_HASH

功能描述

计算参数 expr1、expr2 的哈希值。如果其中一个参数为 NULL，则结果为 NULL。

语法格式

```
GET_HASH(expr1, expr2)
```

参数说明

expr1、expr2：数值类型的字段或表达式。

函数返回类型

BIGINT 数值类型。

示例

```
SELECT GET_HASH(123, 456);

EXPR1 |
------------------------------------------------------------
1507692052 |

SELECT GET_HASH(123, NULL);

EXPR1 |
------------------------------------------------------------
<NULL>|
```

1.32 HEX

功能描述

将 DOUBLE 或 FLOAT 类型的字段或表达式转换为十六进制字符串。该函数和 UNHEX 互为反函数，即返回值可以作为对方的参数。

语法格式

```
HEX(expr)
```

参数说明

expr：DOUBLE 或 FLOAT 类型的字段或表达式。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT HEX(16) FROM dual;

EXPR1 |
------------------------------------------------------------
10|
```

1.33 LCM

功能描述

计算 expr1，expr2 的最小公倍数。

语法格式

```
LCM(expr1,expr2)
```

参数说明

expr1、expr2：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT lcm(15,35) FROM dual;
EXPR1 |
------------------------------------------------------------------
105|
```

1.34 LOG

功能描述

返回参数的自然对数。

与 MySQL 差异：在虚谷数据库中，NUMERIC 类型的参数需要指明类型。

语法格式

```
LOG(expr)
```

参数说明

expr：FLOAT 或 DOUBLE 类型的数值。

函数返回类型

FLOAT 或 DOUBLE 类型。

示例

```
SQL> SELECT LOG(5,25);

EXPR1 |
------------------------------------------------------------------
2.000000e+00 |
```

1.35 LOG2

功能描述

返回以 2 为底的对数。

与 MySQL 差异：在虚谷数据库中，NUMERIC 类型的参数需要指明类型。

语法格式

```
LOG2(expr)
```

参数说明

expr：FLOAT 或 DOUBLE 类型的数值。

函数返回类型

FLOAT 或 DOUBLE 类型。

示例

```
SQL> SELECT log2(4);

EXPR1 |
------------------------------------------------------------
2.000000e+00 |
```

1.36 NANVL

功能描述

判断 expr1 是否为 NaN（非数字），当 expr1 为 NaN 时，函数返回 expr2，否则返回 expr1。

语法格式

```
NANVL(expr1,expr2)
```

参数说明

expr1、expr2：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT nanvl('nan'::double,1.2),nanvl(1.5::double,1.2) FROM
   dual;
EXPR1 | EXPR2 |
------------------------------------------------------------
1.200000e+00 | 1.500000e+00 |
```

1.37 PI

功能描述

返回 π 的值。

与 MySQL 无差异。

语法格式

```
PI()
```

参数说明

无参数。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT PI();

EXPR1 |
------------------------------------------------------------
3.141593e+00 |

SQL> SELECT PI()+1;

EXPR1 |
------------------------------------------------------------
4.141593e+00 |
```

1.38 POW

功能描述

同 POWER，返回 expr1 的 expr2 次方。

与 MySQL 无差异。

语法格式

```
POW(expr1,expr2)
```

参数说明

- expr1：基数。
- expr2：指数。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT POW(5,2);

EXPR1 |
------------------------------------------------------------
2.500000e+01 |
```

1.39 RANDOM

功能描述

获取一个随机数。

输出范围是 [0.0, 1)。

语法格式

```
RANDOM()
```

参数说明

无参数。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT random() FROM dual;
EXPR1 |
------------------------------------------------------------
8.613146e-01 |
```

1.40 RANDOM_NORMAL

功能描述

获取一个遵循正态分布的随机数。

语法格式

```
RANDOM_NORMAL([expr1[,expr2]])
```

参数说明

- expr1：平均值，DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。默认值为 0.0。
- expr2：标准差，DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。默认值为 1.0。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT random_normal(0,1) FROM dual;
EXPR1 |
------------------------------------------------------------
1.140062e+00 |
```

1.41 REMAINDER

功能描述

计算 expr1 除以 expr2 的余数。

语法格式

```
REMAINDER(expr1, expr2)
```

参数说明

- expr1：被除数，数值类型或者能隐式转换为数值类型的其他类型。
- expr2：除数，数值类型或者能隐式转换为数值类型的其他类型。

函数返回类型

与参数相同的数值数据类型。

示例

```
SQL> SELECT remainder(3, 2) FROM dual;
EXPR1 |
------------------------------------------------------------
-1|
```

1.42 ROUND

功能描述

将 expr1 四舍五入到 expr2 指定位置。expr2 为可选参数，默认为 0，将 expr1 裁剪为 0。

语法格式

```
ROUND(expr1[,expr2])
```

参数说明

- expr1：数值类型的字段或表达式。
- expr2：可选参数，保留的小数位。数值类型的字段或表达式。

函数返回类型

与参数相同的数值数据类型。

示例

```
SQL> SELECT ROUND(20.369,2) FROM dual;

EXPR1 |
------------------------------------------------------------
20.37|
```

1.43 ROUND_TIES_TO_EVEN

功能描述

采用银行家舍入法 (四舍六入) 对 expr1 进行舍入。

语法格式

```
ROUND_TIES_TO_EVEN(expr1[,expr2])
```

参数说明

- expr1：NUMERIC 类型或者能隐式转换为 NUMERIC 类型的其他类型。
- expr2：可选参数，INTEGER 类型或者能隐式转换为 INTEGER 类型的其他类型，默认为 0。
  - expr2 为正时，expr1 将舍入到小数点右侧的整数位数。
  - expr2 为负时，expr1 将舍入到小数点左侧的整数位数。

四舍六入规则

舍入位	规则 1	规则 2	规则 3
[0,4]	直接舍去	—	—
[6,9]	进位舍去	—	—
[5]	看舍入位之后	—	—
—	非 0	进位舍去	—
—	为 0	看舍入位前一位的奇偶	—

接下页

舍入位	规则 1	规则 2	规则 3
—	—	为偶数	直接舍去
—	—	为奇数	进位舍去

函数返回类型

NUMERIC 类型。

示例

```
SQL> SELECT round_ties_to_even(1.234,2),round_ties_to_even(1.236,2)
   ,round_ties_to_even(1.225,2),round_ties_to_even(1.235,2) FROM
   dual;
EXPR1 | EXPR2 | EXPR3 | EXPR4 |
-----------------------------------------------------------------
1.23| 1.24| 1.22| 1.24|
```

1.44 SETSEED

功能描述

为后续的 RANDOM 和 RANDOM_NORMAL 设置调用种子。通过设置种子，你可以确保每次运行程序时生成相同的随机数序列。

输入范围是 [-1.0, 1.0]。

语法格式

```
SETSEED(expr)
```

参数说明

expr：expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

CHAR 类型，该函数返回一个空字符串。

示例

```
SQL> SELECT setseed(0.5),random() FROM dual;

EXPR1 | EXPR2 |
-----------------------------------------------------------------
| 9.851677e-01 |
```

1.45 SIN

功能描述

计算参数的正弦值，输入值以弧度为单位。

语法格式

```
SIN(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT sin(1) FROM dual;
EXPR1 |
------------------------------------------------------------
8.414710e-01 |
```

1.46 SIND

功能描述

计算参数的正弦值，输入值以角度为单位。

语法格式

```
SIND(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT sind(30) FROM dual;
EXPR1 |
------------------------------------------------------------
5.000000e-01 |
```

1.47 SINH

功能描述

计算参数的双曲正弦值。

语法格式

```
SINH(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT sinh(1) FROM dual;
EXPR1 |
------------------------------------------------------------
1.175201e+00 |
```

1.48 TAN

功能描述

计算参数的正切值，输入值以弧度为单位。

语法格式

```
TAN(expr)
```

参数说明

expr：DOUBLE/FLOAT 类型，或能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

函数返回类型

DOUBLE/FLOAT 类型，与参数类型一致。

示例

```
SQL> SELECT tan(1) FROM dual;
EXPR1 |
------------------------------------------------------------
1.557408e+00 |
```

1.49 TAND

功能描述

计算参数的正切值，输入值以角度为单位。

语法格式

```
TAND(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT tand(45) FROM dual;
EXPR1 |
------------------------------------------------------------
1.000000e+00 |
```

1.50 TANH

功能描述

计算参数的双曲正切值。

语法格式

```
TANH(expr)
```

参数说明

expr：DOUBLE 类型，或能隐式转换为 DOUBLE 类型的其他类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT tanh(1) FROM dual;
EXPR1 |
------------------------------------------------------------
7.615942e-01 |
```

1.51 TRUNC

功能描述

将数值类型的 expr1 按照小数点后的保留位数 expr2 进行截取，不进行四舍五入。

也可用作日期截断，详细信息请参见时间数据类型函数的 TRUNC 函数。

语法格式

```
TRUNC(expr1[,expr2])
```

参数说明

- expr1：expr1：待截取的数值。
- expr2：expr2：小数点后面的保留位数，可选项，默认为 0。

第二个参数可以为负数，表示将小数点左边指定位数后面的部分截去，且均以 0 记。

函数返回类型

与参数相同的数值数据类型。

示例

```
SQL> SELECT TRUNC(89.985,2),TRUNC(89.985),TRUNC(89.985,-1) FROM
   dual;

EXPR1 | EXPR2 | EXPR3 |
-----------------------------------------------------------
89.98 | 89| 80|
```

1.52 TRUNCATE

功能描述

按照小数位数，进行数值截取。

与 MySQL 无差异。

语法格式

```
TRUNCATE(expr1,expr2)
```

参数说明

- expr1：DOUBLE/FLOAT 类型的数值。
- expr2：小数位数。

函数返回类型

DOUBLE/FLOAT 类型。

示例

```
SQL> SELECT TRUNCATE(123.4567, 2);

EXPR1 |
------------------------------------------------------------
123.45 |

SQL> SELECT TRUNCATE(123.4567, 0);

EXPR1 |
------------------------------------------------------------
123 |

SQL> SELECT TRUNCATE(123.4567, -1);

EXPR1 |
------------------------------------------------------------
120 |
```

1.53 TO_CHAR

功能描述

将数值数据类型的字段或表达式转换为字符串。

语法格式

```
TO_CHAR(expr)
```

参数说明

expr：数值类型的字段或表达式。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT TO_CHAR(123.4) FROM dual;

EXPR1 |
------------------------------------------------------------
123.4|
```

1.54 UNCOMPRESS_FLOAT

功能描述

将 BINARY 类型数据解压为 FLOAT 类型数据，当前版本仅支持两位小数的解压。

FLOAT[] 数组类型的压缩与解压可参考数组类型函数的 COMPRESS_FLOATS 与 UNCOMPRESS_FLOATS 函数。

语法格式

```
UNCOMPRESS_FLOAT(expr1, expr2)
```

参数说明

- expr1：需要解压的数据，BINARY 数据类型。
- expr2：偏移量，INTEGER 数据类型。

函数返回类型

FLOAT 类型。

若任意参数为 NULL，则返回 NULL。

示例

```
SQL> SELECT UNCOMPRESS_FLOAT(HEXTORAW('020000000010110'),2);

EXPR1 |
------------------------------------------------------------
1.0|
```

2 字符数据类型函数

2.1 概述

函数形式	功能
ATOF	将 VARCHAR 或 CHAR 类型的字段或表达式的值转换为 DOUBLE 类型
ATOL	将 VARCHAR 或 CHAR 类型的字段或表达式的值转换为 BIGINT 类型
BASE64_DECODE	将 CHAR 类型的字段或表达式的值按照 BASE64 算法解码为 BINARY 类型的字符串
BASE64_ENCODE	将 BINARY 类型的字段或表达式的值按照 BASE64 算法编码为 CHAR 类型的字符串
BIT_LENGTH	以位为单位返回给定字符串的长度
BTRIM	从给定字符串 1 首、尾删除由指定字符串 2 组成的最长字符串
CHARACTER_LENGTH	字符串长度测试函数，CHAR_LENGTH() 为其同义函数
CONCAT	连接指定字符串
CONCAT_WS	以指定连接符连接指定字符串，第一个参数为连接符
CONVERT	将字符串从原字符集转换为另一个字符集
DECODE_PG	将 CHAR 类型的字段或表达式的值按照指定算法解码为 BINARY 类型的字符串
ELT	返回字符串列表中的第 n 个字符串
接下页	

函数形式	功能
ENCODE_PG	将 BINARY 类型的字段或表达式的值按照指定算法编码为 CHAR 类型的字符串
ESCAPE_DECODE	将 CHAR 类型的字段或表达式的值按照 ESCAPE 算法解码为 BINARY 类型的字符串
ESCAPE_ENCODE	将 BINARY 类型的字段或表达式的值按照 ESCAPE 算法编码为 CHAR 类型的字符串
FIELD	返回 str 在 str1，str2，str3，... 列表中的索引（位置）
FIND_IN_SET	在逗号分隔的字符串列表中查找指定字符串的位置
FROM_BASE64	将 CHAR 类型的字段或表达式的值按照 BASE64 算法解码为 BINARY 类型的数据
HEADING	计算取 VARCHAR 或 CHAR 类型的字段或表达式参数 1 头部的参数 2 个字符的子串
HEX_DECODE	将 CHAR 类型的字段或表达式的值按照 HEX 算法解码为 BINARY 类型的字符串
HEX_ENCODE	将 BINARY 类型的字段或表达式的值按照 HEX 算法编码为 CHAR 类型的字符串
INITCAP	将指定字符串中每个单词的首字母转为大写，其余字母转为小写
INSERT	将源串中指定位置指定长度的字符串替换为新的字符串
INSTR	从字符串参数 1 中获取参数 2 的开始位置
ISNULL	检查表达式是否为 NULL。如果传递的表达式为 NULL，则此函数返回 true，否则返回 false

接下页

函数形式	功能
LCASE	字符串转小写
LEFT	截取指定字符串参数 1 左侧参数 2 个字符
LEFTB	在 VARCHAR 或 CHAR 类型的字段或变量或表达式参数 1 中，从左边开始取参数 2 个字节
LEN	计算 VARCHAR 或 CHAR 类型的字段或表达式的长度。LENGTH 为其同义函数
LENGTH	获取 BIT 类型数据的位数
LENGTHB	计算参数字符串的长度，单位为字节
LOCATE	返回子串在源串中第一次出现的位置
LOWER	将 VARCHAR 或 CHAR 类型的字段或表达式中的字符转换为小写形式
LPAD	使用字符串参数 1 来填充参数 3 的开始处，使字符串长度达到参数 2 长度
LTRIM	将 VARCHAR 或 CHAR 类型的字段或表达式头部空格去掉
MID	字符串截取
OCTET_LENGTH	返回字符串中的字节数
OVERLAY	使用指定字符串替换给定字符串中从指定开始位置的 N 个字符
POSITION	计算串参数 1 在串参数 2 中的起始位置（即从第几个字符开始），参数 2 的第一个字符位置数为 1，其余类推
REGEXP_COUNT	统计字符串出现的次数

接下页

函数形式	功能
REGEXP_LIKE	比较给定的字符串是否与正则表达式匹配
REPEAT	将指定字符串重复参数 2 次
REPLACE	将参数 1 中的子串参数 2 用参数 3 替换，参数 2 是参数 1 的子串
REVERSE_STR	将 VARCHAR 或 CHAR 类型的字段或表达式反写
RIGHT	截取右侧 N 个字符
RIGHTB	在 VARCHAR 或 CHAR 类型的字段或变量或表达式参数 1 中，从右边开始取参数 2 个字节
ROWIDTOCHAR	将 ROWID 数据类型转换为 CHAR 字符类型输出
RPAD	指定字符串参数 3 填充至参数 1 结尾处，使字符串的长度达到参数 2
RTRIM	将 VARCHAR 或 CHAR 类型的字段或表达式尾部空格去掉
SPACE	返回指定长度的空白字符
SPLIT_PART	将给定字符串按指定分隔符分割并返回第 n 部分（n 小于 0，从右到左计算第 n 部分）
SQLCODE	返回错误码
STARTS_WITH	检测给定字符串 expr1 是否以指定前缀 expr2 开头
STRCMP	比较两个字符串
STRPOS	返回指定字符串 expr2 在给定字符串 expr1 中第一次出现的位置

接下页

函数形式	功能
STUFF	将参数 1 字符串中第参数 2 个字符开始的参数 3 个字符用字符串参数 4 替换
STRCMP	比较两个字符串
SUBSTR	当未指定参数 3 时，截取参数 1 中参数 2 到末尾的子串；当指定参数 3 时，则从参数 1 的第参数 2 个字符开始截取参数 3 长度的子串
SUBSTRB	截取目标字符串指定的部分
SUBSTRING	字符串截取
SUBSTRING_INDEX	按指定分隔符截取字符串
TAILING	在 VARCHAR 或 CHAR 类型的字段或表达式参数 1 中取尾部的参数 2 个字符的子串
TO_BASE64	将 BINARY 类型的字段或表达式的值按照 BASE64 算法编码为 CHAR 类型的字符串
TO_HEX	将数字转为等效的 16 进制
TRANSLATE	依次查找参数 1 中的每个字符是否在参数 2 中存在，如果不存在，那么返回参数 1 相应位置的字符；否则，将用参数 3 中与参数 2 同样位置的字符替换参数 1 中的字符
UCASE	字符串转大写
UNHEX	将十六进制值转换为字符串。该函数和 HEX 互为反函数，即返回值可以作为对方的参数
UPPER	将 VARCHAR 或 CHAR 类型的字段或表达式中的字符转换为大写形式

接下页

函数形式	功能
WM_CONCAT	将列参数按照分组条件进行行转列操作，各行记录经过分组后以逗号拼接返回输出

2.2 ATOF

功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式的值转换为 DOUBLE 类型。

语法格式

```
ATOF(expr)
```

参数说明

expr：一个合法的数值字串，如：’123.456’，否则结果将不正确。

函数返回类型

DOUBLE 数值类型。

示例

```
SQL> SELECT ATOF(1) FROM dual;

EXPR1 |
------------------------------------------------------------
1.0 |
```

2.3 ATOL

功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式的值转换为 BIGINT 类型。

语法格式

```
ATOL(expr)
```

参数说明

expr：一个合法的数值字串, 如：’12345’，否则结果将不正确。

函数返回类型

BIGINT 数值类型。

示例

```
SQL> SELECT ATOL(123456) FROM dual;

EXPR1 |
------------------------------------------------------------
123456 |
```

2.4 BASE64_DNCODE

功能描述

将 CHAR 类型的字段或表达式的值按照 BASE64 算法解码为 BINARY 类型的字符串。

语法格式

```
BASE64_DECODE(expr);
```

参数说明

expr：数据类型为 CHAR，待解码字符串。

函数返回类型

BINARY 类型。

示例

```
SQL> SELECT RAWTOHEX(BASE64_DECODE('EjRWeJCrze8AAQ==')) FROM DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |
```

2.5 BASE64_ENCODE

功能描述

将 BINARY 类型的字段或表达式的值按照 BASE64 算法编码为 CHAR 类型的字符串。

语法格式

```
BASE64_ENCODE(expr);
```

参数说明

expr：数据类型为 BINARY，待编码数据。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT BASE64_ENCODE(HEXTORAW('1234567890ABCDEF0001')) FROM
   DUAL;

EXPR1 |
------------------------------------------------------------
EjRWeJCrze8AAQ== |
```

2.6 BIT_LENGTH

功能描述

计算字符串的位长度。

与 MySQL 差异：数据库字符集为 GBK 的情况下，BIT_LENGTH 参数为中文时返回的值与 MySQL 不一致。

语法格式

```
BIT_LENGTH(expr)
```

参数说明

expr：要转换的字符串。

函数返回类型

INTEGER 类型。

示例

字符集为 UTF8 时：

```
SQL> SELECT BIT_LENGTH('WQAQ');

EXPR1 |
------------------------------------------------------------
32 |

SQL> SELECT BIT_LENGTH('一二');

EXPR1 |
------------------------------------------------------------
48 |
```

字符集为 GBK 时：

```
SQL> SELECT BIT_LENGTH('WQAQ');

EXPR1 |
------------------------------------------------------------
32 |

SQL> SELECT BIT_LENGTH('一二');
```

```
EXPR1 |
------------------------------------------------------------------
32 |
```

2.7 BTRIM

功能描述

去除字符串 expr1 两端的指定字符串 expr2。如果不指定 expr2，默认会去除空格。

语法格式

```
BTRIM(expr1[, expr2])
```

参数说明

expr1，expr2: 字符数据类型。

函数返回类型

字符数据类型。

示例

```
SQL> SELECT BTRIM('lxr', 'lr') FROM DUAL;

EXPR1 |
------------------------------------------------------------------
x|
```

2.8 CHARACTER_LENGTH

功能描述

计算字符串的字符长度。CHAR_LENGTH() 为其同义函数。

语法格式

```
CHARACTER_LENGTH(expr)
```

参数说明

expr: 字符串字段或表达式。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT CHAR_LENGTH('ABCDE') FROM dual;

EXPR1 |
-------------------------------------------------------------
5 |
```

2.9 CONCAT

功能描述

连接指定字符串。

与 MySQL 差异：

- 对于参数类型为二进制字符串：MySQL 支持 BLOB 类型系列，而不支持 BINARY 类型；虚谷数据库支持 BINARY 类型，不支持 BLOB 类型。
- 兼容 Oracle 和 PG，跳过 NULL 值。
- CONCAT 函数中，空和空字符串有以下 3 种特殊情况：
  - CONCAT(NULL, NULL) 返回 NULL。
  - CONCAT(”,”) 返回空字符串。
  - CONCAT(”,NULL) 返回空字符串。

  def_empty_str_as_null（系统级参数，可在线修改）仅影响后面两种情况的输出，且结果与 MySQL 存在差异，MySQL 第 2 种情况返回空字符串，第 3 种情况返回 NULL，参数无法同时兼容这两种情况的输出。

语法格式

```
CONCAT(expr1,expr2[,expr3,expr4...])
```

参数说明

expr1、expr2、expr3、expr4：字符串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT CONCAT('一 二 三 四', '-34.56');

EXPR1 |
-------------------------------------------------------------
一 二 三 四-34.56|
```

```
SQL> SELECT CONCAT('一二三四', '-34.56','A','BBBAAA');

EXPR1 |
------------------------------------------------------------
一二三四-34.56ABBBAAA|
```

2.10 CONCAT_WS

功能描述

以指定连接符连接指定字符串，第一个参数为连接符。

与 MySQL 差异：

- 对于参数类型为二进制字符串：MySQL 支持 BLOB 类型系列，而不支持 BINARY 类型；虚谷数据库支持 BINARY 类型，不支持 BLOB 类型。
- 函数无参数或只有一个参数时：MySQL 报错；虚谷数据库可执行，结果返回 NULL。
- CONCAT_WS(‘#’,NULL,NULL) 返回 NULL，其结果不受 def_empty_str_as_null 参数影响，且结果与 MySQL 有差异，MySQL 返回空字符串。

语法格式

```
CONCAT_WS(expr1,expr2,expr3[,expr4....])
```

参数说明

- expr1：连接符，字符串类型。
- expr2、expr3、expr4：字符串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT CONCAT_WS('一二','A','B');

EXPR1 |
------------------------------------------------------------
A一二B|
```

2.11 CONVERT

功能描述

将字符串从一个语言字符集转换为另一个字符集。

语法格式

```
CONVERT(expr1, expr2[, expr3])
```

参数说明

- expr1：源字符串。
- expr2：目的字符集。
- expr3：源字符集。

函数返回类型

CHAR 类型字符串。

示例

```
-- 查看当前数据库信息，字符集为UTF8
SQL> SHOW DB_INFO

DB_NAME | DB_ID | DB_OWNER | DB_CHARSET | DB_TIMEZ |
-----------------------------------------------------------------
SYSTEM| 1 | SYS| UTF8.UTF8_GENERAL_CI| GMT+08:00|

-- 将字符串'test_convert'从默认字符集转换为GBK
SQL> SELECT CONVERT('test_convert','GBK');

EXPR1 |
-----------------------------------------------------------------
test_convert|

-- 查看转换为GBK字符集的字符串字节数
SQL> SELECT LENGTHB(CONVERT('我爱中国','GBK','UTF8'));

EXPR1 |
-----------------------------------------------------------------
8 |

-- 新建数据库，默认字符集为GBK
SQL> CREATE DATABASE db_convert CHARACTER SET 'GBK' TIME ZONE 'GMT
   +08:00';

SQL> USE db_convert

-- 将字符串'test_convert'从默认字符集转换为UTF8
SQL> SELECT CONVERT('test_convert','UTF8');

EXPR1 |
-----------------------------------------------------------------
test_convert|

-- 查看转换为UTF8字符集的字符串字节数
SQL> SELECT LENGTHB(CONVERT('我爱中国','UTF8','GBK'));
```

```
EXPR1 |
------------------------------------------------------------
12 |
```

2.12 DECODE_PG

功能描述

将 CHAR 类型的字段或表达式的值按照指定算法解码为 BINARY 类型的字符串。

语法格式

```
DECODE_PG(expr1,expr2);
```

参数说明

- expr1：数据类型为 CHAR，待解码字符串。
- expr2：数据类型为 CHAR，解码算法名称，可选值为：BASE64、ESCAPE、HEX。

函数返回类型

BINARY 类型。

示例

```
SQL> SELECT RAWTOHEX(DECODE_PG('EjRWeJCrze8AAQ==','BASE64')) FROM
   DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |

SQL> SELECT RAWTOHEX(DECODE_PG(' 4Vx\220\253\315\357\000 ','ESCAPE'
   )) FROM DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |

SQL> SELECT RAWTOHEX(DECODE_PG('1234567890abcdef0001','HEX')) FROM
   DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |
```

2.13 ELT

功能描述

根据索引返回指定位置的值，索引小于 1 或大于字符串列表中字符串个数或者为 NULL 时，函数返回 NULL。

ELT 函数与 FIELD 函数功能互补。

语法格式

```
ELT(n, str1, str2, str3, ...)
```

参数说明

- n：要返回的值的位置（从 1 开始计数），INTEGER 类型。
- str1, str2, str3, ...：要在其中查找的值列表，CHAR 类型。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT ELT(2, 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') FROM DUAL;

EXPR1 |
------------------------------------------------------------
Bb|

SQL> SELECT ELT(10, 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') FROM DUAL;

EXPR1 |
------------------------------------------------------------
<NULL>|
```

2.14 ENCODE_PG

功能描述

将 BINARY 类型的字段或表达式的值按照指定算法编码为 CHAR 类型的字符串。

语法格式

```
ENCODE_PG(expr1,expr2);
```

参数说明

- expr1：数据类型为 BINARY，待编码数据。
- expr2：数据类型为 CHAR，编码算法名称，可选值为：BASE64、ESCAPE、HEX。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT ENCODE_PG(HEXTORAW('1234567890ABCDEF0001'),'BASE64')
   FROM DUAL;

EXPR1 |
------------------------------------------------------------
EjRWeJCrze8AAQ== |

SQL> SELECT ENCODE_PG(HEXTORAW('1234567890ABCDEF0001'),'ESCAPE')
   FROM DUAL;

EXPR1 |
------------------------------------------------------------
 4Vx\220\253\315\357\000  |

SQL> SELECT ENCODE_PG(HEXTORAW('1234567890ABCDEF0001'),'HEX') FROM
   DUAL;

EXPR1 |
------------------------------------------------------------
1234567890abcdef0001 |
```

2.15 ESCAPE_DECODE

功能描述

将 CHAR 类型的字段或表达式的值按照 ESCAPE 算法解码为 BINARY 类型的字符串。

语法格式

```
ESCAPE_DECODE(expr);
```

参数说明

expr：数据类型为 CHAR，待解码字符串。

函数返回类型

BINARY 类型。

示例

```
SQL> SELECT RAWTOHEX(ESCAPE_DECODE(' 4Vx\220\253\315\357\000 '))
   FROM DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |
```

2.16 ESCAPE_ENCODE

功能描述

将 BINARY 类型的字段或表达式的值按照 ESCAPE 算法编码为 CHAR 类型的字符串。

语法格式

```
ESCAPE_ENCODE(expr);
```

参数说明

expr：数据类型为 BINARY，待编码数据。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT ESCAPE_ENCODE(HEXTORAW('1234567890ABCDEF0001')) FROM
   DUAL;

EXPR1 |
------------------------------------------------------------------
 4Vx\220\253\315\357\000  |
```

2.17 FIELD

功能描述

在一组值中查找某个值的位置，返回该值第一次在列表中出现的索引（位置），如果找不到该值，则返回 0。

FIELD 函数与 ELT 函数功能互补。

语法格式

```
FIELD(str, str1, str2, str3, ...)
```

参数说明

- str：要查找的值，INTEGER 或 CHAR 类型。
- str1, str2, str3, ...：要在其中查找的值列表，INTEGER 或 CHAR 类型。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') FROM DUAL;
EXPR1 |
------------------------------------------------------------------
2|

SQL> SELECT FIELD('Gg', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') FROM DUAL;
```

```
EXPR1 |
------------------------------------------------------------
0|
```

2.18 FIND_IN_SET

功能描述

在逗号分隔的字符串列表中查找指定字符串的位置。

语法格式

```
FIND_IN_SET(expr1,expr2)
```

参数说明

- expr1：要查找的字符串。
- expr2：要搜索的逗号分隔的字符串列表。

> 说明
>
> - 如果 expr1 在 expr2 中，则返回一个正整数。
> - 如果 expr1 不在 expr2 中，或者 expr2 是空字符串，则返回零。
> - 如果 expr1 或 expr2 为 NULL，则函数返回 NULL 值。

函数返回类型

INTEGER 数值类型。

示例

- 示例 1

  基本使用。c 是字符串的第 3 个元素，返回 3。

```
SELECT FIND_IN_SET('c', 'a,b,c,d');

EXPR1|
-----+
3|
```

- 示例 2

  在查询中使用。查询表 tab_test 中包含 3 的所有记录。

```
# 创建基础用例表
CREATE TABLE tab_test (
id VARCHAR(100)
);
```

```
#初始化用例表数据
INSERT INTO tab_test (id) VALUES
('1,2,3'),
('2,4,6'),
('3,5,7'),
('4,6,8');

# 在查询语句中使用FIND_IN_SET进行结果集过滤
SELECT * FROM tab_test WHERE FIND_IN_SET('3', id) > 0;

ID    |
------+
1,2,3|
3,5,7|
```

- 示例 3

  与字段结合使用。查询表 tab_test 中 col_list 列包含 col_name 的所有记录。

```
# 创建基础用例表
CREATE TABLE tab_test (
id int NOT NULL,
col_name varchar(255) NOT NULL,
col_list varchar(255) NOT NULL,
PRIMARY KEY (id)
);

# 初始化用例表数据
INSERT INTO tab_test(id,col_name,col_list) VALUES (1, '张三', '张
    三,李四,王武,小王');
INSERT INTO tab_test(id,col_name,col_list) VALUES (2, '李四', '张
    三,李四,王武');
INSERT INTO tab_test(id,col_name,col_list) VALUES (3, '王武', '张
    三,李四');
INSERT INTO tab_test(id,col_name,col_list) VALUES (4, '小王', '张
    三');

# 在查询语句中使用FIND_IN_SET进行结果集过滤
SELECT * FROM tab_test WHERE FIND_IN_SET(col_name,col_list) > 0;

ID|COL_NAME|COL_LIST       |
--+--------+---------------+
1|张三     |张三,李四,王武,小王|
2|李四     |张三,李四,王武     |
```

2.19 FROM_BASE64

功能描述

将 CHAR 类型的字段或表达式的值按照 BASE64 算法解码为 BINARY 类型的数据。

与 MySQL 差异：当解码数据不是 BASE64 编码时，MySQL 返回 NULL，虚谷数据库报错。

语法格式

```
FROM_BASE64(expr)
```

参数说明

expr：数据类型为 CHAR，待解码字符串。

函数返回类型

BINARY 类型。

示例

```
SQL> SELECT TO_CHAR(FROM_BASE64('MTIzNDU2Nzg5MEFCQ0RFRjAwMDE='))
   FROM DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |
```

2.20 HEADING

功能描述

截取字符串 expr1 的前 expr2 个字符。

语法格式

```
HEADING(expr1,expr2)
```

参数说明

- expr1：字串表达式或字段，VARCHAR 或 CHAR 类型。
- expr2：子串长度。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT HEADING('ABCDEFGHIJK',4) FROM dual;

EXPR1 |
------------------------------------------------------------
ABCD|
```

2.21 HEX_DECODE

功能描述

将 CHAR 类型的字段或表达式的值按照 HEX 算法解码为 BINARY 类型的字符串。

语法格式

```
HEX_DECODE(expr);
```

参数说明

expr：数据类型为 CHAR，待解码字符串。

函数返回类型

BINARY 类型。

示例

```
SQL> SELECT RAWTOHEX(HEX_DECODE('1234567890abcdef0001')) FROM DUAL;

EXPR1 |
------------------------------------------------------------
1234567890ABCDEF0001 |
```

2.22 HEX_ENCODE

功能描述

将 BINARY 类型的字段或表达式的值按照 HEX 算法编码为 CHAR 类型的字符串。

语法格式

```
HEX_ENCODE(expr);
```

参数说明

expr：数据类型为 BINARY，待编码数据。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT HEX_ENCODE(HEXTORAW('1234567890ABCDEF0001')) FROM DUAL;

EXPR1 |
------------------------------------------------------------
1234567890abcdef0001 |
```

2.23 INITCAP

功能描述

将指定字符串中每个单词的首字母转为大写，其余字母转为小写。

语法格式

```
INITCAP(expr)
```

参数说明

expr：参数为字符串，并且字符串中的单词以非字母数字字符分割。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT INITCAP('heLlO wOrLd'),INITCAP('aBC#DEF你cEd');

EXPR1 | EXPR2 |
-------------------------------------------------------------
Hello World| Abc#Def你Ced|
```

2.24 INSERT

功能描述

将源串中指定位置指定长度的字符串替换为新的字符串。

与 MySQL 差异：对于参数类型为 BLOB，MySQL 可以返回结果，而虚谷数据库会产生报错。

语法格式

```
INSERT(expr1,expr2,expr3,expr4)
```

参数说明

- expr1：源字符串。
- expr2：替换位置。
- expr3：替换长度。
- expr4：新字符串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT INSERT('abc1ABC1abcABcd',2,3,'WWWW');

EXPR1 |
-------------------------------------------------------------
aWWWWABC1abcABcd|
```

2.25 INSTR

功能描述

从字符串 expr1 中获取 expr2 首次出现的位置。

语法格式

```
INSTR(expr1,expr2[,expr3,expr4])
```

参数说明

- expr1：源字符串。
- expr2：字符串。
- expr3：从字符串 expr1 的哪个位置开始查找，此参数可选。
  - 如果省略，默认为 1，字符串索引从 1 开始。
  - 如果此参数为正，从左到右检索。
  - 如果此参数为负，从右到左检索，返回要查找的字符串在源字符串中的开始索引。
- expr4：查找第几次出现的目标字符串，此参数可选。
  - 如果省略，默认为 1。
  - 如果此参数为负，则报错。

函数返回类型

整型数值类型。

示例

```
SQL> SELECT INSTR('abbaxycdaxzbapqw', 'ba', -2) col_1 FROM dual;

COL_1 |
------------------------------------------------------------
12 |

SQL> SELECT INSTR('abbaxycdaxzbapqwba', 'ba', 1, 1) col_1 FROM dual
   ;

COL_1 |
------------------------------------------------------------
3 |

SQL> SELECT INSTR('abbaxycdaxzbapqwba', 'ba', 1, 2) col_1 FROM dual
   ;

COL_1 |
------------------------------------------------------------
```

```
12 |

SQL> SELECT INSTR('abbaxycdaxzbapqwba', 'ba', 1, 3) col_1 FROM dual
   ;

COL_1 |
------------------------------------------------------------
17 |

SQL> SELECT INSTR('abbaxycdaxzbapqwba', 'ba') col_1 FROM dual;

COL_1 |
------------------------------------------------------------
3 |

SQL> SELECT INSTR('abbaxycdaxzbapqwba', 'ba',2) col_1 FROM dual;

COL_1 |
------------------------------------------------------------
3 |
```

2.26 ISNULL

功能描述

检查表达式是否为 NULL。如果传递的表达式为 NULL，则此函数返回 true，否则返回 false。

与 MySQL 差异：

- MySQL：为 NULL 返回 1，不为 NULL 返回 0。
- 虚谷数据库：为 NULL 返回 true，不为 NULL 返回 false。

语法格式

```
ISNULL(expr)
```

参数说明

expr：要判断是否为 NULL 的表达式。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT ISNULL('12345') FROM dual;

EXPR1 |
------------------------------------------------------------
F |

SQL> SELECT ISNULL('abcd') FROM dual;
```

```
EXPR1 |
------------------------------------------------------------
F |

SQL> SELECT ISNULL(NULL::INT) res FROM dual;

RES |
------------------------------------------------------------
T |
```

2.27 LCASE

功能描述

字符串转小写。

与 MySQL 无差异。

语法格式

```
LCASE(expr)
```

参数说明

expr：字符串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT LCASE('AB123CD');

EXPR1 |
------------------------------------------------------------
ab123cd|

SQL> SELECT LCASE('AB123Cd');

EXPR1 |
------------------------------------------------------------
ab123cd|
```

2.28 LEFT

功能描述

从字符串 expr1 的左侧截取指定数量 expr2 的字符。

与 MySQL 无差异。

语法格式

```
LEFT(expr1,expr2)
```

参数说明

- expr1：字符串。
- expr2：截取长度。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT LEFT('1234321.32123432123',4);

EXPR1 |
------------------------------------------------------------------
1234|


SQL> SELECT LEFT('一二三，。？一二，。？一三四',4);

EXPR1 |
------------------------------------------------------------------
一二三，|
```

2.29 LEFTB

功能描述

从字符串 expr1 的左侧按字节提取指定数量 expr2 的字符。与 LEFT 函数不同，LEFTB 是基于字节而不是字符进行截取的。

语法格式

```
LEFTB(expr1,expr2)
```

参数说明

- expr1：VARCHAR 或 CHAR 类型的字段或表达式。
- expr2：数值类型的字段或表达式。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT LEFTB('ABC',2) FROM dual;

EXPR1 |
------------------------------------------------------------
AB|
```

2.30 LEN

功能描述

计算 VARCHAR 或 CHAR 类型的字段或表达式的长度。LENGTH 为其同义函数。

语法格式

```
LEN(expr)
```

参数说明

expr：字串表达式或字段。

函数返回类型

INT 数值类型。

示例

```
SQL> SELECT LEN('ABC 中 国') FROM dual;

EXPR1 |
------------------------------------------------------------
6 |
```

2.31 LENGTH

功能描述

获取获取字符串或二进制数据的长度。

语法格式

```
LENGTH(expr)
```

参数说明

expr：数据类型为 CHAR、VARBIT、BIT、CLOB、BLOB。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT LENGTH(b'1010101');

EXPR1 |
-----------------------------------------------------------------
7 |

SQL> SELECT LENGTH('中a');

EXPR1 |
-----------------------------------------------------------------
2 |
```

2.32 LENGTHB

功能描述

计算字符串的字节数。

语法格式

```
LENGTHB (expr)
```

参数说明

expr: 待求长度的字符串，类型为 CHAR、VARCHAR、CLOB 或 BLOB。

> **说明**
>
> 多字节字符在不同字符集编码的库中，执行结果可能有差别。例如中文在 UTF8 库中一个字符占 3 个字节，在 GBK 库中一个字符占 2 个字节。

函数返回类型

INTEGER 数值类型。

示例

以字符集为 UTF-8 的数据库为例：

```
-- 单字节字符
SELECT LENGTHB('abcde');

EXPR1 |
-----------------------------------------------------------------
5 |

-- 两字节字符
SELECT LENGTHB('Δ');

EXPR1 |
```

```
----------------------------------------------------------------
2 |

-- 三字节字符
SELECT LENGTHB('字');

EXPR1 |
----------------------------------------------------------------
3 |
```

2.33 LOCATE

功能描述

查找子字符串在源字符串中首次出现的位置。

与 MySQL 差异：

- 对于参数类型为 BLOB：MySQL 可以返回结果；虚谷数据库报错。
- 对于参数类型异常的情况（如 POS 参数为字符，而不是整数）：MySQL 返回 0；虚谷数据库报错。

语法格式

```
LOCATE(expr1,expr2[,expr3])
```

参数说明

- expr1：子字符串。
- expr2：源字符串。
- expr3：搜索的起始位置，它是可选的，默认为位置 1。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT LOCATE('b','abc1ABC1abcABcd');

EXPR1 |
----------------------------------------------------------------
2 |

SQL> SELECT LOCATE('b','abc1ABC1abcABcd',5);

EXPR1 |
----------------------------------------------------------------
10 |
```

2.34 LOWER

功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式中的字符转换为小写形式。

语法格式

```
LOWER(expr)
```

参数说明

expr：字串表达式或字段。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT LOWER('ABCDEF') FROM dual;

EXPR1 |
------------------------------------------------------------
abcdef|
```

2.35 LPAD

功能描述

在字符串 expr1 左侧填充指定字符串 expr3，填充后字符串长度为 expr2。

语法格式

```
LPAD(expr1,expr2[,expr3])
```

参数说明

- expr1：源字符串。
- expr2：填充后的字符串长度。如果长度短于源字符串，将会从左至右截取源字符串。
- expr3：可选参数，填充字符串，填充至源字符串的左边；如果未指定此参数，将会在源字符串的左边填充空格。

函数返回类型

CHAR 类型字符串。

示例

指定填充的字符串，且填充后长度小于源字符串：

```
SQL> SELECT LPAD('woaizhongguo',4,'huacaoshumu') FROM dual;

EXPR1 |
------------------------------------------------------------
woai|
```

不指定填充的字符串，且填充后长度小于源字符串：

```
SQL> SELECT LPAD('woaizhongguo',4) FROM dual;

EXPR1 |
------------------------------------------------------------
woai|
```

不指定填充的字符串，且填充后长度大于源字符串：

```
SQL> SELECT LPAD('woaizhongguo',20) FROM dual;

EXPR1 |
------------------------------------------------------------
        woaizhongguo|
```

2.36 LTRIM

功能描述

去除 VARCHAR 或 CHAR 类型的字段或表达式右侧的空格或从左侧逐个移除指定的字符，直到遇到第一个不匹配的字符为止。

语法格式

```
LTRIM(expr1[,expr2])
```

参数说明

- expr1：要进行左修剪操作的主字符串。
- expr2：可选参数，要从 expr1 左侧移除的字符。如果未指定，则默认去除空格。

函数返回类型

CHAR 类型字符串。

示例

去掉左侧空格：

```
SQL> SELECT LTRIM(' F D D') FROM dual;

EXPR1 |
------------------------------------------------------------
F D D|
```

从左侧逐个移除指定的字符，直到遇到第一个不匹配的字符为止：

```
SQL> SELECT LTRIM('!!#asd','!#') FROM dual;

EXPR1 |
------------------------------------------------------------
asd|
```

2.37 MID

功能描述

从字符串中截取子字符串。

与 MySQL 差异：

- 对于参数类型为 BLOB：MySQL 可以返回结果；虚谷数据库报错。
- 对于 POS 参数为 0 的情况：MySQL 结果为空串；虚谷数据库则是从开始位置截取字符串（类似于 Oracle 和 PG）。

语法格式

```
MID(expr1,expr2[,expr3])
MID(expr1 from expr2[for expr3])
```

参数说明

- expr1：字符串。
- expr2：截取字符串的位置。
- expr3：截取字符串的长度。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT MID('. SPACE TAB123',2,3);

EXPR1 |
------------------------------------------------------------
SP|

SQL> SELECT MID('. SPACE TAB123',2);

EXPR1 |
------------------------------------------------------------
SPACE TAB123|

SQL> SELECT MID('. SPACE TAB123' FROM 2);
```

```
EXPR1 |
------------------------------------------------------------
SPACE TAB123|

SQL> SELECT MID('. SPACE TAB123' FROM 2 FOR 3);

EXPR1 |
------------------------------------------------------------
SP|
```

2.38 OCTET_LENGTH

功能描述

返回字符串中的字节长度。

语法格式

```
OCTET_LENGTH(expr)
```

参数说明

expr：字符数据类型。

函数返回类型

INTEGER 类型。

示例

UTF-8 编码，每个中文字符占用 3 个字节。

```
SQL> SELECT OCTET_LENGTH('dgh 中 文') FROM DUAL;

EXPR1 |
------------------------------------------------------------
9|
```

2.39 OVERLAY

功能描述

使用指定字符串替换给定字符串中从指定开始位置的 N 个字符。

语法格式

```
OVERLAY(expr1 PLACING expr2 FROM expr3 [ FOR expr4 ])
```

参数说明

- expr1：要操作的原始字符串。
- expr2：用于替换原始字符串中部分字符的新子字符串。
- expr3：指定从原始字符串的哪个位置开始替换（从 1 开始计数）。
- FOR expr4：可选参数，指定要替换的子字符串的长度。如果不指定，默认替换从 expr3 位置开始到字符串末尾的所有字符。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT OVERLAY('lxr' PLACING 'r' FROM 1 FOR 2) FROM DUAL;
EXPR1 |
------------------------------------------------------------
rr|
```

2.40 POSITION

功能描述

查找子字符串 expr1 在主字符串 expr2 中首次出现的位置。expr2 的第一个字符位置数为 1，其余类推。

若 expr1 是 expr2 的子串，则返回值大于等于 1，否则返回 0。

语法格式

```
POSITION(expr1 IN expr2)
```

参数说明

- expr1：子字符串。
- expr2：母字符串。

函数返回类型

INT 数值类型。

示例

expr1 是 expr2 的子串：

```
SQL> SELECT POSITION('A'  IN  'BCDABCD') FROM dual;

EXPR1 |
------------------------------------------------------------
4 |
```

expr1 不是 expr2 的子串：

```
SQL> SELECT POSITION('E'  IN  'BCDABCD') FROM dual;

EXPR1 |
------------------------------------------------------------------
0 |
```

2.41 REGEXP_COUNT

功能描述

计算正则表达式在字符串中匹配次数。

语法格式

```
REGEXP_COUNT(expr1,expr2[,expr3[,expr4]])
```

参数说明

- expr1：源字符串，可以是列名或者字符串常量、变量。
- expr2：正则表达式。
- expr3：标识从第几个字符开始正则表达式匹配。
- expr4：匹配选项，取值范围如下：
  - i：大小写不敏感。
  - c：大小写敏感。
  - n：允许句点 (.) 作为通配符去匹配换行符，如果省略该参数，则句点将不匹配换行符。
  - m：将源串视为多行，即将和 ^ $ 分别看作源串中任意位置任何行的开始和结束，而不是仅仅看作整个源串的开始或结束，如果省略该参数，则将源串看作一行。
  - x：忽略空格字符，默认情况下，空格字符与自身相匹配。

> **说明**
>
> - 如果指定了多个相互矛盾的匹配选项，将使用最后一个选项。例如，如果指定'ic'，将使用区分大小写的匹配；如果指定的选项不属于上述范围，则报错。
> - 如果忽略 expr4，则是否区分大小写依照系统参数 CATA_CASE_SENSITIVE 的值确定（此参数的默认值是区分大小写的）。
> - 句点不匹配换行符。
> - 源字符串被视为单行。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT REGEXP_COUNT('asdasadaa','a');

EXPR1 |
------------------------------------------------------------------
5 |
```

2.42 REGEXP_LIKE

功能描述

用于模式匹配，比较给定的字符串是否与正则表达式匹配。如果字符串与正则表达式匹配，如果找到返回 true；如果未找到匹配项，则返回 false。

语法格式

```
REGEXP_LIKE (expression, pattern [, match_type])
```

参数说明

- expression：要进行匹配的表达式或字段，是一个字符串。
- pattern：要匹配的正则表达式模式。
- match_type：用于指定匹配的方式，是一个字符串。

函数返回类型

BOOLEAN 类型。

示例

输入字符串与正则表达式匹配，返回 true（此处“.”在正则表达式中使用时可以匹配任何字符）：

```
SQL> SELECT REGEXP_LIKE ('England or America', 'l.nd') AS Result;

RESULT |
------------------------------------------------------------------
T |
```

输入字符串与正则表达式不匹配，返回 false：

```
SQL> SELECT REGEXP_LIKE ('MCA', 'BCA') AS Result;

RESULT |
------------------------------------------------------------------
F |
```

2.43 REPEAT

功能描述

将指定字符串 expr1 重复 expr2 次。

与 MySQL 无差异。

语法格式

```
REPEAT(expr1,expr2)
```

参数说明

- expr1：字符串。
- expr2：重复的次数，INT 类型。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT REPEAT('ABC12',3);

EXPR1 |
------------------------------------------------------------------
ABC12ABC12ABC12|

SQL> SELECT REPEAT('中 国',5);

EXPR1 |
------------------------------------------------------------------
中 国 中 国 中 国 中 国 中 国|
```

2.44 REPLACE

功能描述

用 expr3 替换字符串 expr1 中指定的子字符串 expr2。

语法格式

```
REPLACE(expr1,expr2,expr3)
```

参数说明

- expr1：进行替换操作的字符串。
- expr2：被替换的子串。
- expr3：替换后的目标串。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT REPLACE('ABCDEF','CD','中 国') FROM dual;

EXPR1 |
------------------------------------------------------------
AB中 国EF|
```

2.45 REVERSE_STR

功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式反转，字符顺序颠倒。

语法格式

```
REVERSE_STR(expr)
```

参数说明

expr：字串表达式或字段。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT REVERSE_STR('ABDCEFG') FROM dual;

EXPR1 |
------------------------------------------------------------
GFECDBA|
```

2.46 RIGHT

功能描述

从字符串 expr1 的右侧截取指定数量 expr2 的字符。

与 MySQL 无差异。

语法格式

```
RIGHT(expr1,expr2)
```

参数说明

- expr1：字符串。
- expr2：截取长度。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT RIGHT('一 二 三，。？ 一 二，。？ 一 三 四',5);

EXPR1 |
------------------------------------------------------------
。？ 一 三 四|
```

2.47 RIGHTB

功能描述

从字符串 expr1 的右侧按字节提取指定数量 expr2 的字符。与 RIGHT 函数不同，RIGHTB 是基于字节而不是字符进行截取的。

语法格式

```
RIGHTB(expr1,expr2)
```

参数说明

- expr1：VARCHAR 或 CHAR 类型的字段或表达式。
- expr2：数值类型的字段或表达式。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT RIGHTB('ABC',2) FROM dual;

EXPR1 |
------------------------------------------------------------
BC|
```

2.48 ROWIDTOCHAR

功能描述

将 ROWID 数据类型转换为 CHAR 字符类型。

ROWID 是数据库系统中用于唯一标识表中每一行的内部标识符。

语法格式

```
ROWIDTOCHAR(expr)
```

参数说明

expr: ROWID 类型的字段或变量或表达式。

函数返回类型

CHAR 类型字符串。

示例

```
SQL> CREATE TABLE  test_rowidtochar(a INT);

SQL> INSERT INTO  test_rowidtochar VALUES(1)(2);

SQL> SELECT ROWIDTOCHAR(ROWID) FROM  test_rowidtochar;

EXPR1 |
------------------------------------------------------------------
AAAAAFUCAAAAAAAAAAAAAA==|
AQAAAFUCAAAAAAAAAAAAAA==|
```

2.49 RPAD

功能描述

在字符串 expr1 的右侧填充指定字符 expr3，填充后的字符串长度为 expr2。

语法格式

```
RPAD(expr1,expr2[,expr3])
```

参数说明

- expr1：源字符串。
- expr2：填充后的字符串长度，如果长度短于源字符串，将会从左至右截取源字符串。
- expr3：可选参数，填充字符串，填充至源字符串的右侧。如果未指定此参数，将会在源字符串的右边填充空格。

函数返回类型

CHAR 类型字符串。

示例

指定填充的字符串，且填充后长度小于 expr1 和 expr3 的长度和：

```
SQL> SELECT RPAD('tt',3,'wzq');

EXPR1 |
------------------------------------------------------------
ttw|
```

指定填充的字符串，且填充后长度等于 expr1 和 expr3 的长度和：

```
SQL> SELECT RPAD('tt',5,'wzq');

EXPR1 |
------------------------------------------------------------
ttwzq|
```

指定填充的字符串，且填充后长度小于 expr1 和 expr3 的长度和：

```
SQL> SELECT RPAD('tt',0,'wzq');

EXPR1 |
------------------------------------------------------------
|
```

不指定填充的字符串，且填充后长度小于 expr1 长度：

```
SQL> SELECT RPAD('tt',1);

EXPR1 |
------------------------------------------------------------
t|
```

2.50 RTRIM

功能描述

去除 VARCHAR 或 CHAR 类型的字段或表达式右侧的空格或从右侧逐个移除指定的字符，直到遇到第一个不匹配的字符为止。

语法格式

```
RTRIM(expr1,[expr2])
```

参数说明

- expr1：要进行右修剪操作的主字符串。
- expr2：可选参数，要从 expr1 右侧移除的字符。如果未指定，则默认去除空格。

函数返回类型

CHAR 类型字符串。

示例

去除右侧空格：

```
SQL> SELECT RTRIM('asd ') FROM dual;

EXPR1 |
------------------------------------------------------------
asd|
```

从右侧逐个移除指定的字符，直到遇到第一个不匹配的字符为止：

```
SQL> SELECT RTRIM('asd!!#','!#') FROM dual;

EXPR1 |
------------------------------------------------------------
asd|

SQL> SELECT RTRIM('asd ','d ') FROM dual;

EXPR1 |
------------------------------------------------------------
as|

SQL> SELECT RTRIM('asd ','d') FROM dual;

EXPR1 |
------------------------------------------------------------
asd |
```

2.51 SPACE

功能描述

返回指定数量的空白字符。

语法格式

```
SPACE(expr)
```

参数说明

expr：指定长度的数值。

函数返回类型

CHAR 类型字符串。

示例

```
SQL> SELECT SPACE(0) FROM dual;

EXPR1 |
------------------------------------------------------------
|
```

```
SQL> SELECT SPACE(1) FROM dual;

EXPR1 |
------------------------------------------------------------
 |

SQL> SELECT SPACE(10) FROM dual;

EXPR1 |
------------------------------------------------------------
          |
```

2.52 SPLIT_PART

功能描述

将给定字符串按指定分隔符分割并返回第 n 部分（n<0，从右到左计算第 n 部分）。

语法格式

```
SPLIT_PART(expr1, expr2, expr3)
```

参数说明

- expr1：源字符串，CHAR，VARCHAR 或者能隐式转换为 CHAR，VARCHAR 的类型。
- expr2：分割字符，CHAR，VARCHAR 或者能隐式转换为 CHAR，VARCHAR 的类型。
- expr3：指定返回的部分 (n)，INTEGER 类型。

函数返回类型

VARCHAR 类型。

示例

```
SQL> SELECT split_part('www.xxxx.com','.',2);

EXPR1 |
------------------------------------------------------------
xxxx|

SQL> SELECT split_part('www.xxxx.com','.',-3);

EXPR1 |
------------------------------------------------------------
www|
```

2.53 SQLCODE

功能描述

返回错误码。

语法格式

```
SQLCODE
```

参数说明

无参数。

函数返回类型

INTEGER 数值类型。

示例

在 PL/SQL 块中创建一个名为 test_sqlcode 的表之后，在 PL/SQL 块中再次创建同名表，使用 SQLCODE 函数返回错误 9016，直接执行 SQL 语句返回错误码和错误信息。

```
SQL> BEGIN
CREATE TABLE test_sqlcode(a INT);
EXCEPTION
WHEN OTHERS THEN
SEND_MSG(SQLCODE);
END;
/

SQL> BEGIN
CREATE TABLE test_sqlcode(a INT);
EXCEPTION
WHEN OTHERS THEN
SEND_MSG(SQLCODE);
END;
/
9016

SQL> CREATE TABLE test_sqlcode(a INT);
Error: [E9016]  同 名 Table 对 象 TEST_SQLCODE 已 存 在
```

2.54 STARTS_WITH

功能描述

检查字符串是否以指定前缀开头。

语法格式

```
STARTS_WITH(expr1, expr2)
```

参数说明

- expr1：要检查的字符串。

- expr2：要检查的前缀。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT STARTS_WITH('hello', 'he') FROM DUAL;

EXPR1 |
------------------------------------------------------------
T |
```

2.55 STRCMP

功能描述

比较两个字符串。

与 MySQL 差异：

- 对于表中的数据，MySQL 默认大小写不敏感，而虚谷大小写敏感，因此会导致一些字符串比较结果有差异。
- 对于参数类型为 BLOB，MySQL 可以返回结果；虚谷数据库报错。
- 忽略大小写的问题与字符集相关，GBK 和 UTF8 下的比较是要区分大小写，MySQL 默认使用的排序规则是不区分大小写。

语法格式

```
    STRCMP(expr1,expr2)
```

参数说明

expr1、expr2：字符串。

函数返回类型

TINYINT 类型。返回值范围：(-1, 0, 1)。

示例

```
SQL> SELECT STRCMP('ABC','abc');

EXPR1 |
------------------------------------------------------------
0 |

SQL> SELECT STRCMP('12','10');

EXPR1 |
```

```
-----------------------------------------------------------------
1 |

SQL> SELECT STRCMP('a','b');

EXPR1 |
-----------------------------------------------------------------
-1 |
```

2.56 STRPOS

功能描述

查找子字符串在主字符串中首次出现的位置。如果未找到子字符串，则返回 0。

语法格式

```
STRPOS(expr1, expr2)
```

参数说明

- expr1：要搜索的主字符串。
- expr2：要查找的子字符串。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT STRPOS('hello','l') FROM DUAL;
EXPR1 |
-----------------------------------------------------------------
3|
```

2.57 STUFF

功能描述

将 expr1 字符串中第 expr2 个字符开始的 expr3 个字符用字符串 expr4 替换。

语法格式

```
STUFF(expr1,expr2,expr3,expr4)
```

参数说明

- expr1：字串表达式或字段。
- expr2：起始替换位置。

- expr3：被替换子串的长度。
- expr4：替换成的目标串。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT STUFF('ABCDEF',2,3,'X') FROM dual;

EXPR1 |
------------------------------------------------------------
AXEF|
```

2.58 SUBSTR

功能描述

从字符串 expr1 中截取第 expr2 个字符到末尾的子字符串。

指定 expr3 时，则从 expr1 的第 expr2 个字符开始截取 expr3 长度的子串。

语法格式

```
SUBSTR(expr1,expr2[,expr3])
```

参数说明

- expr1：要从中截取子字符串的源字符串。
- expr2：为截取的起始位置；无论正负，截取方向总是向右。
  - 为正数时，截取开始位置从左向右数；绝对值大于字符串长度时结果为空。
  - 为 0 时，截取整个字符串。
  - 为负数时，截取开始位置从右向左数；绝对值大于字符串长度时结果为空。
- expr3：要截取的子字符串的长度。该值小于 1 时结果为空。

函数返回类型

CHAR 类型字符串。

示例

从左边数第二个字符开始截取到字符串末尾：

```
SQL> SELECT SUBSTR('ABCDEFG',2) FROM dual;

EXPR1 |
------------------------------------------------------------
```

```
BCDEFG|
```

从右边数第二个字符开始截取到字符串末尾：

```
SQL> SELECT SUBSTR('ABCDEFG',-2) FROM dual;

EXPR1 |
------------------------------------------------------------
FG|
```

从左边数第三个字符开始截取字符串 5 个长度的字符：

```
SQL> SELECT SUBSTR('今天天气特别好',3,5) FROM dual;

EXPR1 |
------------------------------------------------------------
天气特别好|
```

2.59 SUBSTRB

功能描述

从字符串 expr1 中截取第 expr2 个字节到末尾的子字符串。

指定 expr3 时，则从 expr1 的第 expr2 个字节开始截取 expr3 字节长度的子串。

SUBSTRB 是基于字节位置提取子字符串的，SUBSTR 是基于字符位置提取子字符串。

语法格式

```
SUBSTRB (expr1, expr2 [, expr3 ])
```

参数说明

- expr1：要从中截取子字符串的源字符串。
- expr2：为截取的起始位置（单位为字节）；无论正负，截取方向总是向右。
  - 为正数时，截取开始位置从左向右数；绝对值大于字符串长度时结果为 NULL。
  - 为 0 时，截取整个字符串。
  - 为负数时，截取开始位置从右向左数；绝对值大于字符串长度时结果为 NULL。
- expr3：要截取的子字符串的长度（单位为字节）；该值小于 1 时结果为 NULL。
- 两个参数时：expr2 为 INTEGER 类型。
- 三个参数时：expr2 与 expr3 为 NUMERIC 类型。

> **说明**
>
> 如果待截取的字符串为多字节字符，在不同字符集的库中执行结果可能有差别：例如中文在 UTF8 库中一个字符占 3 个字节，在 GBK 库中一个字符占 2 个字节。

函数返回类型

VARCHAR 类型字符串。

示例

从左边数第二个字节处开始截取到字符串末尾：

```
SELECT SUBSTRB('abcdefg', 2);

EXPR1 |
------------------------------------------------------------
bcdefg|
```

从右边数第二个字节处开始截取到字符串末尾：

```
SELECT SUBSTRB('abcdefg', -2);

EXPR1 |
------------------------------------------------------------
fg|
```

从左边数第二个字节处开始截取字符串 5 个字节长度的字符：

```
SELECT SUBSTRB('abcdefg', 2, 3);
EXPR1 |
------------------------------------------------------------
bcd|
```

下面示例以字符集 GBK 为例，一个字符占 2 个字节。

从目标字符串第七个字节（此处为第三个字符）处开始截取，向右截取至目标串结尾：

```
SELECT SUBSTRB('一 二 三 四 五 六', 7);

EXPR1 |
------------------------------------------------------------
三 四 五 六|
```

从目标字符串第七个字节（此处为第三个字符）处开始截取，向右截取 6 个字节（此处为两个字符）：

```
SELECT SUBSTRB('一 二 三 四 五 六', 7, 6);

EXPR1 |
------------------------------------------------------------
三 四|
```

2.60 SUBSTRING

功能描述

字符串截取。

与 MySQL 差异：

- 对于参数类型为 BLOB：MySQL 可以返回结果；虚谷数据库报错。
- 对于 POS 参数为 0 的情况：MySQL 结果为空串；虚谷数据库则是从开始位置截取字符串（类似于 Oracle 和 PG）。

语法格式

```
SUBSTRING(expr1,expr2[,expr3])
SUBSTRING(expr1 from expr2[for expr3])
```

参数说明

- expr1：字符串。
- expr2：截取字符串的位置，INT 类型数值，为负数则从右往左开始数位置。
- expr3：需要截取的长度，INT 类型数值。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT SUBSTRING('. SPACE TAB123',2);

EXPR1 |
------------------------------------------------------------
SPACE TAB123|

SQL> SELECT SUBSTRING('. SPACE TAB123',2,3);

EXPR1 |
------------------------------------------------------------
SP|

SQL> SELECT SUBSTRING('. SPACE TAB123' FROM 2  FOR 3);

EXPR1 |
------------------------------------------------------------
SP|

SQL> SELECT SUBSTRING('. SPACE TAB123' FROM 2);

EXPR1 |
------------------------------------------------------------
```

```
SPACE TAB123|
```

2.61 SUBSTRING_INDEX

功能描述

根据指定的分隔符和次数来分割字符串，并返回分割后的某一部分。

与 MySQL 无差异。

语法格式

```
SUBSTRING_INDEX(expr1,expr2,expr3)
```

参数说明

- expr1：要处理的字符串。
- expr2：用于分割字符串的分隔符。
- expr3：表示要返回的子串的数量。如果为正数，则从左边开始分割并返回前 expr3 个子串；如果为负数，则从右边开始分割并返回后 |expr3| 个子串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT SUBSTRING_INDEX('www.xxx.com', '.', 1) FROM dual;

EXPR1 |
------------------------------------------------------------
www|

SQL> SELECT SUBSTRING_INDEX(SUBSTRING_INDEX('www.xxx.com', '.', 2)
   , '.', -1) FROM dual;

EXPR1 |
------------------------------------------------------------
xxx|

SQL> SELECT SUBSTRING_INDEX(SUBSTRING_INDEX('www.xxx.com', '.', 3)
   , '.', -2) FROM dual;

EXPR1 |
------------------------------------------------------------
xxx.com|
```

2.62 TAILING

功能描述

在 VARCHAR 或 CHAR 类型的字段或表达式 expr1 中取尾部的 expr2 个字符的子串。

语法格式

```
TAILING(expr1,expr2)
```

参数说明

- expr1：字串表达式或字段。
- expr2：子串长度。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT TAILING('ABCDEFGH',4) FROM dual;

EXPR1 |
------------------------------------------------------------
EFGH|
```

2.63 TO_BASE64

功能描述

将 BINARY 类型的字段或表达式的值按照 BASE64 算法编码为 CHAR 类型的字符串。

> **说明**
>
> 与 MySQL 无差异。

语法格式

```
TO_BASE64(expr)
```

参数说明

expr：BINARY 类型，待编码数据。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT TO_BASE64('1234567890ABCDEF0001') FROM DUAL;

EXPR1 |
------------------------------------------------------------
MTIzNDU2Nzg5MEFCQ0RFRjAwMDE= |
```

2.64 TO_HEX

功能描述

将数字转为等效的 16 进制。

语法格式

```
TO_HEX(expr)
```

参数说明

expr：INTEGER/BIGINT 类型。

函数返回类型

VARCHAR 类型。

示例

```
SQL> SELECT to_hex(202301);

EXPR1(CHAR(-1)) |
------------------------------------------------------------
3163d|

Total 1 records.

Use time:0 ms.

SQL> SELECT to_hex(65535);

EXPR1(CHAR(-1)) |
------------------------------------------------------------
ffff|
```

2.65 TRANSLATE

功能描述

依次查找 expr1 中的每个字符是否在 expr2 中存在，如果不存在，那么返回 expr1 相应位置的字符；如果存在，将用 expr3 中与 expr2 同样位置的字符替换 expr1 中的字符。

语法格式

```
TRANSLATE(expr1,expr2,expr3)
```

参数说明

- expr1：要处理的源字符串。
- expr2：包含要被替换的字符的字符串。expr2 中的每个字符都会与 expr3 中相应位置的字符进行一对一替换。
- expr3：包含替换字符的字符串。如果 expr3 的长度小于 expr2，则 expr2 中多余的字符将被删除（即替换为空）。

函数返回类型

VARCHAR 类型字符串。

示例

将字符串 ABCDEFGH 中的 ZACDEF 替换为 12345678。

- 字符 A、C、D、E 和 F 分别被替换为 2、3、4、5 和 6。
- 字符 B、G 和 H 没有在 ZACDEF 中找到匹配项，因此它们保持不变。

```
SQL> SELECT TRANSLATE('ABCDEFGH','ZACDEF','12345678') FROM dual;

EXPR1 |
------------------------------------------------------------------
2B3456GH|
```

2.66 UCASE

功能描述

字符串转大写。

与 MySQL 无差异。

语法格式

```
UCASE(expr)
```

参数说明

expr：CHAR 类型的字符串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT UCASE('abcd');

EXPR1 |
------------------------------------------------------------
ABCD|

SQL> SELECT UCASE('ab123cd');

EXPR1 |
------------------------------------------------------------
AB123CD|
```

2.67 UNHEX

功能描述

将十六进制值转换为字符串。该函数和 HEX(expr) 互为反函数，即返回值可以作为对方的参数。

语法格式

```
UNHEX(expr)
```

参数说明

expr: 十六进制值。

函数返回类型

VARCHAR 类型字符串。

示例

```
SELECT UNHEX('616263') FROM dual;

EXPR1 |
------------------------------------------------------------
abc|

SELECT HEX('abc') FROM dual;

EXPR1 |
------------------------------------------------------------
616263|
```

2.68 UPPER

功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式中的字符转换为大写形式。

语法格式

```
UPPER(expr)
```

参数说明

expr：字串表达式或字段。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT UPPER('abcdefg') FROM dual;

EXPR1 |
------------------------------------------------------------
ABCDEFG|
```

2.69 WM_CONCAT

功能描述

将列按照分组条件进行行转列操作，各行记录经过分组后以逗号拼接返回输出。

语法格式

```
WM_CONCAT(expr)
```

参数说明

expr：列名。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> CREATE TABLE tab(a INT,b VARCHAR(20));

SQL> INSERT INTO tab VALUES(1,'test1')(2,'test2')(3,'test3');

SQL> INSERT INTO tab VALUES(1,'张 三')(2,'李 四')(3,'小 王');

SQL> SELECT a,WM_CONCAT(b) FROM tab GROUP BY a;

A | EXPR1 |
------------------------------------------------------------
1 | test1,张 三|
2 | test2,李 四|
3 | test3,小 王|
```

```
SQL> SELECT a, b FROM tab;

A | B |
-----------------------------------------------------------------
1 | test1|
2 | test2|
3 | test3|
1 | 张 三|
2 | 李 四|
3 | 小 王|
```

3 时间日期数据类型函数

3.1 概述

函数形式	功能
ADDDATE	时间加
ADD_MONTHS	返回在“参数 1”基础上增加“参数 2”个月份后的日期
ADDTIME	将指定的时间间隔（如天数、小时、分钟等）添加到给定的日期或时间值上
CLOCK_TIMESTAMP	返回真正的当前时间，因此它的值甚至在同一条 SQL 命令中都会发生变化
CURDATE	获取当前日期
CURRENT_DATETIME	返回当前时间
CURTIME	返回当前时间
DATE	获取指定日期时间中的日期
DATE_ADD	时间加
DATE_FORMAT	按指定格式串格式化时间
DATE_PART	从日期/时间值中抽取子域，例如年、月、日、时、分等
DATE_SUB	时间减
DATE_TRUNC	将时间戳或时间间隔截取至指定精度
DATEDIFF	计算两个时间差
DAY	返回日期为当月的第几天
	接下页

函数形式	功能
DAYNAME	返回指定日期所在工作日名称
DAYOFMONTH	返回日期为当月的第几天
DAYOFWEEK	返回参数日期为一周中的第几天（值从 1 到 7，周日的值为 1）
DAYOFYEAR	返回参数日期为一年中的第几天
EXTRACT	从时间参数中取出并返回指定的时间域（年、月、日、小时、分钟、秒）
EXTRACT_DAY	取日期的天
EXTRACT_HOUR	取日期的小时
EXTRACT_MINUTE	取日期的分钟
EXTRACT_MONTH	取日期的月
EXTRACT_SECOND	取日期的秒
EXTRACT_YEAR	取日期的年
FROM_DAYS	数字转时间（TO_DAYS 函数的反向函数）
FROM_UNIXTIME	将时间戳（毫秒）转换为时间
GET_BOOT_TIME	获取当前节点的启动时间
GETDAY	获取当前时间数据类型参数中的日
GETHOUR	获取当前时间数据类型参数中的小时
GETMINUTE	获取当前时间数据类型参数中的分钟
GETMONTH	获取当前时间数据类型参数中的月份
GETSECOND	获取当前时间数据类型参数中的秒

接下页

函数形式	功能
GET_UPTIME	获取当前节点上虚谷数据库服务已经运行的秒数
GETYEAR	获取当前时间数据类型参数中的年份
GET_FORMAT	获取时间格式串
HOUR	返回时间中的小时
LAST_DAY	获取当前时间数据类型参数中所在月的最后一天
LOCALTIME	返回系统时间，同 SYSDATE
LOCALTIMESTAMP	返回系统时间，同 SYSDATE
MAKE_DATE	根据年，月，日创建日期
MAKE_TIME	根据时，分，秒创建时间
MAKE_TIMESTAMP	根据年，月，日，时，分，秒创建日期时间
MAKE_TIMESTAMPTZ	根据年，月，日，时，分，秒创建带时区的日期时间
MICROSECOND	返回时间中的微秒
MINUTE	返回时间中的分钟
MONTH	返回日期的月份
MONTHNAME	返回日期/日期时间所在月份的英文名
MONTHS_BETWEEN	取时间的月份差，参数 1 为起始日期，参数 2 为截止日期，若参数 2 大于参数 1，则返回负数
NEXT_DAY	计算距离指定时间最近的周几的日期
NOW	取当前时间
PERIOD_ADD	给指定时间增加特定的月份数

接下页

函数形式	功能
PERIOD_DIFF	返回两个时间相差的月份数，前一个时间减后一个时间
QUARTER	返回指定日期所在季度
SECOND	返回时间中的秒
SEC_TO_TIME	将指定秒数转换为时间
STATEMENT_TIMESTAMP	返回当前语句的开始时刻（收到客户端最后一条命令的时间）
SUBDATE	时间减
SUBTIME	时间减
SYSDATE	取系统当前时间
SYSDATETIME	取系统当前日期时间，格式为 YYYY-MM-DD HH24:MI:SS.FFF
SYSTIME	取系统当前时间，格式为 HH24:MI:SS.FFF
SYSTIMESTAMP	返回数据库所在系统的系统日期
TIME	返回日期时间中的时间
TIME_FORMAT	按指定格式串格式化时间
TIMEDIFF	计算两个时间差
TIMEOFDAY	返回一个格式化的字符串，包括：”星期缩写月份缩写日期时：分：秒. 微秒年份时区缩写”
TIME_TO_SEC	指定时间转换为秒
TIMESTAMP	将传入的参数转为 DATETIME
TIMESTAMPADD	时间加

接下页

函数形式	功能
TIMESTAMPDIFF	时间减
TO_DATE	把参数 1 字符串按照可选的参数 2 格式转换为日期时间类型
TO_DAYS	计算从 [0000-00-00] 开始到指定日期的天数
TO_SECONDS	计算从 [0000-00-00] 开始到指定日期的秒数
TO_TIMESTAMP	将字符串参数 1 按照指定格式转换为 TIMESTAMP 类型
TRANSACTION_TIMESTAMP	获取当前日期时间（事务开始的时间），同一个事务内，该函数返回值不变
TRUNC	将时间戳截取至指定精度
WEEK	返回指定日期所在的周数
WEEKDAY	返回指定日期在工作日中的索引
WEEKOFYEAR	返回指定日期所在的日历周，返回值 [1，53]，等同于 WEEK(date, 3)
YEAR	返回日期的年份
YEARWEEK	计算指定日期所在的周数，并返回年份和周数
UNIX_TIMESTAMP	返回参数对应的 UNIX 格式时间戳。该值为相对于’1970-01-01 00:00:00.000’ 的毫秒数
UTC_DATE	返回 UTC 日期，与 SYSDATE 相差 8 小时
UTC_TIME	返回 UTC 时间
UTC_TIMESTAMP	返回 UTC 日期时间，与 SYSDATE 相差 8 小时

interval expr

时间间隔	格式
INTERVAL YEAR	’YEARS’
INTERVAL MONTH	’MONTHS’
INTERVAL DAY	’DAYS’
INTERVAL HOUR	’HOURS’
INTERVAL MINUTE	’MINUTES’
INTERVAL SECOND	’SECONDS.MICROSECONDS’
INTERVAL YEAR TO MONTH	’YEARS-MONTHS’
INTERVAL DAY TO HOUR	’DAYS HOURS’
INTERVAL DAY TO MINUTE	’DAYS HOURS:MINUTES’
INTERVAL DAY TO SECOND	’DAYSHOURS:MINUTES:SECONDS.MICROSECONDS’
INTERVAL HOUR TO MINUTE	’HOURS:MINUTES’
INTERVAL HOUR TO SECOND	’HOURS:MINUTES:SECONDS.MICROSECONDS’
INTERVAL MINUTE TO SECOND	’MINUTES:SECONDS.MICROSECONDS’

format_str

格式符	描述	备注
%a	缩写的工作日名称	Sun...Sat
%b	缩写的月份	Jan...Dec
%c	月份	数值：1..12
%D	带英文后缀的月份天数	1st,2nd,3rd,....

接下页

格式符	描述	备注
%d	月中的某天	01..31
%e	月中的某天	1..31
%f	微秒	000000...999999
%H	小时	00..23
%h	小时	01..12
%I	小时	01..12
%i	分钟	数值：00..59
%j	年中的某天	001..366
%k	小时	0..23
%l	小时	1..12
%M	月份名称	January..December
%m	月份	00..12
%p	上午/下午	AM/PM
%r	时间	12 小时制带 AM/PM，hh:mm:ss AM/PM
%S	秒	00..59
%s	秒	00..59
%T	时间	24 小时制
%U	周	00..53，周日为一周中的第一天
%u	周	00..53，周一为一周中的第一天

接下页

格式符	描述	备注
%V	周	01..53，周日为一周中的第一天，与%X 一起使用
%v	周	01..53，周一为一周中的第一天，与%X 一起使用
%W	工作日名称	Sunday..Saturday
%w	一周中的天	0=Sunday..6=Saturday
%X	年	每周的第一天为周日，4 位，与%V 一起使用
%x	年	每周的第一天为周一，4 位，与%v 一起使用
%Y	年	四位年
%y	年	两位年
%%	符号%	-

3.2 ADDDATE

功能描述

将指定的时间间隔（如天数、小时、分钟等）添加到给定的日期或时间值上。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷数据库返回具体结果。
- ADDDATE 函数运算 INTERVAL 'YEARS-MONTHS' YEAR TO MONTH 时，当 MONTHS 大于等于 12 时，虚谷数据库报错，MySQL 则在年份上 +1。
- INTERVAL 类型的差异，MySQL 中传入的 INTERVAL 类型进行运算时，最大单位不轮转，而是递增，不受时间格式的最大限制。
- ADDDATE 第二个参数为 INT 型时，传入数据含小数时，MySQL 四舍五入，虚谷数据库舍

弃小数位。如：SELECT ADDDATE('2018-12-31',100.5); 虚谷数据库输出 2019-04-10；MySQL 输出 2019-04-11。

语法格式

```
ADDDATE(expr1,INTERVAL expr2)
```

参数说明

- expr1：DATE/TIME/DATETIME 类型时间，原时间。
- expr2：增加的时间。

函数返回类型

DATE/TIME/DATETIME 类型。

示例

```
SQL> SELECT ADDDATE('2020-12-31 23:59:59',INTERVAL '1' SECOND);

EXPR1 |
------------------------------------------------------------
2021-01-01 00:00:00.000 AD |

SQL> SELECT ADDDATE('2018-12-31 23:59:59',INTERVAL '1' DAY);

EXPR1 |
------------------------------------------------------------
2019-01-01 23:59:59.000 AD |

SQL> SELECT ADDDATE('2100-12-31 23:59:59',INTERVAL '1:1' MINUTE to
   SECOND);

EXPR1 |
------------------------------------------------------------
2101-01-01 00:01:00.000 AD |
```

3.3 ADD_MONTHS

功能描述

返回在日期 expr1 基础上添加或减去 expr2 个月份后的日期。

如果 expr1 是该月的最后一天，或者如果结果月份的天数少于 expr1 的“日”部分（例如从 31 日移到 28 日或 29 日），则返回值是结果月份的最后一天。否则，返回值与 expr1 具有相同的“日”。

语法格式

```
ADD_MONTHS(expr1, expr2)
```

参数说明

- expr1：DATE、DATETIME 或带时区的 DATETIME 类型的列或表达式。
- expr2：NUMERIC 类型，用于指定给 expr1 增加的月份。如果数值为负，则是 expr1 减去相应月份。

函数返回类型

日期、日期时间或带时区的日期时间类型：DATE、DATETIME、DATETIME WITH TIME ZONE。

示例

```
SQL> SELECT ADD_MONTHS('2022-01-30 02:00:00',1) FROM dual;

EXPR1 |
------------------------------------------------------------
2022-02-28 02:00:00.000 AD |
2022-02-28 02:00:00.000 AD |
```

```
SELECT ADD_MONTHS('2024-03-30 02:00:00',1.5) FROM dual;

EXPR1 |
------------------------------------------------------------
2024-04-30 02:00:00.000 AD |
```

```
-- 1.5::NUMERIC 存在进位
SELECT ADD_MONTHS('2024-03-30 02:00:00',1.5::NUMERIC) FROM dual;

EXPR1 |
------------------------------------------------------------
2024-05-30 02:00:00.000 AD |
```

3.4 ADDTIME

功能描述

将指定的时间间隔（如天数、小时、分钟等）添加到给定的日期或时间值上。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷数据库返回具体结果。
- ADDTIME 函数运算 INTERVAL 'YEARS-MONTHS' YEAR TO MONTH 时，当 MONTHS 大于等于 12 时，虚谷数据库报错，MySQL 则在年份上 +1。
- INTERVAL 类型的差异，MySQL 中传入的 INTERVAL 类型进行运算时，最大单位不轮转，而是递增，不受时间格式的最大限制。

语法格式

```
ADDTIME(expr1,INTERVAL expr2)
```

参数说明

- expr1：DATE/TIME/DATETIME 类型时间，原时间。
- expr2：增加的时间。

函数返回类型

DATE/TIME/DATETIME 类型。

示例

```
SQL> SELECT ADDTIME('2020-12-31 23:59:59',INTERVAL '1' SECOND);

EXPR1 |
------------------------------------------------------------
2021-01-01 00:00:00.000 AD |

SQL> SELECT ADDTIME('2018-12-31 23:59:59',INTERVAL '1' DAY);

EXPR1 |
------------------------------------------------------------
2019-01-01 23:59:59.000 AD |

SQL> SELECT ADDTIME('2100-12-31 23:59:59',INTERVAL '1:1' MINUTE to
   SECOND);

EXPR1 |
------------------------------------------------------------
2101-01-01 00:01:00.000 AD |
```

3.5 CLOCK_TIMESTAMP

功能描述

提供系统时钟的实际时间。

每次调用 CLOCK_TIMESTAMP 都会返回当前系统时钟的时间，而不是事务开始时的时间。在同一事务中多次调用 CLOCK_TIMESTAMP 可能会返回不同的时间戳。

> **说明**
>
> 返回值的时区不会根据客户端的时区设置进行偏移，其值精确到毫秒。

语法格式

```
CLOCK_TIMESTAMP()
```

参数说明

无参数。

函数返回类型

DATETIME WITH TIME ZONE 类型。

示例

```
SQL> SELECT CLOCK_TIMESTAMP();

EXPR1 |
------------------------------------------------------------
2024-11-06 17:17:57.295 AD +08:00 |
```

3.6 CURDATE

功能描述

获取当前日期。

与 MySQL 无差异。

语法格式

```
CURDATE()
```

参数说明

无参数。

函数返回类型

DATE 类型。

示例

```
SQL> SELECT CURDATE();

EXPR1 |
------------------------------------------------------------
2022-11-15 AD |
```

3.7 CURRENT_DATETIME

功能描述

返回当前时间。

语法格式

```
CURRENT_DATETIME
```

参数说明

无参数。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT CURRENT_DATETIME FROM dual;

EXPR1 |
------------------------------------------------------------
2022-06-24 10:29:22.037  |
```

3.8 CURTIME

功能描述

返回当前时间。

与 MySQL 无差异。

语法格式

```
CURTIME()
```

参数说明

无参数。

函数返回类型

TIME 类型。

示例

```
SQL> SELECT CURTIME();

EXPR1 |
------------------------------------------------------------
14:58:25.838 |
```

3.9 DATE

功能描述

获取指定日期时间中的日期。

与 MySQL 无差异。

语法格式

```
DATE(expr)
```

参数说明

expr：时间，DATE/DATETIME/DATETIME WITH TIME ZONE 类型。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00, 9999-12-31 23:59:59]

函数返回类型

DATE 类型。

> **说明**
>
> 返回值范围：[0001-01-01, 9999-12-31]

示例

```
SQL> SELECT DATE('1998-02-02');

EXPR1 |
------------------------------------------------------------
1998-02-02 AD |

SQL> SELECT DATE('1998-02-02 10:10:30');

EXPR1 |
------------------------------------------------------------
1998-02-02 AD |

SQL> SELECT DATE('1998-02-02 10:10:30 +8:00');

EXPR1 |
------------------------------------------------------------
1998-02-02 AD |
```

3.10 DATE_ADD

功能描述

将指定的时间间隔（如天数、小时、分钟等）添加到给定的日期或时间值上。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷数据库返回具体结果。

- ADDDATE 函数运算 INTERVAL 'YEARS-MONTHS' YEAR TO MONTH 时，当 MONTHS 大于等于 12 时，虚谷数据库报错，MySQL 则在年份上 +1。
- INTERVAL 类型的差异，MySQL 中传入的 INTERVAL 类型进行运算时，最大单位不轮转，而是递增，不受时间格式的最大限制。

语法格式

```
DATE_ADD(expr1,INTERVAL expr2)
```

参数说明

- expr1：DATE/TIME/DATETIME 类型时间，原时间。
- expr2：增加的时间。

函数返回类型

DATE/TIME/DATETIME 类型。

示例

```
SQL> SELECT DATE_ADD('2020-12-31 23:59:59',INTERVAL '1' SECOND);

EXPR1 |
------------------------------------------------------------
2021-01-01 00:00:00.000 AD |

SQL> SELECT DATE_ADD('2018-05-01',INTERVAL '1' YEAR);

EXPR1 |
------------------------------------------------------------
2019-05-01 00:00:00.000 AD |

SQL> SELECT DATE_ADD('2018-05-01',INTERVAL '30' DAY);

EXPR1 |
------------------------------------------------------------
2018-05-31 00:00:00.000 AD |
```

3.11 DATE_FORMAT

功能描述

按指定格式串格式化时间。

与 MySQL 差异：

- 对于时间格式的字符串：MySQL 执行不报错，会按照规则做转换；虚谷数据库执行报错。

- 对于日期时间格式的字符串有错误的（如 2006-06-00）：MySQL 可正常执行；虚谷数据库执行报错。
- 对于参数类型隐式转换（如 BIGINT 类型隐式转为 VARCHAR）：MySQL 执行正常（合法值转换返回正常结果，非法值转换返回 NULL）；虚谷数据库不支持。
- 对于常值 NULL 为参数：MySQL 返回 NULL；虚谷数据库需要把 NULL 参数转换为 CHAR，否则报错。

语法格式

```
DATE_FORMAT(expr1,expr2)
```

参数说明

- expr1：时间，DATE/DATETIME 类型。
- expr2：格式串，如%y-%c-%D %h:%i:%s。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT DATE_FORMAT('1912-01-03 14:02:03.123456','%y-%c-%D %h:%
   i:%s');

EXPR1 |
------------------------------------------------------------
12-1-3rd 02:02:03|
```

3.12 DATE_PART

功能描述

从日期/时间值中提取特定的部分（如年、月、日、小时、分钟、秒等）。

语法格式

```
DATE_PART(FIELD, SOURCE)
```

参数说明

- FIELD：要提取的时间部分，字符串值。并非每种输入的数据类型都适用于所有字段。例如，无法从 DATE 类型中提取小于一天的字段，也无法从 TIME 类型中提取一天或更长时间的字段。

- SOURCE：要从中提取时间部分的日期/时间值，类型为 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIME、TIME WITH TIME ZONE 或 INTERVAL 的值表达式。

FIELD 参数取值

FIELD	取值说明
millennium	千年。19xx 的年份在第二个千年里，第三个千年从 2001 年 1 月 1 日开始。对于 INTERVAL 类型值，表示年份除以 1000。
century	世纪。对于 INTERVAL 类型值，表示年份除以 100。
decade	十年。即年份除以 10。
year	年份。
quarter	季度。范围为 [1,4]。
month	• 对于 TIMESTAMP 类型值，表示一年里的月份数，范围为 [1,12]。 • 对于 INTERVAL 类型值，表示月数，然后对 12 取模（除以 12 取余数），范围为 [0,11]。
week	该天所在的 ISO 8601 周编号的年份里是第几周。
day	• 对于 TIMESTAMP 类型值，表示日期中的天，范围为 [1,31]。 • 对于 INTERVAL 类型值，表示天数。
hour	小时。范围为 [0,23]，时间间隔不受限制。

接下页

FIELD	取值说明
minute	分钟。范围为 [0,59]。
second	秒。范围为 [0,59]。
milliseconds	毫秒。时间值的秒部分（包括小数部分）乘以 1000。
microseconds	微秒。时间值的秒部分（包括小数部分）乘以 1000000。
isoyear	日期所落在的 ISO 8601 周编号的年，不适用于时间间隔。 根据 ISO 8601 标准，一年的第 1 周是包含该年第一个星期四的那一周。某些年份的第一周可能从上一年的最后一两天开始。某些年份的最后一周可能延续到下一年的前几天。 例如 2024-12-30 的结果是 2025，2022-1-1 的结果是 2021。
dow	星期几。0 表示星期日，1 表示星期一，依此类推到 6 表示星期六。
isodow	星期几。1 表示星期一，2 表示星期二，依此类推到 7 表示星期日。
julian	与日期或时间戳对应的儒略日（JDN，Julian Day Number）。
doy	一年中的第几天。范围为 [1,366]。
	接下页

FIELD	取值说明
epoch	• 对于 TIMESTAMP WITH TIME ZONE 类型值，表示自 1970-01-01 00：00：00 UTC 以来的秒数（之前的时间戳为负值）。 • 对于 DATE 类型值和 TIMESTAMP 类型值，是自 1970-01-01 00：00：00 以来的标称秒数（nominal number of seconds）。 • 对于 INTERVAL 类型值，它是时间间隔的总秒数。
timezone	与 UTC 的时区偏移，以秒为单位。正值对应于 UTC 东边的时区，负值对应 UTC 西边的时区。
timezone_hour	时区偏移的小时部分。
timezone_minute	时区偏移的分钟部分。

函数返回类型

DOUBLE PRECISION 类型。

示例

```
SQL> SELECT DATE_PART('year', '2020-10-20 12:30:25.5020555');

EXPR1 |
------------------------------------------------------------
2020 |

SQL> SELECT DATE_PART('mon', '2020-10-20 12:30:25.5020555');

EXPR1 |
------------------------------------------------------------
10 |

SQL> SELECT DATE_PART('hrs', '2020-10-20 12:30:25.5020555');

EXPR1 |
------------------------------------------------------------
12 |
```

3.13 DATE_SUB

功能描述

从日期或时间中减去指定的时间间隔。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷数据库返回具体结果。
- DATE_SUB 函数运算 INTERVAL 'YEARS-MONTHS' YEAR TO
- MONTH 时，当 MONTHS 大于等于 12 时，虚谷数据库报错，MySQL 则在年份上 +1。
- INTERVAL 类型的差异，MySQL 中传入的 INTERVAL 类型进行运算时，最大单位不轮转，而是递增，不受时间格式的最大限制。

语法格式

```
DATE_SUB(expr1,INTERVAL expr2)
```

参数说明

- expr1：DATE/TIME/DATETIME 类型时间，原时间。
- expr2：减少的时间。

函数返回类型

DATE/TIME/DATETIME 类型。

示例

```
SQL> SELECT DATE_SUB('2018-05-01',INTERVAL '1' YEAR);

EXPR1 |
------------------------------------------------------------
2017-05-01 00:00:00.000 AD |

SQL> SELECT DATE_SUB('2020-12-31 23:59:59',INTERVAL '1' SECOND);

EXPR1 |
------------------------------------------------------------
2020-12-31 23:59:58.000 AD |

SQL> SELECT DATE_SUB('2018-12-31 23:59:59',INTERVAL '1' DAY);

EXPR1 |
------------------------------------------------------------
2018-12-30 23:59:59.000 AD |
```

3.14 DATE_TRUNC

功能描述

将时间戳或时间间隔截断到指定的时间单位。

语法格式

```
DATE_TRUNC(FIELD, SOURCE[, TIMEZONE])
```

参数说明

- FIELD：字符串值。表示要截断到的时间单位。
- SOURCE：类型为 TIMESTAMP、TIMESTAMP WITH TIME ZONE 或 INTERVAL 的值表达式。
- TIMEZONE：字符串值。表示将时间戳截断至指定的时区中。如果不指定，表示将时间戳截断至会话时区中的指定精度。

FIELD 参数取值

FIELD	取值说明
millennium	截断到千年的第一天。
century	截断到百年的第一天。
decade	截断到十年的第一天。
year	截断到年的第一天。
quarter	截断到季度的第一天。
month	截断到月份的第一天。
week	截断到星期一。INTERVAL 类型值不支持指定 week 进行截断。
day	截断到天（即午夜 00:00:00）。
hour	截断到小时。

接下页

FIELD	取值说明
minute	截断到分钟。
second	截断到秒。
milliseconds	截断到毫秒。
microseconds	截断到微秒。

函数返回类型

返回值类型和 SOURCE 的类型保持一致。

示例

```
SQL> SELECT DATE_TRUNC('year', '2020-10-20 12:30:25.5020555');

EXPR1 |
------------------------------------------------------------------
2020-01-01 00:00:00.000 AD |

-- 截 断 时 间 间 隔 类 型
SQL> SELECT DATE_TRUNC('milliseconds', '20 12:30:25.5020555'::
   interval day to second);

EXPR1 |
------------------------------------------------------------------
20 12:30:25.502000 |

-- 截 断 到 小 时
SQL> SELECT DATE_TRUNC('hour', '2020-10-20 12:30:25.5020555','GMT
   +06:00');

EXPR1 |
------------------------------------------------------------------
2020-10-20 12:00:00.000 AD +08:00 |

-- 截 断 到 毫 秒
SQL> SELECT DATE_TRUNC('milliseconds', '2020-10-20 12:30:25.5020555
   ','GMT-04:00');

EXPR1 |
------------------------------------------------------------------
2020-10-20 12:30:25.502 AD +08:00 |
```

3.15 DATEDIFF

功能描述

计算两个时间差。

与 MySQL 无差异。

语法格式

```
DATEDIFF(expr1,expr2)
```

参数说明

expr1、expr2：DATE/TIME/DATETIME 类型的时间。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');

EXPR1 |
------------------------------------------------------------
-31 |

SQL> SELECT DATEDIFF('2010-12-31','2010-11-30 23:59:59');

EXPR1 |
------------------------------------------------------------
31 |
```

3.16 DAY

功能描述

从日期或时间值中提取“日”部分。它返回指定日期的天数（1 到 31），表示该日期是所在月份的第几天。

与 MySQL 无差异。

语法格式

```
DAY(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围为 [0001-01-01, 9999-12-31]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[1, 31]

示例

```
SQL> SELECT DAY('1987-01-01');

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT DAY('1987-01-01 12:12:12');

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT DAY('1987-01-01 12:12:12 +08:00');

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT DAY('9999-12-31');

EXPR1 |
------------------------------------------------------------
31 |
```

3.17 DAYNAME

功能描述

从日期或时间值中提取对应的星期几的名称。

与 MySQL 无差异。

语法格式

```
DAYNAME(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01, 9999-12-31]

函数返回类型

CHAR 类型。

> **说明**
>
> 返回值范围：[Monday, Tuesday, ..., Sunday]

示例

```
SQL> SELECT DAYNAME('1987-01-01');

EXPR1 |
------------------------------------------------------------
Thursday|

SQL> SELECT DAYNAME('2022-01-04 12:34:56');

EXPR1 |
------------------------------------------------------------
Tuesday|

SQL> SELECT DAYNAME('2022-01-04 12:34:56 +08:00');

EXPR1 |
------------------------------------------------------------
Tuesday|
```

3.18 DAYOFMONTH

功能描述

从日期或时间值中提取该日期是所在月份的第几天。

语法格式

```
DAYOFMONTH(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围为 [0001-01-01, 9999-12-31]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[1, 31]

示例

```
SQL> SELECT DAYOFMONTH('1987-01-01');

EXPR1 |
------------------------------------------------------------------
1 |

SQL> SELECT DAYOFMONTH('1987-01-22');

EXPR1 |
------------------------------------------------------------------
22 |

SQL> SELECT DAYOFMONTH('1987-01-01 12:12:12');

EXPR1 |
------------------------------------------------------------------
1 |

SQL> SELECT DAYOFMONTH('1987-01-01 12:12:12 +08:00');

EXPR1 |
------------------------------------------------------------------
1 |
```

3.19 DAYOFWEEK

功能描述

从日期或时间值中提取该日期是星期几的数字表示。返回一个整数，表示该日期是星期几，其中：

- 1 表示星期日
- 2 表示星期一

- 3 表示星期二
- 4 表示星期三
- 5 表示星期四
- 6 表示星期五
- 7 表示星期六

语法格式

```
DAYOFWEEK(expr)
```

参数说明

expr：DATETIME 类型的字段或变量或表达式。

函数返回类型

INTEGER 类型字符串。

示例

```
SQL> SELECT DAYOFWEEK('2022-01-01') FROM dual;

EXPR1 |
------------------------------------------------------------
7 |

SQL> SELECT DAYOFWEEK('2022-01-01 12:34:56 BC') FROM dual;

EXPR1 |
------------------------------------------------------------
2 |

SQL> SELECT DAYOFWEEK('2022-01-01'::DATE + 10) FROM dual;

EXPR1 |
------------------------------------------------------------
3 |

SELECT DAYOFWEEK('') FROM dual;

EXPR1 |
------------------------------------------------------------
<NULL>|
```

3.20 DAYOFYEAR

功能描述

从日期或时间值中提取该日期是所在年份的第几天。返回一个整数，表示该日期在当年中的具体天数（1 到 366）。

其中 1 表示一年的第一天（1 月 1 日），365 或 366 表示一年的最后一天（12 月 31 日），具体取决于该年是否为闰年。

语法格式

```
DAYOFYEAR(expr)
```

参数说明

expr：DATETIME 类型的字段或变量或表达式。

函数返回类型

INTEGER 类型字符串。

示例

```
SQL> SELECT DAYOFYEAR('2022-01-01') FROM dual;

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT DAYOFYEAR('2022-12-31') FROM dual;

EXPR1 |
------------------------------------------------------------
365 |

SQL> SELECT DAYOFYEAR('2022-01-01'::DATE + 10) FROM dual;

EXPR1 |
------------------------------------------------------------
11 |

SQL> SELECT DAYOFYEAR('2022-12-31 12:34:56 BC') FROM dual;

EXPR1 |
------------------------------------------------------------
365 |

SQL> SELECT DAYOFYEAR('2022-12-31 12:34:56 -03:40 BC') FROM dual;

EXPR1 |
------------------------------------------------------------
365 |

SQL> SELECT DAYOFYEAR('') FROM dual;

EXPR1 |
------------------------------------------------------------
<NULL>|
```

3.21 EXTRACT

功能描述

从日期或时间值中提取特定的时间部分。可以提取年、月、日、小时、分钟、秒等不同的时间单位，并返回相应的值。

语法格式

```
EXTRACT(expr1 FROM expr2)
```

参数说明

- expr1：标识符，为 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND 中的一种。
- expr2：TIMESTAMP 或 DATETIME、TIME 类型的字段或表达式。

函数返回类型

INTEGER 类型，与 DATETIME 相对应的时间值。

示例

```
SQL> SELECT EXTRACT(YEAR FROM '2022-05-20 10:10:10'),EXTRACT(MONTH
   FROM '2022-05-20 10:10:10'),
EXTRACT(DAY FROM '2022-05-20 10:10:10'),EXTRACT(HOUR FROM '
   2022-05-20 10:10:10'),
EXTRACT(MINUTE FROM '2022-05-20 10:10:10'),EXTRACT(SECOND FROM '
   2022-05-20 10:10:10') FROM dual;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 |
-------------------------------------------------------------
2022 | 5 | 20 | 10 | 10 | 10 |
```

3.22 EXTRACT_DAY

功能描述

提取日期中的“日”部分。

语法格式

```
EXTRACT_DAY(expr)
```

参数说明

expr：时间类型的字段或表达式。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT EXTRACT_DAY('2022-05-20') FROM dual;

EXPR1 |
------------------------------------------------------------
20 |
```

3.23 EXTRACT_HOUR

功能描述

提取时间中的“小时”部分。

语法格式

```
EXTRACT_HOUR(expr)
```

参数说明

expr：时间类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT EXTRACT_HOUR('2022-05-20 20:10:5') FROM dual;

EXPR1 |
------------------------------------------------------------
20 |
```

3.24 EXTRACT_MINUTE

功能描述

提取时间中的“分钟”部分。

语法格式

```
EXTRACT_MINUTE(expr)
```

参数说明

expr：时间类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT EXTRACT_MINUTE('2022-05-20 20:10:5') FROM dual;

EXPR1 |
------------------------------------------------------------
10 |
```

3.25 EXTRACT_MONTH

功能描述

提取日期中的“月份”部分。

语法格式

```
EXTRACT_MONTH(expr)
```

参数说明

expr：时间类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT EXTRACT_MONTH('2022-5-20 20:10:5') FROM dual;

EXPR1 |
------------------------------------------------------------
5 |
```

3.26 EXTRACT_SECOND

功能描述

提取时间中的“秒”部分。

语法格式

```
EXTRACT_SECOND(expr)
```

参数说明

expr：时间类型的字段或表达式。

函数返回类型

DOUBLE 数值类型。

示例

```
SQL> SELECT EXTRACT_SECOND('2022-5-20 20:10:5') FROM dual;

EXPR1 |
------------------------------------------------------------
5 |
```

3.27 EXTRACT_YEAR

功能描述

提取日期中的“年份”部分。

语法格式

```
EXTRACT_YEAR(expr)
```

参数说明

expr：时间类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT EXTRACT_YEAR('2022-5-20 20:10:5') FROM dual;

EXPR1 |
------------------------------------------------------------
2022 |
```

3.28 FROM_DAYS

功能描述

将天数转换为日期（TO_DAYS 函数的反向函数）。

与 MySQL 无差异。

语法格式

```
FROM_DAYS(expr)
```

参数说明

expr：INT 类型的数值。

> **说明**
>
> 参数取值范围：[366, 3652424]

函数返回类型

DATE 类型。

> **说明**
>
> 返回值范围：[0001-01-01, 9999-12-31]

示例

```
SQL> SELECT FROM_DAYS(4561);

EXPR1 |
------------------------------------------------------------
0012-06-27 AD |
```

3.29 FROM_UNIXTIME

功能描述

将 Unix 时间戳（即从 1970 年 1 月 1 日 00:00:00 UTC 开始的秒数）转换为标准的日期和时间格式。

与 MySQL 差异：

- 返回值与 MySQL 存在差值，其值取决与 MySQL 的时区配置。
- 不支持在 FROM_UNIXTIME 中进行格式转换，如 FROM_UNIXTIME(unix_timestamp, format)。
- 在 MySQL 中，使用该函数和常量进行加减运算时，常量以毫秒为单位；在虚谷数据库中，相同的运算所使用的常量以天为单位。

语法格式

```
FROM_UNIXTIME(expr)
```

参数说明

expr：时间戳，NUMERIC 类型数值（单位：秒）。

> **说明**
>
> 参数取值范围为：[-377673580800, 253402300799.999999]

函数返回类型

DATETIME 类型。

> **说明**
>
> 返回值范围为：[9999-13-31 23:59:59.999999 BC, 9999-12-31 23:59:59.999999 AD]

示例

```
SQL> SELECT FROM_UNIXTIME(1349105565);

EXPR1 |
------------------------------------------------------------
2012-10-01 15:32:45.000 AD |

-- 与常量加运算，运算结果的天数增加
SQL> SELECT FROM_UNIXTIME(1349105565) + 1;

EXPR1 |
------------------------------------------------------------
2012-10-02 15:32:45.000 AD |

SQL> SELECT FROM_UNIXTIME(1349105565.211);

EXPR1 |
------------------------------------------------------------
2012-10-01 15:32:45.211 AD |

SQL> SELECT FROM_UNIXTIME(-291889634.788);

EXPR1 |
------------------------------------------------------------
1960-10-01 15:32:45.212 AD |
```

3.30 GET_BOOT_TIME

功能描述

获取当前节点的启动时间。

语法格式

```
GET_BOOT_TIME()
```

参数说明

无

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT get_boot_time();

EXPR1 |
------------------------------------------------------------
2024-05-10 14:13:50.397 AD |
```

3.31 GETDAY

功能描述

提取日期中的“日”部分。

语法格式

```
GETDAY(expr)
```

参数说明

expr：TIMESTAMP、DATETIME、DATE 类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT GETDAY('2022-05-20 02:00:00') FROM dual;

EXPR1 |
------------------------------------------------------------
20 |
```

3.32 GETHOUR

功能描述

提取时间中的“小时”部分。

语法格式

```
GETHOUR(expr)
```

参数说明

expr：DATETIME、TIME 类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT GETHOUR('2022-05-20 02:00:00') FROM dual;

EXPR1 |
------------------------------------------------------------
2 |
```

3.33 GETMINUTE

功能描述

提取时间中的“分钟”部分。

语法格式

```
GETMINUTE(expr)
```

参数说明

expr：DATETIME、TIME 类型的字段或表达式。

函数返回类型

日期时间类型 DATETIME。

示例

```
SQL> SELECT GETMINUTE('2022-05-20 02:28:00') FROM dual;

EXPR1 |
------------------------------------------------------------
28 |
```

3.34 GETMONTH

功能描述

提取日期中的“月份”部分。

语法格式

```
GETMONTH(expr)
```

参数说明

expr：TIMESTAMP、DATETIME、DATE 类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT GETMONTH('2022-05-20 02:00:00') FROM dual;

EXPR1 |
------------------------------------------------------------
5 |
```

3.35 GETSECOND

功能描述

提取时间中的“秒”部分。

语法格式

```
GETSECOND(expr)
```

参数说明

expr：DATETIME、TIME 类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT GETSECOND('2022-05-20 02:28:55') FROM dual;

EXPR1 |
------------------------------------------------------------
55 |
```

3.36 GET_UPTIME

功能描述

获取当前节点上数据库服务已经运行的秒数。

语法格式

```
GET_UPTIME()
```

参数说明

无

函数返回类型

BIGINT 类型。

示例

```
SQL> SELECT GET_UPTIME();

EXPR1 |
------------------------------------------------------------
1671 |
```

3.37 GETYEAR

功能描述

提取日期中的“年份”部分。

语法格式

```
GETYEAR(expr)
```

参数说明

expr：TIMESTAMP、DATETIME、DATE 类型的字段或表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT GETYEAR('2022-05-20 02:00:00') FROM dual;

EXPR1 |
------------------------------------------------------------
2022 |
```

3.38 GET_FORMAT

功能描述

返回特定类型的日期或时间格式字符串。

与 MySQL 无差异。

语法格式

```
GET_FORMAT(expr1, expr2)
```

参数说明

- expr1：日期、时间类型，可选 DATE|TIME|DATETIME。
- expr2：日期、时间格式化模版，可选’EUR’|’USA’|’JIS’|’ISO’|’INTERNAL’。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT GET_FORMAT(DATE, 'USA');

EXPR1 |
------------------------------------------------------------
%m.%d.%Y|
```

3.39 HOUR

功能描述

提取时间中的“小时”部分。

与 MySQL 差异：

- 参数和取值范围小于 MySQL，因为后者允许小时数大于 23，如 272:59:59 在 MySQL 中合法，而在虚谷数据库中不合法。
- 对于参数为日期时间/时间的字符串，MySQL 不需对参数做转换，虚谷数据库需要先把参数做显式转换。例如：

  MySQL：SELECT HOUR(’03:04:05’);

  虚谷数据库：SELECT HOUR(’03:04:05’::TIME); 或 SELECT HOUR(cast(’03:04:05’ AS TIME));

  注：对于 TIME WITH TIME ZONE 类型的字符串，只支持 CAST() 方式转换，不支持:: 类型名方式转换。
- 所有涉及到支持 DATETIME 和 TIME 的函数，以字符串形式传入参数时均报错：时间常数值错误。

  原因：字符串转时间时，DATETIME 的评分高于 TIME，无法进行隐式转换，需进行强制类型转换。

语法格式

```
HOUR(expr)
```

参数说明

expr：DATETIME/DATETIME WITH TIME ZONE/TIME/TIME WITH TIME ZONE 类型的时间。

> 说明
>
> 参数取值范围：[0001-01-01 00:00:00, 9999-12-31 23:59:59]/[00:00:00, 23:59:59]

函数返回类型

INTEGER 类型。

> 说明
>
> 返回值范围：[0, 23]

示例

```
SQL> SELECT HOUR('23:59:59.999'::TIME);

EXPR1 |
------------------------------------------------------------
23 |

SQL> SELECT HOUR(cast('12:34:56-11:00' AS TIME WITH TIME ZON));

EXPR1 |
------------------------------------------------------------
12 |

SQL> SELECT HOUR(cast('9999-12-31 23:59:59.999' AS DATETIME WITH
   TIME ZONE));

EXPR1 |
------------------------------------------------------------
23 |
```

3.40 LAST_DAY

功能描述

返回给定日期所在月份的最后一天。

语法格式

```
LAST_DAY(expr)
```

参数说明

expr：DATE、DATETIME、DATETIME WITH TIME ZONE 类型的字段或表达式。

函数返回类型

DATE、DATETIME、DATETIME WITH TIME ZONE 类型字符串。

示例

```
SQL> SELECT TO_CHAR(LAST_DAY('2020-02-02')) FROM dual;

EXPR1 |
------------------------------------------------------------
2020-02-29 00:00:00|

SQL> SELECT TO_CHAR(LAST_DAY('2022-02-02 BC')) FROM dual;

EXPR1 |
------------------------------------------------------------
2022-02-28 00:00:00 BC|

SQL> SELECT TO_CHAR(LAST_DAY('2022-02-02 12:34:56 -07:50 BC')) FROM
    dual;

EXPR1 |
------------------------------------------------------------
2022-02-28 12:34:56 BC|

SQL> SELECT LAST_DAY('') FROM dual;

EXPR1 |
------------------------------------------------------------
<NULL>|
```

3.41 LOCALTIME

功能描述

返回当前系统的时间，同 SYSDATE。

与 MySQL 差异：

LOCALTIME 做加减运算同 SYSDATE，虚谷数据库加减以天为单位的数值，MySQL 加减以秒为单位的数值。

语法格式

```
LOCALTIME()
```

参数说明

无参数。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT LOCALTIME();

EXPR1 |
------------------------------------------------------------
2022-11-15 15:00:51.594 AD |

SQL> SELECT SYSDATE();

EXPR1 |
------------------------------------------------------------
2022-11-15 15:01:01.674 AD |
```

3.42 LOCALTIMESTAMP

功能描述

返回当前系统的时间，同 SYSDATE。

与 MySQL 差异：

LOCALTIMESTAMP 做加减运算同 SYSDATE，虚谷数据库加减以天为单位的数值，MySQL 加减以秒为单位的数值。

语法格式

```
LOCALTIMESTAMP()
```

参数说明

无参数。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT LOCALTIMESTAMP();

EXPR1 |
------------------------------------------------------------
2022-11-15 15:03:43.296 AD |

SQL> SELECT SYSDATE();

EXPR1 |
------------------------------------------------------------
2022-11-15 15:03:51.544 AD |
```

3.43 MAKE_DATE

功能描述

根据指定的年份、月份和日期创建一个 DATE 类型的值。

语法格式

```
MAKE_DATE(year,month,day)
```

参数说明

- year：INT 类型，小于 0 表示公元前。
- month：INT 类型。
- day：INT 类型。

函数返回类型

DATE 类型。

示例

```
SQL> SELECT MAKE_DATE(2023,9,6);

EXPR1 |
------------------------------------------------------------
2023-09-06 AD |
```

3.44 MAKE_TIME

功能描述

根据指定的小时、分钟和秒创建一个 TIME 类型的值。

语法格式

```
MAKE_TIME(hour,min,sec)
```

参数说明

- hour：INT 类型。
- min：INT 类型。
- sec：DOUBLE 类型。

函数返回类型

TIME 类型。

示例

```
SQL> SELECT MAKE_TIME(15,32,45.211);

EXPR1 |
------------------------------------------------------------
15:32:45.211 |
```

3.45 MAKE_TIMESTAMP

功能描述

根据指定的年份、月份、日期、小时、分钟和秒创建一个 TIMESTAMP 类型的值。

语法格式

```
MAKE_TIMESTAMP(YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS)
```

参数说明

- YEARS：类型为 INTEGER，表示输入的年份，小于 0 表示公元前。
- MONTHS：类型为 INTEGER，表示输入的月份。
- DAYS：类型为 INTEGER，表示输入的日期。
- HOURS：类型为 INTEGER，表示输入的小时。
- MINUTES：类型为 INTEGER，表示输入的分钟。
- SECONDS：类型为 DOUBLE，表示输入的秒数。

函数返回类型

TIMESTAMP 类型。

示例

```
SQL> SELECT MAKE_TIMESTAMP(2023,9,6,15,32,45.211);

EXPR1 |
------------------------------------------------------------
2023-09-06 15:32:45.211 AD |
```

3.46 MAKE_TIMESTAMPTZ

功能描述

通过指定年、月、日、时、分、秒和时区构建一个 TIMESTAMP WITH TIME ZONE 类型的值。

语法格式

```
MAKE_TIMESTAMPTZ(YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS[,
   TIMEZONE])
```

参数说明

- YEARS：类型为 INTEGER，表示输入的年份。
- MONTHS：类型为 INTEGER，表示输入的月份。
- DAYS：类型为 INTEGER，表示输入的日期。
- HOURS：类型为 INTEGER，表示输入的小时。
- MINUTES：类型为 INTEGER，表示输入的分钟。
- SECONDS：类型为 DOUBLE，表示输入的秒数。
- TIMEZONE：可选值，类型为 CHAR，表示输入的时区。

函数返回类型

DATETIME WITH TIME ZONE 类型。

示例

```
SQL> SELECT MAKE_TIMESTAMPTZ(2024,10,10,15,30,45.123456);

EXPR1 |
------------------------------------------------------------
2024-10-10 15:30:45.123 AD +08:00 |

SQL> SELECT MAKE_TIMESTAMPTZ(2024,10,10,15,30,45.123456,'GMT+09:00'
   );

EXPR1 |
------------------------------------------------------------
2024-10-11 08:30:45.123 AD +08:00 |

SQL> SELECT MAKE_TIMESTAMPTZ(2024,10,10,15,30,45.123456,'GMT-09:00'
   );

EXPR1 |
------------------------------------------------------------
2024-10-10 14:30:45.123 AD +08:00 |
```

3.47 MICROSECOND

功能描述

从给定的时间或日期时间值中提取微秒部分。

与 MySQL 差异：

- 参数和取值范围小于 MySQL，因为 MySQL 允许小时数大于 23，如 272:59:59 在 MySQL 中是合法的，而在虚谷数据库中是不合法的。
- 对于参数中秒的小数部分超过精度时，MySQL 进行四舍五入，虚谷数据库直接截断。
- 对于参数为日期时间/时间的字符串，MySQL 不需对参数做转换，虚谷数据库需要先把参数做显示转换。例如：

  MySQL：SELECT MICROSECOND('03:04:05');

  虚谷数据库：SELECT MICROSECOND('03:04:05'::TIME); 或 SELECT MICROSECOND(CAST('03:04:05' AS TIME));

  注：对 TIME WITH TIME ZONE 类型的字符串，只支持 CAST() 方式转换，不支持:: 类型名方式转换。
- TIME 只存储到毫秒。

语法格式

```
MICROSECOND(expr)
```

参数说明

expr：DATETIME/DATETIME WITH TIME ZONE/TIME/TIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00, 9999-12-31 23:59:59]/[00:00:00, 23:59:59]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[0, 999000]

示例

```
SQL> SELECT MICROSECOND(CAST('2000-01-02 12:34:56.123456' AS
   DATETIME WITH TIME ZONE));

EXPR1 |
```

```
------------------------------------------------------------
123456 |

SQL> SELECT MICROSECOND('12:34:56.123456'::TIME);

EXPR1 |
------------------------------------------------------------
123000 |
```

3.48 MINUTE

功能描述

提取时间中的“分钟”部分。

语法格式

```
MINUTE(expr)
```

参数说明

expr：DATETIME/DATETIME WITH TIME ZONE/TIME/TIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00, 9999-12-31 23:59:59]/[00:00:00, 23:59:59]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[0, 59]

示例

```
SQL> SELECT MINUTE('1987-01-01');

EXPR1 |
------------------------------------------------------------
0 |

SQL> SELECT MINUTE('1987-01-01 12:12:12');

EXPR1 |
------------------------------------------------------------
```

```
12 |

SQL> SELECT MINUTE('1987-01-01 12:12:12 +08:00');

EXPR1 |
------------------------------------------------------------
12 |
```

3.49 MONTH

功能描述

提取日期中的“月份”部分。

与 MySQL 无差异。

语法格式

```
MONTH(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围为 [0001-01-01, 9999-12-31]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[1, 12]

示例

```
SQL> SELECT MONTH('1987-01-01');

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT MONTH('1987-01-01 12:12:12');

EXPR1 |
------------------------------------------------------------
1 |
```

```
SQL> SELECT MONTH('1987-01-01 12:12:12 +08:00');

EXPR1 |
------------------------------------------------------------
1 |
```

3.50 MONTHNAME

功能描述

返回给定日期的月份名称。

语法格式

```
MONTHNAME(expr)
```

参数说明

expr：DATETIME 类型的字段或变量或表达式。

函数返回类型

CHAR 类型字符串。

示例

```
SQL> SELECT MONTHNAME('2022-01-03') FROM dual;

EXPR1 |
------------------------------------------------------------
Janaury|

SQL> SELECT MONTHNAME('2022-02-03 12:34:56 BC') FROM dual;

EXPR1 |
------------------------------------------------------------
February|

SQL> SELECT MONTHNAME('2022-03-03 12:34:56 -07:50 BC') FROM dual;

EXPR1 |
------------------------------------------------------------
March|

SQL> SELECT MONTHNAME('') FROM dual;

EXPR1 |
------------------------------------------------------------
<NULL>|
```

3.51 MONTHS_BETWEEN

功能描述

计算两个日期之间的月份差。

语法格式

```
MONTHS_BETWEEN(expr1,expr2)
```

参数说明

expr1、expr2：DATETIME 类型的字段或变量或表达式。

若 expr2 大于 expr1，则返回负数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT MONTHS_BETWEEN('2021-06-22 02:00:00','
   2021-08-22 02:00:00') FROM dual;

EXPR1 |
------------------------------------------------------------
-2 |
```

3.52 NEXT_DAY

功能描述

返回给定日期之后的第一个指定星期几的日期。

语法格式

```
NEXT_DAY(expr1,expr2)
```

参数说明

- expr1：DATETIME 类型的字段或表达式。
- expr2: 距离指定日期最近周几。

> **说明**
>
> 参数 expr2 可使用 INTEGER 或 VARCHAR 类型的字段或表达式。其中 INTEGER 类型可取值 1 7（1 表示距离指定日期最近的周日，2 表示距离指定日期最近的周一，以此类推）；VARCHAR 类型可取值'MONDAY'至'SUNDAY'。

函数返回类型

DATETIME 日期时间类型。

示例

```
SQL> SELECT NEXT_DAY('2022-02-22 02:00:00',1) FROM dual;

EXPR1 |
------------------------------------------------------------
2022-02-27 02:00:00.000 AD |
```

3.53 NOW

功能描述

返回当前的日期和时间。

语法格式

```
NOW()
```

参数说明

无参数。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT NOW() FROM dual;

EXPR1 |
------------------------------------------------------------
2022-05-14 10:49:24.759 AD |
```

3.54 PERIOD_ADD

功能描述

在给定的时期（以 YYYYMM 或 YYMM 格式表示）上添加指定数量的月份。

语法格式

```
PERIOD_ADD(expr1,expr2)
```

参数说明

- expr1：时间段，格式为 YYMM 或 YYYYMM。

- expr2：增加的月份数值。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT PERIOD_ADD(202212,1) FROM dual;

EXPR1 |
------------------------------------------------------------
202301 |

SQL> SELECT PERIOD_ADD(202212,-1) FROM dual;

EXPR1 |
------------------------------------------------------------
202211 |
```

3.55 PERIOD_DIFF

功能描述

计算两个时期之间的月份数差。

与 MySQL 无差异。

语法格式

```
PERIOD_DIFF(expr1,expr2)
```

参数说明

expr1、expr2：时间段，格式为 YYMM 或 YYYYMM。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT PERIOD_DIFF(202211,202212);

EXPR1 |
------------------------------------------------------------
-1 |

SQL> SELECT PERIOD_DIFF(202211,202210);

EXPR1 |
------------------------------------------------------------
1 |

SQL>  SELECT PERIOD_DIFF(1703,1604);
```

```
EXPR1 |
------------------------------------------------------------
11 |
```

3.56 QUARTER

功能描述

返回给定日期所属的季度。

与 MySQL 无差异。

语法格式

```
QUARTER(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> 说明
>
> 参数取值范围：[0001-01-01, 9999-12-31]

函数返回类型

INTEGER 类型。

> 说明
>
> 返回值范围：[1, 4]

示例

```
SQL> SELECT QUARTER('1987-01-01');

EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT QUARTER('2022-11-04 12:34:56');

EXPR1 |
------------------------------------------------------------
4 |

SQL> SELECT QUARTER('2022-07-04 12:34:56 +08:00');
```

```
EXPR1 |
------------------------------------------------------------
3 |
```

3.57 SECOND

功能描述

提取时间中的“秒”部分。

与 MySQL 差异：

- 参数和取值范围小于 MySQL，因为 MySQL 允许小时数大于 23，如 272:59:59 在 MySQL 中合法，而在虚谷数据库中不合法。
- 对参数为日期时间/时间的字符串：MySQL 不需对参数做转换；虚谷数据库需要先把参数做显示转换。例如：

  MySQL：SELECT SECOND(’03:04:05’);

  虚谷数据库：SELECT SECOND(’03:04:05’::time); 或 SELECT SECOND(CAST(’03:04:05’ AS TIME));

  注：对 TIME WITH TIME ZONE 类型的字符串，只支持 CAST() 方式转换，不支持:: 类型名方式转换。

语法格式

```
SECOND(expr)
```

参数说明

expr：DATETIME/DATETIME WITH TIME ZONE/TIME/TIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00, 9999-12-31 23:59:59]/[00:00:00, 23:59:59]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[0, 59]

示例

```
SQL> SELECT SECOND('1987-01-01');

EXPR1 |
------------------------------------------------------------
0 |

SQL> SELECT SECOND('1987-01-01 12:12:12');

EXPR1 |
------------------------------------------------------------
12 |

SQL> SELECT SECOND('1987-01-01 12:12:12 +08:00');

EXPR1 |
------------------------------------------------------------
12 |
```

3.58 SEC_TO_TIME

功能描述

将给定的秒数转换为 TIME 类型的时间值。

与 MySQL 差异：

- MySQL 支持参数为负数，得到的结果也为负数时间；虚谷数据库中负数参数的结果为 NULL。
- MySQL 的结果中秒可以有小数位；虚谷数据库的结果精确到秒。

语法格式

```
SEC_TO_TIME(expr)
```

参数说明

expr：INT 类型的数值（秒数）。

> **说明**
>
> 参数取值范围：[0, 86399]

函数返回类型

TIME 类型。

> **说明**
>
> 返回值范围：[00:00:00, 23:59:59]

示例

```
SQL> SELECT SEC_TO_TIME(12345);

EXPR1 |
------------------------------------------------------------
03:25:45.000 |

SQL> SELECT SEC_TO_TIME(1);

EXPR1 |
------------------------------------------------------------
00:00:01.000 |
```

3.59 STATEMENT_TIMESTAMP

功能描述

返回当前语句开始执行的时间戳。

与 CLOCK_TIMESTAMP 不同，STATEMENT_TIMESTAMP 返回的是每个 SQL 语句开始执行时的时间，而不是系统时钟的实时时间。

> **说明**
>
> 返回值的时区不会根据客户端的时区设置进行偏移，其值精确到秒。

语法格式

```
STATEMENT_TIMESTAMP()
```

参数说明

无参数。

函数返回类型

DATETIME WITH TIME ZONE 类型。

示例

```
SQL> SELECT STATEMENT_TIMESTAMP();

EXPR1 |
------------------------------------------------------------
2024-11-06 17:20:57.000 AD +08:00 |
```

3.60 SUBDATE

功能描述

从给定的日期中减去指定的时间间隔。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷数据库返回具体结果。
- SUBDATE 函数运算 INTERVAL 'YEARS-MONTHS' YEAR TO MONTH，当 MONTHS 超过 12 时，虚谷数据库报错，MySQL 则在年份上 +1。
- INTERVAL 类型的差异，MySQL 中传入的 INTERVAL 类型进行运算时，最大单位不轮转，而是递增，不受时间格式的最大限制。

语法格式

```
SUBDATE(expr1,INTERVAL expr2)
```

参数说明

- expr1：DATE/TIME/DATETIME 类型时间，原时间。
- expr2：减少的时间。

函数返回类型

DATE/TIME/DATETIME 类型。

示例

```
SQL> SELECT SUBDATE('2018-05-01',INTERVAL '1' YEAR);

EXPR1 |
------------------------------------------------------------
2017-05-01 00:00:00.000 AD |

SQL> SELECT SUBDATE('2020-12-31 23:59:59',INTERVAL '1' SECOND);

EXPR1 |
------------------------------------------------------------
2020-12-31 23:59:58.000 AD |

SQL> SELECT SUBDATE('2018-12-31 23:59:59',INTERVAL '1' DAY);
```

```
EXPR1 |
-------------------------------------------------------------
2018-12-30 23:59:59.000 AD |
```

3.61 SUBTIME

功能描述

用于从一个时间值中减去另一个时间值或指定的时间间隔。

与 MySQL 差异：与 MySQL 差异同 SUBDATE。

语法格式

```
SUBTIME(expr1,INTERVAL expr2)
```

参数说明

- expr1：DATE/TIME/DATETIME 类型时间，原时间。
- expr2：减少的时间。

函数返回类型

DATE/TIME/DATETIME 类型。

示例

```
SQL> SELECT SUBTIME('2020-12-31 23:59:59',INTERVAL '1' SECOND);

EXPR1 |
-------------------------------------------------------------
2020-12-31 23:59:58.000 AD |

SQL> SELECT SUBTIME('2018-05-01',INTERVAL '1' YEAR);

EXPR1 |
-------------------------------------------------------------
2017-05-01 00:00:00.000 AD |

SQL> SELECT SUBTIME('2018-05-01',INTERVAL '30' DAY);

EXPR1 |
-------------------------------------------------------------
2018-04-01 00:00:00.000 AD |
```

3.62 SYSDATE

功能描述

返回系统当前的日期和时间。

语法格式

```
SYSDATE()
```

参数说明

无参数。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT SYSDATE() FROM dual;

EXPR1 |
------------------------------------------------------------
2022-05-14 10:49:52.199 AD |
```

3.63 SYSDATETIME

功能描述

返回系统当前的日期和时间，格式为 YYYY-MM-DD HH24:MI:SS.FFF。

语法格式

```
SYSDATETIME()
```

参数说明

无参数。

函数返回类型

日期时间类型 DATETIME。

示例

```
SQL> SELECT SYSDATETIME() FROM dual;

EXPR1 |
------------------------------------------------------------
2022-05-14 10:50:20.535 AD |
```

3.64 SYSTIME

功能描述

返回系统当前的时间，格式为 HH24:MI:SS.FFF。

语法格式

```
SYSTIME()
```

参数说明

无参数。

函数返回类型

时间类型 TIME。

示例

```
SQL> SELECT SYSTIME() FROM dual;

EXPR1 |
------------------------------------------------------------
10:50:47.408 |
```

3.65 SYSTIMESTAMP

功能描述

返回数据库所在系统的系统日期，包括秒的小数部分和时区。

语法格式

```
SELECT SYSTIMESTAMP();
```

参数说明

无。

函数返回类型

DATETIME WITH TIME ZONE 类型。

示例

返回当前系统时间。

```
SQL> select SYSTIMESTAMP();
EXPR1 |
------------------------------------------------------------
2024-03-06 17:17:57.295 AD +08:00 |
```

3.66 TIME

功能描述

返回日期时间中的时间。

与 MySQL 差异：

- 参数和取值范围小于 MySQL，因为 MySQL 允许小时数大于 23，如 272:59:59 在 MySQL 中合法，而在虚谷数据库中不合法。
- 对于参数为时间类字符串时，MySQL 不需对参数做转换，虚谷数据库需要先把参数做显示转换。例如：

  MySQL：SELECT HOUR('2000-01-01 03:04:05');

  虚谷数据库：SELECT HOUR('03:04:05'::time); 或 SELECT HOUR(CAST('03:04:05' AS TIME);

  注：对 TIME WITH TIME ZONE 类型的字符串，只支持 CAST 方式转换，不支持:: 类型名方式转换。
- 字符串形式的 TIME 值需要指明其要转的具体的类型，否则在隐式转换时会失败。

语法格式

```
TIME(expr)
```

参数说明

expr：DATETIME/DATETIME WITH TIME ZONE/TIME/TIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00, 9999-12-31 23:59:59]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[00:00:00, 23:59:59]

示例

```
SQL> SELECT TIME('12:34:56.123456'::TIME);

EXPR1 |
------------------------------------------------------------------
```

```
12:34:56.123 |

SQL> SELECT TIME(cast('00:00:00.123456 -11:00' AS TIME WITH TIME
   ZONE));

EXPR1 |
------------------------------------------------------------
00:00:00.123 |

SQL> SELECT TIME(cast('9999-12-31 00:00:00.123456' AS DATETIME WITH
    TIME ZONE));

EXPR1 |
------------------------------------------------------------
00:00:00.123 |
```

3.67 TIME_FORMAT

功能描述

根据指定的格式字符串格式化 TIME 类型的时间值。

与 MySQL 差异：

- 对于参数值超出通常范围 [00:00:00, 23:59:59]：MySQL 可正常执行并返回结果；虚谷数据库执行报错。注：该问题是数据库基础数据类型范围的差异。
- 对于格式非标准的参数（如没有连字符）：MySQL 可正常执行并返回结果；虚谷数据库执行报错。
- 对于参数类型隐式转换：MySQL 可正常执行并返回结果；虚谷数据库执行报错。
- 对于常值 NULL 为参数：MySQL 返回 NULL；虚谷数据库需要把 NULL 参数转换为 CHAR，否则报错。

语法格式

```
TIME_FORMAT(expr1,expr2)
```

参数说明

- expr1：时间，DATE/DATETIME 类型。
- expr2：格式串，如%H:%i:%s。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT TIME_FORMAT('13:14:15.123456','%H:%i:%s');

EXPR1 |
------------------------------------------------------------
13:14:15|
```

3.68 TIMEDIFF

功能描述

计算两个时间值之间的差值。

与 MySQL 差异：

- 参数都为 DATE 类型时，MySQL 返回的是年，虚谷数据库返回的是小时。
- 两个参数的类型不相同时，MySQL 直接返回 NULL，虚谷数据库进行运算。

语法格式

```
TIMEDIFF(expr1,expr2)
```

参数说明

expr1、expr2：DATE/TIME/DATETIME 类型的时间。

函数返回类型

INTERVAL HOUR TO SECOND 类型。

> 说明
>
> INTERVAL HOUR TO SECOND 格式：'HOURS:MINUTES:SECONDS.MICROSECONDS'

示例

```
SQL> SELECT TIMEDIFF('2000-01-01 00:00:00','
   2000-01-01 00:00:00.000001');

EXPR1 |
------------------------------------------------------------
-0000000:00:00.000001 |

SQL> SELECT TIMEDIFF('2008-12-31 23:59:59.000001','
   2008-12-30 01:01:01.000002');

EXPR1 |
------------------------------------------------------------
0000046:58:57.999999 |
```

3.69 TIMEOFDAY

功能描述

返回当前的日期和时间，格式为字符串。与 CLOCK_TIMESTAMP 函数类似，返回真正的当前时间。

> 说明
>
> 返回值的时区不会根据客户端的时区设置进行偏移，其值精确至微秒。

语法格式

```
TIMEOFDAY()
```

参数说明

无参数。

函数返回类型

CHAR 类型。

> 说明
>
> 函数返回一个格式化字符串：“星期缩写月份缩写日期时:分:秒.微秒年时区缩写”。

示例

```
SQL> SELECT TIMEOFDAY();

EXPR1 |
------------------------------------------------------------------
Mon Dec 23 16:23:25.363000 2024 GMT+08:00 |
```

3.70 TIME_TO_SEC

功能描述

将 TIME 类型的时间值转换为秒数。

与 MySQL 无差异。

语法格式

```
TIME_TO_SEC(expr)
```

参数说明

expr：TIME 类型的时间。

> **说明**
>
> 参数取值范围：[00:00:00, 23:59:59]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[0, 86399]

示例

```
SQL> SELECT TIME_TO_SEC('12:34:56');

EXPR1 |
------------------------------------------------------------
45296 |

SQL> SELECT TIME_TO_SEC('00:00:00');

EXPR1 |
------------------------------------------------------------
0 |
```

3.71 TIMESTAMP

功能描述

将传入的参数转为 DATETIME 类型。

与 MySQL 差异：TIMESTAMP 第二个参数，虚谷数据库只支持 TIME 格式‘HH:MI:SS’，MySQL 可以对各种格式字符串进行转换解析。

语法格式

```
TIMESTAMP(expr1[,expr2])
```

参数说明

expr1：DATE 或 DATETIME 类型；expr2：TIME 类型。

> **说明**
>
> 当传入一个参数时，expr 为 DATE 或 DATETIME 类型，返回值为 DATETIME 类型。当传入两个参数时，expr1 为 DATE 或 DATETIME 类型，expr2 为 TIME 类型，返回值为 expr1+expr2 且值为 DATETIME 类型。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT TIMESTAMP('2023-03-16'),TIMESTAMP('2023-03-16','
   15:32:45');

EXPR1 | EXPR2 |
------------------------------------------------------------------
2023-03-16 00:00:00.000 AD | 2023-03-16 15:32:45.000 AD |
```

3.72 TIMESTAMPADD

功能描述

在给定的日期时间值上加上指定的时间间隔。

与 MySQL 差异：

TIMESTAMPADD() 函数使用注意事项：当第三个参数为 DATE 类型时，第一个参数仅支持年、季度、月份、周、日；若第一个参数为时、分、秒、微秒，则函数返回 NULL，此时需要将第三个参数由 DATE 类型转换为 DATETIME 类型。

语法格式

```
TIMESTAMPADD(expr1,expr2,expr3)
```

参数说明

- expr1：SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER 或 YEAR。
- expr2：添加的时间，INT 类型数值。
- expr3：原时间（DATE/DATETIME/DATETIME WITH TIME ZONE）。

函数返回类型

DATE/DATETIME/DATETIME WITH TIME ZONE 类型。

示例

```
SQL> SELECT TIMESTAMPADD(DAY,2,'2003-01-02');
```

```
EXPR1 |
--------------------------------------------------------------
2003-01-04 00:00:00.000  |

SQL> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');

EXPR1 |
--------------------------------------------------------------
2003-01-02 00:01:00.000  |
```

3.73 TIMESTAMPDIFF

功能描述

计算两个日期或日期时间值之间的差异，并返回指定时间单位的差值。

与 MySQL 差异：

TIMESTAMPDIFF() 函数使用注意事项：当第三个参数为 DATE 类型时，第一个参数仅支持年、季度、月份、周、日；若第一个参数为时、分、秒、微秒，则函数返回 NULL，此时需要将第三个参数由 DATE 类型转换为 DATETIME 类型。

语法格式

```
TIMESTAMPDIFF(expr1,expr2,expr3)
```

参数说明

- expr1：SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER 或 YEAR。
- expr2：时间（DATE/DATETIME/DATETIME WITH TIME ZONE）。
- expr3：时间（DATE/DATETIME/DATETIME WITH TIME ZONE）。

函数返回类型

BIGINT。

示例

```
SQL> SELECT TIMESTAMPDIFF(DAY,'2003-01-10','2003-01-02');

EXPR1 |
--------------------------------------------------------------
-8 |

SQL> SELECT TIMESTAMPDIFF(YEAR,'2010-01-01','2003-01-02');

EXPR1 |
--------------------------------------------------------------
-6 |
```

```
SQL> SELECT TIMESTAMPDIFF(YEAR,'2010-01-01','2012-01-02');

EXPR1 |
------------------------------------------------------------
2 |
```

3.74 TO_DATE

功能描述

将字符串转换为指定格式日期类型。

语法格式

```
TO_DATE(expr1,[expr2])
```

参数说明

- expr1：CHAR 或 VARCHAR 数据类型的字段或表达式。
- expr2：CHAR 类型，日期时间的格式（如：YYYY-MM-DD HH24:MI:SS），若省略 expr2，默认格式由参数 Time_Format 确定。

> 说明
>
> TIME_FORMAT 为会话级参数，通过 SHOW/SET 查看和设置。

日期时间格式说明：

格式（不区分大小写）	描述
YY/YYY/YYYY	年
MM	月
DD	日
HH/HH24	时
MI	分
SS[.SSSSSS]	秒

> **说明**
>
> 在 MySQL 兼容模式下，支持 YYYY，YYYYMMDD，YYYYMMDDHHMMSS 格式的纯数字转换为日期时间。

函数返回类型

DATETIME 日期时间类型。

示例

```
SQL> SELECT to_date('2022-02-22 02:00:00') FROM dual;

EXPR1 |
------------------------------------------------------------
2022-02-22 02:00:00.000 AD |

Total 1 records.

Use time:0 ms.

SQL> SELECT to_date('02-22-2022 02:00:00','mm-dd-yyyy hh24:mi:ss')
   FROM dual;

EXPR1 |
------------------------------------------------------------
2022-02-22 02:00:00.000 AD |
```

3.75 TO_DAYS

功能描述

计算从 [0000-00-00] 开始到指定日期的天数。

与 MySQL 差异：

- MySQL 支持数值型的日期（如 950501 代表'1995-05-01'）；虚谷数据库不支持。
- 函数使用限制：暂不支持公元前的日期。

语法格式

```
TO_DAYS(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00,9999-12-31 23:59:59]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[366, 3652424]

示例

```
SQL> SELECT TO_DAYS('2022-01-01');

EXPR1 |
------------------------------------------------------------
738521 |

SQL> SELECT TO_DAYS('2022-01-01 12:34:56');

EXPR1 |
------------------------------------------------------------
738521 |

SQL> SELECT TO_DAYS('2022-01-01 12:34:56 +08:00');

EXPR1 |
------------------------------------------------------------
738521 |
```

3.76 TO_SECONDS

功能描述

计算从 [0000-00-00] 开始到指定日期的秒数。

语法格式

```
TO_SECONDS(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01 00:00:00,9999-12-31 23:59:59]

函数返回类型

BIGINT 类型。

> **说明**
>
> 返回值范围：[31622400, 315569519999]

示例

```
SQL> SELECT TO_SECONDS('2022-01-01');

EXPR1 |
------------------------------------------------------------
63808214400 |

SQL> SELECT TO_SECONDS('2022-01-01 12:34:56');

EXPR1 |
------------------------------------------------------------
63808259696 |

SQL> SELECT TO_SECONDS('2022-01-01 12:34:56 +08:00');

EXPR1 |
------------------------------------------------------------
63808259696 |
```

3.77 TO_TIMESTAMP

功能描述

将一个字符串转换为 TIMESTAMP 类型的值。

语法格式

```
TO_TIMESTAMP(expr1, expr2)
```

参数说明

- expr1：待转换的字符串。
- expr2：指定 expr 的格式；典型的值如’YYYY-MM-DD HH24:MI.SS’。

函数返回类型

TIMESTAMP 日期时间类型。

示例

```
SELECT TO_TIMESTAMP('1970-01-01 12:34:56.789', 'yyyy-mm-dd hh24:mi:
   ss.fff'),
TO_TIMESTAMP('10/09/12 05:10:10.123', 'DD/MM/YY HH24:MI:SS.FF')
FROM dual;

EXPR1 | EXPR2 |
------------------------------------------------------------------
1970-01-01 12:34:56.789 AD | 2012-09-10 05:10:10.123 AD |
```

3.78 TRANSACTION_TIMESTAMP

功能描述

返回当前事务开始时的时间戳。这个时间戳是在事务启动时记录的，并且在整个事务的生命周期内保持不变。

语法格式

```
TRANSACTION_TIMESTAMP()
```

参数说明

无参数

函数返回类型

TIMESTAMP WITH TIME ZONE 类型。

示例

```
SQL> set auto_commit off;

Execute successful.
Use time:0 ms.

SQL> SELECT TRANSACTION_TIMESTAMP();

EXPR1 |
------------------------------------------------------------------
2024-04-03 09:21:56.000 AD +08:00 |

Total 1 records.

Use time:0 ms.

SQL> SELECT SLEEP(2000);

EXPR1 |
```

```
-------------------------------------------------------------
2000 |

Total 1 records.

Use time:2000 ms.

SQL> SELECT TRANSACTION_TIMESTAMP();

EXPR1 |
-------------------------------------------------------------
2024-04-03 09:21:56.000 AD +08:00 |
```

3.79 TRUNC

功能描述

将时间戳或时间间隔截断到指定的时间单位。

也可用作数值截断，详细信息请参见数值数据类型函数的 TRUNC 章节。

语法格式

```
TRUNC(SOURCE[, FORMAT])
```

参数说明

- SOURCE：类型为 DATE、TIMESTAMP、TIMESTAMPTZ、TIME 或 TIMETZ 的值表达式
- FORMAT：截取格式。表示对输入值选用什么样的精度进行截断，大小写不敏感。如果省略，则使用默认格式“D”，返回的值将日期截断至天，即午夜 00:00:00。

FORMAT 参数取值

FORMAT	取值说明
• SYYYY • YYYY • YEAR • SYEAR • YYY • YY • Y	截断到年份
• MONTH • MON • MM • RM	截断到月份
• D • d	截断到日
• H • h	截断到小时
	接下页

FORMAT	取值说明
• MINUTE • MIN	截断到分钟
• S • s	截断到秒

函数返回类型

返回值类型和 SOURCE 的类型保持一致。

示例

```
SQL> SELECT DATE_TRUNC('year', '2020-10-20 12:30:25.5020555');

EXPR1 |
------------------------------------------------------------
2020-01-01 00:00:00.000 AD |

-- 截 断 时 间 间 隔 类 型
SQL> SELECT DATE_TRUNC('milliseconds', '20 12:30:25.5020555'::
   interval day to second);

EXPR1 |
------------------------------------------------------------
20 12:30:25.502000 |

-- 截 断 到 小 时
SQL> SELECT DATE_TRUNC('hour', '2020-10-20 12:30:25.5020555','
   GMT+06:00');

EXPR1 |
------------------------------------------------------------
2020-10-20 12:00:00.000 AD +08:00 |

-- 截 断 到 毫 秒
SQL> SELECT DATE_TRUNC('milliseconds', '
   2020-10-20 12:30:25.5020555','GMT-04:00');

EXPR1 |
------------------------------------------------------------
2020-10-20 12:30:25.502 AD +08:00 |
```

3.80 WEEK

功能描述

返回指定日期所在的周数。

与 MySQL 差异：

- 第二个参数为布尔型时，MySQL 可以返回值，虚谷数据库则报错。
- 传入错误时间，MySQL 返回 NULL，虚谷数据库则报错。

语法格式

```
WEEK(expr1[,expr2])
```

参数说明

- expr1：DATE 或 DATETIME 类型，当参数为 TIME 类型时，其对应的日期为当前系统日期。
- expr2：可选参数，缺省值为系统参数 WEEK_MODE，用于确定周数计算的逻辑，INT 类型，可以取值 [0,1,2,3,4,5,6,7]。例如当 expr2 为 2 时，则表示本年的第一个星期天为第一周的开始日。

expr2 详细取值

expr2	一周的第一天	周数范围
0	周日	0-53
1	周一	0-53
2	周日	1-53
3	周一	1-53
4	周日	0-53
5	周一	0-53
6	周日	1-53
7	周一	1-53

函数返回类型

TINYINT 类型。

示例

```
SQL> SELECT WEEK('2023-03-16'),WEEK('2023-03-16',3);

EXPR1 | EXPR2 |
------------------------------------------------------------
11 | 11 |
```

3.81 WEEKDAY

功能描述

从给定的日期中提取该日期是星期几的索引。

与 MySQL 无差异。

语法格式

```
WEEKDAY(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围：[0001-01-01, 9999-12-31]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围：[0, 6]

示例

```
SQL> SELECT WEEKDAY('1987-01-01');

EXPR1 |
------------------------------------------------------------
3 |

SQL> SELECT WEEKDAY('2022-01-04 12:34:56');

EXPR1 |
```

```
-------------------------------------------------------------
1 |

SQL> SELECT WEEKDAY('2022-01-04 12:34:56 +08:00');

EXPR1 |
-------------------------------------------------------------
1 |
```

3.82 WEEKOFYEAR

功能描述

从给定的日期中提取该日期所在的一年的周数。返回值 [1, 53]，等同于 WEEK(date, 3)。

与 MySQL 差异：传入错误时间，MySQL 返回 NULL，虚谷数据库报错。

语法格式

```
WEEKOFYEAR(expr)
```

参数说明

expr：DATE 或 DATETIME 类型。

函数返回类型

TINYINT 类型。

示例

```
SQL> SELECT WEEKOFYEAR('2023-03-16');

EXPR1 |
-------------------------------------------------------------
11 |
```

3.83 YEAR

功能描述

从给定的日期中提取该日期所在的年份。

与 MySQL 差异：虚谷数据库不支持参数为 TIME 类型。当参数为 TIME 类型时，MySQL 可以将’10:10:10’ 格式的时间隐式转换为 DATE 类型。

语法格式

```
YEAR(expr)
```

参数说明

expr：DATE/DATETIME/DATETIME WITH TIME ZONE 类型的时间。

> **说明**
>
> 参数取值范围为 [0001-01-01, 9999-12-31]

函数返回类型

INTEGER 类型。

> **说明**
>
> 返回值范围为 [1, 9999]

示例

```
SQL> SELECT YEAR('1987-01-01');

EXPR1 |
------------------------------------------------------------
1987 |

SQL> SELECT YEAR('1987-01-01 12:12:12');

EXPR1 |
------------------------------------------------------------
1987 |

SQL> SELECT YEAR('1987-01-01 12:12:12 +08:00');

EXPR1 |
------------------------------------------------------------
1987 |
```

3.84 YEARWEEK

功能描述

从给定的日期中提取该日期所在的年份和周数。

与 MySQL 差异：

- 第二个参数为布尔型时，MySQL 可以返回值，虚谷数据库则报错。
- 传入错误时间，MySQL 返回 NULL，虚谷数据库则报错。

语法格式

```
YEARWEEK(expr1[,expr2])
```

参数说明

- expr1：DATE 或 TIME 类型。
- expr2：INT 类型，参数含义与 WEEK 函数中的 expr2 参数一致，缺省值为 0。

函数返回类型

INT 类型。

示例

```
SQL> SELECT YEARWEEK('2023-03-16');

EXPR1 |
------------------------------------------------------------
202311 |
```

3.85 UNIX_TIMESTAMP

功能描述

将日期时间值转换为 Unix 时间戳，或者返回当前的 Unix 时间戳。Unix 时间戳是从 1970 年 1 月 1 日 00:00:00 UTC（协调世界时）开始计算的秒数。

语法格式

```
UNIX_TIMESTAMP(expr)
```

参数说明

- 参数 0 个或 1 个，参数 expr 可以为 DATE、DATETIME、TIMESTAMP 类型。
  - 如果参数 expr 对应的时间早于该时间，则函数返回值为负数。
  - 如果参数 expr 对应的时间晚于该时间，则函数返回值为正数。
- 不传入参数时，默认将系统当前 UTC 时间转换为时间戳。

函数返回类型

NUMERIC 数值类型。

示例

```
SQL> SELECT unix_timestamp('2012-10-01 15:32:45');

EXPR1 |
------------------------------------------------------------
1349105565|
```

```
SQL> SELECT unix_timestamp('2012-10-01 15:32:45.211985');

EXPR1 |
------------------------------------------------------------
1349105565.211985|

SQL> SELECT unix_timestamp('1960-10-01 15:32:45.211985');

EXPR1 |
------------------------------------------------------------
-291889634.788015|

SQL> SELECT unix_timestamp();

EXPR1 |
------------------------------------------------------------
```

3.86 UTC_DATE

功能描述

返回 UTC 日期，与 SYSDATE 相差 8 小时。

与 MySQL 无差异。

语法格式

```
UTC_DATE()
```

参数说明

无参数。

函数返回类型

DATE 类型。

示例

```
SQL> SELECT UTC_DATE();

EXPR1 |
------------------------------------------------------------
2022-11-15 AD |

SQL> SELECT UTC_TIME();

EXPR1 |
------------------------------------------------------------
07:14:37.441 |

SQL> SELECT SYSDATE();
```

```
EXPR1 |
------------------------------------------------------------
2022-11-15 15:14:40.496 AD |
```

3.87 UTC_TIME

功能描述

返回 UTC 时间。

与 MySQL 无差异。

语法格式

```
UTC_TIME()
```

参数说明

无参数。

函数返回类型

TIME 类型。

示例

```
SQL> SELECT UTC_TIME();

EXPR1 |
------------------------------------------------------------
07:09:18.821 |
```

3.88 UTC_TIMESTAMP

功能描述

返回 UTC 日期时间，与 SYSDATE 相差 8 小时。

与 MySQL 无差异。

语法格式

```
UTC_TIMESTAMP()
```

参数说明

无参数。

函数返回类型

DATETIME 类型。

示例

```
SQL> SELECT UTC_TIMESTAMP();

EXPR1 |
------------------------------------------------------------------
2022-11-15 07:12:42.242 AD |

SQL> SELECT SYSDATE();

EXPR1 |
------------------------------------------------------------------
2022-11-15 15:12:54.109 AD |
```

4 类型转换函数

4.1 概述

函数形式	功能
ASCII	返回字符表达式最左端字符的 ASCII 代码值
CHR	将 ASCII 码转换为字符
HEXTORAW	将一个十六进制构成的字符串转换为二进制
NUMTODSINTERVAL	以参数 2 为单位将参数 1 转换为时间间隔类型 INTERVAL DAY TO SECOND
NUMTOYMINTERVAL	以参数 2 为单位将参数 1 转换为时间间隔类型 INTERVAL YEAR TO MONTH
RAWTOHEX	将一个二进制构成的字符串转换为十六进制
TO_BLOB	将十六进制字符串转为 BLOB 类型的值

4.2 ASCII

功能描述

返回字符表达式最左端字符的 ASCII 代码值。

语法格式

```
ASCII(expr)
```

参数说明

expr：字符表达式。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT ASCII('Aaaaa');

EXPR1 |
-------------------------------------------------------------
65 |
```

4.3 CHR

功能描述

将 ASCII 码转换为字符。

语法格式

```
CHR(expr)
```

参数说明

expr：ASCII 码，BIGINT 类型数值。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT CHR(65);

EXPR1 |
-------------------------------------------------------------
A|
```

4.4 HEXTORAW

功能描述

将一个十六进制构成的字符串转换为二进制。

与 Oracle 差异：Oracle 有限制，输入必须为 0 9、a f，虚谷数据库未限制。

语法格式

```
HEXTORAW(expr)
```

参数说明

expr：十六进制的字符串。

函数返回类型

BINARY 类型。

示例

```
SQL> SELECT HEXTORAW('abcdef') FROM dual;

EXPR1 |
------------------------------------------------------------
<BINARY> |
```

4.5 NUMTODSINTERVAL

功能描述

将 expr1 转换为时间间隔类型 INTERVAL DAY TO SECOND，expr2 用于指定 expr1 的单位。

语法格式

```
NUMTODSINTERVAL(expr1,expr2)
```

参数说明

- expr1：NUMERIC 类型或者能隐式转换为 NUMERIC 类型的其他类型。
- expr2：CHAR 类型，其可选值为 day, hour, miute, second，不区分大小写。

函数返回类型

INTERVAL DAY TO SECOND 类型。

示例

```
SQL> SELECT numtodsinterval(1000,'hour') FROM dual;

EXPR1 |
------------------------------------------------------------
 0000041 16:00:00.000000 |
```

4.6 NUMTOYMINTERVAL

功能描述

将 expr1 转换为时间间隔类型 INTERVAL YEAR TO MONTH，expr2 用于指定 expr1 的单位。

语法格式

```
NUMTOYMINTERVAL(expr1,expr2)
```

参数说明

- expr1：NUMERIC 类型或者能隐式转换为 NUMERIC 类型的其他类型。
- expr2：CHAR 类型，其可选值为 year, month，不区分大小写。

函数返回类型

INTERVAL YEAR TO MONTH 类型。

示例

```
SQL> SELECT numtoyminterval(1000,'month') FROM dual;

EXPR1 |
------------------------------------------------------------
 83-04 |
```

4.7 RAWTOHEX

功能描述

将一个二进制构成的字符串转换为十六进制。

与 Oracle 差异：Oracle 只支持输入二进制的字符串，虚谷数据库不支持直接输入数字。

语法格式

```
RAWTOHEX(expr)
```

参数说明

expr：二进制的字符串。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT RAWTOHEX(HEXTORAW('abcdef')) FROM dual;

EXPR1 |
------------------------------------------------------------
ABCDEF|
```

4.8 TO_BLOB

功能描述

将十六进制字符串转为 BLOB 类型的值。

语法格式

```
TO_BLOB(expr)
```

参数说明

expr：十六进制字符串。

函数返回类型

BLOB 类型。

示例

```
SQL> SELECT TO_BLOB('ab');

EXPR1 |
------------------------------------------------------------
<BLOB> |
```

5 空值转换函数

5.1 概述

函数形式	功能
IFNULL	判断参数 1 是否为 NULL，当参数 1 为 NULL 时，函数返回参数 2，否则返回参数 1
NULLIF	比较两个参数的值，若二者相等，则返回 NULL，否则返回第一个参数的值
NVL	若第一个参数不为 NULL，则返回第一个参数的值，否则返回第二个参数的值
NVL2	若第一个参数不为 NULL，则返回第二个参数的值，否则返回第三个参数的值

5.2 IFNULL

功能描述

判断 expr1 是否为 NULL，当 expr1 为 NULL 时，函数返回 expr2，否则返回 expr1。

语法格式

```
IFNULL(expr1,expr2)
```

参数说明

expr1、expr2：任意两个具有公共数据类型的数据类型。

函数返回类型

返回值类型为 expr1 和 expr2 的公共数据类型。

示例

```
SQL> SELECT IFNULL(null,'yes'), IFNULL(1,'yes') FROM dual;

EXPR1 | EXPR2 |
-----------------------------------------------------------------
```

```
yes| 1 |
```

5.3 NULLIF

功能描述

比较两个参数的值，若二者相等，则返回 NULL，否则返回第一个参数的值。

语法格式

```
NULLIF(expr1,expr2)
```

参数说明

expr1、expr2：空值、数值型、字符型和日期型字段。

> ⚠ 注意
>
> expr1 和 expr2 的数据类型必须一致。

函数返回类型

返回值对应类型。

示例

```
SQL> SELECT NULLIF(2,3);

EXPR1 |
------------------------------------------------------------
2 |

SQL> SELECT NULLIF(3,3);

EXPR1 |
------------------------------------------------------------
<NULL>|
```

5.4 NVL

功能描述

若第一个参数不为 NULL，则返回第一个参数的值，否则返回第二个参数的值。

语法格式

```
NVL(expr1,expr2)
```

参数说明

expr1、expr2：空值、数值型、字符型和日期型字段。

> **⚠ 注意**
>
> expr1 和 expr2 的数据类型必须一致。

函数返回类型

返回值对应类型。

示例

```
SQL> SELECT NVL(NULL,2);

EXPR1 |
------------------------------------------------------------------
2 |

SQL> SELECT NVL(1,2);

EXPR1 |
------------------------------------------------------------------
1 |
```

5.5 NVL2

功能描述

若第一个参数不为 NULL，则返回第二个参数的值，否则返回第三个参数的值。

语法格式

```
NVL2(expr1,expr2,expr3)
```

参数说明

expr1、expr2、expr3：空值、数值型、字符型和日期型字段。

函数返回类型

返回值对应类型。

示例

```
SQL> SELECT NVL2(NULL,2,3);

EXPR1 |
------------------------------------------------------------------
3 |
```

```
SQL> SELECT NVL2(NULL,TO_DATE('2022-11-11'),TO_DATE('2022-12-12'));

EXPR1 |
------------------------------------------------------------
2022-12-12 00:00:00.000 AD |
```

6 数组类型函数

6.1 概述

函数形式	功能
ARRAY_APPEND	添加一个数组成员在尾部
ARRAY_CAT	拼接两个数组
ARRAY_DIMS	返回数组维度
ARRAY_FILL	用一个元素填满一个制定了维度的数组
ARRAY_LENGTH	返回所选数组在指定维度的长度
ARRAY_LOWER	返回所选数组在指定维度的下界
ARRAY_POSITION	返回数组成员的下标位置
ARRAY_POSITIONS	以数组格式返回成员的下标位置
ARRAY_PREPEND	添加一个数组成员在头部
ARRAY_REMOVE	删除数组中的某元素
ARRAY_REPLACE	替换数组中的某元素
ARRAY_SAMPLE	随机返回指定大小的子数组
ARRAY_SHUFFLE	将数组成员的次序随机打乱返回
ARRAY_UPPER	返回所选数组在指定维度的上界
CARDINALITY	返回数组元素总数
COMPRESS_FLOATS	32 位 FLOAT[] 数组类型转换为 16 位 FLOAT[] 数组类型，并以 BINARY 类型数据返回

接下页

函数形式	功能
TRIM_ARRAY	倒序移除指定个数的元素
UNCOMPRESS_FLOATS	将 BINARY 类型数据解压为 FLOAT[] 类型数据
ARRAY 数据类型运算符	ARRAY 数组类型支持的运算符

6.2 ARRAY_APPEND

功能描述

将一个元素添加到数组的末尾。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_APPEND(anycompatiblearray, anycompatible)
```

参数说明

- anycompatiblearray：要添加元素的数组，ARRAY 类型数据
- anycompatible：要添加到数组末尾的元素。该元素的类型必须与数组中的元素类型兼容。

函数返回类型

ARRAY 类型。

示例

添加元素 3 到数组 [1,2] 的末尾。

```
SQL> SELECT ARRAY_APPEND(ARRAY[1,2], 3);

EXPR1 |
------------------------------------------------------------------
{1,2,3}|
```

6.3 ARRAY_CAT

功能描述

连接两个数组。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_CAT(anycompatiblearray, anycompatiblearray)
```

参数说明

anycompatiblearray：要连接的数组，ARRAY 类型数据。

函数返回类型

ARRAY 类型。

示例

连接二维数组和一维数组。

```
SQL> SELECT ARRAY_CAT(ARRAY[[1,2],[2,3]], ARRAY[4,5]);

EXPR1 |
------------------------------------------------------------
{{1,2},{2,3},{4,5}}|
```

6.4 ARRAY_DIMS

功能描述

返回 ARRAY 类型参数数组维度。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_DIMS(anyarray)
```

参数说明

anyarray：ARRAY 类型数据。

函数返回类型

CHAR 类型。

示例

二维数组：

- 第一维的下标从 1 到 2（表示有 2 行）。
- 第二维的下标从 1 到 3（表示每行有 3 个元素）。

```
SQL> SELECT ARRAY_DIMS(ARRAY[[1,2,3], [4,5,6]]);

EXPR1 |
------------------------------------------------------------
[1:2][1:3] |
```

6.5 ARRAY_FILL

功能描述

创建一个填充了指定元素的数组，其维度由第二个参数指定。可选的第三个参数为每个维度提供下界值（默认为 1）。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_FILL(anyelement, integer1[][, integer2[]])
```

参数说明

- anyelement：要填充到数组中的元素。
- integer1[]：一个整数数组，表示数组的维度大小。
- integer2[]：一个整数数组，表示数组各维度的下界（即起始下标）。默认数组的下界是 1。

函数返回类型

ARRAY 类型。

示例

创建一个 2x3 的二维数组，其中每个元素都是 11。

```
SQL> SELECT ARRAY_FILL(11, ARRAY[2,3]);

EXPR1 |
------------------------------------------------------------------
{{11,11,11},{11,11,11}}|
```

创建一个长度为 3 的一维数组，所有元素都填充为 7，但数组的下标从 2 开始。

```
SQL> SELECT ARRAY_FILL(7, ARRAY[3], ARRAY[2]);

EXPR1 |
------------------------------------------------------------------
[2:4]={7,7,7}|
```

6.6 ARRAY_LENGTH

功能描述

返回 ARRAY 类型参数在指定维度的长度。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_LENGTH(anyarray, integer)
```

参数说明

- anyarray：要检查其长度的数组，ARRAY 类型数据。
- integer: 指定要查询的维度编号（从 1 开始）。

函数返回类型

INTEGER 类型。

示例

单维数组 [1, 2, 3]，包含 3 个元素。返回第一个维度（也是唯一一个维度）的长度，即 3。

```
SQL> SELECT ARRAY_LENGTH(ARRAY[1,2,3], 1);

EXPR1 |
------------------------------------------------------------
3 |
```

单维数组 [NULL]，包含 1 个元素。返回第一个维度的长度，即 1。

```
SQL> SELECT ARRAY_LENGTH(ARRAY[NULL], 1);

EXPR1 |
------------------------------------------------------------
1 |
```

单维数组 ['text']，包含 1 个字符串元素。返回第二个维度的长度，但这个数组只有一个维度，因此没有第二个维度，返回 NULL。

```
SQL> SELECT ARRAY_LENGTH(ARRAY['text'], 2);

EXPR1 |
------------------------------------------------------------
NULL |
```

6.7 ARRAY_LOWER

功能描述

返回 ARRAY 类型参数在指定维度的下界。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_LOWER(anyarray, integer)
```

参数说明

- anyarray：要检查其下界的数组，ARRAY 类型数据。
- integer：指定要查询的维度编号（从 1 开始）。

函数返回类型

INTEGER 类型。

示例

获取单维数组 [1, 2, 3] 在第一个维度上的下界。

```
SQL> SELECT ARRAY_LOWER(ARRAY[1, 2, 3], 1);

EXPR1 |
------------------------------------------------------------
1 |
```

6.8 ARRAY_POSITION

功能描述

在一个数组中查找指定元素第一次出现的位置。它返回该元素的下标（从 1 开始），如果元素不存在于数组中，则返回 NULL。指定参数 integer 时，则查询从该下标开始。此函数仅支持在一维数组中使用。

ARRAY_POSITION 返回元素第一次出现位置的下标，ARRAY_POSITIONS 返回所有匹配元素的下标（从 1 开始）。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_POSITION(anycompatiblearray, anycompatible[, integer])
```

参数说明

- anycompatiblearray：要查找的一维数组，ARRAY 类型数据。
- anycompatible：要查找的元素。该元素的类型必须与数组中的元素类型兼容。
- integer：可选参数，从哪个位置开始查找，默认为 1（即从数组的第一个元素开始查找）。

函数返回类型

INTEGER 类型。

示例

从默认下标 1 开始查找指定元素’mon’ 第一次出现的位置。

```
SQL> SELECT ARRAY_POSITION(ARRAY['sun', 'mon', 'tue', 'wed', 'thu'
   , 'fri', 'sat'], 'mon');

EXPR1 |
------------------------------------------------------------
2|
```

从下标 3 开始查找指定元素'mon' 第一次出现的位置。

```
SQL> SELECT ARRAY_POSITION(ARRAY['sun', 'mon', 'tue', 'wed', 'thu'
   , 'fri', 'sat', 'mon'], 'mon', 3);

EXPR1 |
------------------------------------------------------------
8|
```

6.9 ARRAY_POSITIONS

功能描述

在一个数组中查找指定元素所有出现位置。它返回该元素的下标（从 1 开始），如果元素不存在于数组中，则返回 NULL。此函数仅支持在一维数组中使用。

ARRAY_POSITION 返回元素第一次出现位置的下标，ARRAY_POSITIONS 返回所有匹配元素的下标（从 1 开始）。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_POSITIONS(anycompatiblearray, anycompatible)
```

参数说明

- anycompatiblearray：要查找的数组，ARRAY 类型数据。
- anycompatible：要查找的元素。该元素的类型必须与数组中的元素类型兼容。

函数返回类型

INTEGER[] 类型（整数数组）。

示例

查找指定元素 mon 在数组中的位置。

```
SQL> SELECT ARRAY_POSITIONS(ARRAY['sun', 'mon', 'tue', 'wed', 'thu'
   , 'fri', 'sat', 'mon'], 'mon');

EXPR1 |
------------------------------------------------------------
{2,8}|
```

6.10 ARRAY_PREPEND

功能描述

将一个元素添加到数组的开头。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_PREPEND(anycompatible, anycompatiblearray)
```

参数说明

- anycompatible：要添加到数组开始的元素。该元素的类型必须与数组中的元素类型兼容。
- anycompatiblearray：要添加元素的数组，ARRAY 类型数据

函数返回类型

ARRAY 类型。

示例

添加元素 1 到数组 [2,3] 的开头。

```
SQL> SELECT ARRAY_PREPEND(1, ARRAY[2,3]);

EXPR1 |
------------------------------------------------------------
{1,2,3}|
```

6.11 ARRAY_REMOVE

功能描述

从数组中删除所有等于给定值的元素，可以删除 NULL。数组必须是一维数组。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_REMOVE(anycompatiblearray, anycompatible)
```

参数说明

- anycompatiblearray：要操作的数组。
- anycompatible：要从数组中删除的元素。该元素的类型必须与数组中的元素类型兼容。

函数返回类型

ARRAY 类型。

示例

删除数组中的元素 2。

```
SQL> SELECT ARRAY_REMOVE(ARRAY[1,2,3,2], 2);

EXPR1 |
------------------------------------------------------------
{1,3}|
```

6.12 ARRAY_REPLACE

功能描述

将数组中所有等于指定元素的值替换为另一个指定的值。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_REPLACE(anycompatiblearray, old_compatible, new_compatible)
```

参数说明

- anycompatiblearray：要操作的数组。
- old_compatible：要被替换的旧元素。该元素的类型必须与数组中的元素类型兼容。
- new_compatible：用于替换旧元素的新元素。该元素的类型必须与数组中的元素类型兼容。

函数返回类型

ARRAY 类型。

示例

将数组中所有元素 5 替换为 3。

```
SQL> SELECT ARRAY_REPLACE(ARRAY[1,2,5,4], 5, 3);

EXPR1 |
------------------------------------------------------------
{1,2,3,4}|
```

6.13 ARRAY_SAMPLE

功能描述

从数组中随机抽取指定数量的元素。抽取的数量不得超过数组第一维的长度。如果数组是多维的，则具有给定第一个下标的切片。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_SAMPLE(anyarray, integer)
```

参数说明

- anyarray：要操作的数组。
- integer：要从数组中抽取的元素数量。必须是非负整数，并且不能大于数组的长度。

函数返回类型

ARRAY 类型。

示例

从一维数组中随机抽取 3 个元素。

```
SQL> SELECT ARRAY_SAMPLE(ARRAY[1,2,3,4,5,6], 3);

EXPR1 |
------------------------------------------------------------
{2,6,1} |
```

从一个二维数组中随机抽取 2 个子数组。

```
SQL> SELECT ARRAY_SAMPLE(ARRAY[[1,2],[3,4],[5,6]], 2);

EXPR1 |
------------------------------------------------------------
{{5,6},{1,2}} |
```

6.14 ARRAY_SHUFFLE

功能描述

随机打乱数组中元素的顺序。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_SHUFFLE(anyarray)
```

参数说明

anyarray：要操作的数组。

函数返回类型

ARRAY 类型。

示例

```
SQL> SELECT ARRAY_SHUFFLE(ARRAY[[1,2],[3,4],[5,6]]);

EXPR1 |
------------------------------------------------------------
{{5,6},{1,2},{3,4}} |
```

6.15 ARRAY_UPPER

功能描述

返回 ARRAY 类型参数在指定维度的上界。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
ARRAY_UPPER(anyarray, integer)
```

参数说明

- anyarray：要检查其上界的数组，ARRAY 类型数据。
- integer: 指定要查询的维度编号（从 1 开始）。

函数返回类型

INTEGER 类型。

示例

获取单维数组 [1, 2, 3] 在第一个维度上的上界。

```
SQL> ELECT ARRAY_UPPER(ARRAY[1, 2, 3], 1);

EXPR1 |
------------------------------------------------------------
3 |
```

6.16 CARDINALITY

功能描述

返回 ARRAY 类型参数数组元素总数。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
CARDINALITY(anyarray)
```

参数说明

anyarray：要检查其元素个数的数组，ARRAY 类型数据。

函数返回类型

INTEGER 类型。

示例

二维数组 [[1,2],[3,4]] 所有嵌套数组的总元素数为 4。

```
SQL> SELECT CARDINALITY(ARRAY[[1,2],[3,4]]);

EXPR1 |
------------------------------------------------------------
4 |
```

6.17 COMPRESS_FLOATS

功能描述

32 位 FLOAT[] 数组类型转换为 16 位 FLOAT[] 数组类型，并以 BINARY 类型数据返回。

函数可指定精度值，精度取值范围 [0, 3]，来源于半精度浮点数的有效范围。压缩后的数据，可通过函数 UNCOMPRESS_FLOATS 将数据解压为 FLOAT[] 类型数据。

语法格式

```
COMPRESS_FLOATS(expr1,expr2)
```

参数说明

- expr1：需要转化的数组，FLOAT[] 数据类型。
- expr2：指定精度，INTEGER 数据类型。

函数返回类型

BINARY 类型。

若任意参数为 NULL，则返回 NULL。

示例

```
SQL> CREATE TABLE table_tt(c1 FLOAT[]);

SQL> INSERT INTO table_tt VALUES('{12.34,1.34,-2.56}');

SQL> SELECT RAWTOHEX(COMPRESS_FLOATS(c1,0)),RAWTOHEX(
   COMPRESS_FLOATS(c1,1)),RAWTOHEX(COMPRESS_FLOATS(c1,2)),RAWTOHEX(
   COMPRESS_FLOATS(c1,3)) FROM table_tt;
```

```
EXPR1 |EXPR2 |EXPR3 |EXPR4 |
-------------------------------------------------------------------
00000000005000200180 0|0100000000330 01A00F600
   |0200000000010418024813|031EC15C3D2B4A|
```

6.18 TRIM_ARRAY

功能描述

从数组的末端移除指定数量的元素。如果数组是多维的，则仅移除第一维。

ARRAY 数据类型的详细信息请参见《SQL 语法参考指南》的 ARRAY 数据类型章节。

语法格式

```
TRIM_ARRAY(anyarray, integer)
```

参数说明

- anyarray：要操作的数组。
- integer：要从数组的末端移除的元素数量。此参数必须小于或等于数组长度，等于数组长度时返回空数组。

函数返回类型

ARRAY 类型。

示例

```
SQL> SELECT TRIM_ARRAY(ARRAY[1,2,3,4,5,6], 2);

EXPR1 |
-------------------------------------------------------------------
{1,2,3,4} |
```

6.19 UNCOMPRESS_FLOATS

功能描述

将 BINARY 类型数据解压为 FLOAT[] 类型数据，根据压缩函数 COMPRESS_FLOATS 指定的精度自动确定解压精度。

语法格式

```
UNCOMPRESS_FLOATS(expr)
```

参数说明

expr：需要解压的数据，BINARY 数据类型。

函数返回类型

FLOAT[] 类型。

若任意参数为 NULL，则返回 NULL。

示例

当压缩函数 COMPRESS_FLOATS 第二个参数为 3 时，指半精度压缩。

```
SQL> CREATE TABLE table_tt(c1 FLOAT[]);

SQL> INSERT INTO table_tt VALUES('{12.34,1.34,-2.56}');

SQL> SELECT UNCOMPRESS_FLOATS(COMPRESS_FLOATS(c1,0)),
   UNCOMPRESS_FLOATS(COMPRESS_FLOATS(c1,1)),UNCOMPRESS_FLOATS(
   COMPRESS_FLOATS(c1,2)),UNCOMPRESS_FLOATS(COMPRESS_FLOATS(c1,3))
   FROM table_tt;

EXPR1 |EXPR2 |EXPR3 |EXPR4 |
------------------------------------------------------------
{-2,1,12}|{-2.5,1.3,12.3}|{-2.56,1.34,12.34}|
{-2.5585938,1.3398438,12.3359375}|
```

6.20 ARRAY 数据类型运算符

@>（包含）

检查一个数组是否包含另一个数组的所有元素。

并不要求两个数组完全相同，只要左侧的数组包含右侧数组的所有元素即可。如果右侧数组中有多个相同的元素，而左侧数组中只有一个该元素，@> 仍然会返回 TRUE。

```
SQL> SELECT ARRAY[1,4,3] @> ARRAY[3,1,3];

EXPR1 |
------------------------------------------------------------
T |
```

<@（被包含）

检查一个数组是否是另一个数组的子集。即左侧的数组中的所有元素都必须存在于右侧的数组中。

并不要求两个数组完全相同，只要左侧的数组中的所有元素都在右侧的数组中即可。如果左侧数组中有多个相同的元素，而右侧数组中只有一个该元素，<@ 仍然会返回 TRUE。

```
SQL> SELECT ARRAY[2,2,7] <@ ARRAY[1,7,4,2,6];

EXPR1 |
```

```
-------------------------------------------------------------------
T |
```

&&（交集）

检查两个数组是否有至少一个共同的元素。

```
SQL> SELECT ARRAY[1,4,3] && ARRAY[2,1];

EXPR1 |
-------------------------------------------------------------------
T |
```

||（连接）

- 连接两个数组。

  ⚠ 注意

  - 数组必须具有相同的维数或维数相差一维。
  - 数组可以和 NULL 连接，不支持空数组连接。
  - 两个数组元素的类型必须相同。

```
-- 相同维数
SQL> SELECT ARRAY[1,2,3] || ARRAY[4,5,6,7];

EXPR1 |
-------------------------------------------------------------------
{1,2,3,4,5,6,7}|

-- 相差一维
SQL> SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]];

EXPR1 |
-------------------------------------------------------------------
{{1,2,3},{4,5,6},{7,8,9}}|
```

- 连接数组和元素。

  ⚠ 注意

  与元素连接的数组必须为一维数组。

  将元素连接到数组的开头：

```
SQL> SELECT 3 || ARRAY[4,5,6];

EXPR1 |
-------------------------------------------------------------------
```

```
{3,4,5,6}|
```

将元素连接到数组的末尾：

```
SQL> SELECT ARRAY[4,5,6] || 7;

EXPR1 |
------------------------------------------------------------------
{4,5,6,7}|
```

7 JSON 数据类型函数

7.1 概述

函数形式	功能
JSON_ARRAY	构建 JSON 数组
JSON_ARRAY_APPEND	添加元素至 JSON 文档
JSON_ARRAY_INSERT	插入元素至 JSON 文档
JSON_CONTAINS	JSON 文档是否在路径处包含特定对象
JSON_CONTAINS_PATH	JSON 文档是否在路径处包含数据
JSON_DEPTH	获取 JSON 文档的最大深度
JSON_EXTRACT	从 JSON 文档返回数据
JSON_INSERT	将数据插入 JSON 文档
JSON_KEYS	获取 JSON 文档的键组成的数组
JSON_LENGTH	获取接送文档中的元素个数
JSON_MERGE	合并 JSON 文档，重复键值合成数组
JSON_MERGE_PATCH	合并 JSON 文档，替换重复键
JSON_MERGE_PRESERVE	合并 JSON 文档，重复键值合成数组
JSON_OBJECT	构建 JSON 对象
JSON_OVERLAPS	比较两个 JSON 文档，如果他们有任何共同的键值对或数组元素，则返回 1，否则返回 0
JSON_PRETTY	格式化文档
	接下页

函数形式	功能
JSON_QUOTE	引用 JSON 文档
JSON_REMOVE	从 JSON 文档中删除数据
JSON_REPLACE	替换 JSON 文档中的值
JSON_SCHEMA_VALID	根据 JSON 模式验证 JSON 文档，成功返回 1，失败返回 0
JSON_SCHEMA_VALIDATION_REPORT	根据 JSON 模式验证 JSON 文档，返回验证报告
JSON_SEARCH	返回给定值在 JSON 文档中的路径
JSON_SET	将数据插入 JSON 文档
JSON_TYPE	返回 JSON 值的类型
JSON_UNQUOTE	取消引用 JSON 文档
JSON_VALID	验证参数是否为有效的 JSON 值

7.2 JSON_ARRAY

功能描述

构建 JSON 数组，返回一个包含了所有参数的 JSON 数组。

语法格式

```
JSON_ARRAY( [val[, val]... )
```

参数说明

0 个或多个，类型取值为 NULL、布尔型、数值型、自定义类型（OBJECT、VARRAY、TABLE）、JSON 类型以及其他能转换成字符型的类型。

函数返回类型

JSON 数值类型。

示例

创建一个包含四个元素的 JSON 数组。

```
SQL> SELECT TO_CHAR(JSON_ARRAY(1, true, null, 'null'));

EXPR1 |
------------------------------------------------------------
[1, true, null, "null"]|
```

创建一个包含多种数据类型的 JSON 数组。

```
SQL> SELECT TO_CHAR(JSON_ARRAY(123, 'abc', NULL, TRUE, FALSE, NOW()
   ));

EXPR1 |
------------------------------------------------------------------
[123, "abc", null, true, false, "2024-04-07 10:03:00"]|
```

创建一个包含数组的 JSON 数组。

```
-- 包含数组的数组
SQL> SELECT TO_CHAR(JSON_ARRAY(123, 456)),TO_CHAR(JSON_ARRAY('abc'
   , 'dec'));

EXPR1 | EXPR2 |
------------------------------------------------------------
[123, 456]| ["abc", "dec"]|
```

7.3 JSON_ARRAY_APPEND

功能描述

添加元素至 JSON 文档指定路径末尾。

语法格式

```
JSON_ARRAY_APPEND( json_doc, path, val[, path, val]... )
```

参数说明

JSON 文档，多个路径 + 多个新值。

- json_doc：JSON 文本，JSON 类型或可转换为 JSON 类型的类型。
- path：JSON 路径表达式，字符类型。
- val：添加的新值，类型取值为（NULL、布尔型、数值型、自定义类型（OBJECT、VARRAY、TABLE）、JSON 类型以及其他能转换成字符型的类型）。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 从左到有计算新值，产生的新值用于后续计算。
- 如果路径选择数组元素，则新值插入到该位置，并旧址与后续元素向后移动。
- 未查找到任何值的路径将被忽略。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式。
  - 路径表达式中包含 * 或 ** 通配符。
  - 路径选择不是数组元素。

函数返回类型

JSON 数值类型。

示例

在数组每个位置追加元素。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_APPEND('["a", ["b", "c"], "d"]', '$
   [0]', 1, '$[1]', 2, '$[3]', 3));

EXPR1 |
------------------------------------------------------------
[["a", 1], ["b", "c", 2], "d"]|
```

在数组末尾追加元素。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_APPEND('[1, 2, 3]', '$', 4));

EXPR1 |
------------------------------------------------------------
[1, 2, 3, 4]|
```

向内嵌数组中追加元素。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_APPEND('[1, [2, 3]]', '$[1]', 4));

EXPR1 |
------------------------------------------------------------
[1, [2, 3, 4]]|
```

向非数组中追加值。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_APPEND('1', '$', 2));

EXPR1 |
------------------------------------------------------------
[1, 2]|
```

7.4 JSON_ARRAY_INSERT

功能描述

插入元素至 JSON 文档指定路径数组位置。

语法格式

```
JSON_ARRAY_INSERT(json_doc, path, val[, path, val]...)
```

参数说明

JSON 文档，多个路径 + 多个新值。

- json_doc：JSON 文本，JSON 类型或可转换为 JSON 类型的类型。
- path：JSON 路径表达式，字符类型。
- val：添加的新值，类型取值为（NULL、布尔型、数值型、自定义类型（OBJECT、VARRAY、TABLE）、JSON 类型以及其他能转换成字符型的类型）。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 从左到有计算新值，产生的新值用于后续计算。
- 如果路径选择数组元素，则新值插入到该位置，并旧址与后续元素向后移动。
- 未查找到任何值的路径将被忽略。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式。
  - 路径表达式中包含 * 或 ** 通配符。
  - 路径选择不是数组元素。

函数返回类型

JSON 数值类型。

示例

在数组中指定位置插入元素。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_INSERT('[1, [2, 3], {"a": [4, 5]}]'
   , '$[0]', 0));

EXPR1 |
------------------------------------------------------------
[0, 1, [2, 3], {"a": [4, 5]}]|
```

```
SQL> SELECT TO_CHAR(JSON_ARRAY_INSERT('["a", ["b", "c"], "d"]', '$
   [0]', 1, '$[1]', 2, '$[3]', 3));

EXPR1 |
------------------------------------------------------------
[1, 2, "a", 3, ["b", "c"], "d"]|
```

向内嵌数组中插入元素。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_INSERT('[1, [2, 3], {"a": [4, 5]}]'
   , '$[1][0]', 'x'));

EXPR1 |
------------------------------------------------------------
[1, ["x", 2, 3], {"a": [4, 5]}]|
```

向对象中的数组中插入元素。

```
SQL> SELECT TO_CHAR(JSON_ARRAY_INSERT('[1, [2, 3], {"a": [4, 5]}]'
   , '$[2].a[0]', 'x'));

EXPR1 |
------------------------------------------------------------
[1, [2, 3], {"a": ["x", 4, 5]}]|
```

7.5 JSON_CONTAINS

功能描述

搜索 json_doc 是否存在于 json_doc_target 中，当 path 存在时，则判断是否存在于指定的路径下。

语法格式

```
JSON_CONTAINS( json_doc_target, json_doc[, path] )
```

参数说明

- json_doc_target：JSON 目标文档，JSON 类型或 JSON String 类型。
- json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。
- path：（可选）JSON 路径表达式，字符类型。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 当对比值为数组时，数组元素都包含在目标文档的某个元素中时，返回 1。
- 当对比值为非数组时，非数组值包含在目标文档中时，返回 1。
- 当对比值为对象时，对象的键值对在目标中存在有相同的键名和值，返回 1。

函数返回类型

INTEGER 数值类型。

示例

不指定路径。

```
SQL> SELECT
TO_CHAR(JSON_CONTAINS('[1, 2, {"x": 3}]', '1')),
TO_CHAR(JSON_CONTAINS('[1, 2, {"x": 3}]', '{"x": 3}')),
TO_CHAR(JSON_CONTAINS('[1, 2, {"x": 3}]', '3'));

EXPR1 | EXPR2 | EXPR3 |
-----------------------------------------------------------------
1| 1| 0|
```

指定路径。

```
SQL> SELECT
TO_CHAR(JSON_CONTAINS('[1, 2, [3, 4]]', '2', '$[2]')),
TO_CHAR(JSON_CONTAINS('[1, 2, [3, 4]]', '2', '$[1]'));

EXPR1 | EXPR2 |
-----------------------------------------------------------------
0| 1|
```

7.6 JSON_CONTAINS_PATH

功能描述

搜索 json_doc 是否在给定的路径下存在数据。

语法格式

```
JSON_CONTAINS_PATH( json_doc, one_or_all, path[, path]... )
```

参数说明

- json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。
- one_or_all：one 或者 all，字符类型。

- one：至少一个 path 存在数据，返回 1。
- all: 所有 path 都存在数据，返回 1。

- path：一个或多个，JSON 路径表达式，字符类型。

函数返回类型

INTEGER 数值类型。

示例

基本使用。

```
SQL> SELECT
TO_CHAR(JSON_CONTAINS_PATH('[1, 2, {"x": 3}]', 'all', '$[0]')) as `
   $[0]`,
TO_CHAR(JSON_CONTAINS_PATH('[1, 2, {"x": 3}]', 'all', '$[3]')) as `
   $[3]`,
TO_CHAR(JSON_CONTAINS_PATH('[1, 2, {"x": 3}]', 'all', '$[2].x')) as
    `$[2].x`;

$[0] | $[3] | $[2]x |
-----------------------------------------------------------
1| 0| 1|
```

one 与 all 对比。

```
SQL> SELECT
TO_CHAR(JSON_CONTAINS_PATH('[1, 2, {"x": 3}]', 'one', '$[0]', '$[3]
   ')) as `one`,
TO_CHAR(JSON_CONTAINS_PATH('[1, 2, {"x": 3}]', 'all', '$[0]', '$[3]
   ')) as `all`;

one | all |
-----------------------------------------------------------
1| 0|
```

7.7 JSON_DEPTH

功能描述

返回 JSON 文档最大深度。

语法格式

```
JSON_DEPTH( json_doc, one_or_all, path[, path]... )
```

参数说明

json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。

> **⚠ 注意**
>
> 空数组、空对象或标量值的深度为 1。包含深度为 1 元素的非空数组，或仅包含深度为 1 成员的分控对象，深度为 2。否则，json_doc 深度大于 2。

函数返回类型

INTEGER 数值类型。

示例

空数组、空对象或标量值。

```
SQL> SELECT TO_CHAR(JSON_DEPTH('{}')), TO_CHAR(JSON_DEPTH('[]')),
   TO_CHAR(JSON_DEPTH('1'));

EXPR1 | EXPR2 | EXPR3 |
-----------------------------------------------------------------
1| 1| 1|
```

其他 JSON 数组。

- 包含深度为 1 元素的非空数组，或仅包含深度为 1 成员的分控对象。

```
SQL> SELECT TO_CHAR(JSON_DEPTH('[1, 2]')), TO_CHAR(JSON_DEPTH('{"
   x": 1}'));

EXPR1 | EXPR2 |
-----------------------------------------------------------------
2| 2|
```

- 包含深度超过 1 的元素的数组，所有成员的值的深度超过 1 的对象。

```
SELECT TO_CHAR(JSON_DEPTH('[1, [2, 3]]')),  TO_CHAR(JSON_DEPTH('
   {"x": {"y": 1}}')),
TO_CHAR(JSON_DEPTH('{"x": {"y": {"z": 1}}}'));

EXPR1 | EXPR2 | EXPR3 |
-----------------------------------------------------------------
3| 3| 4|
```

7.8 JSON_EXTRACT

功能描述

返回 JSON 文档指定路径的值。

语法格式

```
JSON_EXTRACT( json_doc, path[, path]... )
```

参数说明

- json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。
- path：一个或多个路径表达式，字符类型。

> **注意**
>
> 多个路径返回值合并为 JSON 数组。

函数返回类型

JSON 数值类型。

示例

从数组中提取一个元素。

```
SQL> SELECT TO_CHAR(JSON_EXTRACT('[1, 2, {"x": 3}]', '$[2]'));

EXPR1 |
------------------------------------------------------------
{"x": 3}|
```

从对象中提取一个节点的值。

```
SQL> SELECT TO_CHAR(JSON_EXTRACT('{"x": 1, "y": [1, 2]}', '$.y'));

EXPR1 |
------------------------------------------------------------
[1, 2]|
```

提取所有的节点的值。

```
SQL> SELECT TO_CHAR(JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}'
   , '$.*'));

EXPR1 |
------------------------------------------------------------
[1, 2, [3, 4, 5]]|
```

7.9 JSON_INSERT

功能描述

将数据插入 JSON 文档。

语法格式

```
JSON_INSERT( json_doc, path[, path]... )
```

参数说明

- json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。
- path：路径表达式，字符类型。
- val：val 类型取值为（NULL、布尔型、数值型、自定义类型（OBJECT、VARRAY、TABLE）、JSON 类型以及其他能转换成字符型的类型）。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 从左到有计算新值，产生的新值用于后续计算。
- 文档中 path 存在值时，该路径值对将被忽略。
- 若路径表述为对象键，且并不存在于 JSON 文本的指定对象中，则该路径值对将被作为新成员加入对应对象。
- 若路径表述为数组下标，且处于 JSON 文本指定数组末尾之后，数组使用新值扩展。若 JSON 文本路径指定不为数组，则将其包装成数组，然后使用新值扩展。
- 除了上述两类不存在的路径的路径值对都将被忽略。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式。
  - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

JSON 数值类型。

示例

插入到数组。

```
SQL> SELECT TO_CHAR(JSON_INSERT('[1, [2, 3], {"a": [4, 5]}]', '$[0]
   ', 0, '$[3]', 6));

EXPR1 |
------------------------------------------------------------------
[1, [2, 3], {"a": [4, 5]}, 6]|
```

插入 JSON 类型数据。

```
SQL> SELECT TO_CHAR(JSON_INSERT('{"x": 1}', '$.y', 'true'));

EXPR1 |
------------------------------------------------------------------
{"x": 1, "y": "true"}|
```

```
SQL> SELECT TO_CHAR(JSON_INSERT('{"x": 1}', '$.y', CAST('{"z": 2}'
   AS JSON)));

EXPR1 |
------------------------------------------------------------
{"x": 1, "y": {"z": 2}}|
```

7.10 JSON_KEYS

功能描述

从 JSON 对象的顶极值中返回对象键的 JSON 数组。

语法格式

```
JSON_KEYS( json_doc[, path] )
```

参数说明

- json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。
- path：路径表达式，字符类型。

> ⚠ 注意
>
> - 如果任何参数为 NULL 则返回 NULL。
> - 从左到有计算新值，产生的新值用于后续计算。
> - 如果所选对象为空则返回空数组。
> - 若路径为不存在则返回 NULL。
> - 以下情况将导致系统抛出错误：
>   - json_doc 是无效的 JSON 文档。
>   - 任何 path 是无效的路径表达式。
>   - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

JSON 数值类型。

示例

无路径表达式，返回一个 JSON 对象的所有顶层成员组成的数组。

```
SQL> SELECT TO_CHAR(JSON_KEYS('{"x": 1, "y": 2, "z": 3}'));

EXPR1 |
------------------------------------------------------------
["x", "y", "z"]|
```

有路径表达式，返回有路径表达式匹配的 JSON 对象的键。

```
SQL> SELECT TO_CHAR(JSON_KEYS('[0, {"x": 1, "y": 2, "z": 3}]', '$
   [1]'));

EXPR1 |
------------------------------------------------------------
["x", "y", "z"]|
```

非 JSON 对象，如果匹配的 JSON 文档不是 JSON 对象，JSON_KEYS() 返回 NULL。

```
SQL> SELECT
TO_CHAR(JSON_KEYS('1')),
TO_CHAR(JSON_KEYS('true')),
TO_CHAR(JSON_KEYS('"hello"')),
TO_CHAR(JSON_KEYS('[1, 2]')),
TO_CHAR(JSON_KEYS('[0, {"x": 1, "y": 2, "z": 3}]'));

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 |
------------------------------------------------------------
<NULL>| <NULL>| <NULL>| <NULL>| <NULL>|
```

7.11 JSON_LENGTH

功能描述

JSON 对象的长度。

语法格式

```
JSON_LENGTH( json_doc[, path] )
```

参数说明

- json_doc：JSON 比对文本，JSON 类型或 JSON String 类型。
- path：路径表达式，字符类型。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 标量长度为 1。
- 数组长度为元素数量。
- 对象长度为对象成员数量。
- 长度不计算嵌套数组或对象长度。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式。
  - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

JSON 数值类型。

示例

值的长度。

```
SQL> SELECT TO_CHAR(JSON_LENGTH('1')),
TO_CHAR(JSON_LENGTH('true')),
TO_CHAR(JSON_LENGTH('false')),
TO_CHAR(JSON_LENGTH('"abc"'));

EXPR1 | EXPR2 | EXPR3 | EXPR4 |
-------------------------------------------------------------
1| 1| 1| 1|
```

数组的长度。

```
SQL> SELECT
TO_CHAR(JSON_LENGTH('[]')),
TO_CHAR(JSON_LENGTH('[1, 2]')),
TO_CHAR(JSON_LENGTH('[1, {"x": 2}]'));

EXPR1 | EXPR2 | EXPR3 |
-------------------------------------------------------------
0| 2| 2|
```

对象的长度。

```
SQL> SELECT
TO_CHAR(JSON_LENGTH('{"x": 1, "y": 2}')),
TO_CHAR(JSON_LENGTH('{"x": 1, "y": {"z" : 2}}'));

EXPR1 | EXPR2 |
-------------------------------------------------------------
2| 2|
```

路径。

```
SQL> SELECT TO_CHAR(JSON_LENGTH('{"x": 1, "y": [1, 2]}', '$.y'));

EXPR1 |
------------------------------------------------------------
2|
```

7.12 JSON_MERGE

功能描述

合并 JSON 文本。

语法格式

```
JSON_MERGE( json_doc, json_doc[, json_doc]... )
```

参数说明

json_doc：两个以上 JSON 文本，JSON 类型或 JSON String 类型。

> ⚠ 注意
>
> - 如果任何参数为 NULL 则返回 NULL。
> - 两个数组合并为一个数组。
> - 两个对象合并为一个对象。
> - 标量值将被包装为数组并合并为数组。
> - 数组、对象合并，将对象合并到数组。
> - json_doc 不为有效 JSON 文本将抛出错误。

函数返回类型

JSON 数值类型。

示例

合并两个或多个 JSON 数组。

```
SQL> SELECT TO_CHAR(JSON_MERGE('[1, 2]', '[2, 3]', '[3, 4, 5]'));

EXPR1 |
------------------------------------------------------------
[1, 2, 2, 3, 3, 4, 5]|
```

合并两个或多个 JSON 对象。

```
SQL> SELECT TO_CHAR(JSON_MERGE('{"x": 1}', '{"x": 2, "y": 3}'));
```

```
EXPR1 |
------------------------------------------------------------
{"x": [1, 2], "y": 3}|
```

合并纯值。

```
SQL> SELECT TO_CHAR(JSON_MERGE('1', 'true', '"hello"', 'null'));

EXPR1 |
------------------------------------------------------------
[1, true, "hello", null]|
```

合并数组和对象。

```
SQL> SELECT TO_CHAR(JSON_MERGE('{"x": 1}', '[1, 2]'));

EXPR1 |
------------------------------------------------------------
[{"x": 1}, 1, 2]|
```

7.13 JSON_MERGE_PATCH

功能描述

合并 JSON 文本。

语法格式

```
JSON_MERGE_PATCH( json_doc, json_doc[, json_doc]... )
```

参数说明

json_doc：两个以上 JSON 文本，JSON 类型或 JSON String 类型。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 若第一个参数不是对象，合并结果就是第二个参数。
- 若两个参数都为对象，则遵守以下规则：
  - 第一个对象的所有成员键不存在于第二个对象中。
  - 第二个对象的所有成员键不存在于第一个对象中且值不为 null。否则该键将被在结果中忽略。
  - 所有成员键都存在于第一、第二个对象中且第二个对象中的值不为 null。否则该键将被在结果中忽略。
  - json_doc 不为有效 JSON 文本将抛出错误。

函数返回类型

JSON 数值类型。

示例

非 JSON 对象类型合并。

```
SQL> SELECT
TO_CHAR(JSON_MERGE_PATCH('2', 'true')),
TO_CHAR(JSON_MERGE_PATCH('true', '2')),
TO_CHAR(JSON_MERGE_PATCH('[1, 2]', '2')),
TO_CHAR(JSON_MERGE_PATCH('2', '[1, 2]')),
TO_CHAR(JSON_MERGE_PATCH('[1, 2]', '[2, 3]'));

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 |
-------------------------------------------------------------------
true| 2| 2| [1, 2]| [2, 3]|
```

合并两个或多个 JSON 对象。

- 覆盖值：如果第二个对象中有与第一个对象相同的键（x），则第二个对象的值会覆盖第一个对象的值。

```
SQL> SELECT TO_CHAR(JSON_MERGE_PATCH('{"x": 1, "z": 7}', '{"x
   ": 2, "y": 3}'));

EXPR1 |
-------------------------------------------------------------------
{"x": 2, "y": 3, "z": 7}|
```

- 添加新键：如果第二个对象中有第一个对象中不存在的键，则这些键会被添加到结果对象中。

```
SQL> SELECT TO_CHAR(JSON_MERGE_PATCH('{"a": null, "b": 1, "e": 5}
   ', '{"c": 2, "f": 6}'));

EXPR1 |
-------------------------------------------------------------------
{"a": null, "b": 1, "c": 2, "e": 5, "f": 6}|
```

- 删除键：如果第二个对象中的值为 null，则相应的键会被从结果对象中删除。

```
SQL> SELECT TO_CHAR(JSON_MERGE_PATCH('{"x": 1, "z": 7}', '{"x
   ": 2, "z": null}'));

EXPR1 |
-------------------------------------------------------------------
{"x": 2}|
```

7.14 JSON_MERGE_PRESERVE

功能描述

合并 JSON 文本。

语法格式

```
JSON_MERGE_PRESERVE( json_doc, json_doc[, json_doc]... )
```

参数说明

json_doc：两个以上 JSON 文本，JSON 类型或 JSON String 类型。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 两个数组合并为一个数组。
- 两个对象合并为一个对象。
- 标量值将被包装为数组并合并为数组。
- 数组、对象合并，将对象合并到数组。
- json_doc 不为有效 JSON 文本将抛出错误。

函数返回类型

JSON 数值类型。

示例

合并数组。

```
SQL> SELECT TO_CHAR(JSON_MERGE_PRESERVE('[1, 2]', '[2, 3]', '
   [3, 4, 5]'));

EXPR1 |
------------------------------------------------------------
[1, 2, 2, 3, 3, 4, 5]|
```

合并对象。

```
    SQL> SELECT TO_CHAR(JSON_MERGE_PRESERVE('[1, 2]', '[2, 3]', '
       [3, 4, 5]'));

    EXPR1 |
    ------------------------------------------------------------
    [1, 2, 2, 3, 3, 4, 5]|
```

合并数组和对象。

```
SQL> SELECT TO_CHAR(JSON_MERGE_PRESERVE('{"x": 1}', '[1, 2]'));

EXPR1 |
------------------------------------------------------------
[{"x": 1}, 1, 2]|
```

7.15 JSON_OBJECT

功能描述

构建 JSON 对象。

语法格式

```
JSON_OBJECT( [key, val[, key, val]...] )
```

参数说明

0 个或多个键值对，key 必须为可转换为字符型的类型，val 类型取值为（NULL、布尔型、数值型、自定义类型（object、varray、table）、JSON 类型以及其他能转换成字符型的类型）。

> ⚠ 注意
>
> NULL 对应 json null，'null' 对应 json string。

函数返回类型

JSON 数值类型。

示例

常规使用。

```
SQL> SELECT TO_CHAR(JSON_OBJECT('key1', true, 1, 1));

EXPR1 |
--
   ----------------------------------------------------------------------

{"1": 1, "key1": true}|

SQL> SELECT TO_CHAR(JSON_OBJECT('name', 'Jim', 'age', 20));

EXPR1 |
------------------------------------------------------------
{"age": 20, "name": "Jim"}|
```

重复的键，如果 JSON_OBJECT() 的参数中出现了重复的键值对，那么后面的键值对保留在最终返回的对象中。

```
SQL> SELECT TO_CHAR(JSON_OBJECT('name', 'Jim', 'age', 20, 'name', '
   Tim'));

EXPR1 |
------------------------------------------------------------
{"age": 20, "name": "Tim"}|
```

包含复杂的对象。

```
SQL> SELECT TO_CHAR(JSON_OBJECT(
'name',
'Tim',
'age',
20,
'friend',
TO_CHAR(JSON_OBJECT('name', 'Jim', 'age', 20)),
'hobby',
TO_CHAR(JSON_ARRAY('games', 'sports'))
)) AS object;

OBJECT |
-------------------------------------------------------------
{"age": 20, "name": "Tim", "hobby": "[\"games\", \"sports\"]", "
   friend": "{\"age\": 20, \"name\": \"Jim\"}"}|
```

上述 SQL，我们组装了如下 JSON 对象：

```
{
    "age": 20,
    "name": "Tim",
    "hobby": ["games", "sports"],
    "friend": { "age": 20, "name": "Jim" }
}
```

7.16 JSON_OVERLAPS

功能描述

比较两个 JSON 文档，检查两个 JSON 文档是否拥有任何一个相同键值对或数组元素。

语法格式

```
JSON_OVERLAPS( json_doc1, json_doc2 )
```

参数说明

两个 JSON 文本，JSON 类型或 JSON String 类型。

⚠ 注意

- 如果任何参数为 NULL 则返回 NULL。
- 如果两个 JSON 文档有任何共同的键值对或数组元素，则返回 1。
- 如果两个 JSON 文档为标量，则对比两值，相等返回 1。
- json_doc 不为有效 JSON 文本将抛出错误。

函数返回类型

INTEGER 数值类型。

示例

比较数组。

```
SELECT TO_CHAR(JSON_OVERLAPS('[1, 2, 3]', '[3, 4, 5]'));

EXPR1 |
------------------------------------------------------------
1|

SQL> SELECT TO_CHAR(JSON_OVERLAPS('[1, 2, [3]]', '[3, 4, 5]'));

EXPR1 |
------------------------------------------------------------
0|
```

比较对象。

```
SQL> SELECT
TO_CHAR(JSON_OVERLAPS('{"x": 1}', '{"x": 1, "y": 2}')),
TO_CHAR(JSON_OVERLAPS('{"x": 1}', '{"y": 2}'));

EXPR1 | EXPR2 |
------------------------------------------------------------
1| 0|
```

比较纯值和数组。

```
SQL> SELECT
TO_CHAR(JSON_OVERLAPS('[1, 2, 3]', '3')),
TO_CHAR(JSON_OVERLAPS('[1, 2, [3]]', '3'));

EXPR1 | EXPR2 |
------------------------------------------------------------
1| 0|
```

比较纯值。

```
SQL> SELECT TO_CHAR(JSON_OVERLAPS('1', '1')), TO_CHAR(JSON_OVERLAPS
   ('1', '"1"'));

EXPR1 | EXPR2 |
------------------------------------------------------------
1| 0|
```

7.17 JSON_PRETTY

功能描述

格式化 JSON 文本。

语法格式

```
JSON_PRETTY( json_doc )
```

参数说明

JSON 文本，JSON 类型或 JSON String 类型。

函数返回类型

JSON 数值类型

示例

格式化输出数组。

```
SQL> SELECT TO_CHAR(JSON_PRETTY('[1,3,4,5]'));

EXPR1 |
------------------------------------------------------------------
[
1,
3,
4,
5
]|
```

格式化输出对象。

```
SQL> SELECT TO_CHAR(JSON_PRETTY('{"x": 1, "y": 2}'));

EXPR1 |
------------------------------------------------------------------
{
    "x": 1,
    "y": 2
}|
```

格式化输出复杂对象。

```
SQL> SELECT TO_CHAR(JSON_PRETTY('{"x": 1, "y": [1, 2, 3], "z": {"a
   ": "a", "b": true}}'));

EXPR1 |
------------------------------------------------------------------
{
    "x": 1,
    "y": [
    1,
    2,
    3
    ],
    "z": {
        "a": "a",
        "b": true
    }
}|
```

7.18 JSON_QUOTE

功能描述

将字符串转换为有效的 JSON 字符串。

语法格式

```
JSON_QUOTE( string )
```

参数说明

CHAR 类型。

函数返回类型

CHAR 类型。

示例

数组转化为 JSON 格式。

```
SQL> SELECT TO_CHAR(JSON_QUOTE('[1,3,4,5]'));

EXPR1 |
------------------------------------------------------------
"[1,3,4,5]"|
```

数值、NULL 转化为 JSON 格式。

```
SQL> SELECT
TO_CHAR(JSON_QUOTE('123')),
TO_CHAR(JSON_QUOTE('NULL'));

EXPR1 | EXPR2 |
------------------------------------------------------------
"123"| "NULL"|
```

7.19 JSON_REMOVE

功能描述

从 JSON 文档中将指定路径下的数据移除。

语法格式

```
JSON_REMOVE( json_doc, path[, path]... )
```

参数说明

- json_doc：json 文本，JSON 类型或 JSON String 类型。
- path：路径表达式，一个或多个。

⚠ 注意

- 如果任意参数为 NULL 返回 NULL。
- 若路径不存在值，则被忽略。
- 从左到右依次计算路径并删除元素，并将前一次删除后的结果作为下一次计算的输入。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式或是 $。
  - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

JSON 数值类型。

示例

从数组中删除。

下面语句使用 JSON_REMOVE() 从一个 JSON 数组中删除索引为 0 和 2 的元素。先删除索引 [0] 的值，将删除索引 [0] 后的结果 [1,2,[3,4]] 作为删除索引 [2] 的输入再处理。

```
SQL> SELECT TO_CHAR(JSON_REMOVE('[0, 1, 2, [3, 4]]', '$[0]', '$[2]'
   ));

EXPR1 |
------------------------------------------------------------------
[1, 2]|
```

从对象中删除。

下面语句使用 JSON_REMOVE() 从一个 JSON 对象中删除一个成员 x。

```
SQL> SELECT TO_CHAR(JSON_REMOVE('{"x": 1, "y": 2}', '$.x'));

EXPR1 |
------------------------------------------------------------------
{"y": 2}|
```

7.20 JSON_REPLACE

功能描述

从 JSON 文档中将指定路径下的数据替换。

语法格式

```
JSON_REPLACE( json_doc, path, val[, path, val]... )
```

参数说明

- json_doc：json 文本，JSON 类型或 JSON String 类型。
- path：路径表达式，一个或多个。
- val：新值，类型取值为（NULL、布尔型、数值型、自定义类型（object、varray、table）、JSON 类型以及其他能转换成字符型的类型）。

> **⚠ 注意**
>
> - 如果任意参数为 NULL 返回 NULL。
> - 从左到右依次计算路径并删除元素，并将前一次删除后的结果作为下一次计算的输入。
> - 对路径存在值的情况，使用新值覆盖原有值，若路径不存在值，则路径与值被忽略。
> - 以下情况将导致系统抛出错误：
>   - json_doc 是无效的 JSON 文档。
>   - 任何 path 是无效的路径表达式或是 $。
>   - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

CHAR 数值类型。

示例

在数组中替换。将数组的第一个元素和第三个元素替换为新值。

```
SQL> SELECT TO_CHAR(JSON_REPLACE('[1, [2, 3]]', '$[0]', 0, '$[2]'
   , 6));

EXPR1 |
------------------------------------------------------------
[0, [2, 3]]|
```

写入 JSON 类型数据。

```
SQL> SELECT TO_CHAR(JSON_REPLACE('{"x": 1}', '$.x', 'true'));

EXPR1 |
------------------------------------------------------------
{"x": "true"}|


SQL> SELECT TO_CHAR(JSON_REPLACE('{"x": 1}', '$.x', '[1, 2]'));

EXPR1 |
------------------------------------------------------------
{"x": "[1, 2]"}|

SQL> SELECT TO_CHAR(JSON_REPLACE('{"x": 1}', '$.x', CAST('{"z": 2}'
    AS JSON)));
```

```
EXPR1 |
------------------------------------------------------------
{"x": {"z": 2}}|
```

7.21 JSON_SCHEMA_VALID

功能描述

验证 json_doc 是否符合 json_doc_schema，并返回 1 表示验证通过或者返回 0 表示验证不通过。

语法格式

```
JSON_SCHEMA_VALID( json_doc_schema, json_doc )
```

参数说明

- json_doc_schema：json schema 文本，json 或 json string。
- json_doc：json 文本，JSON 类型或 JSON String 类型。

> **⚠ 注意**
>
> - 如果任意参数为 NULL 返回 NULL。
> - 对 json_doc 根据模式验证，通过返回 1。

函数返回类型

INTEGER 数值类型。

示例

json_doc_schema：

```
{
    "type": "object",
    "properties": {
        "id": {
            "type": "number",
            "minimum": 10,
            "maximum": 100
        },
        "num": {
            "type": "number",
            "minimum": 10,
            "maximum": 100
        }
    },
    "required": ["id", "num"]
}
```

json_doc：

```
{
    "id": 100,
    "num": 100
}
```

常规验证。

```
SQL> SELECT TO_CHAR(JSON_SCHEMA_VALID('{"type": "object", "
   properties": { "id": {"type": "number", "minimum": 10, "maximum
   ": 100},
␣␣␣␣␣␣␣␣"num": {"type": "number", "minimum": 10, "maximum
   ": 100} }, "required": ["id", "num"] }', '{"id": 100, "num": 100}
   '));

EXPR1 |
------------------------------------------------------------
1|
```

json_doc 缺少 num。

JSON 模式中定义了 num 是必须的成员，如下 SQL 缺少 num 所以 JSON_SCHEMA_VALID() 函数返回了 0。

```
SQL> SELECT TO_CHAR(JSON_SCHEMA_VALID('{"type": "object", "
   properties": { "id": {"type": "number", "minimum": 10, "maximum
   ": 100},
␣␣␣␣␣␣␣␣"num": {"type": "number", "minimum": 10, "maximum
   ": 100} }, "required": ["id", "num"] }', '{"id": 100}'));

EXPR1 |
------------------------------------------------------------
0|
```

7.22 JSON_SCHEMA_VALIDATION_REPORT

功能描述

验证 json_doc 是否符合 json_doc_schema 并返回验证报告。

语法格式

```
JSON_SCHEMA_VALIDATION_REPORT( json_doc_schema, json_doc )
```

参数说明

- json_doc_schema：json schema 文本，json 或 json string。
- json_doc：json 文本，JSON 类型或 JSON String 类型。

> **⚠ 注意**
>
> 如果任意参数为 NULL 返回 NULL。

返回报告为如下键以及相应值组成的 JSON 对象：

- valid：验证成功为 true，失败为 false。失败才会有下列键值对。
- reason：失败原因。
- schema-location：schema 的失败位置。
- document-location：json 文本的失败位置。
- schema-failed-keyword：schema 的失败的关键字。

函数返回类型

JSON 数值类型。

示例

json_doc_schema：

```
{
    "type": "object",
    "properties": {
        "id": {
            "type": "number",
            "minimum": 10,
            "maximum": 100
        },
        "num": {
            "type": "number",
            "minimum": 10,
            "maximum": 100
        }
    },
    "required": ["id", "num"]
}
```

json_doc：

```
{
    "id": 100,
    "num": 101
}
```

判断 JSON 文本是否符合 JSON 模式，并返回验证报告。

```
SQL> SELECT TO_CHAR(JSON_SCHEMA_VALIDATION_REPORT('{"type": "object
  ", "properties": { "id": {"type": "number", "minimum": 10, "
  maximum": 100},
```

```
␣␣␣␣␣␣␣␣"num": {"type": "number", "minimum": 10, "maximum
   ": 100} }, "required": ["id", "num"] }', '{"id": 100, "num": 101}
   '));

EXPR1 |
------------------------------------------------------------------
{"valid": false, "reason": "The JSON document location '#/num'
   failed requirement 'maximum' at JSON Schema location '#/
   properties/num'",
    "schema-location": "#/properties/num", "document-location": "#/
       num", "schema-failed-keyword": "maximum"}|
```

报告显示：id 的值为 100，符合 Schema 的要求，但 num 的值为 101，超出了 Schema 定义的最大值 100。

7.23 JSON_SEARCH

功能描述

从 JSON 文档中查找给定字符串的路径表达式。

语法格式

```
JSON_SEARCH( json_doc, one_or_all, search_str[, escape_char[, path
   ]...] )
```

参数说明

- json_doc：json 文本，JSON 类型或 JSON String 类型。
- one_or_all：匹配出现该值的一个路径还是全部路径，字符类型。
- search_str：需要搜索的字符串。字符类型。
- escape_char：转义符，字符类型。
- path：路径表达式，字符类型。

⚠ 注意

- 如果 json_doc 或 search_str 或 path 为 NULL 返回 NULL。
- 对路径存在值的情况，使用新值覆盖原有值，若路径不存在值，则路径与值被忽略。
- one_or_all 参数不为'one' 或者'all' 中的一个，则会抛出错误。
- search_str 可以带% 和 _ 字符做 like 运算。
- escape_char 默认值为\，必须为一个常数，null、空串或一个字符。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式。
  - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

JSON 数值类型。

示例

搜索字符串。

```
SQL> SELECT TO_CHAR(JSON_SEARCH('[{"name": "Tim","age": 20,"hobbies
   ": [{ "name": "Car", "weight": 10 },{ "name": "Sports", "weight
   ": 20 }]}
,{"name": "Tom","age": 20,"hobbies": [{ "name": "Reading", "weight
   ": 10 },{ "name": "Sports", "weight": 20 }]}]', 'one', 'Tim'));

EXPR1 |
------------------------------------------------------------
"$[0].name"|
```

one 与 all 对比。

```
SQL> SELECT
TO_CHAR(JSON_SEARCH('[{"name": "Tim","age": 20,"hobbies": [{ "name
   ": "Car", "weight": 10 },{ "name": "Sports", "weight": 20 }]},
{"name": "Tom","age": 20,"hobbies": [{ "name": "Reading", "weight
   ": 10 },{ "name": "Sports", "weight": 20 }]}]', 'one', 'Sports'))
   ,
TO_CHAR(JSON_SEARCH('[{"name": "Tim","age": 20,"hobbies": [{ "name
   ": "Car", "weight": 10 },{ "name": "Sports", "weight": 20 }]},
{"name": "Tom","age": 20,"hobbies": [{ "name": "Reading", "weight
   ": 10 },{ "name": "Sports", "weight": 20 }]}]', 'all', 'Sports'))
   ;

EXPR1 | EXPR2 |
------------------------------------------------------------------
"$[0].hobbies[1].name"| ["$[0].hobbies[1].name", "$[1].hobbies[1].
   name"]|
```

使用通配符。

```
SQL> SELECT  TO_CHAR(JSON_SEARCH('[{"name": "Tim","age": 20,"
   hobbies": [{ "name": "Car", "weight": 10 },{ "name": "Sports", "
   weight": 20 }]},
{"name": "Tom","age": 20,"hobbies": [{ "name": "Reading", "weight
   ": 10 },{ "name": "Sports", "weight": 20 }]}]', 'all', 'S%'));

EXPR1 |
------------------------------------------------------------------
["$[0].hobbies[1].name", "$[1].hobbies[1].name"]|
```

7.24 JSON_SET

功能描述

替换或插入新值，在一个 JSON 文档中插入或更新数据并返回新的 JSON 文档。

语法格式

```
JSON_SET( json_doc, path, val[, path, val]... )
```

参数说明

- json_doc：json 文本，JSON 类型或 JSON String 类型。
- path：路径表达式，字符类型。
- val：新值，类型取值为（NULL、布尔型、数值型、自定义类型（object、varray、table）、JSON 类型以及其他能转换成字符型的类型）。

⚠ 注意

- 如果任意参数为 NULL 返回 NULL。
- 从左到右计算新值，产生的新值用于后续计算。
- 文档中 path 存在值时，则使用新值替换。
- 若路径表述为对象键，且并不存在于 JSON 文本的指定对象中，则该路径值对将被作为新成员加入对应对象。
- 若路径表述为数组下标，且处于 JSON 文本指定数组末尾之后，数组使用新值扩展。若 JSON 文本路径指定不为数组，则将其包装成数组，然后使用新值扩展。
- 除了上述两类不存在的路径的路径值对都将被忽略。
- 以下情况将导致系统抛出错误：
  - json_doc 是无效的 JSON 文档。
  - 任何 path 是无效的路径表达式。
  - 路径表达式中包含 * 或 ** 通配符。

函数返回类型

JSON 数值类型。

示例

数组格式。

```
SQL> SELECT TO_CHAR(JSON_SET('{"x": 1}', '$.x', '10', '$.y', '
   [1, 2]'));

EXPR1 |
------------------------------------------------------------
{"x": "10", "y": "[1, 2]"}|
```

JSON 格式。

```
SQL> SELECT TO_CHAR(JSON_SET('{"x": 1}', '$.x', '10', '$.y', '{"z
   ": 2}'));

EXPR1 |
------------------------------------------------------------
{"x": "10", "y": "{\"z\": 2}"}|
```

数组与 JSON 格式混合使用。

```
SQL> SELECT TO_CHAR(JSON_SET('[1, {"a": 2}]', '$[0]', 10, '$[1].b'
   , 11, '$[3]', 12));

EXPR1 |
------------------------------------------------------------
[10, {"a": 2, "b": 11}, 12]|
```

数组通过 CAST 转化为 JSON 格式。

```
SQL> SELECT TO_CHAR(JSON_SET('{"x": 1}', '$.x', '10', '$.y', CAST('
   [1, 2]' AS JSON)));

EXPR1 |
------------------------------------------------------------
{"x": "10", "y": [1, 2]}|
```

7.25 JSON_TYPE

功能描述

判断 JSON 文本类型。

语法格式

```
JSON_TYPE( json_doc )
```

参数说明

json_doc：json 文本，JSON 类型或 JSON String 类型。

> ⚠ 注意
>
> 返回值为 OBJECT/ARRAY/INTEGER/DOUBLE/BOOLEAN/STRING/NULL。

函数返回类型

CHAR 数值类型。

示例

数字格式。

```
SQL> SELECT TO_CHAR(JSON_TYPE('1')), TO_CHAR(JSON_TYPE('1.23'));

EXPR1 | EXPR2 |
-----------------------------------------------------------------
INTEGER| DOUBLE|
```

数组格式。

```
SQL> SELECT TO_CHAR(JSON_TYPE('[]')), TO_CHAR(JSON_TYPE('[1, 2]'));

EXPR1 | EXPR2 |
-----------------------------------------------------------------
ARRAY| ARRAY|
```

对象。

```
SQL> SELECT TO_CHAR(JSON_TYPE('{}')), TO_CHAR(JSON_TYPE('{"x": 1}')
   );

EXPR1 | EXPR2 |
-----------------------------------------------------------------
OBJECT| OBJECT|
```

更多类型。

```
SQL> SELECT TO_CHAR(JSON_TYPE('true')), TO_CHAR(JSON_TYPE('null'))
   , TO_CHAR(JSON_TYPE('"abc"'));

EXPR1 | EXPR2 | EXPR3 |
-----------------------------------------------------------------
BOOLEAN| NULL| STRING|
```

7.26 JSON_UNQUOTE

功能描述

取消双引号引用 JSON 值，并将结果作为字符串返回。

语法格式

```
JSON_UNQUOTE( json_doc )
```

参数说明

json_doc：json 文本，JSON 类型或 JSON String 类型。

JSON_UNQUOTE 特殊字符

转移	代表字符
”	双引号（”）字符
\b	退格字符
\f	格式提要字符
\n	换行符
\r	回车符
\t	制表符
\\\\	反斜杠
\uXXXX	Unicode 值

函数返回类型

JSON 数值类型。

示例

解析转义字符。

```
SQL> SELECT TO_CHAR(JSON_UNQUOTE('"\u0031\u0032\u0033"'));

EXPR1 |
------------------------------------------------------------
123|
```

返回字符串。

```
SQL> SELECT TO_CHAR(JSON_UNQUOTE('"abc"'));

EXPR1 |
------------------------------------------------------------
abc|
```

去掉 JSON 格式中的双引号。

```
SQL> SELECT TO_CHAR(JSON_UNQUOTE(CAST('"abc"' AS JSON)));

EXPR1 |
------------------------------------------------------------
abc|
```

7.27 JSON_VALID

功能描述

返回 0 或 1 来指示给定的参数是否是一个有效的 JSON 文档。

语法格式

```
JSON_VALID( val )
```

参数说明

VAL 任意类型。

> ⚠ 注意
>
> 只有 CHAR 类型回验证 JSON 格式，其他类型都返回 0。

函数返回类型

INTEGER 数值类型。

示例

带/不带单引号的数值类型。

```
SQL> SELECT JSON_VALID(1), JSON_VALID('1');

EXPR1 | EXPR2 |
------------------------------------------------------------
0 | 1 |
```

带/不带单引号的布尔类型。

```
SQL> SELECT TO_CHAR(JSON_VALID(true)), TO_CHAR(JSON_VALID('true'));

EXPR1 | EXPR2 |
------------------------------------------------------------
0| 1|
```

带/不带双引号的数值类型。

```
SQL> SELECT TO_CHAR(JSON_VALID('abc')), TO_CHAR(JSON_VALID('"abc"')
   );
```

```
EXPR1 | EXPR2 |
-----------------------------------------------------------------
0| 1|
```

8 BIT 数据类型函数

8.1 概述

函数形式	功能
BIT_COUNT	获取 BIT 类型数据中已设置位数，即位值为 1 的位数
BIT_LENGTH	获取 BIT 类型数据的位数
BITTOCHAR	将 BIT 类型数据转换为字符串表示

8.2 BIT_COUNT

功能描述

获取 BIT 类型数据中已设置位数，即位值为 1 的位数。

语法格式

```
BIT_COUNT(expr)
```

参数说明

expr：数据类型为 VARBIT。

函数返回类型

INTEGER 类型。

> **说明**
>
> 若任意参数为 NULL，则返回 NULL。

示例

```
SQL> SELECT BIT_COUNT(b'1010101');

EXPR1 |
------------------------------------------------------------
4 |
```

8.3 BIT_LENGTH

功能描述

获取 BIT 类型数据的位数。

语法格式

```
BIT_LENGTH(expr)
```

参数说明

expr：数据类型为 VARBIT。

函数返回类型

INTEGER 类型。

> **说明**
>
> 若任意参数为 NULL，则返回 NULL。

示例

```
SQL> SELECT BIT_LENGTH(b'1010101');

EXPR1 |
------------------------------------------------------------
7 |
```

8.4 BITTOCHAR

功能描述

将 BIT 类型数据转换为字符串表示，格式为 0、1 组成的字符串。

语法格式

```
BITTOCHAR(expr)
```

参数说明

expr：数据类型为 VARBIT。

函数返回类型

CHAR 类型。

> **说明**
>
> 若任意参数为 NULL，则返回 NULL。

示例

```
SQL> SELECT BITTOCHAR(b'1010101');

EXPR1 |
------------------------------------------------------------------
1010101 |
```

9 XML 数据类型函数

9.1 概述

函数形式	功能
EXTRACT	返回某个标签下的所有标签
EXTRACTVALUE	返回表达式的标签值
XMLCAST	将 XML 节点值转换为字符串
XMLELEMENT	创建一个 XML 标签
XMLEXISTS	判断一个 xpath 表达式中的元素是否存在
XMLFOREST	直接创建一个只有标签名的标签
XMLQUERY	以 xpath 表达式查询 XML 中的元素
XMLSEQUENCE	将非标准的 XML 格式数据转换成数据库表结构
XMLTABLE	将 XML 格式的数据转换为数据库表结构
XMLTYPE	将符合 XML 格式的数据转换为 XML 数据类型

9.2 EXTRACT

功能描述

返回某个标签下的所有标签。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
EXTRACT(XMLdata,xpath)
```

参数说明

- XMLdata：XML 类型数据。

- xpath：查找路径。

函数返回类型

CHAR 类型。

示例

```
SQL> CREATE TABLE test_extract(id INT,xml_col XML,insert_time DATE)
   ;
SQL> insert into test_extract values(1, '<tag><a>AA<b/>aa</a><a
   lang="en">ALPHA<b lang="zh"/>alpha</a><x lang="en">XX<y lang="zh
   ">YY<z/>yy</y></x></tag>', '2020-01-01');
SQL> insert into test_extract values(2, '<country>china<city
   capital="true">beijing</city><city capital="false">shanghai</city
   ></country>', '2020-02-01');
SQL> SELECT EXTRACT(t.xml_col,'/country/city[@capital="true"]')
   FROM test_extract t WHERE t.id = 2;
EXPR1 |
------------------------------------------------------------------
<city capital="true">beijing</city> |

SQL> SELECT EXTRACT(t.xml_col,'data(//@lang)') FROM test_extract t
   WHERE t.id = 1;

EXPR1 |
------------------------------------------------------------------
enzhenzh
```

9.3 EXTRACTVALUE

功能描述

返回表达式的标签值。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
EXTRACTVALUE(XMLdata,xpath)
```

参数说明

- XMLdata：XML 类型数据。
- xpath：查找路径。

函数返回类型

CHAR 类型。

示例

```
SQL> CREATE TABLE test_extractvalue(id INT,xml_col XML,insert_time
   DATE);
```

```
SQL> insert into test_extractvalue values(1, '<tag><a>AA<b/>aa</a><
   a lang="en">ALPHA<b lang="zh"/>alpha</a><x lang="en">XX<y lang="
   zh">YY<z/>yy</y></x></tag>', '2020-01-01');
SQL> insert into test_extractvalue values(2, '<country>china<city
   capital="true">beijing</city><city capital="false">shanghai</city
   ></country>', '2020-02-01');

SQL> SELECT EXTRACTVALUE(t.xml_col,'/country/city[@capital="false"]
   ') AS city FROM test_extractvalue t WHERE t.id = 2;
city |
-----------------------------------------------------------------
shanghai |

SQL> SELECT EXTRACTVALUE(t.xml_col,'/tag//y') AS col FROM
   test_extractvalue t WHERE t.id = 1;
col |
-----------------------------------------------------------------
YYyy |
-----------------------------------------------------------------
```

9.4 XMLCAST

功能描述

XML 转换函数，将 XML 节点值转换为字符串。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
XMLCAST(xmlval as typename)
```

参数说明

- xmlval：XML 类型数据。
- typename：需要转换成的类型，当前仅支持 CHAR 和 CLOB 类型。

函数返回类型

CHAR 类型。

示例

```
-- XML 向 CHAR 转 换
SQL> CREATE TABLE test_xmlcast(id INT,c1 XML);
SQL> INSERT INTO test_xmlcast VALUES(1,'<a>abc</a>');
SQL> SELECT XMLCAST(c1 AS VARCHAR) FROM test_xmlcast;

EXPR1(CHAR(-1)) |
-----------------------------------------------------------------
abc|
```

9.5 XMLELEMENT

功能描述

创建一个 XML 标签。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

XMLELEMENT：

```
XMLELEMENT ( xmlname [, xml_attributes] [, xmlvalue] )
```

xml_attributes：

```
xml_attributes ::= XMLATTRIBUTES ( xml_attribute_list )
xml_attribute_list ::= ( { xml_attribute } [ , ... ] )
xml_attribute ::= val AS name | ident
| ident;
```

> ⚠ 注意
>
> XMLATTRIBUTES 仅支持在 XMLELEMENT 函数中使用。

参数说明

- xmlname：XML 的标签名。
- xml_attributes：属性值，由 XMLATTRIBUTES 函数生成，为可选项。
- xmlvalue：节点值，为可选项。
- [, ...]：表示可以有多个 xml_attribute，每个属性之间用逗号分隔。
- val AS name：指定一个属性，其中 val 是属性值 name 是属性名称。
- ident：直接使用列名或表达式作为属性名称和属性值。

函数返回类型

CHAR 类型。

示例

单参数，只有标签名。

```
SQL> SELECT XMLELEMENT("name");

EXPR1 |
--------------------------------------------------------------
<name></name>|
```

双参数，标签名和属性值。

```
SQL> SELECT XMLELEMENT("name",XMLATTRIBUTES('wang' AS "class"));

EXPR1 |
------------------------------------------------------------
<name class="wang"></name>|
```

双参数，标签名和节点值。

```
SQL> SELECT XMLELEMENT("name",'xxx');

EXPR1 |
------------------------------------------------------------
<name>xxx</name>|
```

三参数，标签名、属性值和节点值。

```
SQL> SELECT XMLELEMENT("name",XMLATTRIBUTES('wang' as "class"),'xxx
   ');

EXPR1 |
------------------------------------------------------------
<name class="wang">xxx</name>|
```

9.6 XMLEXISTS

功能描述

判断一个 xpath 表达式中的元素是否存在。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
XMLEXISTS(xpath PASSING xmlval)
```

参数说明

- xmlval：XML 类型数据
- xpath：查找路径

函数返回类型

BOOL 类型。

示例

存在要查找的数据。

```
SQL> CREATE TABLE test_xmlexists(id INT,c1 XML,insert_time DATE);
SQL> INSERT INTO test_xmlexists VALUES(1,'<a>abc</a>','2020-01-01')
   ;
```

```
SQL> INSERT INTO test_xmlexists VALUES(2,'<country>china</country>'
   ,'2020-02-01');
SQL> INSERT INTO test_xmlexists VALUES(3,'<a>abc</a>','2020-03-01')
   ;
SQL> SELECT t.id,t.insert_time  FROM test_xmlexists t WHERE
   XMLEXISTS('/country' PASSING t.c1);

id |insert_time |
-------------------------------------------------------------
2 |2020-02-01 |
```

不存在要查找的数据。

```
SQL> SELECT t.id,t.insert_time FROM test_xmlexists t WHERE
   XMLEXISTS('/country/city' PASSING t.c1);

id |insert_time |
-------------------------------------------------------------
```

9.7 XMLFOREST

功能描述

直接创建一个只有标签名的标签。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

XMLFOREST：

```
XMLFOREST( xml_attribute_list )
```

xml_attribute_list：

```
xml_attribute_list ::= { xml_attribute } [ , ... ]
xml_attribute ::= val as name | ident;
```

参数说明

- xml_attribute_list：由多个 xml_attribute 组成
- xml_attribute：字段变量或函数

函数返回类型

CHAR 类型。

示例

单独使用。

```
SQL> CREATE TABLE test_xmlforest(id INT,name VARCHAR,addr VARCHAR,
   age INT,height NUMERIC(4,2));
```

```
SQL> INSERT INTO test_xmlforest VALUES(1,'aa','street 01',11,1.55);
SQL> INSERT INTO test_xmlforest VALUES(2,'bb','street 01',12,1.55);
SQL> SELECT XMLFOREST(t.name,max(t.addr) AS addr,max(t.age) AS age)
    AS col FROM test_xmlforest t GROUP BY t.name;

col |
-------------------------------------------------------------
<NAME>bb</NAME><ADDR>street 01</ADDR><AGE>12</AGE>|
<NAME>aa</NAME><ADDR>street 01</ADDR><AGE>11</AGE>|
```

在其他函数中使用。

```
SQL> SELECT XMLELEMENT("person",XMLFOREST(t.name,t.addr,t.age)) AS
   person FROM test_xmlforest t;

person |
-------------------------------------------------------------
<person><NAME>aa</NAME><ADDR>street 01</ADDR><AGE>11</AGE></person
   >|
<person><NAME>bb</NAME><ADDR>street 01</ADDR><AGE>12</AGE></person
   >|
```

9.8 XMLSEQUENCE

功能描述

将非标准的 XML 格式数据转换成数据库表结构。

> ⚠ 注意
>
> 当前版本 XMLSEQUENCE 仅支持在数据库单节点使用，暂不支持集群使用。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
XMLSEQUENCE(xmlvalue)
```

参数说明

xmlvalue：非标准的 XML 格式数据。

函数返回类型

TABLE 类型。

示例

```
-- 查询
SQL> CREATE TABLE test_xmlsequence(id INT,xml_col XML);
```

```
SQL> INSERT INTO test_xmlsequence VALUES(1,'<country>china<city
   capital="true">beijing</city><city capital="false">shanghai</city
   ></country>');
SQL> SELECT p.* FROM test_xmlsequence t,TABLE(XMLSEQUENCE(EXTRACT(t
   .xml_col,'/country'))) p WHERE t.id =1;

COLUMN_VALUE |
-------------------------------------------------------------
<country>china<city capital="true">beijing</city><city capital="
   false">shanghai</city></country> |
```

> ⚠ 注意
>
> 当前版本 XMLSEQUENCE 仅支持在数据库单节点使用，暂不支持集群使用。

9.9 XMLTABLE

功能描述

将 XML 格式的数据转换为数据库表结构。

> ⚠ 注意
>
> 当前版本 XMLTABLE 仅支持在数据库单节点使用，暂不支持集群使用。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

XMLTABLE：

```
XMLTABLE ( xmltable_exprs [ alias_clause ] )
```

xmltable_exprs：

```
xmltable_exprs ::= XQuery_father PASSING XML_data COLUMNS
   xmltable_new_columns

xmltable_new_columns ::= xmltable_new_column | xmltable_new_columns
   , xmltable_new_column

xmltable_new_column ::= name TYPENAME PATH XQuery_child
```

参数说明

- alias_clause：可选，用于为 XMLTABLE 返回的结果集指定一个别名。
- XQuery_father：父级 XQuery 表达式。

- XML_data：XML 数据。
- XQuery_child：子 XQuery 表达式。
- name：返回的表结构的字段名。

函数返回类型

TABLE 类型。

示例

```
-- 查询
SQL> CREATE TABLE test_xmltable(id INT,xml_col XML,insert_time DATE
   );
SQL> INSERT INTO test_xmltable VALUES(1,'<?xml version="1.0"
   encoding="UTF-8"?>
<bookstore>
<book category="COOKING">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="CHILDREN">
<title lang="en">Harry Potter</title>
<author>J K.Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="WEB">
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
<year>2003</year>
<price>49.99</price>
</book>
<book category="WEB">
<title lang="en">Learning XML</title>
<author>Erik T.RAY</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
','2020-03-01');
SQL> SELECT t1.insert_time,t2."title",t2."author",t2."year",t2."
   price"
FROM test_xmltable t1,XMLTABLE(
'//book[@category="CHILDREN"]'
PASSING t1.xml_col
COLUMNS "title" VARCHAR PATH 'title',
"author" VARCHAR PATH 'author',
"year" VARCHAR PATH 'year',
"price" VARCHAR PATH 'price'
) t2
WHERE t1.id =1;
```

```
insert_time |title |author |year |price |
-------------------------------------------------------------
2020-03-01 |Harry Potter |J K.Rowling |2005 |29.99 |
```

9.10 XMLQUERY

功能描述

以 xpath 表达式查询 XML 中的元素。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
XMLQUERY(xpath XML_passing_clause RETURNING CONTENT)

XML_passing_clause : PASSING XML_data
```

参数说明

- XML_data：XML 类型数据。
- xpath：查找路径。

函数返回类型

CHAR 类型。

示例

```
-- 存在要查找的数据
SQL> CREATE TABLE person_data (person_id INT,person_data XML);
SQL> INSERT INTO person_data
(person_id, person_data)
VALUES
(1, XMLTYPE('
<PDRecord>
<PDName>Daniel Morgan</PDName>
<PDDOB>12/1/1951</PDDOB>
<PDEmail>damorgan@u.washington.edu</PDEmail>
</PDRecord>')
);
SQL> INSERT INTO person_data
(person_id, person_data)
VALUES
(2, XMLTYPE('
<PDRecord>
<PDName>Jack Cline</PDName>
<PDDOB>5/17/1949</PDDOB>
<PDEmail>damorgan@u.washington.edu</PDEmail>
</PDRecord>')
);
SQL> INSERT INTO person_data
(person_id, person_data)
```

```
VALUES
(3, XMLTYPE('
<PDRecord>
<PDName>Caleb Small</PDName>
<PDDOB>1/1/1960</PDDOB>
<PDEmail>damorgan@u.washington.edu</PDEmail>
</PDRecord>')
);

SQL> SELECT person_id, XMLQuery(
'for $i in /PDRecord
where $i /PDName = "Daniel Morgan"
order by $i/PDName
return $i/PDName'
PASSING person_data
RETURNING CONTENT) xmlData
FROM person_data;

PERSON_ID |XMLDATA |
-----------------------------------------------------------------
1 |<PDName>Daniel Morgan</PDName> |
2 |NULL |
3 |NULL |
```

9.11 XMLTYPE

功能描述

将符合 XML 格式的数据转换为 XML 数据类型。

XML 数据类型的详细信息请参见《SQL 语法参考指南》的 XML 数据类型章节。

语法格式

```
XMLTYPE(val)
```

参数说明

val：符合 XML 格式的数据，当前版本支持 CHAR 和 CLOB 类型数据。

函数返回类型

XML 类型。

示例

CHAR 向 XML 转换。

```
SQL> SELECT XMLTYPE('<a>wang</a>');
|EXPR1|
|<a>wang</a>|

-- 不符合XML格式的CHAR数据向XML转换
SQL> SELECT XMLTYPE('abc');
```

```
|[E17106] 字 符 串 不 符 合XML类 型 格 式 |
```

CLOB 向 XML 转换。

```
SQL> SELECT XMLTYPE('<a>wang</a>'::CLOB);
|EXPR1|
|<a>wang</a>|

-- 不 符 合XML格 式 的CLOB数 据 向XML转 换
SQL> SELECT XMLTYPE('abc'::clob);
|[E17106] 字 符 串 不 符 合XML类 型 格 式 |
```

10 聚合函数

10.1 概述

函数形式	功能
AVG	求平均值
BIT_AND	执行按位与聚合运算
BIT_OR	执行按位或聚合运算
BIT_XOR	执行按位异或聚合运算
COUNT	统计记录条数
GROUPING	判断该行是否为统计产生的行
GROUP_CONCAT	将 GROUP BY 产生的同一个分组中的值连接起来，返回一个字符串结果
JSON_ARRAYAGG	将结果集聚合成 JSON 数组
JSON_OBJECTAGG	将结果集聚合成 JSON 对象
LISTAGG	将多个行按特定顺序组合为单个字符串
MAX	求最大值
MEDIAN	计算中位数
MIN	求最小值
STDDEV	计算标准差
STDDEV_POP	计算总体标准差
STDDEV_SAMP	计算累积样本标准差

接下页

函数形式	功能
STDEV	计算标准差
STDEVP	计算总体标准差
SUM	求和
VAR	计算方差
VAR_POP	计算总体方差
VAR_SAMP	计算采样方差
VARIANCE	计算方差
VARP	计算总体方差
XMLAGG	将多行数据转换为一个 XML 类型的值

10.2 AVG

功能描述

计算一组数值的平均值。

语法格式

```
AVG([ DISTINCT | ALL ] expr) [ OVER(analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数，analytic_clause 是常用的分析类语句，如 PARTITION BY 等。

函数返回类型

DOUBLE 或 NUMERIC 类型。

示例

```
SQL> CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
manager_id INT NOT NULL,
last_name VARCHAR(50) NOT NULL,
hire_date DATE NOT NULL,
```

```
salary NUMERIC(10, 2) NOT NULL
);

SQL> INSERT INTO employees (manager_id, last_name, hire_date,
   salary) VALUES
(100, 'De Haan', TO_DATE('2001-01-13', 'YYYY-MM-DD'), 17000),
(100, 'Raphaely', TO_DATE('2002-12-07', 'YYYY-MM-DD'), 11000),
(100, 'Kaufling', TO_DATE('2003-05-01', 'YYYY-MM-DD'), 7900),
(100, 'Hartstein', TO_DATE('2004-02-17', 'YYYY-MM-DD'), 13000),
(100, 'Weiss', TO_DATE('2004-07-18', 'YYYY-MM-DD'), 8000),
(100, 'Russell', TO_DATE('2004-10-01', 'YYYY-MM-DD'), 14000);
```

聚合函数。

```
SQL> SELECT AVG(salary) "AVERAGE" FROM employees;

AVERAGE |
-----------------------------------------------------------
11816.6666667|
```

分析函数。

```
SQL> SELECT manager_id, last_name,hire_date,salary,
AVG(salary) OVER (
PARTITION BY manager_id ORDER BY hire_date
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
) AS c_mavg
FROM employees;

MANAGER_ID | LAST_NAME | HIRE_DATE | SALARY | C_MAVG |
-----------------------------------------------------------------
100 | De Haan| 2001-01-13 AD | 17000| 14000|
100 | Raphaely| 2002-12-07 AD | 11000| 11966.6666667|
100 | Kaufling| 2003-05-01 AD | 7900| 10633.3333333|
100 | Hartstein| 2004-02-17 AD | 13000| 9633.3333333|
100 | Weiss| 2004-07-18 AD | 8000| 11666.6666667|
100 | Russell| 2004-10-01 AD | 14000| 11000|
```

10.3 BIT_AND

功能描述

执行按位与聚合运算，返回按位与运算结果数据。

语法格式

```
BIT_AND(expr)
```

参数说明

expr：数据类型为 VARBIT。

函数返回类型

VARBIT 类型。

> **说明**
>
> 若 expr 的某值为 NULL，则忽略跳过此值，并发出警告信息。

示例

```
SQL> CREATE TABLE tab_bit_and(c1 VARBIT(7));

SQL> INSERT INTO tab_bit_and VALUES(b'1010101')(b'1011100')(b'
   0011100');

SQL> SELECT BIT_AND(c1) FROM tab_bit_and;

EXPR1 |
------------------------------------------------------------
b'0010100' |

SQL> INSERT INTO tab_bit_and VALUES(NULL);

SQL> SELECT BIT_AND(c1) FROM tab_bit_and;

EXPR1 |
------------------------------------------------------------
b'0010100' |

Warning: [E17091] 统计函数忽略了部分空值
```

10.4 BIT_OR

功能描述

执行按位或聚合运算，返回按位或运算结果数据。

语法格式

```
BIT_OR(expr)
```

参数说明

expr：数据类型为 VARBIT。

函数返回类型

VARBIT 类型。

> **说明**
>
> 若 expr 的某值为 NULL，则忽略跳过此值，并发出警告信息。

示例

```
SQL> CREATE TABLE tab_bit_or(c1 VARBIT(7));

SQL> INSERT INTO tab_bit_or VALUES(b'1010101')(b'1011100')(b'
   0011100');

SQL> SELECT BIT_OR(c1) FROM tab_bit_or;

EXPR1 |
------------------------------------------------------------
b'1011101' |

SQL> INSERT INTO tab_bit_or VALUES(NULL);

SQL> SELECT BIT_OR(c1) FROM tab_bit_or;

BIT_OR(c1)(VARBIT) |
------------------------------------------------------------
b'1011101' |

Warning: [E17091] 统计函数忽略了部分空值
```

10.5 BIT_XOR

功能描述

执行按位异或聚合运算，返回按位异或运算结果数据。

语法格式

```
BIT_XOR(expr)
```

参数说明

expr：数据类型为 VARBIT。

函数返回类型

VARBIT 类型。

> **说明**
>
> 若 expr 的某值为 NULL，则忽略跳过此值，并发出警告信息。

示例

```
SQL> CREATE TABLE tab_bit_xor(c1 VARBIT(7));

SQL> INSERT INTO tab_bit_xor VALUES(b'1010101')(b'1011100')(b'
   0011100');

SQL> SELECT BIT_XOR(c1) FROM tab_bit_xor;

EXPR1 |
------------------------------------------------------------
b'0010101' |

SQL> INSERT INTO tab_bit_xor VALUES(NULL);

SQL> SELECT BIT_XOR(c1) FROM tab_bit_xor;

BIT_XOR(c1)(VARBIT) |
------------------------------------------------------------
b'0010101' |

Warning: [E17091] 统 计 函 数 忽 略 了 部 分 空 值
```

10.6 COUNT

功能描述

计算返回的行数。

语法格式

```
COUNT({ * | [ DISTINCT | ALL ] expr })
[ OVER (analytic_clause) ]
```

参数说明

- expr：列名、表达式、子查询或符号 *，用来指定要计数的数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 PARTITION BY 等。如果指定了参数 DISTINCT，则该部分只能指定 PARTITION BY 子句，而不能使用 ORDER BY 和开窗相关的语法。

函数返回类型

BIGINT 类型。

示例

```
SQL> CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
manager_id INT NOT NULL,
last_name VARCHAR(50) NOT NULL,
```

```
hire_date DATE NOT NULL,
salary NUMERIC(10, 2) NOT NULL
);

SQL> INSERT INTO employees (manager_id, last_name, hire_date,
   salary) VALUES
(100, 'De Haan', TO_DATE('2001-01-13', 'YYYY-MM-DD'), 17000),
(100, 'Raphaely', TO_DATE('2002-12-07', 'YYYY-MM-DD'), 11000),
(100, 'Kaufling', TO_DATE('2003-05-01', 'YYYY-MM-DD'), 7900),
(100, 'Hartstein', TO_DATE('2004-02-17', 'YYYY-MM-DD'), 13000),
(100, 'Weiss', TO_DATE('2004-07-18', 'YYYY-MM-DD'), 8000),
(100, 'Russell', TO_DATE('2004-10-01', 'YYYY-MM-DD'), 14000);
```

聚合函数。

```
SQL> SELECT COUNT(*) "Total" FROM employees;

Total |
------------------------------------------------------------
6 |

SQL> SELECT COUNT(DISTINCT manager_id) "Managers" FROM employees;

Managers |
------------------------------------------------------------
1 |
```

分析函数。

```
SQL> SELECT last_name, salary,
COUNT(*) OVER (
ORDER BY salary
RANGE BETWEEN 50 PRECEDING AND 150 FOLLOWING
) AS mov_count
FROM employees;

LAST_NAME | SALARY | MOV_COUNT |
------------------------------------------------------------
Kaufling| 7900| 2 |
Weiss| 8000| 1 |
Raphaely| 11000| 1 |
Hartstein| 13000| 1 |
Russell| 14000| 1 |
De Haan| 17000| 1 |
```

10.7 GROUPING

功能描述

判断该行是否为统计产生的行，返回 1 或者 0，返回 1 则表示该行为统计产生的行，返回 0 则表示不是统计产生的行。

语法格式

```
GROUPING(expr)
```

参数说明

expr：字段名。

函数返回类型

INTEGER 类型字符串。

示例

```
SQL> CREATE  TABLE  group_tab_2(id INT,name VARCHAR, salary INT);

SQL> INSERT INTO group_tab_2 VALUES(1,'NAME1',1000);

SQL> INSERT INTO group_tab_2 VALUES(2,'NAME2',2000);

SQL> INSERT INTO group_tab_2 VALUES(3,'NAME3',3000);

SQL> INSERT INTO group_tab_2 VALUES(4,'NAME4',4000);

SQL> SELECT GROUPING(id) g_id,id,SUM(salary) salary from
   group_tab_2 GROUP BY ROLLUP(id) ORDER BY  id;

G_ID | ID | SALARY |
-----------------------------------------------------------------
0 | 1 | 1000|
0 | 2 | 2000|
0 | 3 | 3000|
0 | 4 | 4000|
1 | <NULL>| 10000|
```

10.8 GROUP_CONCAT

功能描述

将 GROUP BY 产生的同一个分组中的值连接起来，返回一个字符串结果。

语法格式

```
GROUP_CONCAT(expr)
```

参数说明

expr：列名。

函数返回类型

VARCHAR 类型。

示例

```
SQL> CREATE TABLE g(a INT,b VARCHAR);

SQL> INSERT INTO g VALUES(1,'sada')(10,'da')(4,'a')(11,'a');

SQL> SELECT * FROM g;

A | B |
-------------------------------------------------------------
1 | sada|
10 | da|
4 | a|
11 | a|

SQL> SELECT GROUP_CONCAT(b) FROM g;

EXPR1 |
-------------------------------------------------------------
sada,da,a,a|

SQL> SELECT GROUP_CONCAT(DISTINCT b) FROM g ORDER BY a;

EXPR1 |
-------------------------------------------------------------
a,da,sada|
```

10.9 JSON_ARRAYAGG

功能描述

将结果集聚合成 JSON 数组。

语法格式

```
JSON_ARRAYAGG( col_or_expr )
```

参数说明

col_or_expr：列或表达式。

> ⚠ 注意
>
> 与其他聚合函数一致，出现 NULL 值（非 JSON NULL）会有“[E17091] 统计函数忽略了部分空值”警告。

函数返回类型

JSON 数值类型。

示例

```
CREATE TABLE t1 (a INT, b VARCHAR(10));
```

```
INSERT INTO t1 (a, b) VALUES (1, '1')(1, '2')(2, 'a')(2, 'b');

SELECT a, JSON_ARRAYAGG(b) AS b_array FROM t1 GROUP BY a;

EXPR1      |
-----------+
["1", "2"]|
["a", "b"]|
```

10.10 JSON_OBJECTAGG

功能描述

将结果集聚合成 JSON 对象。

语法格式

```
JSON_OBJECTAGG( key, value )
```

参数说明

- key：可转换为字符的类型。
- value：取值为 NULL、布尔型、数值型、自定义类型（OBJECT、VARRAY、TABLE）、JSON 类型以及其他能转换成字符型的类型。

> ⚠ 注意
>
> - 与其他聚合函数一致，出现 NULL 值（非 JSON NULL）会有“[E17091] 统计函数忽略了部分空值”警告。
> - key 为 NULL 或参数个数不为 2 会抛出错误。

函数返回类型

JSON 数值类型。

示例

```
create table t1(a int, b varchar);
insert into t1 values(1,'1')(1,'2')(2,'a')(2,'b');
SELECT json_objectagg(a, b) from t1 group by a;
EXPR1      |
-----------+
{"1": "2"}|
{"2": "b"}|
```

10.11 LISTAGG

功能描述

将某个表中的多个行的指定列值合并为一个字符串，以给定常量作为分隔符。

> ⚠ 注意
>
> - 函数返回的字符串长度可能会超过数据库定义的最大长度限制。在使用该函数时，要确保结果不会超过该限制。
> - 如果组合的元素包含特殊字符，如逗号或引号等，可能需要使用适当的转义字符或处理机制。
> - 分区子句可以根据需要来指定，并且允许对结果集进行更细粒度的分隔。在处理大数据集时，可以使用分区来提高性能。
> - 如果 WITHIN GROUP 子句未指定排序顺序，则使用默认的排序顺序。

语法格式

```
LISTAGG([ALL | DISTINCT] expr [,delimiter]) [WITHIN GROUP (
   order_by_clause)] [OVER (partition_clause)]
```

参数说明

- [ALL|DISTINCT] expr：expr 是要组合的列或表达式。你可以使用 ALL 或 DISTINCT 来控制是否包含重复的值。默认情况下，ALL 是隐式使用的。
- delimiter：可选的分隔符，用于分隔组合的元素。如果省略该参数，则使用默认的分隔符。
- WITHIN GROUP (order_by_clause)：order_by_clause 是一个可选的子句，用于指定组合的元素的排序顺序。可以在该子句中使用任何支持排序的表达式。
- OVER (partition_clause)：partition_clause 是一个可选的子句，用于指定要分区的数据集。可以在该子句中使用任何支持分区的表达式。

函数返回类型

STRING 类型。

示例

使用了 LISTAGG 函数聚合 lastname 列，并按 id 列进行分组。

```
SQL> SELECT LISTAGG(DISTINCT lastname, '; ') WITHIN GROUP(ORDER BY
   id DESC) a FROM hrmresource GROUP BY id;

A |
------------------------------------------------------------
xxx1|
xxx2|
```

```
xxx3|
xxx4|
xxx5|
xxx6|
xxx7|
```

10.12 MAX

功能描述

返回 expr 的最大值。

语法格式

```
MAX([ DISTINCT | ALL ] expr)
[ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 PARTITION BY 等。

函数返回类型

DOUBLE 或 NUMERIC 数值类型。

示例

```
SQL> CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
manager_id INT NOT NULL,
last_name VARCHAR(50) NOT NULL,
hire_date DATE NOT NULL,
salary NUMERIC(10, 2) NOT NULL
);

SQL> INSERT INTO employees (manager_id, last_name, hire_date,
   salary) VALUES
(100, 'De Haan', TO_DATE('2001-01-13', 'YYYY-MM-DD'), 17000),
(100, 'Raphaely', TO_DATE('2002-12-07', 'YYYY-MM-DD'), 11000),
(100, 'Kaufling', TO_DATE('2003-05-01', 'YYYY-MM-DD'), 7900),
(100, 'Hartstein', TO_DATE('2004-02-17', 'YYYY-MM-DD'), 13000),
(100, 'Weiss', TO_DATE('2004-07-18', 'YYYY-MM-DD'), 8000),
(100, 'Russell', TO_DATE('2004-10-01', 'YYYY-MM-DD'), 14000);
```

聚合函数。

```
SQL> SELECT MAX(salary) "MAXIMUM" FROM employees;

MAXIMUM |
---------------------------------------------------------------
```

```
17000|
```

分析函数。

```
SQL> SELECT manager_id, last_name, salary,
MAX(salary) OVER (PARTITION BY manager_id) AS  mgr_max
FROM employees;

MANAGER_ID | LAST_NAME | SALARY | MGR_MAX |
-------------------------------------------------------------
100 | De Haan| 17000| 17000|
100 | Raphaely| 11000| 17000|
100 | Kaufling| 7900| 17000|
100 | Hartstein| 13000| 17000|
100 | Weiss| 8000| 17000|
100 | Russell| 14000| 17000|
```

10.13 MEDIAN

功能描述

计算中位数。

语法格式

```
MEDIAN(expr) [ OVER (query_partition_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
  - 如果输入数据为奇数行，则结果为位于中间行的值。
  - 如果输入数据为偶数行，则结果为位于中间两行的均值。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 PARTITION BY 等。

函数返回类型

NUMERIC 类型。

示例

```
SQL> CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
manager_id INT NOT NULL,
last_name VARCHAR(50) NOT NULL,
hire_date DATE NOT NULL,
salary NUMERIC(10, 2) NOT NULL
);
```

```
SQL> INSERT INTO employees (manager_id, last_name, hire_date,
   salary) VALUES
(100, 'De Haan', TO_DATE('2001-01-13', 'YYYY-MM-DD'), 17000),
(100, 'Raphaely', TO_DATE('2002-12-07', 'YYYY-MM-DD'), 11000),
(100, 'Kaufling', TO_DATE('2003-05-01', 'YYYY-MM-DD'), 7900),
(100, 'Hartstein', TO_DATE('2004-02-17', 'YYYY-MM-DD'), 13000),
(100, 'Weiss', TO_DATE('2004-07-18', 'YYYY-MM-DD'), 8000),
(100, 'Russell', TO_DATE('2004-10-01', 'YYYY-MM-DD'), 14000);
```

聚合函数。

```
SQL> SELECT MEDIAN(salary) FROM employees WHERE employee_id > 1;

EXPR1 |
------------------------------------------------------------
11000|
```

分析函数。

```
SQL> SELECT manager_id, salary,
MEDIAN(salary) OVER (PARTITION BY manager_id) "DEPT MEDIAN SALARY"
FROM employees
ORDER BY manager_id, salary;

MANAGER_ID | SALARY | DEPT MEDIAN SALARY |
------------------------------------------------------------
100 | 7900| 12000|
100 | 8000| 12000|
100 | 11000| 12000|
100 | 13000| 12000|
100 | 14000| 12000|
100 | 17000| 12000|
```

10.14 MIN

功能描述

返回 expr 的最小值。

语法格式

```
MIN([ DISTINCT | ALL ] expr)
[ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 PARTITION BY 等。

函数返回类型

DOUBLE 或 NUMERIC 类型。

示例

```
SQL> CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
manager_id INT NOT NULL,
last_name VARCHAR(50) NOT NULL,
hire_date DATE NOT NULL,
salary NUMERIC(10, 2) NOT NULL
);

SQL> INSERT INTO employees (manager_id, last_name, hire_date,
   salary) VALUES
(100, 'De Haan', TO_DATE('2001-01-13', 'YYYY-MM-DD'), 17000),
(100, 'Raphaely', TO_DATE('2002-12-07', 'YYYY-MM-DD'), 11000),
(100, 'Kaufling', TO_DATE('2003-05-01', 'YYYY-MM-DD'), 7900),
(100, 'Hartstein', TO_DATE('2004-02-17', 'YYYY-MM-DD'), 13000),
(100, 'Weiss', TO_DATE('2004-07-18', 'YYYY-MM-DD'), 8000),
(100, 'Russell', TO_DATE('2004-10-01', 'YYYY-MM-DD'), 14000);
```

聚合函数。

```
SQL> SELECT MIN(salary) "MINIMUM" FROM employees;

MINIMUM |
-----------------------------------------------------------------
7900|
```

分析函数。

```
SQL> SELECT manager_id,last_name,salary,
MIN(salary) OVER (PARTITION BY manager_id) AS  mger_min
FROM employees;

MANAGER_ID | LAST_NAME | SALARY | MGER_MIN |
-----------------------------------------------------------------
100 | De Haan| 17000| 7900|
100 | Raphaely| 11000| 7900|
100 | Kaufling| 7900| 7900|
100 | Hartstein| 13000| 7900|
100 | Weiss| 8000| 7900|
100 | Russell| 14000| 7900|
```

10.15 STDDEV

功能描述

计算给定列或表达式的样本标准差。

可以作为聚合函数或分析函数使用，该结果是 VARIANCE 函数计算的方差的平方根。

STDDEV 函数同 STDEV 函数。

语法格式

```
STDDEV([ DISTINCT | ALL ] expr) [ OVER (analytic_clause) ]
```

参数说明

- [DISTINCT | ALL]：DISTINCT 只考虑不同的值，忽略重复值。ALL 考虑所有值，包括重复值。
- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
```

```
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT STDDEV(sal) "STDDEV" FROM employees;

STDDEV |
-------------------------------------------------------------
1182.50322351627169945865|
```

分析函数。

```
SQL> SELECT ename, sal, STDDEV(sal) OVER (ORDER BY hiredate) "
   STDDEV" FROM employees WHERE deptno = 30 ORDER BY ename, sal, "
   STDDEV";

ENAME | SAL | STDDEV |
-------------------------------------------------------------
ALLEN| 1600| 0|
BLAKE| 2850| 841.13019206303610131024|
JAMES| 950| 668.33125519211404502529|
MARTIN| 1250| 666.52081737932236920375|
TURNER| 1500| 715.30879112916448670486|
WARD| 1250| 247.48737341529163335403|
```

10.16 STDDEV_POP

功能描述

计算给定列或表达式的总体标准差并返回总体方差的平方根。

可以作为聚合函数或分析函数使用。STDDEV_POP 函数同 STDEVP 函数。

语法格式

```
STDDEV_POP(expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT STDDEV_POP(sal) "STDDEV_POP" FROM employees;

STDDEV_POP |
-----------------------------------------------------------
1139.488618295281524185 96|
```

分析函数。

```
SQL> SELECT ename, sal, STDDEV_POP(sal) OVER (ORDER BY hiredate) "
   STDDEV_POP" FROM employees WHERE deptno = 30 ORDER BY ename, sal
   , "STDDEV_POP";

ENAME | SAL | STDDEV_POP |
-----------------------------------------------------------------
ALLEN| 1600| 0|
BLAKE| 2850| 686.77992593455049723045|
JAMES| 950| 610.10017392410422277347|
MARTIN| 1250| 596.15434243155521948187|
TURNER| 1500| 619.47558466819335978342|
WARD| 1250| 175|
```

10.17 STDDEV_SAMP

功能描述

计算累积样本标准差并返回样本方差的平方根。

可以作为聚合函数或分析函数使用。

语法格式

```
STDDEV_SAMP(expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
```

```
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT STDDEV_SAMP(sal) "STDDEV_SAMP" FROM employees;

STDDEV_SAMP |
------------------------------------------------------------
1182.503223516271699458651|
```

分析函数。

```
SQL> SELECT ename, sal, STDDEV_SAMP(sal) OVER (ORDER BY hiredate) "
   STDDEV_SAMP" FROM employees WHERE deptno = 30 ORDER BY ename, sal
   , "STDDEV_SAMP";

ENAME | SAL | STDDEV_SAMP |
------------------------------------------------------------
ALLEN| 1600| <NULL>|
BLAKE| 2850| 841.13019206303610131024|
JAMES| 950| 668.331255192114045025291|
MARTIN| 1250| 666.520817379322369203751|
TURNER| 1500| 715.308791129164486704861|
WARD| 1250| 247.487373415291633540311|
```

10.18 STDEV

功能描述

计算给定列或表达式的样本标准差。

可以作为聚合函数或分析函数使用，该结果是 VARIANCE 函数计算的方差的平方根。STDEV 函数同 STDDEV 函数。

语法格式

```
STDEV([ DISTINCT | ALL ] expr) [ OVER (analytic_clause) ]
```

参数说明

- [DISTINCT | ALL]：DISTINCT 只考虑不同的值，忽略重复值。ALL 考虑所有值，包括重复值。
- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
```

```
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT STDEV(sal) "STDEV" FROM employees;

STDEV |
------------------------------------------------------------
1182.50322351627169945865|
```

分析函数。

```
SQL> SELECT ename, sal, STDEV(sal) OVER (ORDER BY hiredate) "STDEV"
    FROM employees WHERE deptno = 30 ORDER BY ename, sal, "STDEV";

ENAME | SAL | STDEV |
------------------------------------------------------------
ALLEN| 1600| 0|
BLAKE| 2850| 841.13019206303610131024|
JAMES| 950| 668.33125519211404502529|
MARTIN| 1250| 666.520817379322369203 75|
TURNER| 1500| 715.308791129164486704 86|
WARD| 1250| 247.48737341529163354 03|
```

10.19 STDEVP

功能描述

计算给定列或表达式的总体标准差并返回总体方差的平方根。

可以作为聚合函数或分析函数使用。STDEVP 函数同 STDDEV_POP 函数。

```
STDEVP(expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause) ：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT STDEVP(sal) "STDEVP" FROM employees;

STDEVP |
-------------------------------------------------------------
1139.48861829528152418596|
```

分析函数。

```
SQL> SELECT ename, sal, STDEVP(sal) OVER (ORDER BY hiredate) "
   STDEVP" FROM employees WHERE  deptno=30 ORDER BY ename, sal, "
   STDEVP";

ENAME | SAL | STDEVP |
-------------------------------------------------------------
ALLEN| 1600| 0|
BLAKE| 2850| 686.77992593455049723045|
JAMES| 950| 610.10017392410422277347|
MARTIN| 1250| 596.15434243155521948187|
TURNER| 1500| 619.47558466819335978342|
WARD| 1250| 175|
```

10.20 SUM

功能描述

返回 expr 的和。

语法格式

```
SUM([ DISTINCT | ALL ] expr)
[ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 PARTITION BY 等。

函数返回类型

DOUBLE 或 NUMERIC 类型。

示例

```
SQL> CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
manager_id INT NOT NULL,
last_name VARCHAR(50) NOT NULL,
hire_date DATE NOT NULL,
salary NUMERIC(10, 2) NOT NULL
);
```

```
SQL> INSERT INTO employees (manager_id, last_name, hire_date,
   salary) VALUES
(100, 'De Haan', TO_DATE('2001-01-13', 'YYYY-MM-DD'), 17000),
(100, 'Raphaely', TO_DATE('2002-12-07', 'YYYY-MM-DD'), 11000),
(100, 'Kaufling', TO_DATE('2003-05-01', 'YYYY-MM-DD'), 7900),
(100, 'Hartstein', TO_DATE('2004-02-17', 'YYYY-MM-DD'), 13000),
(100, 'Weiss', TO_DATE('2004-07-18', 'YYYY-MM-DD'), 8000),
(100, 'Russell', TO_DATE('2004-10-01', 'YYYY-MM-DD'), 14000);
```

聚合函数。

```
SQL> SELECT SUM(salary) "TOTAL" FROM employees;

TOTAL |
------------------------------------------------------------
70900|
```

分析函数。

```
SQL> SELECT manager_id,last_name,salary,
SUM(salary) OVER (
PARTITION BY manager_id ORDER BY salary
) l_csum
FROM employees;

MANAGER_ID | LAST_NAME | SALARY | L_CSUM |
------------------------------------------------------------
100 | Kaufling| 7900| 7900|
100 | Weiss| 8000| 15900|
100 | Raphaely| 11000| 26900|
100 | Hartstein| 13000| 39900|
100 | Russell| 14000| 53900|
100 | De Haan| 17000| 70900|
```

10.21 VAR

功能描述

计算数据的方差。

可以作为聚合函数或分析函数使用。VAR 函数同 VARIANCE 函数。

语法格式

```
VAR([ DISTINCT | ALL ] expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
  - 当 expr 的行数 = 1 时，Oracle 兼容模式下返回值为 0，否则返回 NULL。
  - 当 expr 的行数 > 1 时，返回值为各行数字（去除空值外）的方差。

- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT VAR(sal) "VAR" FROM employees;

VAR |
```

```
-------------------------------------------------------------
1398313.87362637362637362637|
```

分析函数。

```
SQL> SELECT ename, sal, VAR(sal) OVER (ORDER BY hiredate) "VAR"
   FROM employees WHERE deptno = 30;

ENAME | SAL | VAR |
-------------------------------------------------------------
ALLEN| 1600| 0|
WARD| 1250| 61250|
BLAKE| 2850| 707500|
TURNER| 1500| 511666.66666666666666666667|
MARTIN| 1250| 444250|
JAMES| 950| 446666.66666666666666666667|
```

10.22 VAR_POP

功能描述

计算数据的总体方差。

可以作为聚合函数或分析函数使用。VAR_POP 函数同 VARP 函数。

语法格式

```
VAR_POP(expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);
```

```
SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT VAR_POP(sal) "VAR_POP" FROM employees;

VAR_POP |
--------------------------------------------------------------
1298434.31122448979591836735|
```

分析函数。

```
SQL> SELECT hiredate, VAR_POP(sum(sal)) OVER (ORDER BY hiredate) "
   VAR_POP" FROM employees GROUP BY hiredate ORDER BY hiredate, "
   VAR_POP";

HIREDATE | VAR_POP |
--------------------------------------------------------------
1980-12-17 AD | 0|
1981-02-20 AD | 160000|
1981-02-22 AD | 107222.22222222222222222|
1981-04-02 AD | 660117.1875|
1981-05-01 AD | 756100|
1981-06-09 AD | 672864.58333333333333333|
1981-09-08 AD | 605841.83673469387755102041|
1981-09-28 AD | 578896.484375|
1981-11-17 AD | 1504320.98765432098765432099|
```

```
1981-12-03 AD | 1633906.25|
1982-01-23 AD | 1578667.355371900826446280 99|
1987-07-13 AD | 1704075.520833333333333333 33|
```

10.23 VAR_SAMP

功能描述

计算数据的样本方差。

可以作为聚合函数或分析函数使用。

语法格式

```
VAR_SAMP(expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
```

```
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT VAR_SAMP(sal) "VAR_SAMP" FROM employees;

VAR_SAMP |
---------------------------------------------------------
1398313.873626373626373626373|
```

分析函数。

```
SQL> SELECT hiredate, VAR_SAMP(SUM(sal)) OVER (ORDER BY hiredate) "
   VAR_SAMP" FROM employees GROUP BY hiredate ORDER BY hiredate, "
   VAR_SAMP";

HIREDATE | VAR_SAMP |
-----------------------------------------------------------------
1980-12-17 AD | <NULL>|
1981-02-20 AD | 320000|
1981-02-22 AD | 160833.33333333333333333333|
1981-04-02 AD | 880156.25|
1981-05-01 AD | 945125|
1981-06-09 AD | 807437.5|
1981-09-08 AD | 706815.476190476190476190476|
1981-09-28 AD | 661595.982142857142857142857|
1981-11-17 AD | 1692361.111111111111111111111|
1981-12-03 AD | 1815451.388888888888888888889|
1982-01-23 AD | 1736534.090909090909090909091|
1987-07-13 AD | 1858991.477272727272727272727|
```

10.24 VARIANCE

功能描述

计算数据的方差。

语法格式

```
VARIANCE([ DISTINCT | ALL ] expr)
[ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
  - 当 expr 的行数 = 1 时，返回值为 0。
  - 当 expr 的行数 > 1 时，返回值为各行数字（去除空值外）的方差。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
```

```
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT VARIANCE(sal) "VARIANCE" FROM employees;

VARIANCE |
-----------------------------------------------------------
1398313.873626373626373626373|
```

分析函数。

```
SQL> SELECT ename, sal, VARIANCE(sal) OVER (ORDER BY hiredate) "
   VARIANCE" FROM employees WHERE deptno = 30;

ENAME | SAL | VARIANCE |
-----------------------------------------------------------
ALLEN| 1600| 0|
WARD| 1250| 61250|
BLAKE| 2850| 707500|
TURNER| 1500| 511666.66666666666666666667|
MARTIN| 1250| 444250|
JAMES| 950| 446666.66666666666666666667|
```

10.25 VARP

功能描述

计算数据的总体方差。

可以作为聚合函数或分析函数使用。VARP 函数同 VAR_POP 函数。

语法格式

```
VARP(expr) [ OVER (analytic_clause) ]
```

参数说明

- expr：列名或表达式，用来指定输入数据。
- OVER (analytic_clause)：可选的表达式，用于分析函数。analytic_clause 是常用的分析类语句，如 ORDER BY 等。

函数返回类型

NUMERIC/DOUBLE 数值类型。

示例

```
SQL> CREATE TABLE employees
(
empno    NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
ename    VARCHAR2(10),
job      VARCHAR2(9),
mgr      NUMBER(4),
hiredate DATE,
sal      NUMBER(7,2),
comm     NUMBER(7,2),
deptno   NUMBER(2)
);

SQL> INSERT INTO employees VALUES
(7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-12-1980', 'DD-MM-YYYY')
   , 800, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-2-1981', 'DD-MM-YYYY'
   ), 1600, 300, 30),
(7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-2-1981', 'DD-MM-YYYY')
   , 1250, 500, 30),
(7566, 'JONES', 'MANAGER', 7839, TO_DATE('2-4-1981', 'DD-MM-YYYY')
   , 2975, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, TO_DATE('28-9-1981', 'DD-MM-YYYY
   '), 1250, 1400, 30),
(7698, 'BLAKE', 'MANAGER', 7839, TO_DATE('1-5-1981', 'DD-MM-YYYY')
   , 2850, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, TO_DATE('9-6-1981', 'DD-MM-YYYY')
   , 2450, NULL, 10),
(7788, 'SCOTT', 'ANALYST', 7566, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7839, 'KING', 'PRESIDENT', NULL, TO_DATE('17-11-1981', 'DD-MM-YYYY
   '), 5000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, TO_DATE('8-9-1981', 'DD-MM-YYYY'
   ), 1500, 0, 30),
(7876, 'ADAMS', 'CLERK', 7788, TO_DATE('13-7-1987', 'DD-MM-YYYY')
   , 1100, NULL, 20),
(7900, 'JAMES', 'CLERK', 7698, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 950, NULL, 30),
(7902, 'FORD', 'ANALYST', 7566, TO_DATE('3-12-1981', 'DD-MM-YYYY')
   , 3000, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, TO_DATE('23-1-1982', 'DD-MM-YYYY')
   , 1300, NULL, 10);
```

聚合函数。

```
SQL> SELECT VARP(sal) "VARP" FROM employees;

VARP |
-------------------------------------------------------------
1298434.31122448979591836735|
```

分析函数。

```
SQL> SELECT hiredate, VARP(SUM(sal)) OVER (ORDER BY hiredate) "VARP
   " FROM employees GROUP BY hiredate ORDER BY hiredate, "VARP";

HIREDATE | VARP |
-------------------------------------------------------------
1980-12-17 AD | 0|
1981-02-20 AD | 160000|
1981-02-22 AD | 107222.22222222222222222|
1981-04-02 AD | 660117.1875|
1981-05-01 AD | 756100|
1981-06-09 AD | 672864.58333333333333333|
1981-09-08 AD | 605841.836734693877551020 41|
1981-09-28 AD | 578896.484375|
1981-11-17 AD | 1504320.987654320987654320 99|
1981-12-03 AD | 1633906.25|
1982-01-23 AD | 1578667.355371900826446280 99|
1987-07-13 AD | 1704075.520833333333333333 33|
```

10.26 XMLAGG

功能描述

将多行数据转换为一个 XML 类型的值。

语法格式

```
XMLAGG((XMLELEMENT ( xmlname, xml_attributes, xmlvalue )))

xml_attributes ::= XMLATTRIBUTES ( xml_attribute_list )
xml_attribute_list ::= ( { xml_attribute } [ , ... ] )
xml_attribute ::= val AS name | ident
```

参数说明

- xmlname：XML 的标签名。
- xml_attributes：属性值，由 XMLATTRIBUTES 函数生成，为可选项。
- xmlvalue：节点值，为可选项。
- [, ...]：表示可以有多个 xml_attribute，每个属性之间用逗号分隔。
- val AS name：指定一个属性，其中 val 是属性值 name 是属性名称。
- ident：直接使用列名或表达式作为属性名称和属性值。

函数返回类型

CHAR 类型。

示例

使用 XMLAGG 函数将多行数据聚合成一个 XML 元素，并通过 ORDER BY 子句对结果进行排序。

```
SQL> CREATE TABLE test_xmlagg(id INT,name VARCHAR,addr VARCHAR,age
   INT);

SQL> INSERT INTO test_xmlagg VALUES(1,'aa','street 01',11);
SQL> INSERT INTO test_xmlagg VALUES(2,'bb','street 01',12);

-- ORDER BY age
SQL> SELECT XMLAGG(XMLELEMENT("person",t.name || ' ' || t.addr)) AS
    person_info FROM test_xmlagg t ORDER BY t.age;

person_info |
------------------------------------------------------------------
<person>aa street 01</person>person>bb street 01</person>|

-- ORDER BY age DESC
SQL> SELECT XMLAGG(XMLELEMENT("person",t.name || ' ' || t.addr)) AS
    person_info FROM test_xmlagg t ORDER BY t.age DESC;

person_info |
------------------------------------------------------------------
<person>aa street 01</person>person>bb street 01</person>|
```

11 窗口函数

11.1 概述

函数形式	功能
ROW_NUMBER	根据指定的列或多个列对结果集进行排序，并为每一行分配一个行号

11.2 ROW_NUMBER

功能描述

ROW_NUMBER 是一种窗口函数，它为结果集中的每一行分配一个唯一的数字值。这个数字值是根据指定的排序顺序获得，可以用来对结果集中的行进行编号或排序。

ROW_NUMBER 函数会根据排序顺序为每一行分配连续的数字值，但不会跳过重复的值。如果有多行具有相同的排序值，则它们具有相同的行号。如果想要跳过重复的值，可以使用其他窗口函数如 RANK 或 DENSE_RANK。

语法格式

```
ROW_NUMBER() OVER (ORDER BY column1, column2, ...)
```

参数说明

- ORDER BY：可选，用于指定排序的列和排序顺序。
- column：排序的列。

函数返回类型

INTEGER 或 BIGINT 类型。

示例

- 示例 1

  从 EMP 表中选择的员工编号 (EMPNO)、经过四舍五入的薪水 (SAL) 以及为每一行分配的唯一行号 (SO_SAL)。

```
SQL> SELECT EMPNO,ROUND(SAL,0) SAL,ROW_NUMBER() OVER() AS SO_SAL
   FROM EMP;
```

```
EMPNO | SAL | SO_SAL |
-----------------------------------------------------------------
7369| 800| 1 |
7499| 1600| 2 |
7521| 1250| 3 |
7566| 2975| 4 |
7654| 1250| 5 |
7698| 2850| 6 |
```

- 示例 2

  按部门 (DEPTNO) 对员工进行分组，并在每个部门内按 DEPTNO 和 EMPNO 排序。

```
SQL> SELECT EMPNO,ROUND(SAL,0) SAL,ROW_NUMBER() OVER() AS SO_SAL
   FROM EMP;

EMPNO | SAL | SO_SAL |
-----------------------------------------------------------------
7369| 800| 1 |
7499| 1600| 2 |
7521| 1250| 3 |
7566| 2975| 4 |
7654| 1250| 5 |
7698| 2850| 6 |
```

12 内部功能性函数

12.1 概述

函数形式	功能
FLUSH_COMMAND_LOG	将 COMMAND.LOG 的缓存立即写入磁盘
FLUSH_ERROR_LOG	系统错误日志 ERROR.LOG 的即时缓存写盘函数
FLUSH_MODIFY_LOG	同步等候变更日志的写盘动作
FORMAT_GSTO_NOS	格式化全局存储号信息

12.2 FLUSH_COMMAND_LOG

功能描述

将 COMMAND.LOG 的缓存立即写入磁盘。

用于将当前系统缓存中的 SQL 请求信息立即写入磁盘，可避免在读取 COMMAND.LOG 日志过程中出现的读取内容与实际内容不一致的问题。该函数只影响当前连接的节点，并不会向集群内的其他远程节点发起同步请求。在调用该函数后，若写盘动作正常执行完成则返回 true，否则返回 false。

语法格式

```
FLUSH_COMMAND_LOG()
```

参数说明

无参数

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT FLUSH_COMMAND_LOG();

FLUSH_COMMAND_LOG() |
--------------------------------------------------------------
```

```
T |

SQL> SELECT * FROM SYS_COMMAND_LOG;
```

12.3 FLUSH_ERROR_LOG

功能描述

系统错误日志 ERROR.LOG 的即时缓存写盘函数。

用于将当前系统缓存中的异常信息立即写入磁盘，可避免在读取 ERROR.LOG 日志过程中出现的读取内容与实际内容不一致的问题。该函数只影响当前连接的节点，并不会向集群内的其他远程节点发起同步请求。在调用该函数后，若写盘动作正常执行完成则返回 true，否则返回 false。

语法格式

```
FLUSH_ERROR_LOG()
```

参数说明

无参数

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT FLUSH_ERROR_LOG();

FLUSH_ERROR_LOG() |
------------------------------------------------------------
T |

SQL> SELECT * FROM SYS_ERROR_LOG;
```

12.4 FLUSH_MODIFY_LOG

功能描述

FLUSH_MODIFY_LOG 函数用于同步等候变更日志的写盘动作。

在实际业务请求与变更记载写盘行为之间属于异步设计，故在使用 poll_modify_data 对变更日志进行消费时可能存在因数据不及时落盘而出现的无法消费到最新的变更内容，这里存在一个微小的数据消费时间窗口。可通过在每次进行变更数据消费之前调用 FLUSH_MODIFY_LOG()

函数来进行最长不超过 1 秒的同步等候动作，若限制的时间窗口内同步等候完成该函数返回 true，否则返回 false。

> **说明**
>
> 该函数只能支持在 G 角色节点上执行，在非 G 角色节点上执行将返回 false。

语法格式

```
    FLUSH_MODIFY_LOG()
```

参数说明

无参数

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT FLUSH_MODIFY_LOG();

FLUSH_MODIFY_LOG()(BOOLEAN) |
-------------------------------------------------------------
F |

SQL> EXEC dbms_replication.poll_modify_data('sup', 0, null
   , 1024*1024, false, -1, 0);
```

12.5 FORMAT_GSTO_NOS

功能描述

格式化全局存储号信息。

二级分区表的全局存储号信息在系统表中常以一个不可读取的魔数表示，需要由数据库内部系统函数 format_gsto_nos 来进行格式化展示。format_gsto_nos 有以下两种重载形式：

- 接受 1 个参数，输出结果为逗号分隔的全局存储号字符串。
- 接受 2 个参数，分别是全局存储号的魔数和子分区号，输出结果为全局存储号魔数中子分区号指定的全局存储号数值。

语法格式

```
FORMAT_GSTO_NOS(expr1[,expr2])
```

参数说明

- expr1：一个整形的全局存储信息的魔数，这个魔数通过系统表查询获取。
- expr2：子分区号。

函数返回类型

- 参数一位时，返回 CHAR
- 参数两位时，返回 INTEGER

示例

- 示例 1

  接受 1 个参数，输出结果为逗号分隔的全局存储号字符串。

```
SQL> SELECT format_gsto_nos(gsto_nos) AS gsto_nos FROM dba_partis
    WHERE table_id=(SELECT table_id FROM dba_tables WHERE
   table_name='T1');

GSTO_NOS |
-------------------------------------------------------------
201,202,203
204,205,206
```

- 示例 2

  接受 2 个参数，分别是全局存储号的魔数和子分区号，输出结果为全局存储号魔数中子分区号指定的全局存储号数值。

```
SQL> SELECT parti_no, subparti_no, format_gsto_nos(gsto_nos,
   subparti_no) AS gsto_no FROM dba_partis a, dba_subpartis b
   WHERE a.table_id = (SELECT table_id FROM dba_tables WHERE
   table_name='T1');

PARTI_NO | SUBPARTI_NO | GSTO_NO |
-------------------------------------------------------------
0 | 0 | 201
0 | 1 | 202
0 | 2 | 203
1 | 0 | 204
1 | 1 | 205
1 | 2 | 206
```

13 特殊表达式

13.1 概述

函数形式	功能
CURRENT_DATE	取当前日期
CURRENT_DATABASE	取当前数据库名
CURRENT_DB	取当前数据库名
CURRENT_DB_ID	取当前数据库 ID
CURRENT_IP	取当前连接会话的登录来源 IP
CURRENT_NODEID	返回当前连接节点 nodeid
CURRENT_SCHEMA	获取当前连接的模式（默认情况下与当前登录用户同名）
CURRENT_TIME	取当前时间，格式为 HH24:MI:SS.FFF
CURRENT_TIMESTAMP	取当前日期时间，格式为 YYYY-MM-DD HH24:MI:SS.FFF
CURRENT_USER	取当前用户名
CURRENT_USERID	取当前用户（调用者）的用户 ID 号
CURRVAL	获取指定序列值对象在当前用户会话中最后一次获取到的序列值
DATABASE	同 CURRENT_DB，返回当前数据库
DIR_EXISTS	判断数据库 HOME 目录下文件夹是否存在

接下页

函数形式	功能
FILE_EXISTS	判断数据库文件是否存在
FILELEN	返回一个 Long，代表一个文件的长度，单位是字节
GEN_RANDOM_UUID	兼容 PostgreSQL 的 GEN_RANDOM_UUID 函数生成 UUID
GET_INSTALL_PATH	获取指定节点安装目录
GET_TYPE_SPACE	获取指定类型在存储上占用的空间大小，单位是字节
GET_WORK_PATH	获取指定节点工作目录
GREATEST	返回一组表达式中的最大值，以第一个表达式的数据类型为标准
IF	如果参数 1 为真，则返回参数 2，否则返回参数 3
INET_ATON	IPV4 转整数
INET_NTOA	整数转 IPV4
JSON_VALUE	在给定的路径中获取一个值，并可选择返回类型将其转换返回。
LEAST	返回一组表达式中的最小值
MAX_NODE_NUM	查询 cluster.ini 中 MAX_NODES 值
MEMBER OF	判断 json_doc 和 value 是否为 JSON 数组的元素
NEWID	产生一个全球唯一的标识
NUM_NONNULLS	统计参数列表中非空值 (NULL) 的数量
NUM_NULLS	统计参数列表中空值 (NULL) 的数量
RENAME_FILE	重命名数据库文件

接下页

函数形式	功能
SLEEP	执行挂起一段时间
SQLERRM	返回与参数指定的错误代码相关的错误消息
SYS_CONTEXT	返回与上下文命名空间相关的参数值
SYS_GUID	通用唯一识别码，通过算法生成全局唯一 ID
SYS_USERID	返回建立当前连接时的登录用户的 ID
SYS_UUID	通用唯一识别码，通过算法生成全局唯一 ID
SYSTEM_USER	返回建立当前连接时的登录用户
UID	取当前用户（调用者）的用户 ID 号
USER	同 current_user，返回当前用户
UUID	通用唯一识别码，通过算法生成全局唯一 ID
USERENV	获取连接用户关键信息
VERSION	同 show version，返回当前数据库版本

13.2 CURRENT_DATABASE

功能描述

取当前数据库名。

语法格式

```
CURRENT_DATABASE
```

参数说明

无参数。

函数返回类型

VARCHAR 字符串。

示例

```
SQL> SELECT CURRENT_DATABASE FROM dual;
```

```
EXPR1 |
------------------------------------------------------------
SYSTEM|
```

13.3 CURRENT_DATE

功能描述

取当前日期。

语法格式

```
CURRENT_DATE
```

参数说明

无参数。

函数返回类型

DATE 日期类型。

示例

```
SQL> SELECT CURRENT_DATE FROM dual;

EXPR1 |
------------------------------------------------------------
2022-05-14 AD |
```

13.4 CURRENT_DB

功能描述

取当前数据库名。

语法格式

```
CURRENT_DB
```

参数说明

无参数。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT CURRENT_DB FROM dual;

EXPR1 |
------------------------------------------------------------
SYSTEM|
```

13.5 CURRENT_DB_ID

功能描述

取当前数据库 ID。

语法格式

```
CURRENT_DB_ID
```

参数说明

无参数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT CURRENT_DB_ID FROM dual;

EXPR1 |
------------------------------------------------------------
1 |
```

13.6 CURRENT_IP

功能描述

取当前连接会话的登录来源 IP。

语法格式

```
CURRENT_IP
```

参数说明

无参数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT CURRENT_IP FROM dual;

EXPR1 |
------------------------------------------------------------
192.168.2.116|
```

13.7 CURRENT_NODEID

功能描述

返回当前连接节点 nodeid。

语法格式

```
CURRENT_NODEID
```

参数说明

无参数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT CURRENT_NODEID  FROM dual;

EXPR1 |
------------------------------------------------------------
1 |
```

13.8 CURRENT_SCHEMA

功能描述

获取当前连接的模式（默认情况下与当前登录用户同名）。

语法格式

```
CURRENT_SCHEMA
```

参数说明

无参数。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT current_schema FROM dual;

EXPR1 |
------------------------------------------------------------
SYSDBA|
```

13.9 CURRENT_TIME

功能描述

取当前时间，格式为 HH24:MI:SS.FFF。

语法格式

```
CURRENT_TIME
```

参数说明

无参数。

函数返回类型

TIME 时间类型。

示例

```
SQL> SELECT CURRENT_TIME FROM dual;

EXPR1 |
------------------------------------------------------------
15:42:23.820 |
```

13.10 CURRENT_TIMESTAMP

功能描述

取当前日期时间，格式为 YYYY-MM-DD HH24:MI:SS.FFF。

语法格式

```
CURRENT_TIMESTAMP
```

参数说明

无参数。

函数返回类型

TIMESTAMP 日期时间类型。

示例

```
SQL> SELECT CURRENT_TIMESTAMP FROM dual;

EXPR1 |
-----------------------------------------------------------
2022-05-14 15:42:01.601 AD |
```

13.11 CURRENT_USER

功能描述

取当前用户名。

语法格式

```
CURRENT_USER
```

参数说明

无参数。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT CURRENT_USER FROM dual;

EXPR1 |
-----------------------------------------------------------
SYSDBA|
```

13.12 CURRENT_USERID

功能描述

取当前用户（调用者）的 ID。

语法格式

```
CURRENT_USERID()
```

参数说明

无参数。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT CURRENT_USERID() FROM dual;

EXPR1 |
------------------------------------------------------------
1 |
```

13.13 CURRVAL

功能描述

获取指定序列值对象在当前用户会话中最后一次获取到的序列值。

语法格式

```
CURRVAL(expr)
```

参数说明

expr：序列值名称。

> **说明**
>
> 可指定模式名，数据类型为 CHAR，格式为 [schema_name.]sequence_name。

函数返回类型

BIGINT 类型。

> **说明**
>
> 若指定序列值在当前用户会话中还未获取过序列值，则此函数将抛出错误。

示例

```
SQL> CREATE SEQUENCE sysdba.seq;

SQL> SELECT sysdba.seq.nextval;
EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT currval('seq');
EXPR1 |
------------------------------------------------------------
1 |

SQL> SELECT currval('sysdba.seq');
EXPR1 |
```

```
-------------------------------------------------------------------
1 |
```

13.14 DATABASE

功能描述

同 CURRENT_DB，返回当前数据库。

与 MySQL 无差异。

语法格式

```
DATABASE()
```

参数说明

无参数。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT DATABASE();

EXPR1 |
-------------------------------------------------------------------
SYSTEM|
```

13.15 DIR_EXISTS

功能描述

判断数据库 HOME 目录下文件夹是否存在。

语法格式

```
DIR_EXISTS(expr)
```

参数说明

expr：文件夹名字。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT DIR_EXISTS('/DATA');

EXPR1 |
------------------------------------------------------------
T |
```

13.16 FILE_EXISTS

功能描述

判断数据库文件是否存在。

语法格式

```
FILE_EXISTS(expr)
```

参数说明

expr：文件名字。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT FILE_EXISTS(123) FROM dual;

EXPR1 |
------------------------------------------------------------
F |
```

13.17 FILELEN

功能描述

获取一个文件的长度，单位是字节。

语法格式

```
FILELEN(expr)
```

参数说明

expr：文件名字。

函数返回类型

BIGINT 类型。

示例

```
SQL> SELECT FILELEN('/DATA/DATA1.DBF') FROM dual;

EXPR1 |
------------------------------------------------------------
402653184 |
```

13.18 GEN_RANDOM_UUID

功能描述

兼容 PostgreSQL 的 GEN_RANDOM_UUID 函数生成 UUID。

语法格式

```
GEN_RANDOM_UUID
```

参数说明

无参数。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT GEN_RANDOM_UUID FROM dual;

EXPR1 |
------------------------------------------------------------
660FEDAD-0AC3-4EDC-B511-773C21108B24|
```

13.19 GET_INSTALL_PATH

功能描述

获取指定节点安装目录。

语法格式

```
GET_INSTALL_PATH(expr)
```

参数说明

expr：指定节点。

函数返回类型

VARCHAR 类型。

示例

```
SQL> SELECT GET_INSTALL_PATH(1);

EXPR1 |
------------------------------------------------------------
/RAID0_1/wq/db/BIN|

SQL> SELECT GET_INSTALL_PATH(2);

EXPR1 |
------------------------------------------------------------
/RAID0_1/wq/db/BIN|
```

13.20 GET_TYPE_SPACE

功能描述

获取指定类型在存储上占用的空间大小，单位是字节。

语法格式

```
GET_TYPE_SPACE(expr)
```

参数说明

expr：数据类型的名称字符串，可以带精度。

> **说明**
>
> 参数为虚谷数据库支持的数据类型。

函数返回类型

INTEGER 类型。

- 变长类型（VARCHAR/BLOB/CLOB/BINARY）：返回 0。
- 非内置数据类型：返回异常，数据类型
- NULL 或空字符串：返回 NULL。
- NUMERIC 类型：返回 2 17byte（不同精度对应不同的存储长度）。
- 字符类型：
  - 定长类型（CHAR/NCHAR[(SIZE)]）：返回 SIZE * mbmaxlen。GBK 库返回 SIZE * 2，UTF8 库返回 SIZE * 3。
  - 变长类型（VARCHAR/VARCHAR2[(SIZE)]）：返回 0。

> **说明**
>
> SIZE 单位为字符；字符长度由不同字符集的库决定，字符集中最大字符长度大小为 mbmaxlen。

示例

```
SQL> SELECT GET_TYPE_SPACE('NUMERIC(38,12)') FROM DUAL;

EXPR1 |
------------------------------------------------------------
17 |

SQL> SELECT GET_TYPE_SPACE('INTERVAL MINUTE(7) TO SECOND(6)') FROM
   DUAL;

EXPR1 |
------------------------------------------------------------
8 |
```

13.21 GET_WORK_PATH

功能描述

获取指定节点工作目录。

语法格式

```
GET_WORK_PATH(expr)
```

参数说明

expr：指定节点。

函数返回类型

VARCHAR 类型。

示例

```
SQL> SELECT GET_WORK_PATH(1);

EXPR1 |
------------------------------------------------------------
/RAID0_1/wq/db/BIN|

SQL> SELECT GET_WORK_PATH(2);

EXPR1 |
------------------------------------------------------------
/RAID0_1/wq/db/BIN|
```

13.22 GREATEST

功能描述

返回一组表达式中的最大值，以第一个表达式的数据类型为标准。

语法格式

```
GREATEST(expr1, expr2, ...exprn)
```

参数说明

expr1,2,...,n：同类型表达式。

函数返回类型

同数据类型。

示例

```
SQL> SELECT GREATEST(2, 5, 3) colo1, GREATEST(2, '5', 3) col2,
   GREATEST('b', 'B', 'a') col3, GREATEST('b', 'cB', 'a') col4,
   GREATEST(SYSDATE,SYSDATE+1) col5,GREATEST('中', '华', '人') col6
   , GREATEST(NULL, 'a', 'b') col7 FROM dual;

COLO1(TINYINT) | COL2(BIGINT) | COL3(CHAR(-1)) | COL4(CHAR(-1)) |
   COL5(DATETIME) | COL6(CHAR(-1)) | COL7(CHAR(-1)) |
-------------------------------------------------------------
5 | 5 | b| cB| 2022-06-25 14:43:41.984  | 华| b|
```

13.23 IF

功能描述

条件判断。如果 expr1 为真，则返回 expr2，否则返回 expr3。

语法格式

```
IF(expr1,expr2,expr3)
```

参数说明

- expr1：布尔表达式。
- expr2/expr3：任意两个具有公共数据类型的数据类型。

函数返回类型

返回值类型为 expr2 和 expr3 的公共数据类型。

示例

```
SQL> SELECT IF(1>0,'yes','no'), IF(1<0,'yes','no') FROM dual;
```

```
EXPR1 | EXPR2 |
-------------------------------------------------------------
yes| no|

SQL> SELECT IF(1>0, 1=1, 1=0), IF(1<0, 2>1, 1<>1) FROM dual;

EXPR1 | EXPR2 |
-------------------------------------------------------------
T | F |
```

13.24 INET_ATON

功能描述

IPV4 转整数。

与 MySQL 无差异。

语法格式

```
INET_ATON(expr)
```

参数说明

expr：IP。

> **说明**
>
> 参数取值范围：[0.0.0.0, 255.255.255.255]

函数返回类型

BIGINT 类型。

> **说明**
>
> 返回值范围：[0, 4294967295]

示例

```
SQL> SELECT INET_ATON('192.168.2.218');

EXPR1 |
-------------------------------------------------------------
3232236250 |
```

13.25 INET_NTOA

功能描述

整数转 IPV4。

与 MySQL 无差异。

语法格式

```
INET_NTOA(expr)
```

参数说明

expr：整数。

> **说明**
>
> 参数取值范围：[0, 4294967295]

函数返回类型

CHAR 类型。

> **说明**
>
> 返回值范围：[0.0.0.0, 255.255.255.255]

示例

```
SQL> SELECT INET_NTOA('4294967295');

EXPR1 |
------------------------------------------------------------------
255.255.255.255|
```

13.26 JSON_VALUE

功能描述

在给定的路径中获取一个值，并可选择返回类型将其转换返回。

语法格式

```
JSON_VALUE ( json_doc, path [RETURNING type] [on_empty] [on_error])
on_empty:
{ NULL | ERROR | DEFAULT value } ON EMPTY
on_error:
```

```
{ NULL | ERROR | DEFAULT value } ON ERROR
```

参数说明

- json_doc：一个有效的 JSON 文档，如果为 NULL，表达式则返回 NULL。
- path：JSON 路径，必须为字符串, 如果为 NULL，表达式则返回 null。
- type：可以是以下虚谷基础类型。
  - CHAR
  - VARCHAR
  - BOOLEAN
  - TINYINT
  - SMALLINT
  - INTEGER
  - BIGINT
  - FLOAT
  - DOUBLE
  - NUMERIC
  - DATE
  - TIME
  - DATETIME
  - JSON
- on_empty：如果在表达式使用，则当表达式没有查找到数据时指定表达式的行为，可以是如下列表。
  - NULL ON EMPTY：返回 NULL；这也是默认的 ON EMPTY 行为。
  - DEFAULT value ON EMPTY：返回 value 值，且 value 值需与返回类型转换。
  - ERROR ON EMPTY：表达式抛出错误。
- on_error：如果在表达式使用，则当表达式出现错误时指定表达式的行为，可以是如下列

表。

- NULL ON ERROR：返回 NULL；这也是默认的 ON ERROR 行为。
- DEFAULT value ON ERROR：返回 value 值，且 value 值需与返回类型转换。
- ERROR ON ERROR：表达式抛出错误。

备注

ON EMPTY 如果使用，则必须在 ON_ERROR 子句之前，顺序错误会导致表达式抛出语法错误。

错误处理：

当 json_doc 或 jsonpath 其中一个无效，则 SQL 错误抛出而不触发 ON ERROR 规则。

ON ERROR 触发规则：

- 尝试提取对象或数组，例如从 JSON 多个路径位置生成对象和数组。
- 转换错误，例如字符串”abc” 转换为数值型。
- 值的截断。

转换错误总会发送警告消息，即使指定不抛出错误，例如： default value | null on error。

ON EMPTY 触发规则：

当按照 path 路径无法找到值时触发。

13.27 LEAST

功能描述

返回一组表达式中的最小值。

语法格式

```
LEAST(expr1, expr2, ...exprn)
```

参数说明

expr1,2,...,n：同数据类型。

函数返回类型

同数据类型。

示例

```
SQL> SELECT LEAST(2, 5, 3) colo1, LEAST(2, '5', 3) col2, LEAST('b'
   , 'B', 'a') col3, LEAST('b', 'cB', 'a') col4,LEAST(sysdate,
```

```
   sysdate+1) col5,LEAST('中', '华', '人') col6,LEAST(NULL, 'a', 'b'
   ) col7  FROM dual;

COLO1(TINYINT) | COL2(BIGINT) | COL3(CHAR(-1)) | COL4(CHAR(-1)) |
   COL5(DATETIME) | COL6(CHAR(-1)) | COL7(CHAR(-1)) |
-----------------------------------------------------------------
2 | 2 | a| a| 2022-06-24 14:14:26.624  | 中| <NULL>|
```

13.28 MAX_NODE_NUM

功能描述

查询 cluster.ini 中 MAX_NODES 值。

语法格式

```
MAX_NODE_NUM
```

参数说明

无参数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT MAX_NODE_NUM  FROM dual;

EXPR1(INTEGER) |
---------------------------------------------------------------
32 |
```

13.29 MEMBER OF

功能描述

- 如果 json_doc 是 JSON 数组，则判断 value 值是否为 JSON 数组的元素，是则返回 1，不是则返回 0。
- 如果 json_doc 不是 JSON 数组，则判断 value 值与 JSON 值是否一致，一致返回 1，不一致返回 0，如果 value 或 json_value 是 NULL，表达式则返回 NULL。

语法格式

```
value MEMBER OF(json_doc)
```

参数说明

- value：类型取值为 NULL、布尔型、数值型、自定义类型（object、varray、table）、JSON 类型以及其他能转换成字符型的类型。
- json_doc: JSON 类型或者可转换为 JSON 类型的数据类型。

13.30 NEWID

功能描述

产生一个全球唯一的标识。

语法格式

```
NEWID
```

参数说明

无参数。

函数返回类型

CHAR 类型字符串。

示例

```
SQL> CREATE TABLE test_newid(a INT);

SQL> INSERT INTO test_newid VALUES(1)(2)(3);

SQL> SELECT NEWID FROM test_newid;

EXPR1 |
------------------------------------------------------------
A1A486CD-04B1-4C83-B9AE-7AD8AE15EC4C|
6C9D4623-8E14-464D-8BF3-D84EF0255C39|
269B968C-4D15-4107-98D2-EDB759DFEDEE|
```

13.31 NUM_NONNULLS

功能描述

统计参数列表中非空值（NULL）的数量。

语法格式

```
NUM_NONNULLS(expr1[,expr2...])
```

参数说明

expr1：数据库支持的所有数据类型，且至少传入 1 个参数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT num_nonnulls(1,NULL,'abc',NULL,'') FROM dual;
EXPR1 |
------------------------------------------------------------
3 |
```

13.32 NUM_NULLS

功能描述

统计参数列表中的空值（NULL）数量。

语法格式

```
NUM_NULLS(expr1[,expr2...])
```

参数说明

expr1：数据库支持的所有数据类型，且至少传入 1 个参数。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT num_nulls(1,NULL,'abc',NULL,'') FROM dual;
EXPR1 |
------------------------------------------------------------
2 |
```

13.33 RENAME_FILE

功能描述

重命名数据库文件。

语法格式

```
RENAME_FILE(expr1,expr2)
```

参数说明

- expr1：原文件名。
- expr2：新文件名。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT  RENAME_FILE('/BACKUP/BACK_DB_FILE.exp','/BACKUP/
   BACK_DB_FILE1.exp');

EXPR1 |
------------------------------------------------------------
T |
```

13.34 SLEEP

功能描述

执行挂起一段时间。

语法格式

```
SLEEP(expr)
```

参数说明

expr：挂起时间。

函数返回类型

INTEGER 类型。

示例

```
SQL> SELECT SLEEP(10000) FROM dual;

EXPR1 |
------------------------------------------------------------
10000 |
```

13.35 SQLERRM

功能描述

返回与参数指定的错误代码相关的错误消息。

语法格式

```
SQLERRM [ ( expr ) ]
```

参数说明

expr：代表错误编号的整数；如“Error: [E19138 L1 C8] 时间值常数错误”中的编号 19138。

> 说明
>
> 如果不指定参数，则函数结果为 NULL。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT SQLERRM(19138);

EXPR1 |
------------------------------------------------------------
ERR-19138: 时 间 值 常 数 错 误 |
```

13.36 SYS_CONTEXT

功能描述

返回与上下文命名空间相关的参数值。

语法格式

```
SYS_CONTEXT(expr1, expr2)
```

参数说明

- expr1：当前只能为'USERENV'。
- expr2：当前支持的值包括以下列项。
  - 'LANGUAGE' - 当前会话所连接数据库的时区和字符集。
  - 'SESSIONID' - 当前会话的会话 ID。
  - 'SESSION_USERID' - 当前会话的用户 ID。
  - 'SESSION_USER' - 当前会话的用户名。
  - 'IP_ADDRESS' - 连接客户端的机器的 IP 地址。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT SYS_CONTEXT('USERENV', 'LANGUAGE') FROM dual;

EXPR1 |
```

```
-------------------------------------------------------------
GMT+08:00.utf8|

SQL> SELECT SYS_CONTEXT('USERENV', 'SESSION_USER') FROM dual;

EXPR1 |
-------------------------------------------------------------
U1|

SQL> SELECT SYS_CONTEXT('USERENV', 'IP_ADDRESS') FROM dual;

EXPR1 |
-------------------------------------------------------------
192.168.2.223|
```

13.37 SYS_GUID

功能描述

返回一个全局唯一的标识符。

语法格式

```
SYS_GUID
```

参数说明

无参数。

函数返回类型

GUID 类型。

示例

```
SQL> SELECT SYS_GUID FROM dual;

EXPR1 |
-------------------------------------------------------------
<2EAF7CB7CC1E441B920F3EAFC64493DB> |
```

13.38 SYS_USERID

功能描述

返回建立当前连接时的登录用户的 ID。

语法格式

```
SYS_USERID
```

参数说明

无参数。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT SYS_USERID FROM dual;

EXPR1 |
------------------------------------------------------------
1 |

SQL> CREATE USER TEST_USER1 IDENTIFIED BY '123QWE###';

SQL> SET SESSION AUTHORIZATION TEST_USER1;

SQL> SELECT SYS_USERID FROM dual;

EXPR1 |
------------------------------------------------------------
1 |
```

13.39 SYS_UUID

功能描述

通用唯一识别码，通过算法生成全局唯一 ID。

语法格式

```
SYS_UUID
```

参数说明

无参数。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT SYS_UUID FROM dual;

EXPR1 |
------------------------------------------------------------
7E63A51F-A16D-FFC1-B618-D41EF2DA6618|
```

13.40 SYSTEM_USER

功能描述

返回建立当前连接时的登录用户。

语法格式

```
SYSTEM_USER
```

参数说明

无参数。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT SYSTEM_USER FROM dual;

EXPR1 |
------------------------------------------------------------
SYSDBA|

SQL> CREATE USER TEST_USER2 IDENTIFIED BY '123QWE###';

SQL> SET SESSION AUTHORIZATION TEST_USER2;

SQL> SELECT SYSTEM_USER FROM dual;

EXPR1 |
------------------------------------------------------------
SYSDBA|
```

13.41 UID

功能描述

取当前用户（调用者）的 ID。

语法格式

```
UID()
```

参数说明

无参数。

函数返回类型

INTEGER 数值类型。

示例

```
SQL> SELECT UID() FROM dual;

EXPR1 |
------------------------------------------------------------
1 |
```

13.42 USER

功能描述

同 CURRENT_USER，返回当前用户。

语法格式

```
USER
```

参数说明

无参数。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT USER;

EXPR1 |
------------------------------------------------------------
SYSDBA|
```

13.43 UUID

功能描述

通用唯一识别码，通过算法生成全局唯一 ID。

与 MySQL 无差异。

语法格式

```
UUID
```

参数说明

无参数。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT UUID FROM dual;

EXPR1 |
------------------------------------------------------------------
8063A51F-B048-03C2-B618-D41EF2DA6618|

SQL> SELECT LENGTH(UUID) FROM dual;

EXPR1 |
------------------------------------------------------------------
36 |
```

13.44 USERENV

功能描述

获取连接用户关键信息。

语法格式

```
USERENV(expr)
```

参数说明

expr：要查看的信息名，当前支持的值包括以下列项。

- 'SID'：当前会话的会话 ID。
- 'SESSIONID'：当前会话的会话 ID。
- 'SESSION_USER'：当前会话的用户名。
- 'SESSION_USERID'：当前会话的用户 ID。
- 'CURRENT_USER'：当前会话的用户名。
- 'CURRENT_USERID'：当前会话的用户 ID。
- 'IP_ADDRESS'：连接客户端的机器的 IP 地址。
- 'ISDBA'：当前登陆用户是否为 DBA。
- 'LANG'：当前会话所连接数据库的字符集。
- 'LANGUAGE'：当前会话所连接数据库的时区和字符集。
- 'PID'：当前线程的线程 ID。

> **说明**
>
> - SID 和 SESSIONID 取值相同。
> - SESSION_USER 和 CURRENT_USER 取值相同。
> - SESSION_USERID 和 CURRENT_USERID 取值相同。

函数返回类型

VARCHAR 类型字符串。

示例

```
SQL> SELECT USERENV('LANGUAGE'),USERENV('SESSIONID'),USERENV('
   SESSION_USERID'),USERENV('SESSION_USER'),USERENV('IP_ADDRESS')
   FROM dual;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 |
-------------------------------------------------------------
GMT+08:00.gbk| 44| 1| SYSDBA| 192.168.2.116|
```

13.45 VERSION

功能描述

同 SHOW VERSION，返回当前数据库版本。

与 MySQL 无差异。

语法格式

```
VERSION()
```

参数说明

无参数。

函数返回类型

CHAR 类型。

示例

```
SQL> SELECT VERSION();

EXPR1 |
-------------------------------------------------------------
XuGu SQL Server 12.0.0|
```

14 特殊操作符

14.1 概述

函数形式	功能
+	加运算
-	减运算
||	字符串拼接
->	返回对应路径数据
->>	返回对应路径数据并取消对 JSON 类型的引用
&	将两个位操作数做按位与计算
|	将两个位操作数做按位或计算
^	将两个位操作数做按位异或计算
<<	将位操作数左移指定位数
>>	将位操作数右移指定位数
~	将操作数做按位非运算

14.2 +

功能描述

加运算。

数据类型

以下是一些常见的数据类型使用方式。

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
INTEGER	INTEGER	INTEGER
DATETIME	INTERGE	DATETIME
FLOAT	FLOAT	DOUBLE

完整的数据返回类型请在数据库中执行命令“SELECT*FROM sys_operators WHERE NAME = '+';”进行查看。

示例

- 示例 1

  算术运算，计算数值 5 加上数值 2 的结果。

```
--返 回 类 型 为INT
SQL> SELECT 5 + 2;

EXPR1 |
------------------------------------------------------------------
7 |
```

- 示例 2

  计算当前时间（以 2024/9/23 16:48 为例）再过 30 天后的时间。

```
-- 返 回 类 型 为DATETIME
SQL> SELECT CURRENT_TIMESTAMP + 30 ;

EXPR1 |
------------------------------------------------------------------
2024-10-23 16:48:43.869 AD |
```

- 示例 3

  浮点数运算，将数值 1.5 和 1.1 显式地转换为 FLOAT 类型，然后进行加法运算。

```
--返 回 类 型 为DOUBLE
SQL> SELECT 1.5::FLOAT + 1.1::FLOAT;

EXPR1 |
------------------------------------------------------------------
2.6 |
```0
```
```

14.3 -

功能描述

减运算。

数据类型

以下是一些常见的数据类型使用方式。

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
INTEGER	INTEGER	INTEGER
DATETIME	DATETIME	NUMERIC
FLOAT	FLOAT	DOUBLE

完整的数据返回类型请在数据库中执行命令“SELECT*FROM sys_operators WHERE NAME = '-';”进行查看。

示例

- 示例 1

  算术运算，计算数值 10 减去数值 2 的结果。

```
--返回类型为INT
SQL> SELECT 10 - 2;

EXPR1 |
------------------------------------------------------------------
8 |
```

- 示例 2

  查询当前时间（以 2024/9/23 15:36 为例）与 2019/8/2 15:30 之间的时间差。

```
--  返回类型为NUMERIC
SQL> SELECT CURRENT_TIMESTAMP - TO_DATE('2019-08-02 15:30:00','
   YYYY-MM-DD HH24:MI:SS');

EXPR1 |
------------------------------------------------------------------
1879.004339058553241|
```

- 示例 3

  浮点数运算，将数值 1.2 和 1.1 显式地转换为 FLOAT 类型，然后进行减法运算。

```
--返回类型为DOUBLE
SQL> SELECT 1.2::FLOAT - 1.1::FLOAT;

EXPR1 |
------------------------------------------------------------
0.1 |
```0
```

14.4 ||

功能描述

字符串拼接。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
CHAR	CHAR	CHAR
VARCHAR	VARCHAR	VARCHAR

示例

- 示例 1

  连接 2019 和 ABC。

```
SQL> SELECT 2019 || 'ABC' FROM DUAL;

EXPR1 |
------------------------------------------------------------
2019ABC|
```

- 示例 2

  连接多个字符串。

```
SQL> SELECT 'xxxx' || 'db' || ' and ' || 'xxxx' FROM DUAL;

EXPR1 |
------------------------------------------------------------
xxxxdb and xxxx|
```
```

14.5 ->

功能描述

返回对应路径数据。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
JSON	CHAR(JSONPath)	JSON

JSONPath 的详细内容请参见《SQL 语法参考》的数据类型 > JSON 数据类型章节。

示例

查询 JSON 数组 [1,2,”abc”] 中的第 3 个元素（索引从 0 开始）。

```
SQL> SELECT '[1,2,"abc"]' -> '$[2]';

EXPR1 |
------------------------------------------------------------
"abc"|
```

14.6 ->>

功能描述

返回对应路径数据并取消对 JSON 类型的引用。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
JSON	CHAR(JSONPath)	JSON

JSONPath 的详细内容请参见《SQL 语法参考》的数据类型 > JSON 数据类型章节。

示例

查询 JSON 数组 [1,2,”abc”] 中的第 3 个元素（索引从 0 开始），并将其转换为文本类型。

```
SQL> SELECT '[1,2,"abc"]' ->> '$[2]';
```

```
EXPR1 |
------------------------------------------------------------
abc|
```

14.7 &（按位与）

功能描述

将两个位操作数做按位与计算。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
VARBIT	VARBIT	VARBIT
BIGINT	BIGINT	BIGINT

示例

- 示例 1

  对二进制字符串 101 和 1011 进行按位与运算。

```
-- VARBIT 类 型
SQL> SELECT b'101' & b'1011';

EXPR1 |
------------------------------------------------------------
0001 |
```

> **说明**
>
> 当两个操作数为 VARBIT 类型且位数不等时，对较短的操作数左侧补零，再执行计算。

- 示例 2

  对 BITINT 类型数据进行按位与运算。

```
-- BIGINT 类 型
SQL> SELECT 9223372036854775800 & 9223372036854775705;

EXPR1 |
------------------------------------------------------------
9223372036854775704 |
```

- 示例 3

  隐式转换，支持的数据类型包括：TINYINT、SMALLINT、INT、FLOAT、DOUBLE、STR、NUMERIC。

```
-- 字符串'1'隐式转换为整数1，然后执行按位与运算
SQL> SELECT '1' & 2;

EXPR1 |
------------------------------------------------------------------
0 |

-- 浮点数1.5隐式转换为整数2，然后执行按位与运算
SQL> SELECT 1.5 & 2;

EXPR1 |
------------------------------------------------------------------
2 |

-- 更多类型转换示例
SQL> CREATE TABLE tab_tt(c1 TINYINT, c2 SMALLINT, c3 INT, c4
   FLOAT, c5 DOUBLE, c6 VARCHAR, c7 NUMERIC);

SQL> INSERT INTO tab_tt VALUES(1, 2, 3, 4.3, 5.6, '4', 1.5);

SQL> SELECT c1 & c1, c2 & c2, c3 & c3, c4 & c4, c5 & c5, c6 & c6
   , c7 & c7 FROM tab_tt;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 | EXPR7 |
------------------------------------------------------------------
1 | 2 | 3 | 4 | 6 | 4 | 2 |
```

> **说明**
>
> BIGINT 类型的按位与运算支持隐式转换成 BIGINT 类型运算。

14.8 |（按位或）

功能描述

将两个位操作数做按位或计算。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
VARBIT	VARBIT	VARBIT
BIGINT	BIGINT	BIGINT

示例

- 示例 1

  对二进制字符串 101 和 1011 进行按位与或运算。

```
-- VARBIT类型
SQL> SELECT b'101' | b'1011';

EXPR1 |
------------------------------------------------------------
1111 |
```

> **说明**
>
> 当两个操作数为 VARBIT 类型且位数不等时，对较短的操作数左侧补零，再执行计算。

- 示例 2

  对 BITINT 类型数据进行按位与运算。

```
-- BIGINT类型
SQL> SELECT 9223372036854775800 | 9223372036854775705;

EXPR1 |
------------------------------------------------------------
9223372036854775801 |
```

- 示例 3

  隐式转换，支持的数据类型包括：TINYINT、SMALLINT、INT、FLOAT、DOUBLE、STR、NUMERIC。

```
-- 字符串'1'隐式转换为整数1，然后执行按位或运算
SQL> SELECT '1' | 2;

EXPR1 |
------------------------------------------------------------
3 |

-- 浮点数1.5隐式转换为整数2，然后执行按位或运算
```

```
SQL> SELECT 1.5 | 2;

EXPR1 |
------------------------------------------------------------
2 |

-- 更多类型转换示例
SQL> CREATE TABLE tab_tt(c1 TINYINT, c2 SMALLINT, c3 INT, c4
   FLOAT, c5 DOUBLE, c6 VARCHAR, c7 NUMERIC);

SQL> INSERT INTO tab_tt VALUES(1, 2, 3, 4.3, 5.6, '4', 1.5);

SQL> SELECT c1 | c1, c2 | c2, c3 | c3, c4 | c4, c5 | c5, c6 | c6
   , c7 | c7 FROM tab_tt;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 | EXPR7 |
------------------------------------------------------------
1 | 2 | 3 | 4 | 6 | 4 | 2 |
```

> 说明
>
> BIGINT 类型的按位或运算支持隐式转换成 BIGINT 类型运算。

14.9 ^（按位异或）

功能描述

将两个位操作数做按位异或计算。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
VARBIT	VARBIT	VARBIT
BIGINT	BIGINT	BIGINT

示例

- 示例 1

  对二进制字符串 101 和 1011 进行按位异或运算。

```
-- VARBIT类型
SQL> SELECT b'101' & b'1011';

EXPR1 |
```

```
------------------------------------------------------------------
1110 |
```

> **说明**
>
> 当两个操作数为 VARBIT 类型且位数不等时，对较短的操作数左侧补零，再执行计算。

- 示例 2

  对 BITINT 类型数据进行按位异或运算。

```
-- BIGINT类型
SQL> SELECT 9223372036854775800 ^ 9223372036854775705;

EXPR1 |
------------------------------------------------------------------
97 |
```

- 示例 3

  隐式转换，支持的数据类型包括：TINYINT、SMALLINT、INT、FLOAT、DOUBLE、STR、NUMERIC。

```
-- 字符串'1'隐式转换为整数1，然后执行按位与运算
SQL> SELECT '1' ^ 2;

EXPR1 |
------------------------------------------------------------------
3 |

-- 浮点数1.5隐式转换为整数2，然后执行按位与运算
SQL> SELECT 1.5 & 2;

EXPR1 |
------------------------------------------------------------------
0 |

-- 更多类型转换示例
SQL> CREATE TABLE tab_tt(c1 TINYINT, c2 SMALLINT, c3 INT, c4
   FLOAT, c5 DOUBLE, c6 VARCHAR, c7 NUMERIC);

SQL> INSERT INTO tab_tt VALUES(1, 2, 3, 4.3, 5.6, '4', 1.5);

SQL> SELECT c1 ^ c2, c2 ^ c2, c3 ^ c4, c4 ^ c4, c5 ^ c5, c6 ^ c6
   , c7 ^ c7 FROM tab_tt;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 | EXPR7 |
------------------------------------------------------------------
3 | 0 | 7 | 0 | 0 | 0 | 0 |
```

> 说明
>
> BIGINT 类型的按位异或运算支持隐式转换成 BIGINT 类型运算。

14.10 <<（位左移）

功能描述

将位操作数左移指定位数。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
VARBIT	VARBIT	VARBIT
BIGINT	BIGINT	BIGINT

示例

- 示例 1

  对二进制字符串 1011 位左移 2。

```
-- VARBIT 类 型
SQL> SELECT b'1011' << 2;

EXPR1 |
----------------------------------------------------------------
1100 |
```

> 说明
>
> 当移位数负数时，将往反方向移位。

- 示例 2

  对 BITINT 类型数据进行位左移运算。

```
-- BIGINT 类 型
SQL> SELECT 1 << 2;

EXPR1 |
----------------------------------------------------------------
4 |
```

- 示例 3

  隐式转换，支持的数据类型包括：TINYINT、SMALLINT、INT、FLOAT、DOUBLE、STR、NUMERIC。

```
-- 字符串'1'隐式转换为整数1，然后执行位左移运算
SQL> SELECT '1' << 2;

EXPR1 |
------------------------------------------------------------------
4 |

-- 浮点数1.5隐式转换为整数2，然后执行位左移运算
SQL> SELECT 1.5 << 2;

EXPR1 |
------------------------------------------------------------------
8 |

-- 更多类型转换示例
SQL> CREATE TABLE tab_tt(c1 TINYINT, c2 SMALLINT, c3 INT, c4
   FLOAT, c5 DOUBLE, c6 VARCHAR, c7 NUMERIC);

SQL> INSERT INTO tab_tt VALUES(1, 2, 3, 4.3, 5.6, '4', 1.5);

SQL> SELECT c1 << c1, c2 << c2, c3 << c4, c4 << c4, c5 << c5, c6
    << c6, c7 << c7 FROM tab_tt;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 | EXPR7 |
------------------------------------------------------------------
2 | 8 | 48 | 64 | 384 | 64 | 8 |
```

> **说明**
>
> BIGINT 类型的按位左移运算支持隐式转换成 BIGINT 类型运算。

14.11 >>（位右移）

功能描述

将位操作数右移指定位数。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
VARBIT	VARBIT	VARBIT
BIGINT	BIGINT	BIGINT

示例

- 示例 1

  对二进制字符串 1011 位右移 2。

```
-- VARBIT类型
SQL> SELECT b'1011' >> 2;

EXPR1 |
------------------------------------------------------------
0010 |
```

> **说明**
>
> 当移位数负数时，将往反方向移位。

- 示例 2

  对 BITINT 类型数据进行位右移运算。

```
-- BIGINT类型
SQL> SELECT 8 >> 2;

EXPR1 |
------------------------------------------------------------
2 |
```

- 示例 3

  隐式转换，支持的数据类型包括：TINYINT、SMALLINT、INT、FLOAT、DOUBLE、STR、NUMERIC。

```
-- 字符串'8'隐式转换为整数1，然后执行位右移运算
SQL> SELECT '8' >> 2;

EXPR1 |
------------------------------------------------------------
2 |

-- 浮点数6.5隐式转换为整数7，然后执行位右移运算
```

```
SQL> SELECT 6.5 >> 2;

EXPR1 |
------------------------------------------------------------
1 |

-- 更多类型转换示例
SQL> CREATE TABLE tab_tt(c1 TINYINT, c2 SMALLINT, c3 INT, c4
   FLOAT, c5 DOUBLE, c6 VARCHAR, c7 NUMERIC);

SQL> INSERT INTO tab_tt VALUES(1, 2, 3, 4.3, 5.6, '4', 1.5);

SQL> SELECT c1 >> c1, c2 >> c2, c3 >> c4, c4 >> c4, c5 >> c5, c6
    >> c6, c7 >> c7 FROM tab_tt;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 | EXPR7 |
------------------------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

> 说明
>
> BIGINT 类型的按位右移运算支持隐式转换成 BIGINT 类型运算。

14.12 ~（按位非）

功能描述

将操作数做按位非运算。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
无	VARBIT	VARBIT
无	BIGINT	BIGINT

示例

- 示例 1

  对二进制字符串 101 进行按位非运算。

```
-- VARBIT类型
SQL> SELECT ~ b'101';

EXPR1 |
```

```
-------------------------------------------------------------------
010 |
```

- 示例 2

  对 BITINT 类型数据进行按位非运算。

```
-- BIGINT类型
SQL> SELECT ~ 1;

EXPR1 |
-------------------------------------------------------------------
-2 |
```

- 示例 3

  隐式转换，支持的数据类型包括：TINYINT、SMALLINT、INT、FLOAT、DOUBLE、STR、NUMERIC。

```
-- 字符串'1'隐式转换为整数1，然后进行按位非运算。
SQL> SELECT ~ '1';

EXPR1 |
-------------------------------------------------------------------
-2 |

-- 浮点数6.5隐式转换为整数7，然后进行按位非运算。
SQL> SELECT ~ 6.5;

EXPR1 |
-------------------------------------------------------------------
-8 |

-- 更多类型转换示例
SQL> CREATE TABLE tab_tt(c1 TINYINT, c2 SMALLINT, c3 INT, c4
   FLOAT, c5 DOUBLE, c6 VARCHAR, c7 NUMERIC);

SQL> INSERT INTO tab_tt VALUES(1, 2, 3, 4.3, 5.6, '4', 1.5);

SQL> SELECT ~ c1, ~ c2, ~ c3, ~ c4, ~ c5, ~ c6, ~ c7 FROM tab_tt;

EXPR1 | EXPR2 | EXPR3 | EXPR4 | EXPR5 | EXPR6 | EXPR7 |
-------------------------------------------------------------------
-2 | -3 | -4 | -5 | -7 | -5 | -3 |
```

> **说明**
>
> BIGINT 类型的按位非运算支持隐式转换成 BIGINT 类型运算，对于非位运算，返回结果支持有符号数。

15 几何函数

15.1 概述

函数形式	功能
AREA	计算几何对象面积
CENTER	返回几何对象的中心点
DIAGONAL	提取 BOX 类型几何对象对角线作为线段返回（与 LSEG(BOX) 相同）
DIAMETER	返回 CIRCLE 类型几何对象的直径
HEIGHT	返回 BOX 类型几何对象的竖直高度
ISCLOSED	判断 PATH 类型几何对象是否封闭
ISOPEN	判断 PATH 类型几何对象是否开放
LENGTH	计算线段或路径总长度
NPOINTS	返回路径或多边形点的数量
PCLOSE	将 PATH 类型几何对象转换为封闭形式
POPEN	将 PATH 类型几何对象转换为开放形式
RADIUS	计算 CIRCLE 类型几何对象的半径
SLOPE	计算通过两个点类型几何对象所画直线的斜率
WIDTH	返回 BOX 类型几何对象的水平宽度

15.2 AREA

功能描述

计算几何对象面积。适用于 CIRCLE、BOX、PATH 类型几何对象。

封闭类型几何对象计算面积，返回实际结果。开放类型几何对象计算面积，返回 NULL。

语法格式

```
AREA(expr)
```

参数说明

expr：目标 CIRCLE、BOX 或 PATH 对象，为 CIRCLE、BOX 或 PATH 类型。

函数返回类型

DOUBLE 类型。

示例

CIRCLE：

```
SQL> SELECT AREA(CIRCLE('((5,0),1)'));

EXPR1 |
------------------------------------------------------------
3.141593e+00 |

Total 1 records.
```

BOX：

```
SQL> SELECT AREA(BOX('(2,2),(0,0)'));

EXPR1 |
------------------------------------------------------------
4.000000e+00 |

Total 1 records.
```

PATH：

- 封闭路径计算面积，返回实际结果。

```
SQL> SELECT AREA(PATH('((0,0),(1,1),(2,0))') );

EXPR1 |
------------------------------------------------------------
1.000000e+00 |

Total 1 records.
```

- 开放路径计算面积，返回 NULL。

```
SQL> SELECT AREA(PATH('[(0,0),(1,1),(2,0)]') );

EXPR1 |
------------------------------------------------------------
<NULL>|
```

```
Total 1 records.
```

15.3 CENTER

功能描述

返回几何对象的中心点。适用于 BOX、CIRCLE 类型几何对象。

语法格式

```
CENTER(expr)
```

参数说明

expr：目标 BOX 或 CIRCLE 对象，为 BOX 或 CIRCLE 类型。

函数返回类型

POINT 类型。

示例

BOX：

```
SELECT CENTER(CIRCLE('((5,2),1)'));
EXPR1     |
----------+
(5.0,2.0)|
```

CIRCLE：

```
SELECT CENTER(CIRCLE('((5,2),1)'));
EXPR1     |
----------+
(5.0,2.0)|
```

15.4 DIAGONAL

功能描述

提取 BOX 的对角线作为线段返回（与 LSEG(BOX) 相同）。

语法格式

```
DIAGONAL (expr)
```

参数说明

expr：BOX 类型几何对象。

函数返回类型

LSEG 类型。

示例

```
SELECT DIAGONAL(BOX('(1,2),(0,0)'));
EXPR1                |
---------------------+
[(1.0,2.0),(0.0,0.0)]|
```

15.5 DIAMETER

功能描述

返回 CIRCLE 类型几何对象的直径。

语法格式

```
DIAMETER(expr)
```

参数说明

expr：CIRCLE 类型几何对象。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT DIAMETER(CIRCLE( '<(0,0),2>')) ;

EXPR1 |
------------------------------------------------------------
4.000000e+00 |

Total 1 records.
```

15.6 HEIGHT

功能描述

返回 BOX 类型几何对象的竖直高度。

语法格式

```
HEIGHT(expr)
```

参数说明

expr：BOX 类型几何对象。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT HEIGHT(BOX('(1,2),(0,0)'));

EXPR1 |
------------------------------------------------------------
2.000000e+00 |

Total 1 records.
```

15.7 ISCLOSED

功能描述

判断 PATH 类型几何对象是否封闭。

- 若路径封闭，返回 TRUE。
- 若路径开放，返回 FALSE。

语法格式

```
ISCLOSED(expr)
```

参数说明

expr：PATH 类型几何对象。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT ISCLOSED(PATH('((0,0),(1,1),(2,0))'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

15.8 ISOPEN

功能描述

判断 PATH 类型几何对象是否开放。

- 若路径开放，返回 TRUE。

- 若路径封闭，返回 FALSE。

语法格式

```
ISOPEN(expr)
```

参数说明

expr：PATH 类型几何对象。

函数返回类型

BOOLEAN 类型。

示例

```
SQL> SELECT ISOPEN(PATH('[(0,0),(1,1),(2,0)]'));

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

15.9 LENGTH

功能描述

计算线段或路径总长度。适用于 LSEG、PATH 类型几何对象。

语法格式

```
LENGTH(expr)
```

参数说明

expr：LSEG、PATH 类型几何对象。

函数返回类型

DOUBLE 类型。

示例

LSEG：

```
SQL> SELECT LENGTH(LSEG('((0,0),(1,1))'));

EXPR1 |
------------------------------------------------------------------
1.414214e+00 |

Total 1 records.
```

PATH：

```
SQL> SELECT LENGTH(PATH('[(0,0),(1,1),(2,0)]'));

EXPR1 |
------------------------------------------------------------
2.828427e+00 |

Total 1 records.
```

15.10 NPOINTS

功能描述

返回路径或多边形点的数量。适用于 PATH、POLYGON 类型几何对象。

语法格式

```
NPOINTS(expr)
```

参数说明

expr：PATH、POLYGON 类型几何对象。

函数返回类型

INTEGER 类型。

示例

PATH：

```
SQL> SELECT NPOINTS(PATH('[(0,0),(1,1),(2,0)]'));

EXPR1 |
------------------------------------------------------------
3 |

Total 1 records.
```

POLYGON：

```
SQL> SELECT NPOINTS(POLYGON('
   ( 10, 190, 10, 70, 80, 70, 80, 130, 50, 160, 120, 160, 120, 190, 10, 190
   '));

EXPR1 |
------------------------------------------------------------
8 |

Total 1 records.
```

15.11 PCLOSE

功能描述

将 PATH 类型几何对象转换为封闭形式。

语法格式

```
PCLOSE(expr)
```

参数说明

expr：PATH 类型几何对象。

函数返回类型

PATH 类型。

示例

```
SELECT PCLOSE(PATH('[(0,0),(1,1),(2,0)]'));
EXPR1                          |
-------------------------------+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

15.12 POPEN(PATH)

功能描述

将 PATH 类型几何对象转换为开放形式。

语法格式

```
POPEN(expr)
```

参数说明

expr：PATH 类型几何对象。

函数返回类型

PATH 类型。

示例

```
SELECT POPEN(PATH('((0,0),(1,1),(2,0))'));
EXPR1                          |
-------------------------------+
[(0.0,0.0),(1.0,1.0),(2.0,0.0)]|
```

15.13 RADIUS

功能描述

计算 CIRCLE 类型几何对象的半径。

语法格式

```
RADIUS(expr)
```

参数说明

expr：目标 CIRCLE 对象，为 CIRCLE 类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT RADIUS(CIRCLE('((5,2),1)'));

EXPR1 |
------------------------------------------------------------
1.000000e+00 |

Total 1 records.
```

15.14 SLOPE

功能描述

计算通过两个点类型几何对象所画直线的斜率。

语法格式

```
SLOPE(expr1, expr2)
```

参数说明

expr1、expr2：目标 POINT 对象，为 POINT 类型。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT SLOPE(POINT('(0,0)'), POINT('(2,1)'));

EXPR1 |
------------------------------------------------------------
5.000000e-01 |
```

```
Total 1 records.
```

15.15 WIDTH

功能描述

返回 BOX 类型几何对象的水平宽度。

语法格式

```
WIDTH(expr)
```

参数说明

expr：BOX 类型几何对象。

函数返回类型

DOUBLE 类型。

示例

```
SQL> SELECT WIDTH(BOX('(1,2),(0,0)'));

EXPR1 |
------------------------------------------------------------
1.000000e+00 |

Total 1 records.
```

16 几何类型转换函数

16.1 概述

函数形式	功能
BOX	将目标对象转换为 BOX 类型几何对象
BOUND_BOX	计算两个 BOX 类型几何对象的边界框
CIRCLE	将目标对象转换为 CIRCLE 类型几何对象
LINE	将两点或字符串转换一条直线
LSEG	将目标对象转换为 LSEG 类型几何对象
PATH	将目标对象转换为 PATH 类型几何对象
POINT	将目标对象转换为 POINT 类型几何对象，计算几何对象的中心
POLYGON	将目标对象转换为 POLYGON 类型几何对象

16.2 BOUND_BOX

功能描述

计算两个方框的边界框。

语法格式

```
BOUND_BOX(expr1, expr2)
```

参数说明

expr1、expr2：BOX 类型几何对象。

函数返回类型

BOX 类型。

示例

```
SELECT BOUND_BOX(BOX('(1,1),(0,0)'), BOX('(4,4),(3,3)'));
EXPR1               |
--------------------+
(4.0,4.0),(0.0,0.0)|
```

16.3 BOX(CIRCLE)

功能描述

根据输入的参数类型有以下功能：

- CIRCLE：将圆形转换为矩形，返回 CIRCLE 内切正方形的 BOX。
- POINT：将点转换为空框。
- POINT, POINT：将任意两个点转换为框。
- POLYGON：计算多边形的边界框。
- CHAR：将字符串转为 BOX。

语法格式

```
BOX(expr1 [, expr2])
```

参数说明

- expr1：几何对象或字符串。支持以下类型：
  - CIRCLE
  - POINT
  - POLYGON
  - CHAR
- expr2：仅适用于 POINT, POINT 的转换。

函数返回类型

BOX 类型。

示例

CIRCLE：

```
SELECT BOX(CIRCLE('<(0,0),1>'));
EXPR1
```

```
-------------------------------------------------------------------+
(0.7071067811865475,0.7071067811865475)
   ,(-0.7071067811865475,-0.7071067811865475)|
```

POINT：

```
SELECT BOX(POINT('(5,6)'));
EXPR1                |
---------------------+
(5.0,6.0),(5.0,6.0)|
```

POINT, POINT：

```
SELECT BOX(POINT('(0,1)'),POINT('(1,0)'));
EXPR1                |
---------------------+
(1.0,1.0),(0.0,0.0)|
```

POLYGON：

```
SELECT BOX(POLYGON('((0,0),(1,1),(2,0))'));
EXPR1                |
---------------------+
(2.0,1.0),(0.0,0.0)|
```

CHAR：

```
SELECT BOX('(0,0),(1,1)');
EXPR1                |
---------------------+
(1.0,1.0),(0.0,0.0)|
```

16.4 CIRCLE

功能描述

根据输入的参数类型有以下功能：

- BOX：计算最小的圆形包围框，返回边框的外接圆。
- POINT, DOUBLE：从圆心和半径构造圆。
- POLYGON：将多边形转换为圆。圆心是多边形各点位置的平均值，半径是多边形各点到圆心的平均距离。
- CIRCLE：将圆转换为圆。
- CHAR：将字符串转换为圆。

语法格式

```
CIRCLE(expr)
```

参数说明

- expr1：几何对象或字符串。支持以下类型：
  - BOX
  - POINT, DOUBLE
  - POLYGON
  - CIRCLE
  - CHAR
- expr2：目标半径，为 DOUBLE 类型，仅适用于 POINT, DOUBLE 的转换。

函数返回类型

CIRCLE 类型。

示例

BOX：

```
SELECT CIRCLE(BOX('(1,1),(-1,-1)'));
EXPR1                          |
-------------------------------+
<(0.0,0.0),1.4142135623730951>|
```

POINT, DOUBLE：

```
SELECT CIRCLE(POINT ('(3,4)'), 2.0);
EXPR1           |
----------------+
<(3.0,4.0),2.0>|
```

POLYGON：

```
SELECT CIRCLE(POLYGON('((0,0),(1,3),(2,0))'));
EXPR1                          |
-------------------------------+
<(1.0,1.0),1.6094757082487299>|
```

CIRCLE：

```
SELECT CIRCLE('<(3,4), 2.0>'::CIRCLE);
EXPR1           |
----------------+
<(3.0,4.0),2.0>|
```

CHAR：

```
SELECT CIRCLE('<(3,4), 2.0>');
EXPR1           |
----------------+
<(3.0,4.0),2.0>|
```

16.5 LINE

功能描述

将两点或字符串转换一条直线。

语法格式

```
LINE(expr1 [, expr2])
```

参数说明

- expr1：CHAR 类型字符串或 POINT 类型几何对象。
- expr2：POINT 类型几何对象，仅适用于 POINT, POINT 转换。

函数返回类型

LINE 类型。

示例

POINT, POINT：

```
SELECT LINE(POINT('(-1,0)'), POINT('(1,0)'));
EXPR1          |
---------------+
{0.0,-1.0,0.0}|
```

CHAR：

```
SELECT LINE('(-1,0),(1,0)');
EXPR1          |
---------------+
{0.0,-1.0,0.0}|
```

16.6 LSEG

功能描述

根据输入的参数类型有以下功能：

- BOX：提取方框的对角线作为线段。
- POINT, POINT：使用两个端点构造线段。

- CHAR：将字符串转换为线段。

语法格式

```
LSEG(expr1 [, expr2])
```

参数说明

- expr1：几何对象或字符串。支持以下类型：
  - BOX
  - POINT
  - CHAR
- expr2：POINT 类型几何对象，仅适用于 POINT, POINT 转换。

函数返回类型

LSEG 类型。

示例

BOX：

```
SELECT LSEG(BOX ('(1,0),(-1,0)')) ;
EXPR1                    |
-------------------------+
[(1.0,0.0),(-1.0,0.0)]|
```

POINT, POINT：

```
SELECT LSEG(BOX ('(1,0),(-1,0)')) ;
EXPR1                    |
-------------------------+
[(1.0,0.0),(-1.0,0.0)]|
```

CHAR：

```
SELECT LSEG('(1,0),(-1,0)') ;
EXPR1                    |
-------------------------+
[(1.0,0.0),(-1.0,0.0)]|
```

16.7 PATH

功能描述

根据输入的参数类型有以下功能：

- POLYGON：将多边形转换为具有相同点列表的闭合路径。

- PATH：将路径转换为路径。
- CHAR：将字符串转换为路径。

语法格式

```
PATH(expr)
```

参数说明

expr：几何对象或字符串。支持以下类型：

- POLYGON
- PATH
- CHAR

函数返回类型

PATH 类型。

示例

POLYGON：

```
SELECT PATH('((0,0),(1,1),(2,0))');
EXPR1                              |
-----------------------------------+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

PATH：

```
SELECT PATH('((0,0),(1,1),(2,0))'::PATH);
EXPR1                              |
-----------------------------------+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

CHAR：

```
SELECT PATH('((0,0),(1,1),(2,0))');
EXPR1                              |
-----------------------------------+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

16.8 POINT

功能描述

- 构造点
  - DOUBLE, DOUBLE：用坐标构造点。

  - CHAR：将字符串转换为点。

- 计算几何对象中心
  - BOX：计算 BOX 的中心。
  - CIRCLE：计算圆心。
  - LSEG：计算线段的中心。
  - POLYGON：计算多边形的中心（多边形的点位置的平均值）。

语法格式

```
POINT(expr1 [, expr2])
```

参数说明

- expr1：支持以下类型：
  - DOUBLE
  - CHAR
  - BOX
  - CIRCLE
  - LSEG
  - POLYGON
- expr2：仅适用于 DOUBLE, DOUBLE 转换，其中 expr1 是 X 值，expr2 是 Y 值，DOUBLE 类型。

函数返回类型

POINT 类型。

示例

DOUBLE, DOUBLE：

```
SELECT POINT(23.4, -44.5);
EXPR1        |
-------------+
(23.4,-44.5)|
```

CHAR：

```
SELECT POINT('23.4, -44.5');
EXPR1        |
-------------+
```

```
(23.4,-44.5)|
```

BOX：

```
SELECT POINT(BOX('(1,0),(-1,0)'));
EXPR1    |
---------+
(0.0,0.0)|
```

CIRCLE：

```
SELECT POINT(CIRCLE('<(0,0),2>'));
EXPR1    |
---------+
(0.0,0.0)|
```

LESG：

```
SELECT POINT(LSEG('[(-1,0),(1,0)]'));
EXPR1    |
---------+
(0.0,0.0)|
```

POLYGON：

```
SELECT POINT(POLYGON('((0,0),(1,1),(2,0))'));
EXPR1                    |
-------------------------+
(1.0,0.3333333333333333)|
```

16.9 POLYGON

功能描述

根据输入的参数类型有以下功能：

- BOX：将 BOX 转换为 4 点多边形。
- CIRCLE：将 CIRCLE 转换为 12 点多边形。
- INTEGER, CIRCLE：将 CIRCLE 转换为 N 点多边形。
- PATH：将封闭路径转换为具有相同点列表的多边形。
- CHAR：将字符串转换为多边形。

语法格式

```
POLYGON(expr)
```

参数说明

expr：目标几何对象或字符串。支持以下类型：

- BOX
- CIRCLE
- PATH
- CHAR

“INTEGER,CIRCLE”转换输入 2 个参数，其中 expr1 为点个数，为 INTEGER 类型。expr2 为 CIRCLE 类型几何对象。

函数返回类型

POLYGON 类型。

示例

BOX：

```
SELECT POLYGON(BOX('(1,1),(0,0)'));
EXPR1                                      |
-------------------------------------------+
((0.0,0.0),(0.0,1.0),(1.0,1.0),(1.0,0.0))|
```

CIRCLE：

```
SELECT POLYGON(CIRCLE('<(0,0),2>'));
EXPR1                                      |
-------------------------------------------+
((-2.0,0.0),(-1.7320508075688774,0.9999999999999999)
   ,(-1.0000000000000002,1.7320508075688772),(-1.2246467991473532E
   -16,2.0),(0.9999999999999996,1.7320508075688774)
   ,(1.732050807568877,1.0000000000000007),(2.0,2.4492935982947064E
   -16),(1.7320508075688776,-0.9999999999999994)
   ,(1.0000000000000009,-1.7320508075688767),(3.6739403974420594E
   -16,-2.0),(-0.9999999999999987,-1.732050807568878)
   ,(-1.7320508075688767,-1.0000000000000009))|
```

INTEGER, CIRCLE：

```
SELECT POLYGON(4, CIRCLE('<(3,0),1>'));
EXPR1                                                                |
---------------------------------------------------------------------+
((2.0,0.0),(3.0,1.0),(4.0,1.2246467991473532E-16),(3.0,-1.0))|
```

PATH：

```
SELECT POLYGON(PATH('(0,0),(1,1),(2,0)'));
EXPR1                            |
---------------------------------+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

CHAR：

```
SELECT POLYGON('(0,0),(1,1),(2,0)');
```

```
EXPR1                            |
---------------------------------+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

17 几何操作符

17.1 概述

函数形式	功能
+	• 平移，将第二个参数 POINT 的坐标添加到第一个对象的坐标中 • 连接两个打开的路径，如果其中一个路径是关闭的，则返回 NULL
-	平移，从第一个参数对象坐标中减去第二个 POINT 的坐标
*	伸展/旋转，将第一个参数的每个点乘以第二个 POINT
/	收缩/旋转，将第一个参数的每个点除以第二个 POINT
@-@	计算总长度
@@	计算中心点
#	• 返回点的数量 • 计算交点，如果没有交点则返回 NULL • 计算两个方框的交集，如果没有则为 NULL
##	计算第二个对象上离第一个对象最近的点
<->	计算对象之间的距离
	接下页

函数形式	功能
@>	判断第一个对象是否包含第二个对象
<@	判断第一个对象是否被包含于第二个对象
&&	返回多边形点的数量
«	判断第一个对象是否完全位于第二个对象的左边
»	判断第一个对象是否完全位于第二个对象的右侧
&<	判断第一个对象是否不超过第二个对象的右边
&>	判断第一个对象是否完全位于第二个对象的右边
«|	判断第一个对象是否完全位于第二个对象的下方
|»	判断第一个对象是否完全位于第二个对象的上方
&<|	判断第一个对象是否不超过第二个对象的上方
|&>	判断第一个对象是否不超过第二个对象的上方
<^	判断第一个对象是否位于第二个对象的下方，允许边缘相切
>^	判断第一个对象是否位于第二个对象的上方，允许边缘相切
?#	判断两个对象是否相交
?-|	判断两条线是否相互垂直
~=	判断两个对象是否一般相等
<>	判断两个对象是否不等
<	判断第一个对象的面积是否小于第二个对象的面积
>	判断第一个对象的面积是否大于第二个对象的面积

接下页

函数形式	功能
<=	判断第一个对象的面积是否小于等于第二个对象的面积
>=	判断第一个对象的面积是否大于等于第二个对象的面积
=	• 判断第一个对象的面积是否等于第二个对象的面积 • 判断第一个对象是否等于第二个对象

17.2 +

功能描述

+ 操作符在几何数据类型中有以下两种功能：

- 平移，将第二个参数 POINT 的坐标添加到第一个对象的坐标中。适用于 POINT、BOX、PATH、CIRCLE。
- 连接两个打开的路径，如果其中一个路径是关闭的，则返回 NULL。适用于 PATH。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	POINT
BOX	POINT	BOX
PATH	POINT	PATH
CIRCLE	POINT	CIRCLE
PATH（连接两个路径）	PATH	PATH

示例

平移：

- 平移 POINT

```
SELECT POINT('(2.0,0)') + POINT('(0,2.0)');
EXPR1    |
---------+
(2.0,2.0)|
```

- 平移 BOX

```
SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) + POINT('(0,2.0)');
EXPR1               |
--------------------+
(1.0,3.0),(0.0,2.0)|
```

- 平移 PATH

```
SELECT PATH('[(0,0),(1,1)]') + POINT('(0,2.0)');
EXPR1                  |
-----------------------+
[(0.0,2.0),(1.0,3.0)]|
```

- 平移 CIRCLE

```
SELECT CIRCLE('<(3,4),2>') + POINT('(1,2)');
EXPR1           |
----------------+
<(4.0,6.0),2.0>|
```

连接路径：

- 连接两个打开的路径

```
SELECT PATH('[(0,0),(1,1)]') + PATH('[(0,0),(-1,-1)]');
EXPR1                                          |
-----------------------------------------------+
[(0.0,0.0),(1.0,1.0),(0.0,0.0),(-1.0,-1.0)]|
```

- 第二个路径为闭合路径，则返回 NULL

```
SELECT PATH('[(0,0),(1,1)]') + PATH('((1,0),(0,0),(-1,-1))');

EXPR1 |
------------------------------------------
<NULL> |
```

17.3 -

功能描述

平移，从第一个参数对象坐标中减去第二个 POINT 的坐标。适用于 POINT、BOX、PATH、CIRCLE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	POINT
BOX	POINT	BOX
PATH	POINT	PATH
CIRCLE	POINT	CIRCLE

示例

- 平移 POINT

```
SELECT POINT('(2.0,0)') - POINT('(0,2.0)');
EXPR1      |
-----------+
(2.0,-2.0)|
```

- 平移 BOX

```
SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) - POINT('(0,2.0)');
EXPR1                   |
------------------------+
(1.0,-1.0),(0.0,-2.0)|
```

- 平移 PATH

```
SELECT PATH('[(0,0),(1,1)]') - POINT('(0,2.0)');
EXPR1                      |
---------------------------+
[(0.0,-2.0),(1.0,-1.0)]|
```

- 平移 CIRCLE

```
SELECT CIRCLE('<(3,4),2>') - POINT('(1,2)');
EXPR1             |
------------------+
<(2.0,2.0),2.0>|
```

17.4 *

功能描述

伸展/旋转，将第一个参数的每个点乘以第二个 POINT。适用于 POINT、BOX、PATH、CIRCLE。

> **说明**
>
> - 将点视为由实部和虚部表示的复数，并执行标准的复数乘法。
> - 如果将第二个 POINT 解释为向量，这等价于将对象的大小和到原点的距离按向量的长度缩放，并以向量与 x 轴的夹角绕原点逆时针旋转。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	POINT
BOX	POINT	BOX
PATH	POINT	PATH
CIRCLE	POINT	CIRCLE

示例

- POINT

```
SELECT POINT('(2.0,0)') * POINT('(0,2.0)');
EXPR1     |
----------+
(0.0,4.0)|
```

- BOX

```
SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) * POINT('(0,2.0)');
EXPR1                    |
-------------------------+
(1.0,-1.0),(0.0,-2.0)|
```

- PATH

```
SELECT PATH('((1,2),(3,4),(5,6))') * point('(2,2)');
EXPR1                                |
-------------------------------------+
((2.0,4.0),(6.0,8.0),(10.0,12.0))|

SELECT PATH('((0,0),(1,0),(1,1))') * point(cosd(45), sind(45));
```

```
EXPR1


-------------------------------------------------------------+
((0.0,0.0),(0.7071067811865475,0.7071067811865475)
   ,(0.0,1.414213562373095))|
```

- CIRCLE

```
SELECT CIRCLE('<(3,4),2>') * POINT('(1,2)');
EXPR1                          |
-------------------------------+
<(-5.0,10.0),4.47213595499958>|
```

17.5 /

功能描述

收缩/旋转，将第一个参数的每个点除以第二个 POINT。适用于 POINT、BOX、PATH、CIRCLE。

> **说明**
>
> - 将点视为由实部和虚部表示的复数，并执行标准的复数除法。
> - 如果将第二个 POINT 解释为向量，等价于将对象的大小和到原点的距离按向量的长度缩放，并以向量与 X 轴的夹角绕原点顺时针旋转。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	POINT
BOX	POINT	BOX
PATH	POINT	PATH
CIRCLE	POINT	CIRCLE

示例

- POINT

```
SELECT POINT('(2.0,0)') / POINT('(0,2.0)');
EXPR1     |
----------+
(0.0,-1.0)|
```

- BOX

```
SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) / POINT('(0,2.0)');
EXPR1                |
---------------------+
(0.5,0.0),(0.0,-0.5)|
```

- PATH

```
SELECT PATH('((0,0),(1,0),(1,1))') / POINT('(2.0,0)');
EXPR1                              |
-----------------------------------+
((0.0,0.0),(0.5,0.0),(0.5,0.5))|

SELECT PATH('((0,0),(1,0),(1,1))') / POINT(cosd(45), sind(45));
EXPR1


-------------------------------------------------------------------+
((0.0,0.0),(0.7071067811865476,-0.7071067811865476)
   ,(1.4142135623730951,0.0))|
```

- CIRCLE

```
SELECT CIRCLE('<(3,4),2>') / POINT('(1,2)');
EXPR1                              |
-----------------------------------+
<(2.2,-0.4),0.8944271909999159>|
```

17.6 @-@

功能描述

计算总长度。适用于 LSEG、PATH。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
无	LSEG	DOUBLE
无	PATH	

示例

- LSEG

```
SQL> SELECT @-@ LSEG ('[(-1,0),(1,0)]') ;

EXPR1 |
------------------------------------------------------------
2.000000e+00 |

Total 1 records.
```

- PATH

```
SELECT @-@ PATH('[(0,0),(1,0),(1,1)]');
EXPR1|
-----+
2.0|
```

17.7 @@

功能描述

计算中心点。适用于 BOX、LSEG、POLYGON、CIRCLE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
无	BOX	POINT
无	LESG	
无	POLYGON	

接下页

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
无	CIRCLE	

示例

- BOX

```
SELECT @@ BOX('(2,2),(0,0)');
EXPR1     |
----------+
(1.0,1.0)|
```

- LSEG

```
SELECT @@ LSEG('[(0,0),(1,1)]');
EXPR1     |
----------+
(0.5,0.5)|
```

- POLYGON

```
SELECT @@ POLYGON('( 10, 110, 20, 120, 30, 130, 40, 140)');
EXPR1         |
--------------+
(25.0,125.0)|
```

- CIRCLE

```
SELECT @@ CIRCLE('<(3,4),2>');
EXPR1     |
----------+
(3.0,4.0)|
```

17.8

功能描述

在几何数据类型中有以下 3 种功能：

- 返回点的数量。适用于 PATH、POLYGON。
- 计算交点，如果没有交点则返回 NULL。适用于 LSEG、LINE。
- 计算两个方框的交集，如果没有则为 NULL。适用于 BOX。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
无	PATH	INTEGER
无	POLYGON	INTEGER
LSEG	LSEG	POINT
LINE	LINE	POINT
BOX	BOX	BOX

示例

返回点的数量

- PATH

```
SELECT # PATH('((1,0),(0,1),(-1,0))');
EXPR1|
-----+
3|
```

- POLYGON

```
SELECT # POLYGON('( 10, 110, 20, 120, 30, 130, 40, 140)');
EXPR1|
-----+
4|
```

计算交点

- LSEG

```
SELECT LSEG('[(0,0),(1,1)]') # LSEG('[(1,0),(0,1)]');
EXPR1     |
----------+
(0.5,0.5)|
```

- LINE

```
SELECT LINE('[(0,0),(1,1)]') # LINE('[(1,0),(0,1)]');
EXPR1     |
----------+
(0.5,0.5)|
```

计算两个方框的交集

```
SELECT BOX('(2,2),(-1,-1)') # BOX('(1,1),(-2,-2)');
EXPR1                    |
-------------------------+
(1.0,1.0),(-1.0,-1.0)|
```

17.9

功能描述

计算第二个对象上离第一个对象最近的点。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	BOX	POINT
POINT	LSEG	
POINT	LINE	
LSEG	BOX	
LSEG	LSEG	
LINE	LSEG	

示例

- ##(POINT, BOX) 返回边框到点的最近的点。

```
SQL> SELECT POINT('(0,0)') ## BOX('(2,2),(4,4)');

EXPR1     |
----------+
(2.0,2.0)|
```

- ##(POINT, LSEG) 返回线段到点的最近的点。

```
SQL> SELECT POINT('(0,0)') ## LSEG('[(2,0),(0,2)]');

EXPR1     |
----------+
(1.0,1.0)|
```

- ##(POINT, LINE) 返回直线到点的最近的点。

```
SQL> SELECT point('(0,0)') ## LINE('[(2,0),(0,2)]');

EXPR1     |
----------+
(1.0,1.0)|
```

- ##(LSEG, BOX) 返回边框到线段的最近的点。

```
SQL> SELECT LSEG('[(2,0),(0,2)]') ## BOX('(2,2),(4,4)');

EXPR1     |
----------+
(2.0,2.0)|
```

- ##(LSEG, LSEG) 返回边框到线段的最近的点。

```
SQL> SELECT LSEG('[(2,0),(0,2)]') ## LSEG('[(-2,0),(0,0)]');

EXPR1     |
----------+
(0.0,0.0)|

-- 两条平行线段，返回 NULL。
SQL> SELECT lseg('[(2,0),(0,2)]') ## lseg('[(-2,0),(0,-2)]');

EXPR1 |
------+
<NULL>|

Total 1 records.
```

- ##(LINE, LSEG) 返回边框到线段的最近的点。

```
SQL> SELECT LINE('[(-2,0),(0,0)]') ## lseg('[(2,0),(0,2)]');

EXPR1     |
----------+
(2.0,0.0)|
```

17.10 <->

功能描述

计算对象之间的距离。

适用于 POINT 与另一种几何类型的所有组合，以及这些额外的类型对：(BOX, LSEG)，(BOX, LINE)，(LSEG, LINE)，(POLYGON, CIRCLE)，以及易子情况。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	DOUBLE
POINT	BOX	
POINT	LSEG	
POINT	LINE	
POINT	PATH	
POINT	POLYGON	
POINT	CIRCLE	
BOX	POINT	
BOX	BOX	
BOX	LSEG	
LINE	POINT	
LINE	LSEG	
LINE	LINE	
LSEG	POINT	
LSEG	BOX	
LSEG	LSEG	
LSEG	LINE	
POLYGON	POINT	
POLYGON	POLYGON	
POLYGON	CIRCLE	

接下页

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
PATH	POINT	
PATH	PATH	
CIRCLE	POINT	
CIRCLE	POLYGON	
CIRCLE	CIRCLE	

示例

- 计算圆到圆的距离

```
SQL> SELECT CIRCLE('<(0,0),1>') <-> CIRCLE('<(5,0),1>');

EXPR1 |
------------------------------------------------------------
3.000000e+00 |

Total 1 records.
```

- 计算点到线段的距离

```
SQL> SELECT POINT('(0,0)') <-> LSEG('[(2,0),(0,2)]') ;

EXPR1 |
------------------------------------------------------------
1.414214e+00 |

Total 1 records.
```

- 计算圆到多边形的距离

```
SQL> SELECT CIRCLE('<(0,0),1>') <-> POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');

EXPR1 |
------------------------------------------------------------
1.094536e+02 |

Total 1 records.
```

17.11 @>

功能描述

检查第一个对象是否包含第二个对象。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	POINT	BOOLEAN
BOX	BOX	
PATH	POINT	
POLYGON	POINT	
POLYGON	POLYGON	
CIRCLE	POINT	
CIRCLE	CIRCLE	

示例

- 方框不包含

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) @> BOX(POINT('
   (0,1)'),POINT('(-1,0)'));

EXPR1 |
------------------------------------------------------------
F |

Total 1 records.
```

- 圆包含点

```
SQL> SELECT CIRCLE('<(0,0),2>') @> POINT('(1,1)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.12 <@

功能描述

检查第一个对象是否被包含于第二个对象。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	BOX	BOOLEAN
POINT	LSEG	
POINT	LINE	
POINT	PATH	
POINT	POLYGON	
POINT	CIRCLE	
BOX	BOX	
LSEG	BOX	
LSEG	LINE	
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 方框被包含

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(-1,0)')) <@ BOX(POINT('
   (1,1)'),POINT('(-1,-1)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 圆被包含

```
SQL> SELECT CIRCLE('<(1,1),2>') <@ CIRCLE('<(0,0),5>') ;

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 多边形被包含

```
SQL> SELECT POLYGON('(  30, 130, 40, 140)') <@ POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.13 &&

功能描述

检查两个图形是否重叠（有一个共同点就为真）。适用于 BOX、POLYGON、CIRCLE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个方框重叠

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) && BOX(POINT('
   (0,1)'),POINT('(-1,0)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆重叠

```
SQL> SELECT CIRCLE('<(1,1),2>') && CIRCLE('<(0,0),5>') ;

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形重叠

```
SQL> SELECT POLYGON('(3, 13, 4, 14)') && POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');

EXPR1 |
------------------------------------------------------------------
F |

Total 1 records.
```

17.14 «

功能描述

判断第一个对象是否完全位于第二个对象的左侧。适用于 POINT、BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个点

```
SQL> SELECT POINT('(0,2.0)') << POINT('(2.0,0)');

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) << BOX(POINT('
   (4,5)'),POINT('(6,7)'));

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') << CIRCLE('<(5,0),1>');

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('(3, 13, 4, 14)') << POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

17.15 »

功能描述

判断第一个对象是否完全位于第二个对象的右侧。适用于 POINT、BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。

- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个点

```
SQL> SELECT POINT('(2.0,0)') >> POINT('(0,2.0)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个方框

```
SQL> SELECT BOX(POINT('(4,5)'),POINT('(6,7)')) >> BOX(POINT('
   (0,1)'),POINT('(1,0)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(5,0),1>') >> CIRCLE('<(0,0),1>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('( 10, 110, 20, 120, 30, 130, 40, 140)') >>
   POLYGON('(3, 13, 4, 14)') ;

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.16 &<

功能描述

判断第一个对象是否不超过第二个对象的右边。适用于 BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) &< BOX(POINT('
   (4,5)'),POINT('(6,7)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('(3, 13, 4, 14)')  &< POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');
```

```
EXPR1 |
-------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') &< CIRCLE('<(5,0),1>');

EXPR1 |
-------------------------------------------------------------
T |

Total 1 records.
```

17.17 &>

功能描述

判断第一个对象是否完全位于第二个对象的右边。适用于 BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(4,5)'),POINT('(6,7)')) &> BOX(POINT('
   (0,1)'),POINT('(1,0)'));

EXPR1 |
-------------------------------------------------------------
T |
```

```
Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('( 10, 110, 20, 120, 30, 130, 40, 140)') &>
   POLYGON('(3, 13, 4, 14)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(5,0),1>') &> CIRCLE('<(0,0),1>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.18 «|

功能描述

判断第一个对象是否完全位于第二个对象的下方。适用于 POINT、BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	
POLYGON	POLYGON	

接下页

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
CIRCLE	CIRCLE	

示例

- 两个点

```
SQL> SELECT POINT('(2.0,0)') <<| POINT('(0,2.0)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) <<| BOX(POINT('
   (4,5)'),POINT('(6,7)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,-3),1>') <<| CIRCLE('<(0,0),1>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('(3, 13, 4, 14)') <<| POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.19 |»

功能描述

判断第一个对象是否完全位于第二个对象的上方。适用于 POINT、BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个点

```
SQL> SELECT POINT('(2.0,4)') |>> POINT('(0,2.0)');

EXPR1 |
------------------------------------------------------------
T |
```

- 两个方框

```
SQL> SELECT BOX('(5,5),(3,4)') |>> BOX('(3,3),(0,0)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,3),1>') |>>  CIRCLE('<(0,0),1>');
```

```
EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('( 10, 110, 20, 120, 30, 130, 40, 140)') |>>
   POLYGON('(3, 13, 4, 14)') ;

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.20 &<|

功能描述

判断第一个对象是否不超过第二个对象的上方。适用于 BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) &<| BOX(POINT('
   (4,5)'),POINT('(6,7)'));

EXPR1 |
------------------------------------------------------------
T |
```

```
Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,-3),1>') &<| CIRCLE('<(0,0),1>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('(3, 13, 4, 14)') &<| POLYGON('
   ( 10, 110, 20, 120, 30, 130, 40, 140)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.21 |&>

功能描述

判断第一个对象是否不超过第二个对象的下方。适用于 BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(4,5)'),POINT('(6,7)')) |&> BOX(POINT('
   (0,1)'),POINT('(1,0)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') |&> CIRCLE('<(0,-3),1>') ;

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形

```
SQL> SELECT POLYGON('( 10, 110, 20, 120, 30, 130, 40, 140)') |&>
   POLYGON('(3, 13, 4, 14)') ;

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.22 >^

功能描述

判断第一个对象是否位于第二个对象的下方，允许边缘相切。适用于 POINT、BOX。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	

示例

- 两个点

```
SQL> SELECT POINT('(2.0,2)') <^ POINT('(0,2.1)');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个方框

```
SQL> SELECT BOX('((1,1),(0,0))') <^ BOX('((2,2),(1,1))');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.23 >^

功能描述

判断第一个对象是否位于第二个对象的上方，允许边缘相切。适用于 POINT、BOX。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	

示例

- 两个点

```
SQL> SELECT POINT('(2.0,2.1)') >^ POINT('(0,2)');

EXPR1 |
------------------------------------------------------------
T |
```

- 两个方框

```
SQL> SELECT BOX('((2,2),(1,1))') >^ BOX('((1,1),(0,0))');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.24 ?#

功能描述

判断两个对象是否相交。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	

接下页

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
LSEG	BOX	BOOLEAN
LSEG	LSEG	
LSEG	LINE	
LINE	BOX	
LINE	LINE	
PATH	PATH	

示例

- 两个方框相交

```
SQL> SELECT BOX('((2,2),(1,1))') ?# BOX('((1,1),(0,0))');

EXPR1 |
------------------------------------------------------------
T |
```

- 线段和方框相交

```
SQL> SELECT LSEG('[(-1,0),(1,0)]') ?# BOX('(2,2),(-2,-2)');

EXPR1 |
------------------------------------------------------------
T |
```

- 线和方框相交

```
SQL> SELECT LINE('[(-1,0),(1,0)]') ?# BOX('(2,2),(-2,-2)');

EXPR1 |
------------------------------------------------------------
T |
```

- 两条路径相交

```
SQL> SELECT PATH('((15,55),(34,134))')  ?# PATH('((10, 110)
   , (20, 120), (30, 130), (40, 140))');

EXPR1 |
------------------------------------------------------------
T |
```

17.25 ?-|

功能描述

判断两条线是否相互垂直。适用于 LSEG 和 LINE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
LSEG	LSEG	BOOLEAN
LINE	LINE	

示例

- 线段相互垂直

```
SQL> SELECT LSEG('[(0,0),(0,1)]') ?-| LSEG('[(0,0),(1,0)]' );

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 直线相互垂直

```
SQL> SELECT LINE('[(0,0),(0,1)]') ?-| LINE('[(0,0),(1,0)]' );

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.26 ~=

功能描述

判断两个对象是否一般相等。适用于 POINT、BOX、POLYGON、CIRCLE。

- 是，返回 TRUE。

- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	BOOLEAN
BOX	BOX	
POLYGON	POLYGON	
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(4,5)'),POINT('(6,7)')) ~= BOX(POINT('
   (6,7)'),POINT('(4,5)'));

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个多边形一般相等，但点的顺序不同

```
SQL> SELECT POLYGON('((0,0),(1,1))') ~= POLYGON('((1,1),(0,0))');

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆不等

```
SQL> SELECT CIRCLE('<(0,0),1>') ~= CIRCLE('<(0,-3),1>') ;

EXPR1 |
------------------------------------------------------------------
F |

Total 1 records.
```

17.27 <>

功能描述

判断两个对象是否不等。适用于 POINT、LSEG、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
POINT	POINT	DOUBLE
LSEG	LSEG	
CIRCLE	CIRCLE	

示例

- 两个点不等

```
SQL> SELECT POINT('(1,0)') <> POINT('(0,0)');

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两条线段

```
SQL> SELECT LSEG('[(-1,0),(1,0)]') <> LSEG('[(1,0),(-1,0)]' );

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') <> CIRCLE('<(0,-3),1>');

EXPR1 |
------------------------------------------------------------------
F |
```

```
Total 1 records.
```

17.28 <

功能描述

判断第一个对象的面积是否小于第二个对象的面积。适用于 BOX、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	DOUBLE
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) < BOX(POINT('(4,5)
   '),POINT('(6,8)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') < CIRCLE('<(5,0),2>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.29 >

功能描述

判断第一个对象的面积是否大于第二个对象的面积。适用于 BOX、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,2)'),POINT('(2,0)')) > BOX(POINT('(4,5)
   '),POINT('(6,7)'));

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),2>') > CIRCLE('<(5,0),1>');

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```

17.30 <=

功能描述

判断第一个对象的面积是否小于等于第二个对象的面积。适用于 BOX、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) <= BOX(POINT('
   (4,5)'),POINT('(6,8)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') <= CIRCLE('<(5,0),1>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.31 >=

功能描述

判断第一个对象的面积是否大于等于第二个对象的面积。适用于 BOX、CIRCLE。

- 是，返回 TRUE。
- 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
CIRCLE	CIRCLE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,2)'),POINT('(2,0)')) >= BOX(POINT('
   (4,5)'),POINT('(6,7)'));

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

- 两个圆

```
SQL> SELECT CIRCLE('<(0,0),1>') >= CIRCLE('<(5,0),1>');

EXPR1 |
------------------------------------------------------------
T |

Total 1 records.
```

17.32 =

功能描述

= 在几何数据类型中有以下两种功能：

- 判断第一个对象的面积是否等于第二个对象的面积。适用于 BOX。
  - 是，返回 TRUE。
  - 否，返回 FALSE。
- 判断第一个对象是否等于第二个对象。适用于 LINE。
  - 是，返回 TRUE。
  - 否，返回 FALSE。

数据类型

L_OPERAND_TYP（左操作数类型）	R_OPERAND_TYP（右操作数类型）	RET_TYPE（返回类型）
BOX	BOX	BOOLEAN
LINE	LINE	

示例

- 两个方框

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) = BOX(POINT('(4,5)
   '),POINT('(5,4)'));

EXPR1 |
------------------------------------------------------------------
T |

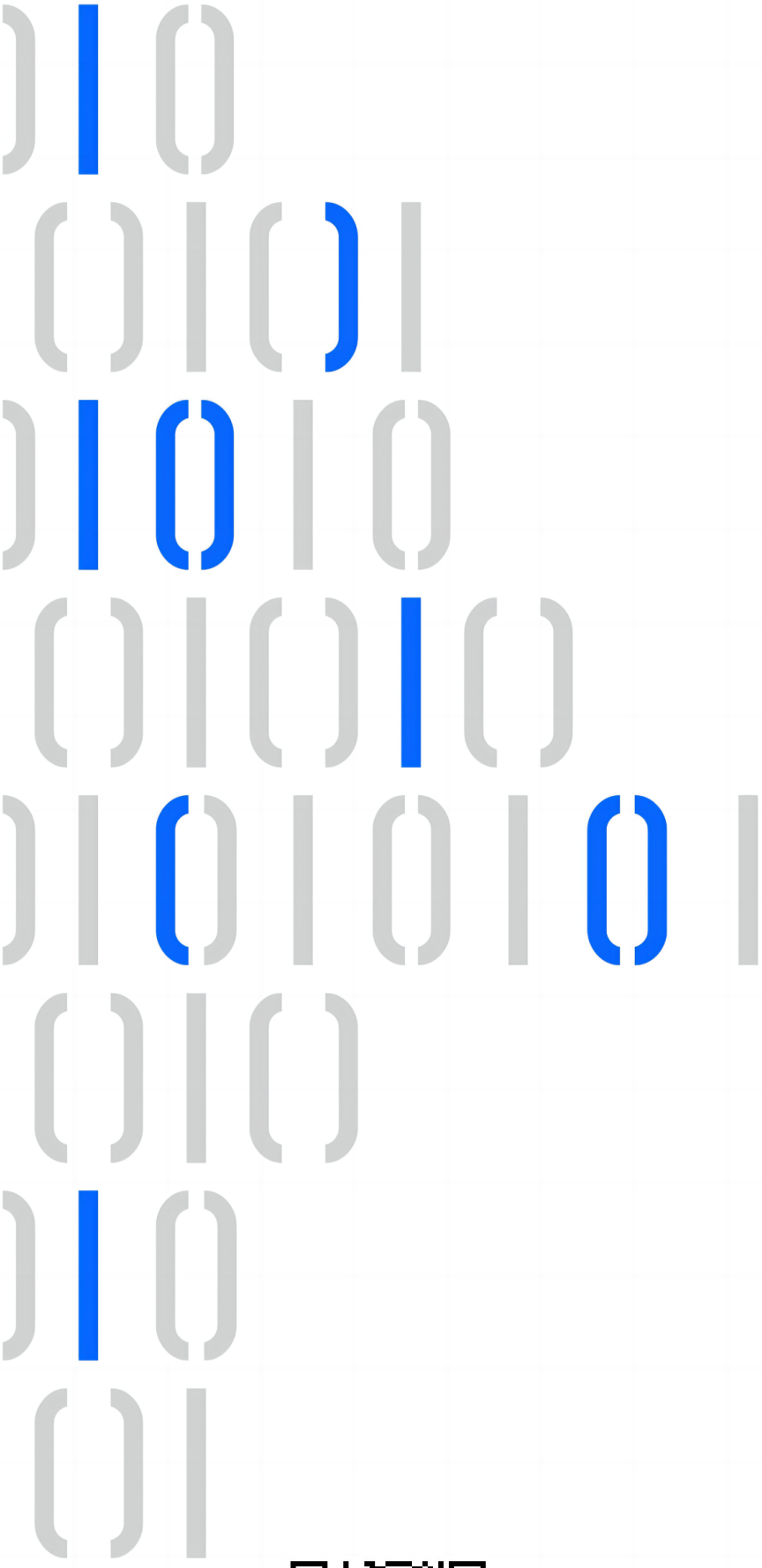
Total 1 records.
```

- 两条直线

```
SQL> SELECT BOX(POINT('(0,1)'),POINT('(1,0)')) = BOX(POINT('(4,5)
   '),POINT('(5,4)'));

EXPR1 |
------------------------------------------------------------------
T |

Total 1 records.
```



成都虚谷伟业科技有限公司

联系电话：400-8886236

官方网站：www.xugudb.com