

# 虚谷数据库 V12.6

## 系统函数参考指南

文档版本 01

发布日期 2024-10-08



版权所有 © 2024 成都虚谷伟业科技有限公司。

## 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



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](http://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
 <b>注意</b>	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。
 <b>说明</b>	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	2024-10-08	第一次发布

# 目录

1	数值数据类型函数	1
1.1	概述	1
1.2	ABS	3
1.3	ACOS	4
1.4	ACOSD	4
1.5	ACOSH	5
1.6	ASIN	5
1.7	ASIND	6
1.8	ASINH	6
1.9	ATAN	7
1.10	ATAND	7
1.11	ATANH	8
1.12	ATAN2	8
1.13	ATAN2D	9
1.14	BIN	9
1.15	BITAND	10
1.16	CBRT	11
1.17	CEILING	11
1.18	CONV	12
1.19	COS	12
1.20	COSD	13
1.21	COSH	13
1.22	COT	14
1.23	COTD	14
1.24	DEGREES	15
1.25	DIV	15
1.26	ERF	16

1.27	ERFC	16
1.28	FACTORIAL	17
1.29	FLOOR	17
1.30	GCD	18
1.31	GET_HASH	18
1.32	HEX	19
1.33	LCM	19
1.34	LOG	20
1.35	LOG2	20
1.36	NANVL	21
1.37	PI	21
1.38	POW	22
1.39	RANDOM	23
1.40	RANDOM_NORMAL	23
1.41	REMAINDER	24
1.42	ROUND	24
1.43	ROUND_TIES_TO_EVEN	25
1.44	SETSEED	26
1.45	SIN	27
1.46	SIND	27
1.47	SINH	28
1.48	TAN	28
1.49	TAND	29
1.50	TANH	29
1.51	TRUNC	30
1.52	TRUNCATE	31
1.53	TO_CHAR	32
2	字符数据类型函数	33
2.1	概述	33
2.2	ATOF	37

2.3	ATOL	38
2.4	BASE64_DNCODE	38
2.5	BASE64_ENCODE	39
2.6	BIT_LENGTH	39
2.7	CHARACTER_LENGTH	40
2.8	CONCAT	41
2.9	CONCAT_WS	42
2.10	CONVERT	43
2.11	DECODE_PG	44
2.12	ENCODE_PG	44
2.13	ESCAPE_DECODE	45
2.14	ESCAPE_ENCODE	46
2.15	FIND_IN_SET	46
2.16	FROM_BASE64	48
2.17	HEADING	49
2.18	HEX_DECODE	49
2.19	HEX_ENCODE	50
2.20	INITCAP	50
2.21	INSERT	51
2.22	INSTR	52
2.23	ISNULL	53
2.24	LCASE	54
2.25	LEFT	54
2.26	LEFTB	55
2.27	LEN	55
2.28	LENGTH	56
2.29	LENGTHB	57
2.30	LOCATE	58
2.31	LOWER	58
2.32	LPAD	59

2.33 LTRIM .....	60
2.34 MID .....	60
2.35 POSITION .....	61
2.36 REGEXP_COUNT .....	62
2.37 REGEXP_LIKE .....	63
2.38 REPEAT .....	64
2.39 REPLACE .....	65
2.40 REVERSE_STR .....	65
2.41 RIGHT .....	66
2.42 RIGHTB .....	66
2.43 ROWIDTOCHAR .....	67
2.44 RPAD .....	67
2.45 RTRIM .....	68
2.46 SPACE .....	69
2.47 SPLIT_PART .....	70
2.48 SQLCODE .....	71
2.49 STUFF .....	72
2.50 STRCMP .....	72
2.51 SUBSTR .....	73
2.52 SUBSTRB .....	74
2.53 SUBSTRING .....	76
2.54 SUBSTRING_INDEX .....	77
2.55 TAILING .....	77
2.56 TO_BASE64 .....	78
2.57 TO_HEX .....	78
2.58 TRANSLATE .....	79
2.59 UCASE .....	80
2.60 UNHEX .....	80
2.61 UPPER .....	81
2.62 WM_CONCAT .....	81

3	时间日期数据类型函数 .....	83
3.1	概述 .....	83
3.2	ADDDATE .....	90
3.3	ADD_MONTHS .....	91
3.4	ADDTIME .....	91
3.5	CURDATE .....	92
3.6	CURRENT_DATETIME .....	93
3.7	CURTIME .....	93
3.8	DATE .....	94
3.9	DATE_ADD .....	95
3.10	DATE_FORMAT .....	95
3.11	DATE_SUB .....	96
3.12	DATEDIFF .....	97
3.13	DAY .....	98
3.14	DAYNAME .....	99
3.15	DAYOFMONTH .....	100
3.16	DAYOFWEEK .....	101
3.17	DAYOFYEAR .....	102
3.18	EXTRACT .....	103
3.19	EXTRACT_DAY .....	104
3.20	EXTRACT_HOUR .....	104
3.21	EXTRACT_MINUTE .....	105
3.22	EXTRACT_MONTH .....	105
3.23	EXTRACT_SECOND .....	106
3.24	EXTRACT_YEAR .....	106
3.25	FROM_DAYS .....	107
3.26	FROM_UNIXTIME .....	107
3.27	GET_BOOT_TIME .....	108
3.28	GETDAY .....	109
3.29	GETHOUR .....	109

3.30	GETMINUTE	110
3.31	GETMONTH	110
3.32	GETSECOND	111
3.33	GET_UPTIME	111
3.34	GETYEAR	112
3.35	GET_FORMAT	112
3.36	HOUR	113
3.37	LAST_DAY	114
3.38	LOCALTIME	115
3.39	LOCALTIMESTAMP	116
3.40	MAKE_DATE	117
3.41	MAKE_TIME	117
3.42	MAKE_TIMESTAMP	118
3.43	MICROSECOND	118
3.44	MINUTE	120
3.45	MONTH	121
3.46	MONTHNAME	122
3.47	MONTHS_BETWEEN	123
3.48	NEXT_DAY	123
3.49	NOW	124
3.50	PERIOD_ADD	124
3.51	PERIOD_DIFF	125
3.52	QUARTER	126
3.53	SECOND	127
3.54	SEC_TO_TIME	128
3.55	SUBDATE	129
3.56	SUBTIME	130
3.57	SYSDATE	131
3.58	SYSDATETIME	131
3.59	SYSTEMTIME	132

3.60	SYSTIMESTAMP	132
3.61	TIME	133
3.62	TIME_FORMAT	134
3.63	TIMEDIFF	135
3.64	TIME_TO_SEC	136
3.65	TIMESTAMP	137
3.66	TIMESTAMPADD	138
3.67	TIMESTAMPDIFF	139
3.68	TO_DATE	139
3.69	TO_DAYS	141
3.70	TO_SECONDS	142
3.71	TO_TIMESTAMP	143
3.72	TRANSACTION_TIMESTAMP	144
3.73	WEEK	145
3.74	WEEKDAY	146
3.75	WEEKOFYEAR	147
3.76	YEAR	147
3.77	YEARWEEK	148
3.78	UNIX_TIMESTAMP	149
3.79	UTC_DATE	150
3.80	UTC_TIME	151
3.81	UTC_TIMESTAMP	151
4	类型转换函数	153
4.1	概述	153
4.2	ASCII	153
4.3	CHR	154
4.4	HEXTORAW	154
4.5	NUMTODSINTERVAL	155
4.6	NUMTOYMINTERVAL	155
4.7	RAWTOHEX	156

4.8	TO_BLOB .....	156
5	空值转换函数 .....	158
5.1	概述 .....	158
5.2	IFNULL .....	158
5.3	NULLIF .....	159
5.4	NVL .....	159
5.5	NVL2 .....	160
6	JSON 数据类型函数 .....	162
6.1	概述 .....	162
6.2	JSON_ARRAY .....	163
6.3	JSON_ARRAY_APPEND .....	164
6.4	JSON_ARRAY_INSERT .....	166
6.5	JSON_CONTAINS .....	167
6.6	JSON_CONTAINS_PATH .....	168
6.7	JSON_DEPTH .....	170
6.8	JSON_EXTRACT .....	171
6.9	JSON_INSERT .....	172
6.10	JSON_KEYS .....	174
6.11	JSON_LENGTH .....	175
6.12	JSON_MERGE .....	177
6.13	JSON_MERGE_PATCH .....	178
6.14	JSON_MERGE_PRESERVE .....	180
6.15	JSON_OBJECT .....	181
6.16	JSON_OVERLAPS .....	182
6.17	JSON_PRETTY .....	184
6.18	JSON_QUOTE .....	185
6.19	JSON_REMOVE .....	186
6.20	JSON_REPLACE .....	187
6.21	JSON_SCHEMA_VALID .....	188
6.22	JSON_SCHEMA_VALIDATION_REPORT .....	189

6.23	JSON_SEARCH	190
6.24	JSON_SET	192
6.25	JSON_TYPE	194
6.26	JSON_UNQUOTE	195
6.27	JSON_VALID	196
7	BIT 数据类型函数	198
7.1	概述	198
7.2	BIT_COUNT	198
7.3	BIT_LENGTH	199
7.4	BITTOCHAR	199
8	XML 数据类型函数	201
8.1	概述	201
8.2	EXTRACT	201
8.3	EXTRACTVALUE	202
8.4	XMLCAST	203
8.5	XMLELEMENT	204
8.6	XMLEXISTS	205
8.7	XMLFOREST	206
8.8	XMLSEQUENCE	207
8.9	XMLTABLE	207
8.10	XMLQUERY	209
8.11	XMLTYPE	210
9	聚合函数	212
9.1	概述	212
9.2	AVG	213
9.3	BIT_AND	214
9.4	BIT_OR	215
9.5	BIT_XOR	215
9.6	COUNT	216
9.7	GROUPING	217

9.8	GROUP_CONCAT	218
9.9	JSON_ARRAYAGG	219
9.10	JSON_OBJECTAGG	220
9.11	LISTAGG	220
9.12	MAX	222
9.13	MEDIAN	223
9.14	MIN	224
9.15	STDDEV	225
9.16	STDDEV_POP	225
9.17	STDDEV_SAMP	226
9.18	SUM	227
9.19	VAR_POP	228
9.20	VAR_SAMP	229
9.21	VARIANCE	230
9.22	XMLAGG	231
10	窗口函数	232
10.1	概述	232
10.2	ROW_NUMBER	232
11	内部功能性函数	234
11.1	概述	234
11.2	FLUSH_COMMAND_LOG	234
11.3	FLUSH_ERROR_LOG	235
11.4	FLUSH_MODIFY_LOG	235
11.5	FORMAT_GSTO_NOS	236
12	特殊表达式	238
12.1	概述	238
12.2	CURRENT_DATABASE	240
12.3	CURRENT_DATE	241
12.4	CURRENT_DB	241
12.5	CURRENT_DB_ID	242

12.6 CURRENT_IP .....	242
12.7 CURRENT_NODEID .....	243
12.8 CURRENT_SCHEMA .....	243
12.9 CURRENT_TIME .....	244
12.10 CURRENT_TIMESTAMP .....	244
12.11 CURRENT_USER .....	245
12.12 CURRENT_USERID .....	245
12.13 CURRVAL .....	246
12.14 DATABASE .....	247
12.15 DIR_EXISTS .....	247
12.16 FILE_EXISTS .....	248
12.17 FILELEN .....	248
12.18 GEN_RANDOM_UUID .....	249
12.19 GET_INSTALL_PATH .....	249
12.20 GET_TYPE_SPACE .....	250
12.21 GET_WORK_PATH .....	251
12.22 GREATEST .....	252
12.23 IF .....	252
12.24 INET_ATON .....	253
12.25 INET_NTOA .....	254
12.26 JSON_VALUE .....	254
12.27 LEAST .....	256
12.28 MAX_NODE_NUM .....	257
12.29 MEMBER OF .....	257
12.30 NEWID .....	258
12.31 NUM_NONNULLS .....	258
12.32 NUM_NULLS .....	259
12.33 RENAME_FILE .....	259
12.34 SLEEP .....	260
12.35 SQLERRM .....	260

12.36	SYS_CONTEXT .....	261
12.37	SYS_GUID .....	262
12.38	SYS_USERID .....	262
12.39	SYS_UUID .....	263
12.40	SYSTEM_USER .....	264
12.41	UID .....	264
12.42	USER .....	265
12.43	UUID .....	265
12.44	USERENV .....	266
12.45	VERSION .....	267
13	特殊操作符 .....	268
13.1	概述.....	268
13.2	+ .....	268
13.3	- .....	270
13.4	.....	271
13.5	-> .....	272
13.6	->> .....	272
13.7	&（按位与） .....	273
13.8	（按位或） .....	274
13.9	^（按位异或） .....	276
13.10	<<（位左移） .....	278
13.11	>>（位右移） .....	279
13.12	~（按位非） .....	281
14	几何函数 .....	284
14.1	概述.....	284
14.2	AREA(BOX).....	285
14.3	AREA(CIRCLE).....	285
14.4	AREA(PATH) .....	286
14.5	CENTER(BOX) .....	287
14.6	CENTER(CIRCLE) .....	287

14.7	DIAGONAL(BOX)	288
14.8	DIAMETER(CIRCLE)	288
14.9	HEIGHT(BOX)	289
14.10	ISCLOSED(PATH)	289
14.11	ISOPEN(PATH)	290
14.12	LENGTH(LSEG)	291
14.13	LENGTH(PATH)	291
14.14	NPOINTS(PATH)	292
14.15	NPOINTS(POLYGON)	292
14.16	PCLOSE(PATH)	293
14.17	POPEN(PATH)	293
14.18	RADIUS(CIRCLE)	294
14.19	SLOPE(POINT, POINT)	294
14.20	WIDTH(BOX)	295
15	几何类型转换函数	296
15.1	概述	296
15.2	BOUND_BOX(BOX, BOX)	297
15.3	BOX(CIRCLE)	298
15.4	BOX(CHAR)	298
15.5	BOX(POINT)	299
15.6	BOX(POLYGON)	299
15.7	BOX(POINT, POINT)	300
15.8	CIRCLE(BOX)	300
15.9	CIRCLE(CHAR)	301
15.10	CIRCLE(CIRCLE)	301
15.11	CIRCLE(POLYGON)	302
15.12	CIRCLE(POINT, DOUBLE)	302
15.13	LINE(CHAR)	303
15.14	LINE(POINT, POINT)	303
15.15	LSEG(BOX)	304

15.16	LSEG(CHAR)	304
15.17	LSEG(POINT, POINT)	305
15.18	PATH(CHAR)	305
15.19	PATH(PATH)	306
15.20	PATH(POLYGON)	306
15.21	POINT(BOX)	307
15.22	POINT(CHAR)	307
15.23	POINT(CIRCLE)	308
15.24	POINT(CIRCLE)	308
15.25	POINT(POLYGON)	309
15.26	POINT(DOUBLE, DOUBLE)	309
15.27	POLYGON(BOX)	310
15.28	POLYGON(CIRCLE)	310
15.29	POLYGON(CHAR)	311
15.30	POLYGON(INTEGER, CIRCLE)	311
15.31	POLYGON(PATH)	312
16	几何操作符	313
16.1	概述	313
16.2	+	315
16.3	-	316
16.4	*	317
16.5	/	319
16.6	@-@	320
16.7	@@	321
16.8	#	322
16.9	##	324
16.10	<->	325
16.11	@>	328
16.12	<@	329
16.13	&&	330

16.14 «	331
16.15 »	332
16.16 &<	334
16.17 &>	335
16.18 «	336
16.19  »	338
16.20 &<	339
16.21  &>	340
16.22 >^	341
16.23 >^	342
16.24 ?#	343
16.25 ?-	345
16.26 ~=	345
16.27 <>	347
16.28 <	348
16.29 >	349
16.30 <=	349
16.31 >=	350
16.32 =	351

# 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
接下一页	

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 $\pi$ 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

## 1.2 ABS

### 功能描述

计算数值类型的字段或表达式的绝对值。

### 语法格式

```
ABS (expr)
```

### 参数说明

expr: 数值类型的字段或表达式。

### 返回值

同数值数据类型。

### 示例

```
SQL> SELECT ABS(-100.4),ABS(100.4) FROM dual;  
EXPR1 | EXPR2 |  
-----  
100.4 | 100.4 |
```

## 1.3 ACOS

### 功能描述

计算 expr 的反余弦值，结果为弧度。函数返回值在 0 到  $\pi$  之间。

### 语法格式

```
ACOS (expr)
```

### 参数说明

expr: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型，其取值范围为: [-1.0, 1.0]。

### 返回值

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 `expr` 的反余弦值，结果为角度。函数返回值在 0 到 180 之间。

### 语法格式

```
ACOSD (expr)
```

### 参数说明

`expr`: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，其取值范围为：[-1.0, 1.0]。

### 返回值

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 `expr` 的反双曲余弦值。

### 语法格式

```
ACOSH (expr)
```

### 参数说明

`expr`: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，其取值大于 1.0。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.6 ASIN

### 功能描述

计算 `expr` 的反正弦值，结果为弧度。函数返回值在  $-\pi/2$  到  $\pi/2$  之间。

### 语法格式

```
ASIN(expr)
```

### 参数说明

`expr`: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型，其取值范围为：[-1.0, 1.0]。

### 返回值

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 `expr` 的反正弦值，结果为角度。函数返回值在 -90 到 90 之间。

### 语法格式

```
ASIND(expr)
```

### 参数说明

`expr`: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，其取值范围为：[-1.0, 1.0]。

### 返回值

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 `expr` 的反双曲正弦值。

### 语法格式

```
ASINH(expr)
```

### 参数说明

`expr`: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.9 ATAN

### 功能描述

计算 `expr1` 的反正切值，结果为弧度。函数返回值在  $-\pi/2$  到  $\pi/2$  之间。当函数传入两个参数时，功能同 ATAN2。

### 语法格式

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

### 参数说明

`expr1`: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

### 返回值

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 `expr` 的反正切值，结果为角度。函数返回值在-90 到 90 之间。

### 语法格式

```
ATAND (expr)
```

### 参数说明

`expr`: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.11 ATANH

### 功能描述

计算 `expr` 的反双曲正切值。

### 语法格式

```
ATANH (expr)
```

### 参数说明

`expr`: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，其取值范围为：[-1.0, 1.0]。

### 返回值

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  $\text{expr1}/\text{expr2}$  的反正切值，结果为弧度。函数返回值在  $-\pi$  到  $\pi$  之间。

### 语法格式

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

### 参数说明

$\text{expr1}/\text{expr2}$ : DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

### 返回值

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  $\text{expr1}/\text{expr2}$  的反正切值，结果为角度。

### 语法格式

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

### 参数说明

$\text{expr1}/\text{expr2}$ : DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

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。

### 返回值

CHAR 类型。

### 示例

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

EXPR1 |
-----|
1100 |

SQL> SELECT BIN(12.1) FROM dual;

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

## 1.15 BITAND

### 功能描述

返回两个参数进行二进制 AND 运算后对应的十进制数。

函数将两个参数分别转为二进制后，对各位进行逻辑 AND，最终的结果再转为十进制。

### 语法格式

`BITAND(expr1, expr2)`

### 参数说明

- expr1: 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。
- 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。

### 返回值

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
-- 逐位做 AND 运算的结果为 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 expr 的立方根。

### 语法格式

```
CBRT(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

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。

### 返回值

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。

### 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，取值 [2, 36]，其正负号指示 expr1 是有符号数还是无符号数。

### 返回值

VARCHAR。

### 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 expr 的余弦值。

### 语法格式

```
COS(expr)
```

### 参数说明

expr: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型，为以弧度表示的角度。

### 返回值

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 expr 的余弦值。

### 语法格式

```
COSD(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，为角度。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.21 COSH

### 功能描述

计算 expr 的双曲余弦值。

### 语法格式

COSH (expr)

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.22 COT

### 功能描述

计算 expr 的余切值。

### 语法格式

COT (expr)

### 参数说明

expr: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型，为以弧度表示的角度。

### 返回值

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 expr 的余切值。

### 语法格式

COTD(expr)

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，为角度。

### 返回值

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 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

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/expr2: NUMERIC 类型或者能隐式转换为 NUMERIC 类型的其他类型。

### 返回值

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 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

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。

### 语法格式

```
ERFC(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT erfc(1.0) FROM dual;  
EXPR1 |  
-----
```

## 1.28 FACTORIAL

### 功能描述

计算 expr 的阶乘。

### 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。其取值范围为: [0, 34), 受限于返回值类型的取值范围。

### 返回值

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。

## 返回值

BIGINT 或 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: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

## 返回值

DOUBLE 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。

## 返回值

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(expr) 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。

### 返回值

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 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

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 xugu 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/DOUBLE 类型的数值。

### 返回值

FLOAT/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 xugu 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/DOUBLE 类型的数值。

### 返回值

FLOAT/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 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型。

### 返回值

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  $\pi$  的值。

与 MySQL 无差异。

### 语法格式

```
PI ()
```

### 参数说明

无参数。

### 返回值

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。

### 返回值

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。

### 语法格式

```
RANDOM()
```

### 参数说明

无参数。

### 返回值

DOUBLE 类型，返回值取值范围为：[0.0, 1)。

### 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/expr2: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

expr1 表示平均值，expr2 表示标准差；其默认值分别为 0.0 和 1.0。

### 返回值

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/expr2: 数值类型或者能隐式转换为数值类型的其他类型。

### 返回值

数值类型。

### 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: 保留的小数位；可选，数值类型的字段或表达式。

### 返回值

同数值数据类型。

### 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: INTEGER 类型或者能隐式转换为 INTEGER 类型的其他类型, 可选, 默认为 0。
  - 如果 expr2 为正, 则 expr1 将四舍五入到小数点右侧的整数位数。
  - 如果 expr2 为负, 则 expr1 将四舍五入到小数点左侧的整数位数。

**注意**

四舍六入规则如下：

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	—
—	—	为偶数	直接舍去
—	—	为奇数	进位舍去

**返回值**

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。

**语法格式**

```
SETSEED(expr)
```

**参数说明**

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。其取值范围为: [-1.0, 1.0]。

### 返回值

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 expr 的正弦值。

### 语法格式

```
SIN(expr)
```

### 参数说明

expr: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型, 为以弧度表示的角度。

### 返回值

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 expr 的正弦值。

### 语法格式

```
SIND(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，为角度。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.47 SINH

### 功能描述

计算 expr 的双曲正弦值。

### 语法格式

```
SINH(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.48 TAN

### 功能描述

计算 expr 的正切值。

### 语法格式

```
TAN(expr)
```

### 参数说明

expr: DOUBLE/FLOAT 类型或者能隐式转换为 DOUBLE/FLOAT 类型的其他类型，为以弧度表示的角度。

### 返回值

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 expr 的正切值。

### 语法格式

```
TAND(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型，为角度。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.50 TANH

### 功能描述

计算 expr 的双曲正切值。

### 语法格式

```
TANH(expr)
```

### 参数说明

expr: DOUBLE 类型或者能隐式转换为 DOUBLE 类型的其他类型。

### 返回值

DOUBLE 类型。

### 示例

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

## 1.51 TRUNC

### 功能描述

- 语法一  
将日期值 expr1，按照 expr2 日期格式裁剪。
- 语法二  
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。

计算 expr 的双曲正切值。

### 语法格式

- 语法一

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

- 语法二

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

### 参数说明

- 语法一
    - expr1: 日期格式的值
    - expr2: 日期格式, 'yyyy'、'mm'、'dd' 等, 默认返回当前年月日
  - 语法二
    - 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。

## 返回值

同日期数据类型。

## 示例

- 语法一

```
SQL> SELECT TRUNC('2022-02-22 22:22:22', 'yyyy'), TRUNC('
2022-02-22 22:22:22', 'mm'), TRUNC('2022-02-22 22:22:22', 'dd')
FROM dual;

EXPR1 | EXPR2 | EXPR3 |
-----
2022-01-01 00:00:00.000 AD | 2022-02-01 00:00:00.000 AD
      | 2022-02-22 00:00:00.000 AD |
```

- 语法二

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

EXPR1 | EXPR2 | EXPR3 |
-----
```

# 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。

## 返回值

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。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT TO_CHAR(123.4) FROM dual;  
  
EXPR1 |  
-----  
123.4 |
```

# 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
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
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
接下页	

函数形式	功能
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
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
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
接下页	

函数形式	功能
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
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
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
接下页	

函数形式	功能
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
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
接下页	

函数形式	功能
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
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: 应是一个合法的数值字符串，如：'123.456'，否则结果将不正确。

### 返回值

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: 应是一个合法的数值字符串, 如: '12345', 否则结果将不正确。

### 返回值

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。

### 返回值

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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT BASE64_ENCODE (HEXTORAW ('1234567890ABCDEF0001')) FROM
      DUAL;
EXPR1 |
-----
EjRWeJCrze8AAQ== |
```

## 2.6 BIT\_LENGTH

### 功能描述

以位为单位返回给定字符串的长度。

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。

## 返回值

INTEGER 数值类型。

## 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 CHARACTER\_LENGTH

## 功能描述

字符串长度测试函数。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。

## 返回值

INTEGER 数值类型。

## 示例

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

EXPR1 |
```

## 2.8 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；xugu 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`：字符串
- `expr2`：字符串
- `expr3`：字符串
- `expr4`：字符串

### 返回值

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 -34.56 BBBAAA |
```

## 2.9 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；xugu 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；xugu 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：字符串
- expr2：字符串
- expr3：字符串
- expr4：字符串

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT CONCAT_WS(' - ', 'A', 'B');  
  
EXPR1 |  
-----  
A - B |
```

## 2.10 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

### 返回值

CHAR 类型字符串。

### 示例

```
SQL> SHOW DB_INFO

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

SQL> SELECT CONVERT('test_convert','GBK');

EXPR1 |
-----|
test_convert|

Use time:0 ms.
SQL> SELECT LENGTHB(CONVERT('我爱中国','GBK','UTF8'));

EXPR1 |
-----|
8 |

SQL> CREATE DATABASE db_convert CHARACTER SET 'GBK' TIME ZONE 'GMT
+08:00';

SQL> USE db_convert

SQL> SELECT CONVERT('test_convert','UTF8');

EXPR1 |
-----|
test_convert|

SQL> SELECT LENGTHB(CONVERT('我爱中国','UTF8','GBK'));
```

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

## 2.11 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

### 返回值

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 |
```

## 2.12 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

### 返回值

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 |
```

## 2.13 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。

### 返回值

BINARY 类型。

### 示例

```
SQL> SELECT RAWTOHEX (ESCAPE_ENCODE (' 4vx\220\253\315\357\000 '))  
      FROM DUAL;  
  
EXPR1 |  
-----  
1234567890ABCDEF0001 |
```

## 2.14 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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT ESCAPE_ENCODE (HEXTORAW ('1234567890ABCDEF0001')) FROM  
      DUAL;  
  
EXPR1 |  
-----  
4vx\220\253\315\357\000 |
```

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

## 返回值

INTEGER 数值类型。

## 示例

- 示例 1

基本使用。

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

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

上述示例结果返回 3，因为 c 是列表的第 3 个元素。

- 示例 2

在查询中使用。

```
# 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 |
```

上述示例表示查询表 tab\_test 中包含 3 的所有记录。

- 示例 3

与字段结合使用。

```
# 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

上述示例表示查询表 tab\_test 中 col\_list 列包含 col\_name 的所有记录。

## 2.16 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。

### 返回值

BINARY 类型。

### 示例

```
SQL> SELECT TO_CHAR (FROM_BASE64 ('MTIzNDU2Nzg5MEFCQ0RFRjAwMDE='))  
      FROM DUAL;  
  
EXPR1 |  
-----  
1234567890ABCDEF0001 |
```

## 2.17 HEADING

### 功能描述

取 VARCHAR 或 CHAR 类型的字段或表达式 expr1 头部的 expr2 个字符的子串。

### 语法格式

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

### 参数说明

- expr1: 字串表达式或字段。
- expr2: 子串长度。

expr: 字符串。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT HEADING ('ABCDEFGHIJK',4) FROM dual;  
  
EXPR1 |  
-----  
ABCD |
```

## 2.18 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。

### 返回值

BINARY 类型。

### 示例

```
SQL> SELECT RAWTOHEX (HEX_DECODE ('1234567890abcdef0001')) FROM DUAL;  
EXPR1 |  
-----  
1234567890ABCDEF0001 |
```

## 2.19 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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT HEX_ENCODE (HEXTORAW ('1234567890ABCDEF0001')) FROM DUAL;  
EXPR1 |  
-----  
1234567890abcdef0001 |
```

## 2.20 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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT INITCAP('heLlO wOrLd'), INITCAP('aBC#DEF 你 cEd');  
EXPR1 | EXPR2 |  
-----  
Hello World| Abc#Def 你 Ced|
```

## 2.21 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 xugu 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

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT INSERT('abc1ABC1abcABcd', 2, 3, 'WWW');  
EXPR1 |  
-----  
aWWWABC1abcABcd|
```

## 2.22 INSTR

### 功能描述

从字符串 expr1 中获取 expr2 的开始位置。

### 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，从左到右开始检索；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，若省略，默认为 1；若为负数，则报错

### 返回值

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 |
```

## 2.23 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。
- xugu：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。

### 返回值

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 |
```

## 2.24 LCASE

### 功能描述

字符串转小写。

### 语法格式

```
LCASE (expr)
```

### 参数说明

expr: 字符串。

### 返回值

CHAR 类型。

### 示例

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

```
EXPR1 |
```

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

```
SQL> SELECT LCASE ('AB123Cd');
```

```
EXPR1 |
```

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

## 2.25 LEFT

### 功能描述

截取字符串 expr1 左侧 expr2 个字符。

### 语法格式

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

### 参数说明

- expr1: 字符串。
- expr2: 截取长度。

## 返回值

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, |
```

## 2.26 LEFTB

### 功能描述

在 VARCHAR 或 CHAR 类型的字段或变量或表达式 expr1 中，从左边开始取 expr2 个字节。

### 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。

## 返回值

VARCHAR 类型字符串。

## 示例

```
SQL> SELECT LEFTB('ABC',2) FROM dual;  
  
EXPR1 |  
-----  
AB |
```

## 2.27 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: 字符串表达式或字段。

### 返回值

INT 数值类型。

### 示例

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

```
EXPR1 |
```

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

## 2.28 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。

### 返回值

INTEGER 类型。

### 示例

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

```
EXPR1 |
```

```
-----  
7 |
```

```
SQL> SELECT LENGTH('中a');
```

```
EXPR1 |
```

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

## 2.29 LENGTHB

### 功能描述

计算参数字符串的长度，单位为字节。

### 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。

### 返回值

INTEGER 数值类型。

### 示例

```
-- 以下执行用例的库，字符集为 UTF-8
```

```
-- 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 |
```

## 2.30 LOCATE

### 功能描述

返回子串在源串中第一次出现的位置。

与 MySQL 差异：

- 对于参数类型为 blob：MySQL 可以返回结果；xugu 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；xugu 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

### 返回值

INTEGER 数值类型。

### 示例

```
SQL> SELECT LOCATE ('b', 'abc1ABC1abcABcd');  
EXPR1 |  
-----  
2 |  
SQL> SELECT LOCATE ('b', 'abc1ABC1abcABcd', 5);  
EXPR1 |  
-----  
10 |
```

## 2.31 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: 字符串表达式或字段。

### 返回值

VARCHAR 类型字符串。

### 示例

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

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

## 2.32 LPAD

### 功能描述

对字符串 expr1 使用指定的字符 [expr3] 进行填充，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: 填充字符串，可选参数，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。

### 返回值

CHAR 类型字符串。

### 示例

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

EXPR1 |
-----|
woai |

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

EXPR1 |
-----|
woai |

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

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

## 2.33 LTRIM

### 功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式头部空格去掉。

### 语法格式

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

### 参数说明

- expr1: 字符串表达式或字段。
- expr2: 字符串表达式或字段。

传入一个参数时，去掉该字段或表达式前面的空格；传入两个参数时，在 expr1 中去掉 expr2。

### 返回值

CHAR 类型字符串。

### 示例

```
SQL> SELECT LTRIM(' F D D') FROM dual;  
  
EXPR1 |  
-----  
F D D|  
  
SQL> SELECT LTRIM('asd','a') FROM dual;  
  
EXPR1 |  
-----  
sd|
```

## 2.34 MID

### 功能描述

字符串截取。

### 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。

### 返回值

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|
```

## 2.35 POSITION

### 功能描述

计算串 expr1 在串 expr2 中的起始位置（即从第几个字符开始），expr2 的第一个字符位置数为 1，其余类推。

### 语法格式

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

### 参数说明

- expr1: 子字符串。
- expr2: 母字符串。

### 返回值

INT 数值类型。

### 示例

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

```
EXPR1 |
```

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

## 2.36 REGEXP\_COUNT

### 功能描述

统计字符串出现的次数。

### 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。

### 返回值

INTEGER 类型。

### 示例

```
SQL> SELECT REGEXP_COUNT('asdasadaa','a');  
  
EXPR1 |  
-----  
5 |
```

## 2.37 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。

### 返回值

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 |
```

## 2.38 REPEAT

### 功能描述

将指定字符串重复 N 次。

### 语法格式

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

### 参数说明

- expr1：字符串。
- expr2：重复的次数，INT 类型。

### 返回值

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 |
```

## 2.39 REPLACE

### 功能描述

将 expr1 中的子串 expr2 用 expr3 替换，expr2 是 expr1 的子串。

### 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。

### 返回值

VARCHAR 类型字符串。

### 示例

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

```
EXPR1 |
```

```
-----  
AB中国EF|
```

## 2.40 REVERSE\_STR

### 功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式反写。

### 语法格式

```
REVERSE_STR(expr)
```

### 参数说明

expr: 字符串表达式或字段。

### 返回值

VARCHAR 类型字符串。

### 示例

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

```
EXPR1 |
```

GFECDBA |

## 2.41 RIGHT

### 功能描述

截取右侧 N 个字符。

### 语法格式

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

### 参数说明

- expr1: 字符串。
- expr2: 截取长度。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT RIGHT ('一二三, 。? 一二, 。? 一三四', 5);
```

```
EXPR1 |
```

```
-----  
。? 一三四 |
```

## 2.42 RIGHTB

### 功能描述

在 VARCHAR 或 CHAR 类型的字段或变量或表达式 expr1 中，从右边开始取 expr2 个字节。

### 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。

### 返回值

VARCHAR 类型字符串。

### 示例

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

```
EXPR1 |
```

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

## 2.43 ROWIDTOCHAR

### 功能描述

将 ROWID 数据类型转换为 CHAR 字符类型输出。

### 语法格式

```
ROWIDTOCHAR(expr)
```

### 参数说明

expr: ROWID 类型的字段或变量或表达式。

### 返回值

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 |
```

```
-----  
AAAAAFUCAAAAAAAAAAAAAAAAAA== |
```

```
AQAAAAFUCAAAAAAAAAAAAAAAAAA== |
```

## 2.44 RPAD

### 功能描述

指定字符串 expr3 填充至 expr1 结尾处，使字符串的长度达到 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: 填充字符串, 可选参数, 填充至源字符串的右边; 如果未指定此参数, 将会在源字符串的右边填充空格。

### 返回值

CHAR 类型字符串。

### 示例

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

```
EXPR1 |
```

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

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

```
EXPR1 |
```

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

```
SQL> SELECT RPAD('tt', 4, 'wzq');
```

```
EXPR1 |
```

```
-----  
ttwz|
```

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

```
EXPR1 |
```

```
-----  
|
```

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

```
EXPR1 |
```

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

## 2.45 RTRIM

### 功能描述

将 VARCHAR 或 CHAR 类型的字段或表达式尾部空格去掉。

### 语法格式

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

### 参数说明

- expr1: 字符串表达式或字段。
- expr2: 字符串表达式或字段。

### 返回值

CHAR 类型字符串。

### 示例

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

EXPR1 |
-----
asd|

SQL> SELECT RTRIM('a s d ') FROM dual;

EXPR1 |
-----
a s d|

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

EXPR1 |
-----
as|

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

EXPR1 |
-----
as|

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

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

## 2.46 SPACE

### 功能描述

返回指定长度的空白字符。

### 语法格式

```
SPACE (expr)
```

### 参数说明

expr: 指定长度的数值。

## 返回值

CHAR 类型字符串。

## 示例

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

```
EXPR1 |
```

```
-----  
|
```

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

```
EXPR1 |
```

```
-----  
|
```

```
SELECT SPACE(4) FROM dual;
```

```
EXPR1 |
```

```
-----  
|
```

## 2.47 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

### 返回值

VARCHAR 类型。

### 示例

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

```
EXPR1 |
```

```
-----  
xugudb|
```

```
Total 1 records.

Use time:0 ms.

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

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

## 2.48 SQLCODE

### 功能描述

返回错误码。

### 语法格式

```
SQLCODE
```

### 参数说明

无参数。

### 返回值

INTEGER 数值类型。

### 示例

```
SQL> CREATE DATABASE db_sqlcode;

SQL> USE db_sqlcode

SQL> BEGIN
2 CREATE TABLE test_sqlcode(a INT);
3 EXCEPTION
4 WHEN OTHERS THEN
5 SEND_MSG(SQLCODE);
6 END;
7 /

SQL> BEGIN
2 CREATE TABLE test_sqlcode(a INT);
3 EXCEPTION
4 WHEN OTHERS THEN
5 SEND_MSG(SQLCODE);
6 END;
7 /
9016

SQL> BEGIN
2 DROP TABLE test_sqlcode2;
3 EXCEPTION
```

```
4  WHEN OTHERS THEN
5  SEND_MSG(SQLCODE);
6  END;
7  /
5021
```

## 2.49 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: 字符串表达式或字段。
- expr2: 起始替换位置。
- expr3: 被替换子串的长度。
- expr4: 替换成的目标串。

### 返回值

VARCHAR 类型字符串。

### 示例

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

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

## 2.50 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 xugu 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; xugu 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的。

### 语法格式

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

### 参数说明

- expr1: 字符串。
- expr2: 字符串。

### 返回值

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 |
```

## 2.51 SUBSTR

### 功能描述

当未指定 expr3 时, 截取 expr1 中 expr2 到末尾的子串; 当指定 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: 字符串表达式或字段。
- expr2: 子串的起始位置。
- expr3: 子串长度。

### 返回值

CHAR 类型字符串。返回值范围: (-1, 0, 1)。

### 示例

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

EXPR1 |
-----|
BCDEFG|

SQL> SELECT SUBSTR('ABCDEF',2,3) FROM dual;

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

## 2.52 SUBSTRB

### 功能描述

截取目标字符串指定的部分。

### 语法格式

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

### 参数说明

- expr1: 为待截取的字符串。
- expr2: 为截取的开始位置（单位是字节）；截取开始位置不管从哪边数，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: expr3: 指定截取字符的字节数；该值小于 1 时结果为 null。

### 📖 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。

### 返回值

VARCHAR 类型字符串。

### 示例

```
-- 示例 1 ~ 3 中，目标字符串为英文，一个字符占一个字节
-- 示例 1：从目标字符串左边数第二个字节处开始截取，向右截取至目标串
      结尾
SELECT SUBSTRB('abcdefg', 2);

EXPR1 |
-----
bcdefg|

-- 示例 2：从目标字符串右边数第二个字节处开始截取，向右截取至目标串
      结尾
SELECT SUBSTRB('abcdefg', -2);

EXPR1 |
-----
fg|

-- 示例 3：从目标字符串左边数第二个字节处开始截取，向右截取三个字节
      结束
SELECT SUBSTRB('abcdefg', 2, 3);
EXPR1 |
-----
bcd|

-- 示例 4 ~ 5 中，目标字符串为中文，在 UTF8 库中一个字符占三个字节
-- 示例 4：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|

-- 示例 5：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 |
-----
```

## 2.53 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；xugu 报错。

对于 pos 参数为 0 的情况：MySQL 结果为空串；xugu 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。

### 返回值

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 |
```

## 2.54 SUBSTRING\_INDEX

### 功能描述

按指定分隔符截取字符串。

### 语法格式

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

### 参数说明

- expr1: 字符串。
- expr2: 分隔符。
- expr3: 序号，为非 0 整数。若为整数则表示从左到右数，若为负数则从右到左数。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT SUBSTRING_INDEX('www.xugu.com', '.', 1) FROM dual;  
EXPR1 |  
-----  
www |
```

## 2.55 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: 字符串表达式或字段。
- expr2: 子串长度。

## 返回值

VARCHAR 类型字符串。

## 示例

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

```
EXPR1 |
```

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

## 2.56 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 的 TO\_BASE64 函数使用无差异。

### 语法格式

```
TO_BASE64(expr)
```

### 参数说明

expr: 字符串。

### 返回值

CHAR 类型。

### 示例

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

```
EXPR1 |
```

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

## 2.57 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

### 返回值

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|
```

## 2.58 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；否则，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: 字符串字段或表达式。
- expr2: 字符串字段或表达式。
- expr3: 字符串字段或表达式。

### 返回值

VARCHAR 类型字符串。

### 示例

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

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

## 2.59 UCASE

### 功能描述

字符串转大写。

### 语法格式

```
UCASE (expr)
```

### 参数说明

expr: CHAR 类型的字符串。

### 返回值

CHAR 类型。

### 示例

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

EXPR1 |
-----|
ABCD|

SQL> SELECT UCASE ('ab123cd') ;

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

## 2.60 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。

### 返回值

VARCHAR 类型字符串。

### 示例

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

```
EXPR1 |
```

```
-----  
abc |
```

```
SELECT HEX('abc') FROM dual;
```

```
EXPR1 |
```

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

## 2.61 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: 字符串表达式或字段。

### 返回值

VARCHAR 类型字符串。

### 示例

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

```
EXPR1 |
```

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

## 2.62 WM\_CONCAT

### 功能描述

将列 expr 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。

### 返回值

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(3,'小王')(1,'张三')(2,'李四');  
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 |
```

# 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
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_SUB	时间减
DATEDIFF	计算两个时间差
DAY	返回日期为当月的第几天
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
接下页	

函数形式	功能
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
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
接下页	

函数形式	功能
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
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
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
SUBDATE	时间减
接下页	

函数形式	功能
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
SYSTEMTIME	取系统当前时间，格式为 HH24:MI:SS.FFF
SYSTEMSTAMP	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
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
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
WEEK	返回指定日期所在的周数
接下页	

函数形式	功能
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'
接下页	

时间间隔	格式
INTERVAL DAY TO MINUTE	'DAYS HOURS:MINUTES'
INTERVAL DAY TO SECOND	'DAYSHOURS:MINUTES:SECONDS.MICROSECOND S'
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,....
%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
%l	小时	01..12
%i	分钟	数值: 00..59
%j	年中的某天	001..366
接下页		

格式符	描述	备注
%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
%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
接下页		

格式符	描述	备注
%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

时间加。

### 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。

### 返回值

DATE/TIME/DATETIME。

### 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 `expr1` 基础上增加 `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，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`: INTEGER 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。

### 返回值

日期、日期时间或带时区的日期时间类型，包括 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 |
```

## 3.4 ADDTIME

### 功能描述

时间加。

### 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。

## 返回值

DATE/TIME/DATETIME。

## 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 CURDATE

### 功能描述

获取当前日期。

### 语法格式

```
CURDATE ()
```

### 参数说明

无参数。

### 返回值

DATE 类型。

### 示例

```
SELECT CURDATE ();  
  
EXPR1 |  
-----  
2022-11-15 AD |
```

## 3.6 CURRENT\_DATETIME

### 功能描述

输出当前时间。

### 语法格式

```
CURRENT_DATETIME
```

### 参数说明

无参数。

### 返回值

DATETIME 类型。

### 示例

```
SQL> SELECT CURRENT_DATETIME FROM dual;

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

## 3.7 CURTIME

### 功能描述

返回当前时间。

### 语法格式

```
CURTIME ()
```

### 参数说明

无参数。

### 返回值

TIME 类型。

### 示例

```
SQL> SELECT CURTIME ();

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

## 3.8 DATE

### 功能描述

获取指定日期时间中的日期。

### 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]

### 返回值

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 |
```

## 3.9 DATE\_ADD

### 功能描述

时间加。

### 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。

### 返回值

DATE/TIME/DATETIME。

### 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 |
```

## 3.10 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, 会按照自己的规则做转换; xugu 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；xugu 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）；xugu 不支持。
- 对于常值 NULL 为参数：MySQL 返回 NULL；xugu 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。

### 返回值

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|
```

## 3.11 DATE\_SUB

### 功能描述

时间减。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，xugu 返回具体结果。
- DATE\_SUB 函数运算 INTERVAL xx YEAR TO MONTH 时，当 MONTH 超过 12 时，虚谷报错，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。

### 返回值

DATE/TIME/DATETIME。

### 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 |
```

## 3.12 DATEDIFF

### 功能描述

计算两个时间差。

### 语法格式

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

### 参数说明

- expr1: DATE/TIME/DATETIME 类型时间。
- expr2: DATE/TIME/DATETIME 类型时间。

### 返回值

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 |
```

## 3.13 DAY

### 功能描述

返回日期为当月的第几天。

### 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]

### 返回值

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 |
```

## 3.14 DAYNAME

### 功能描述

返回指定日期所在工作日名称。

### 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]

### 返回值

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|
```

## 3.15 DAYOFMONTH

### 功能描述

返回日期为当月的第几天。

### 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]

### 返回值

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 |
```

## 3.16 DAYOFWEEK

### 功能描述

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）。

### 语法格式

```
DAYOFWEEK (expr)
```

### 参数说明

expr: DATETIME 类型的字段或变量或表达式。

### 返回值

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>|
```

## 3.17 DAYOFYEAR

### 功能描述

返回参数日期为一年中的第几天。

### 语法格式

```
DAYOFYEAR(expr)
```

### 参数说明

expr: DATETIME 类型的字段或变量或表达式。

### 返回值

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>|
```

## 3.18 EXTRACT

### 功能描述

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）。

### 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。

### 返回值

INTEGER, 与 DETEFIELD 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 |
```

## 3.19 EXTRACT\_DAY

### 功能描述

取日期的天。

### 语法格式

```
EXTRACT_DAY(expr)
```

### 参数说明

expr: 时间类型的字段或表达式。

### 返回值

INTEGER。

### 示例

```
SQL> SELECT EXTRACT_DAY('2022-05-20') FROM dual;  
  
EXPR1 |  
-----  
20 |
```

## 3.20 EXTRACT\_HOUR

### 功能描述

取日期的小时。

### 语法格式

```
EXTRACT_HOUR(expr)
```

### 参数说明

expr: 时间类型的字段或表达式。

### 返回值

INTEGER 类型。

### 示例

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

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

## 3.21 EXTRACT\_MINUTE

### 功能描述

取日期的分钟。

### 语法格式

```
EXTRACT_MINUTE(expr)
```

### 参数说明

expr: 时间类型的字段或表达式。

### 返回值

INTEGER 类型。

### 示例

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

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

## 3.22 EXTRACT\_MONTH

### 功能描述

取日期的月。

### 语法格式

```
EXTRACT_MONTH(expr)
```

### 参数说明

expr: 时间类型的字段或表达式。

### 返回值

INTEGER 类型。

### 示例

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

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

## 3.23 EXTRACT\_SECOND

### 功能描述

取日期的秒。

### 语法格式

```
EXTRACT_SECOND(expr)
```

### 参数说明

expr: 时间类型的字段或表达式。

### 返回值

数值类型 DOUBLE。

### 示例

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

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

## 3.24 EXTRACT\_YEAR

### 功能描述

取日期的年。

### 语法格式

```
EXTRACT_YEAR(expr)
```

### 参数说明

expr: 时间类型的字段或表达式。

### 返回值

INTEGER 类型。

### 示例

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

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

## 3.25 FROM\_DAYS

### 功能描述

数字转时间（TO\_DAYS 函数的反向函数）。

### 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]

### 返回值

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 |
```

## 3.26 FROM\_UNIXTIME

### 功能描述

将时间戳（秒）转换为时间（0 时区的时间）。

### 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]

### 返回值

DATETIME。

#### 📖 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 |  
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 |
```

## 3.27 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

## 返回值

DATETIME 类型。

## 示例

```
SQL> SELECT get_boot_time();  
  
EXPR1 |  
-----  
2024-05-10 14:13:50.397 AD |
```

## 3.28 GETDAY

### 功能描述

获取当前时间数据类型参数 expr 中的日。

### 语法格式

```
GETDAY(expr)
```

### 参数说明

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

### 返回值

INTEGER 类型。

### 示例

```
SQL> SELECT GETDAY('2022-05-20 02:00:00') FROM dual;  
  
EXPR1 |  
-----  
20 |
```

## 3.29 GETHOUR

### 功能描述

获取当前时间数据类型参数 expr 中的小时。

### 语法格式

```
GETHOUR(expr)
```

### 参数说明

expr: DATETIME、TIME 类型的字段或表达式。

### 返回值

INTEGER 类型。

### 示例

```
SQL> SELECT GETHOUR('2022-05-20 02:00:00') FROM dual;  
  
EXPR1 |  
-----  
2 |
```

## 3.30 GETMINUTE

### 功能描述

获取当前时间数据类型参数 expr 中的分钟。

### 语法格式

```
GETMINUTE(expr)
```

### 参数说明

expr: DATETIME、TIME 类型的字段或表达式。

### 返回值

日期时间类型 DATETIME。

### 示例

```
SQL> SELECT GETMINUTE('2022-05-20 02:28:00') FROM dual;  
  
EXPR1 |  
-----  
28 |
```

## 3.31 GETMONTH

### 功能描述

获取当前时间数据类型参数 expr 中的月份。

### 语法格式

```
GETMONTH (expr)
```

### 参数说明

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

### 返回值

INTEGER 类型。

### 示例

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

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

## 3.32 GETSECOND

### 功能描述

获取当前时间数据类型参数 expr 中的秒。

### 语法格式

```
GETSECOND (expr)
```

### 参数说明

expr: DATETIME、TIME 类型的字段或表达式。

### 返回值

INTEGER 类型。

### 示例

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

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

## 3.33 GET\_UPTIME

### 功能描述

获取当前节点上虚谷数据库服务已经运行的秒数。

### 语法格式

```
GET_UPTIME()
```

### 参数说明

无

### 返回值

BIGINT 类型。

### 示例

```
SQL> SELECT GET_UPTIME();
```

```
EXPR1 |
```

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

## 3.34 GETYEAR

### 功能描述

获取当前时间数据类型参数 `expr` 中的年份。

### 语法格式

```
GETYEAR(expr)
```

### 参数说明

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

### 返回值

INTEGER 类型。

### 示例

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

```
EXPR1 |
```

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

## 3.35 GET\_FORMAT

### 功能描述

返回一个时间格式，固定搭配，常与 DATE\_FORMAT、STR\_TO\_DATE 结合使用。

### 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'。

### 返回值

CHAR。

### 示例

```
SQL> SELECT GET_FORMAT (DATE, 'USA');  
  
EXPR1 |  
-----  
%m.%d.%Y|
```

## 3.36 HOUR

### 功能描述

返回时间中的小时。

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 xugu 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, xugu 需要先把参数做显式转换。例如: MySQL: SELECT HOUR('03:04:05'); xugu: 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]

#### 返回值

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 |
```

## 3.37 LAST\_DAY

#### 功能描述

获取当前时间数据类型参数 expr 中所在月的最后一天。

#### 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。

## 返回值

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>|
```

## 3.38 LOCALTIME

### 功能描述

返回系统时间，同 SYSDATE。

与 MySQL 差异：

LOCALTIME 做加减运算同 SYSDATE，xugu 加减一个以天为单位的数值，MySQL 加减一个以秒为单位的数值。

### 语法格式

```
LOCALTIME ()
```

### 参数说明

无参数。

## 返回值

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 |
```

## 3.39 LOCALTIMESTAMP

### 功能描述

返回系统时间，同 SYSDATE。与 MySQL 差异：

LOCALTIMESTAMP 做加减运算同 SYSDATE，xugu 加减一个以天为单位的数值，MySQL 加减一个以秒为单位的数值。

### 语法格式

```
LOCALTIMESTAMP ()
```

### 参数说明

无参数。

### 返回值

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 |
```

## 3.40 MAKE\_DATE

### 功能描述

根据年，月，日创建日期。

### 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

### 返回值

DATE

### 示例

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

```
EXPR1 |
```

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

## 3.41 MAKE\_TIME

### 功能描述

根据时，分，秒创建时间。

### 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

### 返回值

TIME 类型。

### 示例

```
SQL> SELECT MAKE_TIME(15,32,45.211);  
  
EXPR1 |  
-----  
15:32:45.211 |
```

## 3.42 MAKE\_TIMESTAMP

### 功能描述

根据年，月，日，时，分，秒创建日期时间。

### 语法格式

```
MAKE_TIMESTAMP(year,month,day,hour,min,sec)
```

### 参数说明

- year: INT 类型，小于 0 表示公元前
- month: INT 类型
- day: INT 类型
- hour: INT 类型
- min: INT 类型
- sec: DOUBLE 类型

### 返回值

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 |
```

## 3.43 MICROSECOND

### 功能描述

返回时间中的微秒。

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 中是合法的，而在 xugu 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，xugu 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，xugu 需要先把参数做显示转换。例如：MySQL: SELECT MICROSECOND('03:04:05'); xugu: 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]

### 返回值

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 |
```

## 3.44 MINUTE

### 功能描述

返回时间中的分钟。

### 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]

### 返回值

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 |
```

## 3.45 MONTH

### 功能描述

返回日期的月份。

### 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]

### 返回值

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 |
```

## 3.46 MONTHNAME

### 功能描述

返回日期/日期时间所在月份的英文名。

### 语法格式

```
MONTHNAME (expr)
```

### 参数说明

expr: DATETIME 类型的字段或变量或表达式。

### 返回值

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>|
```

## 3.47 MONTHS\_BETWEEN

### 功能描述

取时间的月份差，expr1 为起始日期，expr2 为截至日期，若 expr1 大于 expr2，则返回负数。

### 语法格式

```
MONTHS_BETWEEN(expr1,expr2)
```

### 参数说明

- expr1: DATETIME 类型的字段或变量或表达式。
- expr2: DATETIME 类型的字段或变量或表达式。

### 返回值

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 |
```

## 3.48 NEXT\_DAY

### 功能描述

计算距离指定时间最近的周几的日期。

### 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`。

### 返回值

日期时间类型 `DATETIME`。

### 示例

```
SQL> SELECT NEXT_DAY('2022-02-22 02:00:00',1) FROM dual;

EXPR1 |
-----
2022-02-27 02:00:00.000 AD |
```

## 3.49 NOW

### 功能描述

取当前时间。

### 语法格式

```
NOW()
```

### 参数说明

无参数。

### 返回值

`DATETIME` 类型。

### 示例

```
SQL> SELECT NOW() FROM dual;

EXPR1 |
-----
2022-05-14 10:49:24.759 AD |
```

## 3.50 PERIOD\_ADD

### 功能描述

给指定时间增加特定的月份数。

### 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。

### 返回值

INTEGER。

### 示例

```
SQL> SELECT PERIOD_ADD(202212,1) FROM dual;
```

```
EXPR1 |
```

```
-----  
202301 |
```

```
SQL> SELECT PERIOD_ADD(202212,-1) FROM dual;
```

```
EXPR1 |
```

```
-----  
202211 |
```

## 3.51 PERIOD\_DIFF

### 功能描述

返回两个时间相差的月份数, 前一个时间减后一个时间。

### 语法格式

```
PERIOD_DIFF(expr1,expr2)
```

### 参数说明

- expr1: 时间段, 格式为 YYMM 或 YYYYMM。
- expr2: 时间段, 格式为 YYMM 或 YYYYMM。

### 返回值

INTEGER。

### 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 |
```

## 3.52 QUARTER

### 功能描述

返回指定日期所在季度。

### 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]

### 返回值

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 |
```

## 3.53 SECOND

### 功能描述

返回时间中的秒。

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 xugu 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；xugu 需要先把参数做显示转换。例如：MySQL: SELECT SECOND('03:04:05'); xugu: 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]

## 返回值

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 |
```

## 3.54 SEC\_TO\_TIME

### 功能描述

将指定秒数转换为时间。

### 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]

## 返回值

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 |
```

## 3.55 SUBDATE

### 功能描述

时间减。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷返回具体结果。
- SUBDATE 函数运算 INTERVAL xx YEAR TO MONTH，当 month 超过 12 时，虚谷报错，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。

### 返回值

DATE/TIME/DATETIME。

### 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 |
```

## 3.56 SUBTIME

### 功能描述

时间减。

与 MySQL 差异：

- 当计算结果超过范围后，MySQL 返回 NULL，虚谷返回具体结果。
- SUBDATE 函数运算 INTERVAL xx YEAR TO MONTH，当 month 超过 12 时，虚谷报错，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

```
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。

### 返回值

DATE/TIME/DATETIME。

## 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 |
```

## 3.57 SYSDATE

### 功能描述

取系统当前时间。

### 语法格式

```
SYSDATE ()
```

### 参数说明

无参数。

### 返回值

DATETIME 类型。

### 示例

```
SQL> SELECT SYSDATE () FROM dual;  
  
EXPR1 |  
-----  
2022-05-14 10:49:52.199 AD |
```

## 3.58 SYSDATETIME

### 功能描述

取系统当前日期时间，格式为 YYYY-MM-DD HH24:MI:SS.FFF。

## 语法格式

```
SYSDATETIME ()
```

## 参数说明

无参数。

## 返回值

日期时间类型 DATETIME。

## 示例

```
SQL> SELECT SYSDATETIME () FROM dual;
```

```
EXPR1 |
```

```
-----  
2022-05-14 10:50:20.535 AD |
```

## 3.59 SYSTIME

### 功能描述

取系统当前时间，格式为 HH24:MI:SS.FFF。

### 语法格式

```
SYSTIME ()
```

### 参数说明

无参数。

### 返回值

时间类型 TIME。

### 示例

```
SQL> SELECT SYSTIME () FROM dual;
```

```
EXPR1 |
```

```
-----  
10:50:47.408 |
```

## 3.60 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。

### 返回值

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 |
```

## 3.61 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，因为后者允许小时数大于 23，如 272:59:59 在 MySQL 中合法，而在 xugu 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，xugu 需要先把参数做显示转换。例如：MySQL: SELECT HOUR('2000-01-01 03:04:05'); xugu: 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]

### 返回值

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 |
```

## 3.62 TIME\_FORMAT

### 功能描述

按指定格式串格式化时间。

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；xugu 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；xugu 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；xugu 执行报错。
- 对于常值 NULL 为参数：MySQL 返回 NULL；xugu 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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT TIME_FORMAT('13:14:15.123456', '%H:%i:%s');  
  
EXPR1 |  
-----  
13:14:15|
```

## 3.63 TIMEDIFF

### 功能描述

计算两个时间差。

与 MySQL 差异：

- 参数都为 DATE 类型时，MySQL 返回的是年，xugu 返回的是小时。
- 两个参数的类型不相同时，MySQL 直接返回 NULL，xugu 能做运算。

### 语法格式

```
TIMEDIFF(expr1,expr2)
```

### 参数说明

- expr1：DATE/TIME/DATETIME 类型时间。
- expr2：DATE/TIME/DATETIME 类型时间。

### 返回值

INTERVAL HOUR TO SECOND。

#### 📖 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 |
```

## 3.64 TIME\_TO\_SEC

### 功能描述

指定时间转换为秒。

### 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]

### 返回值

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 |
```

## 3.65 TIMESTAMP

### 功能描述

将传入的参数转为 DATETIME。

与 MySQL 差异：

TIMESTAMP 第二个参数，xugu 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。

### 返回值

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 |
```

## 3.66 TIMESTAMPADD

### 功能描述

时间加。

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）。

### 返回值

DATE/DATETIME/DATETIME WITH TIME ZONE。

### 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 |
```

## 3.67 TIMESTAMPDIFF

### 功能描述

时间减。

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)。

### 返回值

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 |
```

## 3.68 TO\_DATE

### 功能描述

把字符串 expr1 转换为日期时间值，expr2 指定 expr1 的日期时间格式。

### 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。详细信息请参见《SQL 语法参考指南》的会话变量 > 连接会话参数章节。

日期时间格式说明：

格式（不区分大小写）	描述
YY/YYYY/YYYYY	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。

### 返回值

日期时间类型 DATETIME。

## 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 |
```

## 3.69 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'）；xugu 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]

### 返回值

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 |
```

## 3.70 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]

### 返回值

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 |
```

## 3.71 TO\_TIMESTAMP

### 功能描述

将字符串 expr1 按照指定格式转换为 TIMESTAMP 类型。

### 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'。

### 返回值

日期时间类型 TIMESTAMP。

### 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 |
```

## 3.72 TRANSACTION\_TIMESTAMP

### 功能描述

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。

### 语法格式

```
TRANSACTION_TIMESTAMP()
```

### 参数说明

无参数

### 返回值

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 |
```

## 3.73 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，xugu 则报错。
- 传入错误时间，MySQL 返回 NULL，xugu 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

### 返回值

TINYINT。

### 示例

```
SQL> SELECT WEEK('2023-03-16'), WEEK('2023-03-16', 3);  
EXPR1 | EXPR2 |  
-----  
11 | 11 |
```

## 3.74 WEEKDAY

### 功能描述

返回指定日期在工作日中的索引。

### 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]

### 返回值

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 |
```

## 3.75 WEEKOFYEAR

### 功能描述

返回指定日期所在的日历周，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，xugu 则报错。

### 语法格式

```
WEEKOFYEAR(expr)
```

### 参数说明

expr: DATE 或 DATETIME 类型。

### 返回值

TINYINT。

### 示例

```
SQL> SELECT WEEKOFYEAR('2023-03-16');  
EXPR1 |  
-----  
11 |
```

## 3.76 YEAR

### 功能描述

返回日期的年份。

与 MySQL 差异：

虚谷不支持参数为 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，这是时间格式上的差异，暂不处理。

### 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]

### 返回值

INTEGER。

#### 📖 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 |
```

## 3.77 YEARWEEK

### 功能描述

计算指定日期所在的周数，并返回年份和周数。

与 MySQL 差异：

- 第二个参数为布尔型时，MySQL 可以返回值，xugu 则报错。
- 传入错误时间，MySQL 返回 NULL，xugu 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。

### 返回值

INT。

### 示例

```
SQL> SELECT YEARWEEK('2023-03-16');  
  
EXPR1 |  
-----  
202311 |
```

## 3.78 UNIX\_TIMESTAMP

### 功能描述

返回参数对应的 UNIX 时间戳。其值是从'1970-01-01 00:00:00.000' 开始经过的秒数。

### 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。

### 返回值

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 |  
-----
```

## 3.79 UTC\_DATE

### 功能描述

返回 UTC 日期，与 SYSDATE 相差 8 小时。

### 语法格式

```
UTC_DATE()
```

### 参数说明

无参数。

### 返回值

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 |
```

## 3.80 UTC\_TIME

### 功能描述

返回 UTC 时间。

### 语法格式

```
UTC_TIME ()
```

### 参数说明

无参数。

### 返回值

TIME。

### 示例

```
SQL> SELECT UTC_TIME ();  
  
EXPR1 |  
-----  
07:09:18.821 |
```

## 3.81 UTC\_TIMESTAMP

### 功能描述

返回 UTC 日期时间，与 SYSDATE 相差 8 小时。

### 语法格式

```
UTC_TIMESTAMP ()
```

### 参数说明

无参数。

### 返回值

DATETIME。

### 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。

### 返回值

INTEGER。

### 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。

### 返回值

CHAR。

### 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, xugu 未限制。

### 语法格式

```
HEXTORAW (expr)
```

### 参数说明

expr: 十六进制的字符串。

### 返回值

BINARY。

### 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 指定了 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。

### 返回值

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 指定了 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。

### 返回值

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, xugu 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。

### 返回值

CHAR。

### 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。

## 返回值

BLOB。

## 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。

### 返回值

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: 空值、数值型、字符型和日期型字段。
- 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。

### 返回值

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: 空值、数值型、字符型和日期型字段。
- 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。

### 返回值

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: 空值、数值型、字符型和日期型字段。
- expr2: 空值、数值型、字符型和日期型字段。
- expr3: 空值、数值型、字符型和日期型字段。

### 返回值

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 JSON 数据类型函数

## 6.1 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 JSON 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

## 6.2 JSON\_ARRAY

### 功能描述

构建 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）。

### 返回值

JSON 数值类型。

### 示例

```
-- 参数组合形成 JSON 数组
```

```
SQL> SELECT TO_CHAR(JSON_ARRAY(1, true, null, 'null'));

EXPR1 |
-----
[1, true, null, "null"]|

SQL> SELECT TO_CHAR(JSON_ARRAY(123, 'abc', NULL, TRUE, FALSE, NOW()
));

EXPR1 |
-----
[123, "abc", null, true, false, "2024-04-07 10:03:00"]|

-- 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"]|
```

## 6.3 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**。
- 如果 **json\_doc** 是无效的 **JSON** 文档或任何 **path** 是无效的路径表达式或路径表达式中包含 **\*** 或 **\*\*** 通配符或路径选择不是数组元素，都将抛出错误。
- 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。

## 返回值

JSON 数值类型。

## 示例

```
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]|
```

## 6.4 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。
- 如果 json\_doc 是无效的 JSON 文档或任何 path 是无效的路径表达式或路径表达式中包含 \* 或 \*\* 通配符或路径选择不是数组元素，都将抛出错误。
- 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。

### 返回值

JSON 数值类型。

### 示例

```
SELECT TO_CHAR(JSON_ARRAY_INSERT('["a", ["b", "c"], "d"]', '$[0]',  
    , 1, '$[1]', 2, '$[3]', 3));  
  
EXPR1 |  
-----  
[1, 2, "a", 3, ["b", "c"], "d"]  
  
-- 在数组指定位置插入元素
```

```
SQL> SELECT TO_CHAR(JSON_ARRAY_INSERT('[1, [2, 3], {"a": [4, 5]}]',  
    , '$[0]', 0));  
  
EXPR1 |  
-----  
[0, 1, [2, 3], {"a": [4, 5]}]|  
  
-- 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]}]|
```

## 6.5 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**。

### 返回值

INTEGER 数值类型。

### 示例

```
SELECT TO_CHAR(JSON_CONTAINS( '[1, [1,2], 3]', '1', '$[1]'));
```

```
EXPR1 |
```

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

```
-- 数组类型
```

```
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')),  
TO_CHAR(JSON_CONTAINS('[1, 2, [3, 4]]', '2', '$[2]'));
```

```
EXPR1 | EXPR2 |
```

```
-----  
1 | 0 |
```

## 6.6 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。
- path: 一个或多个, JSON 路径表达式, 字符类型。

### 注意

- 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。

## 返回值

INTEGER 数值类型。

## 示例

```
SQL> SELECT TO_CHAR(JSON_CONTAINS_PATH( '[1,[1,2],3]', 'one', '$[1]', '$[3]'));
```

EXPR1
1

```
SQL> SELECT TO_CHAR(JSON_CONTAINS_PATH( '[1,[1,2],3]', 'all', '$[1]', '$[3]'));
```

EXPR1
0

```
-- 数组类型
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 vs all
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`;
```

```
')) as `all`';  
  
one | all |  
-----  
1 | 0 |
```

## 6.7 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, 接送文档深度大于 2。

### 返回值

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 |  
  
-- 包含深度为 1 的元素的数组, 所有成员的值的深度为 1 的对象  
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 |
```

## 6.8 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。

### 返回值

JSON 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '  
$.name'));  
  
EXPR1 |  
-----  
"Aztalan"|  
  
SELECT TO_CHAR(JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$  
.*'));  
  
EXPR1 |  
-----  
[1, 2, [3, 4, 5]]|  
  
-- 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] |
```

## 6.9 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。
- 如果 **json\_doc** 与任意 **path** 无效或 **path** 包含 **\*** 或 **\*\*** 通配符，都将抛出错误。

**返回值**

JSON 数值类型。

**示例**

```
SQL> SELECT TO_CHAR(JSON_INSERT('["a", {"b": 1, "c": 2}]', '$[0]',
    1, '$[1].d', 2, '$[4]', 3));

EXPR1 |
-----
["a", {"b": 1, "c": 2, "d": 2}, 3]|

-- 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 } }
```

## 6.10 JSON\_KEYS

### 功能描述

从 JSON 对象的顶层值中返回对象键的 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。
- json\_doc 不为有效 JSON 文本或 path 不为有效路径表达式或路径表达式中包含 \* 或 \*\* 通配符, 都将抛出错误。

### 返回值

JSON 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_KEYS('{ "a": 1, "b": 2 }', '$'));
EXPR1 |
-----
["a", "b"]

-- 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"] |  
-- 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"] |  
-- 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]'));  
EXPR1 | EXPR2 | EXPR3 | EXPR4 |  
-----  
<NULL>| <NULL>| <NULL>| <NULL>|
```

## 6.11 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。
- **json\_doc** 不为有效 **JSON** 文本或 **path** 不为有效路径表达式或路径表达式中包含 **\*** 或 **\*\*** 通配符，都将抛出错误。

**返回值**

JSON 数值类型。

**示例**

```
SQL> SELECT TO_CHAR(JSON_LENGTH('{ "a": 1, "b": 2}', '$'));
EXPR1 |
-----
2 |

-- 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 |
```

## 6.12 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。

### 返回值

JSON 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_MERGE('{"a": 1, "b": 2}', '{"a": 3}'));  
  
EXPR1 |
```

```
-----  
{"a": [1, 3], "b": 2}|  
-- 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]|
```

## 6.13 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。

## 返回值

JSON 数值类型。

## 示例

```
SQL> SELECT TO_CHAR(JSON_MERGE_PATCH('{"a": null, "b": 1, "e": 5}',
    , '{"c": 2, "b": null, "d": null, "e": 10}'));

EXPR1 |
-----
{"a": null, "c": 2, "e": 10}|

-- 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
SQL> SELECT TO_CHAR(JSON_MERGE_PATCH('{"x": 1, "z": 7}', '{"x":
    ": 2, "y": 3}'));

EXPR1 |
-----
{"x": 2, "y": 3, "z": 7}|
```

```
-- 若第二个参数中存在值为 null 的键，则该键不会出现结果对象中
SQL> SELECT TO_CHAR(JSON_MERGE_PATCH('{"x": 1, "z": 7}', '{"x
      ": 2, "z": null}'));
EXPR1 |
-----
{"x": 2}|
```

## 6.14 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。

### 返回值

JSON 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_MERGE_PRESERVE('{"a": 1, "b": 2}', '1'));
EXPR1 |
-----
[{"a": 1, "b": 2}, 1]|
```

```

-- 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('{"x": 1}', '{"x": 2, "y": 3}'));

EXPR1 |
-----|
{"x": [1, 2], "y": 3}|

-- 合并数组和对象
SQL> SELECT TO_CHAR(JSON_MERGE_PRESERVE('{"x": 1}', '[1, 2]'));

EXPR1 |
-----|
[{"x": 1}, 1, 2]|
    
```

## 6.15 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。

### 返回值

JSON 数值类型。

### 示例

```
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" }
}

```

## 6.16 JSON\_OVERLAPS

### 功能描述

比较两个 JSON 文档。

### 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。

## 返回值

INTEGER 数值类型。

## 示例

```
SQL> SELECT TO_CHAR(JSON_OVERLAPS('[1,3,4,5]', '[3,9,10]'));
EXPR1 |
-----
1 |

-- 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 |
```

## 6.17 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。

### 返回值

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
  }
}
```

## 6.18 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。

### 返回值

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" |
```

## 6.19 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

- 如果任何参数为 NULL 则返回 NULL。
- 从左往右计算, 产生的新值用于后续计算。
- 若路径不存在值, 则被忽略。
- json\_doc 不为有效 JSON 文本或 path 不为有效路径表达式, 或者为 \$ 或包含 \*、\*\* 通配符, 都将抛出错误。

### 返回值

JSON 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_REMOVE('[1,3,4,5]', '$[3]', '$[100]'));  
EXPR1 |  
-----  
[1, 3, 4] |  
  
-- 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 3 的元素  
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
SQL> SELECT TO_CHAR(JSON_REMOVE('{"x": 1, "y": 2}', '$.x'));
EXPR1 |
-----
{"y": 2}|
```

## 6.20 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

- 如果任何参数为 NULL 则返回 NULL。
- 从左往右计算, 产生的新值用于后续计算。
- 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。
- json\_doc 不为有效 JSON 文本或 path 不为有效路径表达式, 或者为 \$ 或包含 \*、\*\* 通配符, 都将抛出错误。

### 返回值

CHAR 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_REPLACE('[1, 3, 4, 5]', '$[3]', 100));
```

```
EXPR1 |
-----
[1, 3, 4, 100]|

-- 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}}|
```

## 6.21 JSON\_SCHEMA\_VALID

### 功能描述

验证 json\_doc 是否符合 json\_doc\_schema。

### 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。

**返回值**

INTEGER 数值类型

**示例**

```
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 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 |
```

## 6.22 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。



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。

### 返回值

JSON 数值类型。

### 示例

```
-- JSON文本是否符合JSON模式
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"}|
```

## 6.23 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。
- `json_doc` 不为有效 json 文本或 `path` 不为有效路径表达式, 或者包含 \*、\*\* 通配符, 都将抛出错误。

## 返回值

JSON 数值类型。

## 示例

```
SQL> SELECT TO_CHAR(JSON_SEARCH('["测试"]', 'one', '测%', '', '$'))
;
EXPR1 |
-----
"$[0]" |

-- 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(第一个匹配的路径) vs all(返回所有匹配的路径)
```

```
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"] |
```

## 6.24 JSON\_SET

### 功能描述

替换或插入新值。

### 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。
- 如果 json\_doc 与任意 path 无效或 path 包含 \* 或 \*\* 通配符，都将抛出错误。

**返回值**

JSON 数值类型。

**示例**

```
-- 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] }
```

## 6.25 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。

### 返回值

CHAR 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_TYPE('true')), TO_CHAR(JSON_TYPE('null'))  
      , TO_CHAR(JSON_TYPE('"abc"'));  
  
EXPR1 | EXPR2 | EXPR3 |  
-----  
BOOLEAN| NULL | STRING|  
  
-- 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 |
```

## 6.26 JSON\_UNQUOTE

### 功能描述

取消 JSON 值引用并返回。

### 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

### 返回值

JSON 数值类型。

### 示例

```
SQL> SELECT TO_CHAR(JSON_UNQUOTE('"\u0031\u0032\u0033"));  
  
EXPR1 |  
-----  
123 |  
  
-- 去除带双引号字符串中的双引号，返回字符串  
SQL> SELECT TO_CHAR(JSON_UNQUOTE('"abc"'));  
  
EXPR1 |  
-----  
abc |  
  
-- 去除JSON格式中的双引号  
SQL> SELECT TO_CHAR(JSON_UNQUOTE(CAST('"abc"' AS JSON)));  
  
EXPR1 |  
-----  
abc |
```

## 6.27 JSON\_VALID

### 功能描述

判断 VAL 是否为合格的 JSON 文本。

### 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。

### 返回值

INTEGER 数值类型。

### 示例

```
-- 数值类型、带单引号数值类型字符串判断是否是一个有效的 JSON 文档  
SELECT TO_CHAR(JSON_VALID(1)), TO_CHAR(JSON_VALID('1'));  
) ;  
EXPR1 | EXPR2 |  
-----  
0 | 1 |
```

```
-- 布尔类型、带单引号的布尔类型字符串判断是否是一个有效的 JSON 文档
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判断是否是一个有效的 JSON 文档
SELECT TO_CHAR(JSON_VALID('abc')), TO_CHAR(JSON_VALID('"abc"'));

EXPR1 | EXPR2 |
-----
0 | 1 |
```

# 7 BIT 数据类型函数

## 7.1 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

## 7.2 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。

### 返回值

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 |
```

## 7.3 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。

### 返回值

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 |
```

## 7.4 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。

### 返回值

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 |
```

# 8 XML 数据类型函数

## 8.1 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

## 8.2 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

### 返回值

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
```

## 8.3 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

### 返回值

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 |
-----
```

## 8.4 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

### 返回值

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 |
```

## 8.5 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

XMLEMENT:

```
XMLEMENT(xmlname[,xml_attributes,xmlvalue])
```

xml\_attributes:

```
xml_attributes : XMLATTRIBUTES(xml_attribute_list)
xml_attribute_list : xml_attribute.....
xml_attribute: val as name
| ident;
```



注意

XMLATTRIBUTES 仅支持在 XMLEMENT 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

### 返回值

CHAR 类型。

### 示例

```
-- 单参数, 只有标签名
SQL> SELECT XMLEMENT("name");

EXPR1 |
-----|
<name></name>|

-- 双参数, 标签名和属性值
SQL> SELECT XMLEMENT("name",XMLATTRIBUTES('wang' AS "class"));
EXPR1 |
-----|
<name class="wang"></name>|
```

```
-- 双参数, 标签名和节点值
SQL> SELECT XMLELEMENT("name", 'xugu');
EXPR1 |
-----
<name>xugu</name>|

-- 三参数, 标签名、属性值和节点值
SQL> SELECT XMLELEMENT("name", XMLATTRIBUTES('wang' as "class"), '
      xugu');
EXPR1 |
-----
<name class="wang">xugu</name>|
```

## 8.6 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

### 返回值

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 |  
-----
```

## 8.7 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(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

### 返回值

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>|
```

## 8.8 XMLSEQUENCE

### 功能描述

将非标准的 XML 格式数据转换成数据库表结构。

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

### 返回值

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。

## 8.9 XMLTABLE

### 功能描述

将 XML 格式的数据转换为数据库表结构。

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)
          | XMLTABLE(xmltable_exprs) alias_clause
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 TYPE_NAME PATH XQuery_child
```

## 参数说明

- 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

## 返回值

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 |
```



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。

## 8.10 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

### 返回值

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('
<PRecord>
<PDName>Daniel Morgan</PDName>
<PDOB>12/1/1951</PDOB>
<PEmail>damorgan@u.washington.edu</PEmail>
</PRecord>')
);
SQL> INSERT INTO person_data
(person_id, person_data)
VALUES
(2, XMLTYPE('
<PRecord>
<PDName>Jack Cline</PDName>
<PDOB>5/17/1949</PDOB>
<PEmail>damorgan@u.washington.edu</PEmail>
</PRecord>')
);
SQL> INSERT INTO person_data
(person_id, person_data)
VALUES
(3, XMLTYPE('
<PRecord>
<PDName>Caleb Small</PDName>
<PDOB>1/1/1960</PDOB>
<PEmail>damorgan@u.washington.edu</PEmail>
</PRecord>')
);

SQL> SELECT person_id, XMLQuery(
'for $i in /PRecord
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 |
```

## 8.11 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

## 返回值

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|
```

# 9 聚合函数

## 9.1 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
SUM	求和
VAR_POP	计算总体方差
VAR_SAMP	计算采样方差
VARIANCE	计算方差
XMLAGG	将多行数据转换为一个 XML 类型的值

## 9.2 AVG

### 功能描述

计算指定字段的平均值。

### 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。

### 返回值

数值类型 DOUBLE 或 NUMERIC。

### 示例

```
-- 用作聚合函数
SELECT AVG(salary) "AVERAGE" FROM employees;

AVERAGE |
-----
6548.2307692 |

-- 用作分析函数
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| 11833.3333333|
...
```

## 9.3 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。

### 返回值

VARBIT 类型。

#### 说明

若某值为 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));
Execute successful.

SQL> INSERT INTO tab_bit_and VALUES (b'1010101') (b'1011100') (b'
0011100');
Total 3 records effected.

SQL> SELECT BIT_AND(c1) FROM tab_bit_and;

EXPR1 |
-----
b'0010100' |
```

## 9.4 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。

### 返回值

VARBIT 类型。

#### 📖 说明

若某值为 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));  
Execute successful.  
  
SQL> INSERT INTO tab_bit_or VALUES (b'1010101') (b'1011100') (b'  
0011100');  
  
Total 3 records effected.  
  
SQL> SELECT BIT_OR(c1) FROM tab_bit_or;  
  
EXPR1 |  
-----  
b'1011101' |
```

## 9.5 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。

## 返回值

VARBIT 类型。

### 📖 说明

若某值为 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));
Execute successful.
SQL> INSERT INTO tab_bit_xor VALUES (b'1010101') (b'1011100') (b'
0011100');
Total 3 records effected.
SQL> SELECT BIT_XOR(c1) FROM tab_bit_xor;
EXPR1 |
-----
b'0010101' |
```

## 9.6 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。

## 返回值

整数类型 BIGINT。

## 示例

```
-- 用作聚合函数
SELECT COUNT(*) "Total" FROM employees;

Total |
-----
104 |

SELECT COUNT(DISTINCT manager_id) "Managers" FROM employees;

Managers |
-----
18 |

-- 用作分析函数
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 |
-----
Olson| 2100| 3 |
Markle| 2200| 2 |
Philtanker| 2200| 2 |
Landry| 2400| 8 |
Gee| 2400| 8 |
...
```

# 9.7 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。

## 返回值

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

## 9.8 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。

### 返回值

VARCHAR。

### 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|
    
```

## 9.9 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

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。

### 返回值

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_arrayagg(b) from t1 group by a;
EXPR1      |
-----+
["1", "2"]|
["a", "b"]|
    
```

## 9.10 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。

### 返回值

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"} |
```

## 9.11 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。

### 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。

## 返回值

STRING 类型。

## 示例

```
-- 将员工姓名按逗号分隔合并为一个字符串
SELECT LISTAGG(last_name, ',') WITHIN GROUP (ORDER BY last_name) AS
employees
FROM employees;

-- 按部门ID将员工姓名按逗号分隔合并为一个字符串, 每个部门一个字符串
SELECT department_id, LISTAGG(last_name, ',') WITHIN GROUP (ORDER
BY last_name) AS employees
FROM employees
GROUP BY department_id;

-- 将员工姓名按逗号分隔合并为一个字符串, 并在整个结果集上进行分区
SELECT LISTAGG(last_name, ',') WITHIN GROUP (ORDER BY last_name)
OVER () AS all_employees
FROM employees;
```

**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。

## 9.12 MAX

### 功能描述

获取指定字段的最大值。

### 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。

### 返回值

数值类型 **DOUBLE** 或 **NUMERIC**。

### 示例

```
-- 用作聚合函数  
SELECT MAX(salary) "MAXIMUM" FROM employees;  
  
MAXIMUM |  
-----  
24000 |  
  
-- 用作分析函数  
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| Kochhar| 17000| 17000|  
100| De Haan| 17000| 17000|  
100| Raphaely| 11000| 17000|  
100| Weiss| 8000| 17000|  
100| Fripp| 8200| 17000|  
...
```

## 9.13 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。

### 返回值

数值类型 NUMERIC。

### 示例

```
-- 作为聚合函数  
SELECT MEDIAN(salary) FROM employees WHERE employee_id > 200;  
  
EXPR1 |  
-----  
9150|  
  
-- 作为分析函数  
SELECT department_id, salary,  
MEDIAN(salary) OVER (PARTITION BY department_id) "DEPT MEDIAN  
SALARY"  
FROM employees  
ORDER BY department_id, salary;  
  
DEPARTMENT_ID | SALARY | DEPT MEDIAN SALARY |
```

```
-----  
10| 4400| 4400|  
20| 6000| 9500|  
20| 13000| 9500|  
30| 2500| 2850|  
30| 2600| 2850|
```

## 9.14 MIN

### 功能描述

获取指定字段的最小值。

### 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。

### 返回值

- 数值类型 `DOUBLE` 或 `NUMERIC`

### 示例

```
-- 用作聚合函数  
SELECT MIN(salary) "MINIMUM" FROM employees;  
  
MINIMUM |  
-----  
2100|  
  
-- 用作分析函数  
SELECT manager_id,last_name,salary,  
MIN(salary) OVER (PARTITION BY manager_id) AS mger_min  
FROM employees;  
  
MANAGER_ID | LAST_NAME | SALARY | MGR_MIN |  
-----  
100| Kochhar| 17000| 5800|  
100| De Haan| 17000| 5800|  
100| Raphaely| 11000| 5800|  
100| Weiss| 8000| 5800|  
100| Fripp| 8200| 5800|  
...
```

## 9.15 STDDEV

### 功能描述

计算数据的样本标准差。结果是 VARIANCE 函数计算的方差的平方根。

### 语法格式

```
STDDEV([ 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 ORDER BY 等。

### 返回值

数值类型 NUMERIC。

### 示例

```
-- 作为聚合函数  
SELECT STDDEV(salary) "DEVIATION" FROM employees;  
  
DEVIATION |  
-----  
3191.089337|  
  
-- 作为分析函数  
SELECT first_name, salary,  
STDDEV(salary) OVER (ORDER BY hire_date) "STDDEV"  
FROM employees  
WHERE department_id = 30  
ORDER BY first_name, salary, "STDDEV";  
  
FIRST_NAME | SALARY | STDDEV |  
-----  
Alexander| 3100| 5586.14357137372544276667|  
Den| 11000| 0|  
Guy| 2600| 3649.24649756631266126818|  
Karen| 2500| 3362.58828880373042912264|  
Shelli| 2900| 4035.26124722220845374227|  
Sigal| 2800| 4650.08960487143014508823|
```

## 9.16 STDDEV\_POP

### 功能描述

计算总体标准差并返回总体方差的平方根。

## 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。

## 返回值

数值类型 NUMERIC。

## 示例

```
-- 作为聚合函数  
SELECT STDDEV_POP(salary) "POP" FROM employees;  
  
POP |  
-----  
3912.694087|  
  
-- 作为分析函数  
SELECT first_name, salary,  
STDDEV_POP(salary) OVER (ORDER BY hire_date) "STDDEV_POP"  
FROM employees  
WHERE department_id= 30  
ORDER BY first_name, salary, "STDDEV_POP";  
  
FIRST_NAME | SALARY | STDDEV_POP |  
-----  
Alexander| 3100| 3950|  
Den| 11000| 0|  
Guy| 2600| 3263.98529408451838998893|  
Karen| 2500| 3089.60909563416559798111|  
Shelli| 2900| 3494.63875100131057881124|  
Sigal| 2800| 3796.78226338508323415112|
```

# 9.17 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。

## 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。

## 返回值

数值类型 `NUMERIC`。

## 示例

```
-- 作为聚合函数
SELECT STDDEV_SAMP(salary) "SAMP" FROM employees;

SAMP |
-----
3931.641869|

-- 作为分析函数
SELECT first_name, salary,
STDDEV_SAMP(salary) OVER (ORDER BY hire_date) "STDDEV_SAMP"
FROM employees
WHERE department_id = 30
ORDER BY first_name, salary, "STDDEV_SAMP";

FIRST_NAME | SALARY | STDDEV_SAMP |
-----
Alexander| 3100| 5586.14357137372544276667|
Den| 11000| <NULL>|
Guy| 2600| 3649.24649756631266126818|
Karen| 2500| 3362.58828880373042912264|
Shelli| 2900| 4035.26124722220845374227|
Sigal| 2800| 4650.08960487143014508823|
```

# 9.18 SUM

## 功能描述

计算指定字段的累计值。

## 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。

## 返回值

数值类型 DOUBLE 或 NUMERIC。

## 示例

```
-- 用作聚合函数
SELECT SUM(salary) "TOTAL" FROM employees;

TOTAL |
-----
681016|

-- 用作分析函数
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| Mourgos| 5800| 5800|
100| Vollman| 6500| 12300|
100| Kaufling| 7900| 20200|
100| Weiss| 8000| 28200|
100| Fripp| 8200| 36400|
...
```

# 9.19 VAR\_POP

## 功能描述

计算数据的总体方差。

## 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。

## 返回值

数值类型 NUMERIC。

## 示例







# 10 窗口函数

## 10.1 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

## 10.2 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。

### 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。

### 返回值

INTEGER 或 BIGINT 类型。

### 示例

- 示例 1

```
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

```
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 |
```

# 11 内部功能性函数

## 11.1 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

## 11.2 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

### 返回值

BOOLEAN

### 示例

```
SELECT FLUSH_COMMAND_LOG();  
SELECT * FROM SYS_COMMAND_LOG;
```

## 11.3 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

### 返回值

BOOLEAN

### 示例

```
SELECT FLUSH_ERROR_LOG();  
SELECT * FROM SYS_ERROR_LOG;
```

## 11.4 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

## 返回值

BOOLEAN

## 示例

```
SELECT FLUSH_MODIFY_LOG();  
EXEC dbms_replication.poll_modify_data('sup', 0, null  
    , 1024*1024, false, -1, 0);
```

# 11.5 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。

## 返回值

- 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
```

# 12 特殊表达式

## 12.1 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基准
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

## 12.2 CURRENT\_DATABASE

### 功能描述

取当前数据库名。

### 语法格式

```
CURRENT_DATABASE
```

### 参数说明

无参数且不使用 ()。

### 返回值

VARCHAR 字符串。

### 示例

```
SQL> SELECT CURRENT_DATABASE FROM dual;
```

```
EXPR1 |
```

```
-----  
SYSTEM|
```

## 12.3 CURRENT\_DATE

### 功能描述

取当前日期。

### 语法格式

```
CURRENT_DATE
```

### 参数说明

无参数且不使用 ()。

### 返回值

日期类型 DATE。

### 示例

```
SQL> SELECT CURRENT_DATE FROM dual;
```

```
EXPR1 |
```

```
-----  
2022-05-14 AD |
```

## 12.4 CURRENT\_DB

### 功能描述

取当前数据库名。

### 语法格式

```
CURRENT_DB
```

### 参数说明

无参数且不使用 ()。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT CURRENT_DB FROM dual;

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

## 12.5 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

无参数且不使用 ()。

### 返回值

整数类型 INTEGER。

### 示例

```
SQL> SELECT CURRENT_DB_ID FROM dual;

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

## 12.6 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

无参数且不使用 ()。

### 返回值

INTEGER。

### 示例

```
SQL> SELECT CURRENT_IP FROM dual;

EXPR1 |
-----|
192.168.2.116|
```

## 12.7 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

无参数且不使用 ()。

### 返回值

INTEGER 类型。

### 示例

```
SQL> SELECT CURRENT_NODEID FROM dual;

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

## 12.8 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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT current_schema FROM dual;

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

## 12.9 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

无参数且不使用 ()。

### 返回值

时间类型 TIME。

### 示例

```
SQL> SELECT CURRENT_TIME FROM dual;

EXPR1 |
-----
15:42:23.820 |
```

## 12.10 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

无参数且不使用 ()。

### 返回值

日期时间类型 TIMESTAMP。

### 示例

```
SQL> SELECT CURRENT_TIMESTAMP FROM dual;

EXPR1 |
-----
2022-05-14 15:42:01.601 AD |
```

## 12.11 CURRENT\_USER

### 功能描述

取当前用户名。

### 语法格式

```
CURRENT_USER
```

### 参数说明

无参数且不使用 ()。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT CURRENT_USER FROM dual;

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

## 12.12 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。

### 返回值

INTEGER 数值类型。

### 示例

```
SQL> SELECT CURRENT_USERID() FROM dual;

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

## 12.13 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。

### 返回值

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 |
```

## 12.14 DATABASE

### 功能描述

同 CURRENT\_DB，返回当前数据库。

### 语法格式

```
DATABASE ()
```

### 参数说明

无参数。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT DATABASE ();  
EXPR1 |  
-----  
SYSTEM|
```

## 12.15 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。

### 返回值

BOOLEAN 类型。

### 示例

```
SQL> SELECT DIR_EXISTS ('/DATA');  
  
EXPR1 |  
-----  
T |
```

## 12.16 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。

### 返回值

BOOLEAN 类型。

### 示例

```
SQL> SELECT FILE_EXISTS (123) FROM dual;  
  
EXPR1 |  
-----  
F |
```

## 12.17 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。

### 返回值

BIGINT 类型。

### 示例

```
SQL> SELECT FILELEN('/DATA/DATA1.DBF') FROM dual;

EXPR1 |
-----
402653184 |
```

## 12.18 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

无参数且不使用 ()。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT GEN_RANDOM_UUID FROM dual;

EXPR1 |
-----
660FEDAD-0AC3-4EDC-B511-773C21108B24 |
```

## 12.19 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。

### 返回值

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|
```

## 12.20 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。

### 返回值

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 |
```

## 12.21 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。

### 返回值

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|
```

## 12.22 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: 同类型表达式。
- expr2: 同类型表达式。
- exprn: 同类型表达式。

### 返回值

同数据类型。

### 示例

```
SQL> SELECT GREATEST(2, 5, 3) col01, 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;  
  
COL01(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|
```

## 12.23 IF

### 功能描述

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。

### 返回值

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 |
```

## 12.24 INET\_ATON

### 功能描述

IPV4 转整数。

### 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]

### 返回值

BIGINT 类型。

#### 📖 说明

返回值范围: [0, 4294967295]

### 示例

```
SQL> SELECT INET_ATON('192.168.2.218');  
EXPR1 |  
-----  
3232236250 |
```

## 12.25 INET\_NTOA

### 功能描述

整数转 IPV4。

### 语法格式

```
INET_NTOA(expr)
```

### 参数说明

expr: 整数。

#### 📖 说明

参数取值范围: [0, 4294967295]

### 返回值

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|
```

## 12.26 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。

### 返回值

INTEGER 数值类型。

### 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 当按照 path 路径无法找到值时触发。

## 12.27 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: 同类型表达式。
- expr2: 同类型表达式。
- exprn: 同类型表达式。

### 返回值

同数据类型。

### 示例

```
SQL> SELECT LEAST(2, 5, 3) col01, 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;
```

```
COL01(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> |
```

## 12.28 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

无参数且不使用 ()。

### 返回值

INTEGER 类型。

### 示例

```
SQL> SELECT MAX_NODE_NUM FROM dual;
```

```
EXPR1(INTEGER) |  
-----  
32 |
```

## 12.29 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。

## 12.30 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

无参数且不使用 ()。

### 返回值

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-EDB759DFEDEC |
```

## 12.31 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。

## 返回值

INTEGER 类型。

## 示例

```
SQL> SELECT num_nonnulls(1,NULL,'abc',NULL,'') FROM dual;  
EXPR1 |  
-----  
3 |
```

# 12.32 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。

## 返回值

INTEGER 类型。

## 示例

```
SQL> SELECT num_nulls(1,NULL,'abc',NULL,'') FROM dual;  
EXPR1 |  
-----  
2 |
```

# 12.33 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。

## 返回值

BOOLEAN 类型。

## 示例

```
SQL> SELECT RENAME_FILE('/BACKUP/BACK_DB_FILE.exp', '/BACKUP/  
BACK_DB_FILE1.exp');  
  
EXPR1 |  
-----  
T |
```

# 12.34 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。

## 返回值

INTEGER 类型。

## 示例

```
SQL> SELECT SLEEP(10000) FROM dual;  
  
EXPR1 |  
-----  
10000 |
```

# 12.35 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。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT SQLERRM(19138);  
  
EXPR1 |  
-----  
ERR-19138: 时间值常数错误 |
```

## 12.36 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。

### 返回值

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|
```

## 12.37 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

无参数且不使用 ()。

### 返回值

GUID。

### 示例

```
SQL> SELECT SYS_GUID FROM dual;  
  
EXPR1 |  
-----  
<2EAF7CB7CC1E441B920F3EAF64493DB> |
```

## 12.38 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

无参数且不使用 ()。

### 返回值

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 |
```

## 12.39 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

无参数且不使用 ()。

### 返回值

VARCHAR 类型字符串。

### 示例

```
SQL> SELECT SYS_UUID FROM dual;

EXPR1 |
-----
7E63A51F-A16D-FFC1-B618-D41EF2DA6618 |
```

## 12.40 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

无参数且不使用 ()。

### 返回值

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|
```

## 12.41 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。

### 返回值

INTEGER 数值类型。

### 示例

```
SQL> SELECT UID() FROM dual;
```

```
EXPR1 |
```

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

## 12.42 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。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT USER;
```

```
EXPR1 |
```

```
-----  
SYSDBA |
```

## 12.43 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

```
UUID
```

### 参数说明

无参数且不使用 ()。

### 返回值

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 |
```

## 12.44 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。
  - '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。

### 返回值

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 |
```

## 12.45 VERSION

### 功能描述

同 SHOW VERSION，返回当前数据库版本。

### 语法格式

```
VERSION()
```

### 参数说明

无参数。

### 返回值

CHAR 类型。

### 示例

```
SQL> SELECT VERSION();
```

```
EXPR1 |
```

```
-----  
XuGu SQL Server 12.0.0|
```

# 13 特殊操作符

## 13.1 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

## 13.2 +

### 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
```

## 13.3 -

### 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
```

## 13.4 ||

### 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|
```

## 13.5 ->

### 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" |
```

## 13.6 ->>

### 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 |
```

## 13.7 &（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。

## 13.8 | (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。

## 13.9 ^（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。

## 13.10 <<（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。

## 13.11 >>（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

当移位数为负数时，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。

## 13.12 ~（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。

# 14 几何函数

## 14.1 概述

函数形式	功能
AREA(BOX)	计算 BOX 对象面积
AREA(CIRCLE)	计算 CIRCLE 对象面积
AREA(PATH)	计算 PATH 对象面积
CENTER(BOX)	返回 BOX 的中心点
CENTER(CIRCLE)	返回 CIRCLE 的中心点
DIAGONAL(BOX)	提取 BOX 的对角线作为线段返回
DIAMETER(CIRCLE)	返回 CIRCLE 的直径
HEIGHT(BOX)	返回边框的垂直尺寸
ISCLOSED(PATH)	判断路径是否封闭
ISOPEN(PATH)	判断路径是否开放
LENGTH(LSEG)	计算线段总长度
LENGTH(PATH)	计算路径总长度
NPOINTS(PATH)	返回 PATH 点的数量
NPOINTS(POLYGON)	返回多边形点的数量
PCLOSE(PATH)	将路径转换为封闭形式
POPEN(PATH)	将路径转换为开放形式
接下一页	

函数形式	功能
RADIUS(CIRCLE)	计算 CIRCLE 的半径
SLOPE(POINT, POINT)	计算通过两点所画直线的斜率
WIDTH(BOX)	返回边框的水平大小

## 14.2 AREA(BOX)

### 功能描述

计算 BOX 对象面积。

### 语法格式

```
AREA (expr)
```

### 参数说明

expr: 目标 BOX 对象，为 BOX 类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT AREA(BOX(' (2,2), (0,0) '));
EXPR1 |
-----
4.000000e+00 |
Total 1 records.
```

## 14.3 AREA(CIRCLE)

### 功能描述

计算 CIRCLE 对象面积。

### 语法格式

```
AREA (expr)
```

### 参数说明

expr: 目标 CIRCLE 对象，为 CIRCLE 类型。

## 返回值

DOUBLE 类型。

## 示例

```
SQL> SELECT AREA(CIRCLE('((5,0),1)'));  
  
EXPR1 |  
-----  
3.141593e+00 |  
  
Total 1 records.
```

# 14.4 AREA(PATH)

## 功能描述

计算 PATH 对象面积。

### 📖 说明

- 输入的路径必须封闭，否则返回 NULL。
- 输入的路径是自交叉时，结果可能是无意义。

## 语法格式

```
AREA(expr)
```

## 参数说明

expr: 目标 PATH 对象，为 PATH 类型。

## 返回值

DOUBLE 类型。

## 示例

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.
```

## 14.5 CENTER(BOX)

### 功能描述

返回 BOX 的中心点。

### 语法格式

```
CENTER(expr)
```

### 参数说明

expr: 目标 BOX 对象，为 BOX 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT CENTER(BOX('(1,2),(0,0)')) ;  
EXPR1 |  
-----+  
(0.5,1.0)|
```

## 14.6 CENTER(CIRCLE)

### 功能描述

返回 CIRCLE 的中心点。

### 语法格式

```
CENTER(expr)
```

### 参数说明

expr: 目标 CIRCLE 对象，为 CIRCLE 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT CENTER(CIRCLE(' (5,2),1')) ;
EXPR1      |
-----+
(5.0,2.0) |
```

## 14.7 DIAGONAL(BOX)

### 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: 目标 BOX 对象，为 BOX 类型。

### 返回值

LSEG 类型。

### 示例

```
SELECT DIAGONAL(BOX(' (1,2), (0,0)')) ;
EXPR1      |
-----+
[(1.0,2.0), (0.0,0.0)] |
```

## 14.8 DIAMETER(CIRCLE)

### 功能描述

返回 CIRCLE 的直径。

### 语法格式

```
DIAMETER (expr)
```

### 参数说明

expr: 目标 CIRCLE 对象，为 CIRCLE 类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT DIAMETER(CIRCLE( '(0,0),2)' ) ;  
  
EXPR1 |  
-----  
4.000000e+00 |  
Total 1 records.
```

## 14.9 HEIGHT(BOX)

### 功能描述

返回边框的垂直尺寸。

### 语法格式

```
HEIGHT(expr)
```

### 参数说明

expr: 目标 BOX 对象, 为 BOX 类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT HEIGHT(BOX( '(1,2), (0,0)' ) );  
  
EXPR1 |  
-----  
2.000000e+00 |  
Total 1 records.
```

## 14.10 ISCLOSED(PATH)

### 功能描述

判断路径是否封闭。

- 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: 目标 PATH 对象, 为 PATH 类型。

### 返回值

BOOLEAN 类型。

### 示例

```
SQL> SELECT ISCLOSED(PATH('((0,0),(1,1),(2,0))'));  
  
EXPR1 |  
-----  
T |  
  
Total 1 records.
```

## 14.11 ISOPEN(PATH)

### 功能描述

判断路径是否开放。

- 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: 目标 PATH 对象, 为 PATH 类型。

### 返回值

BOOLEAN 类型。

### 示例

```
SQL> SELECT ISOPEN(PATH('[ (0,0), (1,1), (2,0) ]'));  
  
EXPR1 |  
-----  
T |  
  
Total 1 records.
```

## 14.12 LENGTH(LSEG)

### 功能描述

计算线段总长度。

### 语法格式

```
LENGTH(expr)
```

### 参数说明

expr: 目标 LSEG 对象, 为 LSEG 类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT LENGTH(LSEG('((0,0),(1,1))'));  
  
EXPR1 |  
-----  
1.414214e+00 |  
  
Total 1 records.
```

## 14.13 LENGTH(PATH)

### 功能描述

计算路径总长度。

### 语法格式

```
LENGTH(expr)
```

### 参数说明

expr: 目标 PATH 对象, 为 PATH 类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT LENGTH(PATH('[ (0,0), (1,1), (2,0) ]'));  
  
EXPR1 |  
-----  
2.828427e+00 |
```

Total 1 records.

## 14.14 NPOINTS(PATH)

### 功能描述

返回 PATH 点的数量。

### 语法格式

```
NPOINTS(expr)
```

### 参数说明

expr: 目标 PATH 对象, 为 PATH 类型。

### 返回值

INTEGER 类型。

### 示例

```
SQL> SELECT NPOINTS(PATH('[(0,0),(1,1),(2,0)]'));  
  
EXPR1 |  
-----  
3 |  
  
Total 1 records.
```

## 14.15 NPOINTS(POLYGON)

### 功能描述

返回多边形点的数量。

### 语法格式

```
NPOINTS(expr)
```

### 参数说明

expr: 目标 POLYGON 对象, 为 POLYGON 类型。

### 返回值

INTEGER 类型。

### 示例

```
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.
```

## 14.16 PCLOSE(PATH)

### 功能描述

将路径转换为封闭形式。

### 语法格式

```
PCLOSE (expr)
```

### 参数说明

expr: 目标 PATH 对象，为 PATH 类型。

### 返回值

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)) |
```

## 14.17 POPEN(PATH)

### 功能描述

将路径转换为开放形式。

### 语法格式

```
POPEN (expr)
```

### 参数说明

expr: 目标 PATH 对象，为 PATH 类型。

### 返回值

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)]|
```

## 14.18 RADIUS(CIRCLE)

### 功能描述

计算 CIRCLE 的半径。

### 语法格式

```
RADIUS(expr)
```

### 参数说明

expr: 目标 CIRCLE 对象，为 CIRCLE 类型。

### 返回值

DOUBLE 类型。

### 示例

```
SQL> SELECT RADIUS(CIRCLE('(5,2),1'));
EXPR1 |
-----+-----
1.000000e+00 |
Total 1 records.
```

## 14.19 SLOPE(POINT, POINT)

### 功能描述

计算通过两点所画直线的斜率。

### 语法格式

```
SLOPE(expr1, expr2)
```

### 参数说明

- expr1: 第一个目标 POINT 对象，为 POINT 类型。
- expr2: 第二个目标 POINT 对象，为 POINT 类型。

## 返回值

DOUBLE 类型。

## 示例

```
SQL> SELECT SLOPE (POINT ('(0,0)'), POINT ('(2,1)'));  
  
EXPR1 |  
-----  
5.000000e-01 |  
  
Total 1 records.
```

# 14.20 WIDTH(BOX)

## 功能描述

返回边框的水平大小。

## 语法格式

```
WIDTH (expr)
```

## 参数说明

expr: 目标 BOX 对象, 为 BOX 类型。

## 返回值

DOUBLE 类型。

## 示例

```
SQL> SELECT WIDTH (BOX ('(1,2)', '(0,0)'));  
  
EXPR1 |  
-----  
1.000000e+00 |  
  
Total 1 records.
```

# 15 几何类型转换函数

## 15.1 概述

函数形式	功能
BOUND_BOX(BOX, BOX)	计算两个方框的边界框
BOX(CIRCLE)	将圆形转换为矩形，返回 CIRCLE 内切正方形的 BOX
BOX(CHAR)	将字符串转为 BOX
BOX(POINT)	将点转换为空框
BOX(POLYGON)	计算多边形的边界框
BOX(POINT, POINT)	将任意两个点转换为框
CIRCLE(BOX)	计算最小的圆形包围框，返回边框的外接圆
CIRCLE(CHAR)	将字符串转换为圆
CIRCLE(CIRCLE)	将圆转换为圆
CIRCLE(POLYGON)	将多边形转换为圆
CIRCLE(POINT, DOUBLE)	从圆心和半径构造圆
LINE(CHAR)	将字符串转换一条直线
LINE(POINT, POINT)	两点转换一条直线
LSEG(BOX)	提取方框的对角线作为线段
LSEG(CHAR)	将字符串转换为线段
LSEG(POINT, POINT)	使用两个端点构造线段
接下一页	

函数形式	功能
PATH(CHAR)	将字符串转换为路径
PATH(PATH)	将路径转换为路径
PATH(POLYGON)	将多边形转换为具有相同点列表的闭合路径
POINT(BOX)	计算 BOX 的中心
POINT(CHAR)	将字符串转换为点
POINT(CIRCLE)	计算圆心
POINT(LSEG)	计算线段的中心
POINT(POLYGON)	计算多边形的中心（多边形的点位置的平均值）
POINT(DOUBLE, DOUBLE)	用坐标构造点
POLYGON(BOX)	将 BOX 转换为 4 点多边形
POLYGON(CIRCLE)	将 CIRCLE 转换为 12 点多边形
POLYGON(CHAR)	将字符串转换为多边形
POLYGON(INTEGER, CIRCLE)	将 CIRCLE 转换为 N 点多边形
POLYGON(PATH)	将封闭路径转换为具有相同点列表的多边形

## 15.2 BOUND\_BOX(BOX, 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: 第一个目标 BOX 对象，为 BOX 类型。
- expr2: 第二个目标 BOX 对象，为 BOX 类型。

## 返回值

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)|
```

# 15.3 BOX(CIRCLE)

## 功能描述

将圆形转换为矩形，返回 CIRCLE 内切正方形的 BOX。

## 语法格式

```
BOX(expr)
```

## 参数说明

expr: 目标 CIRCLE 对象，为 CIRCLE 类型。

## 返回值

BOX 类型。

## 示例

```
SELECT BOX (CIRCLE ('<(0,0),1>'));
EXPR1
-----+
(0.7071067811865475,0.7071067811865475)
,(-0.7071067811865475,-0.7071067811865475)|
```

# 15.4 BOX(CHAR)

## 功能描述

将字符串转为 BOX。

## 语法格式

```
BOX(expr)
```

## 参数说明

expr: 目标 CHAR 字符串，为 CHAR 类型。

## 返回值

BOX 类型。

## 示例

```
SELECT BOX ('(0,0), (1,1)');
EXPR1          |
-----+
(1.0,1.0), (0.0,0.0) |
```

# 15.5 BOX(POINT)

## 功能描述

将点转换为空框。

## 语法格式

```
BOX(expr)
```

## 参数说明

expr: 目标 POINT 对象, 为 POINT 类型。

## 返回值

BOX 类型。

## 示例

```
SELECT BOX (POINT ('(5,6)'));
EXPR1          |
-----+
(5.0,6.0), (5.0,6.0) |
```

# 15.6 BOX(POLYGON)

## 功能描述

计算多边形的边界框。

## 语法格式

```
BOX(expr)
```

## 参数说明

expr: 目标 POLYGON 对象, 为 POLYGON 类型。

## 返回值

BOX 类型。

### 示例

```
SELECT BOX (POLYGON (' ((0,0), (1,1), (2,0)) '));
EXPR1          |
-----+
(2.0,1.0), (0.0,0.0) |
```

## 15.7 BOX(POINT, POINT)

### 功能描述

将任意两个点转换为框。

### 语法格式

```
BOX(expr1, expr2)
```

### 参数说明

- expr1: 第一个目标 POINT 对象，为 POINT 类型。
- expr2: 第二个目标 POINT 对象，为 POINT 类型。

### 返回值

BOX 类型。

### 示例

```
SELECT BOX (POINT (' (0,1) '), POINT (' (1,0) '));
EXPR1          |
-----+
(1.0,1.0), (0.0,0.0) |
```

## 15.8 CIRCLE(BOX)

### 功能描述

计算最小的圆形包围框，返回边框的外接圆。

### 语法格式

```
CIRCLE(expr)
```

### 参数说明

expr: 目标 BOX 对象，为 BOX 类型。

### 返回值

CIRCLE 类型。

### 示例

```
SELECT CIRCLE (BOX (' (1, 1) , (-1, -1) ')) ;
EXPR1      |
-----+
<(0.0,0.0) , 1.4142135623730951>|
```

## 15.9 CIRCLE(CHAR)

### 功能描述

将字符串转换为圆。

### 语法格式

```
CIRCLE (expr)
```

### 参数说明

expr: 目标字符串, 为 CHAR 类型。

### 返回值

CIRCLE 类型。

### 示例

```
SELECT CIRCLE ('<(3,4) , 2.0>');
EXPR1      |
-----+
<(3.0,4.0) , 2.0>|
```

## 15.10 CIRCLE(CIRCLE)

### 功能描述

将圆转换为圆。

### 语法格式

```
CIRCLE (expr1)
```

### 参数说明

expr1: 目标圆, 为 CIRCLE 类型。

### 返回值

CIRCLE 类型。

### 示例

```
SELECT CIRCLE ('<(3,4), 2.0>'::CIRCLE);
EXPR1      |
-----+
<(3.0,4.0),2.0>|
```

## 15.11 CIRCLE(POLYGON)

### 功能描述

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。

### 语法格式

```
CIRCLE(expr1)
```

### 参数说明

expr1: 目标 POLYGON 对象，为 POLYGON 类型。

### 返回值

CIRCLE 类型。

### 示例

```
SELECT CIRCLE (POLYGON ('((0,0),(1,3),(2,0))'));
EXPR1      |
-----+
<(1.0,1.0),1.6094757082487299>|
```

## 15.12 CIRCLE(POINT, DOUBLE)

### 功能描述

从圆心和半径构造圆。

### 语法格式

```
CIRCLE(expr1, expr2)
```

### 参数说明

- expr1: 目标圆心，为 POINT 类型。
- expr2: 目标半径，为 DOUBLE 类型。

### 返回值

CIRCLE 类型。

### 示例

```
SELECT CIRCLE (POINT ('(3,4)'), 2.0);
EXPR1      |
-----+
<(3.0,4.0),2.0>|
```

## 15.13 LINE(CHAR)

### 功能描述

将字符串转换一条直线。

### 语法格式

```
LINE (expr)
```

### 参数说明

expr: 目标字符串, 为 CHAR 类型。

### 返回值

LINE 类型。

### 示例

```
SELECT LINE ('(-1,0),(1,0)');
EXPR1      |
-----+
{0.0,-1.0,0.0}|
```

## 15.14 LINE(POINT, POINT)

### 功能描述

两点转换一条直线。

### 语法格式

```
LINE (expr1, expr2)
```

### 参数说明

- expr1: 第一个目标 POINT 对象, 为 POINT 类型。
- expr2: 第二个目标 POINT 对象, 为 POINT 类型。

### 返回值

LINE 类型。

### 示例

```
SELECT LINE (POINT (' (-1,0) '), POINT (' (1,0) '));
EXPR1      |
-----+
{0.0, -1.0, 0.0} |
```

## 15.15 LSEG(BOX)

### 功能描述

提取方框的对角线作为线段。

### 语法格式

```
LSEG (expr)
```

### 参数说明

expr: 目标 BOX 对象，为 BOX 类型。

### 返回值

LSEG 类型。

### 示例

```
SELECT LSEG (BOX (' (1,0) , (-1,0) ')) ;
EXPR1      |
-----+
[(1.0, 0.0) , (-1.0, 0.0)] |
```

## 15.16 LSEG(CHAR)

### 功能描述

将字符串转换为线段。

### 语法格式

```
LSEG (expr)
```

### 参数说明

expr: 目标字符串，为 CHAR 类型。

### 返回值

LSEG 类型。

### 示例

```
SELECT LSEG(' (1,0) , (-1,0) ' ) ;
EXPR1          |
-----+
[ (1.0,0.0) , (-1.0,0.0) ] |
```

## 15.17 LSEG(POINT, POINT)

### 功能描述

使用两个端点构造线段。

### 语法格式

```
LSEG(expr1, expr2)
```

### 参数说明

- expr1: 第一个目标 POINT 对象, 为 POINT 类型。
- expr2: 第二个目标 POINT 对象, 为 POINT 类型。

### 返回值

LSEG 类型。

### 示例

```
SELECT LSEG(POINT(' (-1,0) '), POINT(' (1,0) '));
EXPR1          |
-----+
[ (-1.0,0.0) , (1.0,0.0) ] |
```

## 15.18 PATH(CHAR)

### 功能描述

将字符串转换为路径。

### 语法格式

```
PATH(expr)
```

### 参数说明

expr: 目标字符串, 为 CHAR 类型。

### 返回值

PATH 类型。

### 示例

```
SELECT PATH('((0,0),(1,1),(2,0))');
EXPR1      |
-----+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

## 15.19 PATH(PATH)

### 功能描述

将路径转换为路径。

### 语法格式

```
PATH(expr)
```

### 参数说明

expr: 目标 PATH 对象, 为 PATH 类型。

### 返回值

PATH 类型。

### 示例

```
SELECT PATH('((0,0),(1,1),(2,0))'::PATH);
EXPR1      |
-----+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

## 15.20 PATH(POLYGON)

### 功能描述

将多边形转换为具有相同点列表的闭合路径。

### 语法格式

```
PATH(expr)
```

### 参数说明

expr: 目标 POLYGON 对象, 为 POLYGON 类型。

### 返回值

PATH 类型。

### 示例

```
SELECT PATH (POLYGON (' ((0,0), (1,1), (2,0)) '));
EXPR1      |
-----+
((0.0,0.0), (1.0,1.0), (2.0,0.0)) |
```

## 15.21 POINT(BOX)

### 功能描述

计算 BOX 的中心。

### 语法格式

```
POINT (expr)
```

### 参数说明

expr: 目标 BOX 对象, 为 BOX 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT POINT (BOX (' (1,0), (-1,0) '));
EXPR1      |
-----+
(0.0,0.0) |
```

## 15.22 POINT(CHAR)

### 功能描述

将字符串转换为点。

### 语法格式

```
POINT (expr)
```

### 参数说明

expr: 目标字符串, 为 CHAR 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT POINT ('23.4, -44.5');
EXPR1      |
-----+
(23.4, -44.5) |
```

## 15.23 POINT(CIRCLE)

### 功能描述

计算圆心。

### 语法格式

```
POINT (expr)
```

### 参数说明

expr: 目标 CIRCLE 对象，为 CIRCLE 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT POINT (CIRCLE ('<(0,0), 2>'));
EXPR1      |
-----+
(0.0, 0.0) |
```

## 15.24 POINT(LSEG)

### 功能描述

计算线段的中心。

### 语法格式

```
POINT (expr)
```

### 参数说明

expr: 目标 LSEG 对象，为 LSEG 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT POINT(LSEG(' [(-1,0), (1,0)] '));
EXPR1      |
-----+
(0.0,0.0) |
```

## 15.25 POINT(POLYGON)

### 功能描述

计算多边形的中心（多边形的点位置的平均值）。

### 语法格式

```
POINT(expr)
```

### 参数说明

expr: 目标 POLYGON 对象，为 POLYGON 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT POINT(POLYGON(' ((0,0), (1,1), (2,0)) '));
EXPR1      |
-----+
(1.0,0.3333333333333333) |
```

## 15.26 POINT(DOUBLE, DOUBLE)

### 功能描述

用坐标构造点。

### 语法格式

```
POINT(expr1,expr2)
```

### 参数说明

- expr1: 目标 X 值，DOUBLE 类型。
- expr2: 目标 Y 值，DOUBLE 类型。

### 返回值

POINT 类型。

### 示例

```
SELECT POINT(23.4, -44.5);
EXPR1      |
-----+
(23.4, -44.5) |
```

## 15.27 POLYGON(BOX)

### 功能描述

将 BOX 转换为 4 点多边形。

### 语法格式

```
POLYGON(expr)
```

### 参数说明

expr: 目标 BOX 对象, 为 BOX 类型。

### 返回值

POLYGON 类型。

### 示例

```
SELECT POLYGON(BOX(' (1,1), (0,0) '));
EXPR1      |
-----+
((0.0,0.0), (0.0,1.0), (1.0,1.0), (1.0,0.0)) |
```

## 15.28 POLYGON(CIRCLE)

### 功能描述

将 CIRCLE 转换为 12 点多边形。

### 语法格式

```
POLYGON(expr)
```

### 参数说明

expr: 目标 CIRCLE 对象, 为 CIRCLE 类型。

### 返回值

POLYGON 类型。

### 示例

```
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))|
```

## 15.29 POLYGON(CHAR)

### 功能描述

将字符串转换为多边形。

### 语法格式

```
POLYGON(expr)
```

### 参数说明

expr: 目标字符串, 为 CHAR 类型。

### 返回值

POLYGON 类型。

### 示例

```
SELECT POLYGON('(0,0),(1,1),(2,0)');
EXPR1
-----+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

## 15.30 POLYGON(INTEGER, CIRCLE)

### 功能描述

将 CIRCLE 转换为 N 点多边形。

### 语法格式

```
POLYGON(expr1, expr2)
```

### 参数说明

- expr1: 点个数, 为 INTEGER 类型。

- expr2: 目标 CIRCLE 对象, 为 CIRCLE 类型。

### 返回值

POLYGON 类型。

### 示例

```
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))|
```

## 15.31 POLYGON(PATH)

### 功能描述

将封闭路径转换为具有相同点列表的多边形。

### 语法格式

```
POLYGON(expr)
```

### 参数说明

expr: 目标 PATH 对象, 为 PATH 类型。

### 返回值

POLYGON 类型。

### 示例

```
SELECT POLYGON(PATH('(0,0),(1,1),(2,0)'));
EXPR1
-----+
((0.0,0.0),(1.0,1.0),(2.0,0.0))|
```

# 16 几何操作符

## 16.1 概述

函数形式	功能
+	<ul style="list-style-type: none"> <li>• 平移，将第二个参数 POINT 的坐标添加到第一个对象的坐标中</li> <li>• 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</li> </ul>
-	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
#	<ul style="list-style-type: none"> <li>• 返回点的数量</li> <li>• 计算交点，如果没有交点则返回 NULL</li> <li>• 计算两个方框的交集，如果没有则为 NULL</li> </ul>
##	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
=	<ul style="list-style-type: none"> <li>判断第一个对象的面积是否等于第二个对象的面积</li> <li>判断第一个对象是否等于第二个对象</li> </ul>

## 16.2 +

### 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> |
```

## 16.3 -

### 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> |
```

## 16.4 \*

### 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>|
```

## 16.5 /

### 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

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	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> |
```

## 16.6 @-@

### 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 |
```

# 16.7 @@

## 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	LSEG	
		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	POLYGON	
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) |
```

## 16.8 #

### 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) |
```

## 16.9 ##

### 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) |
```

## 16.10 <->

### 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.
```

## 16.11 @>

### 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.
```

## 16.12 <@

### 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.
```

## 16.13 &&

### 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.
```

## 16.14 «

### 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.
```

## 16.15 »

### 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.
```

## 16.16 &<

### 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.
    
```

## 16.17 &>

### 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.
```

## 16.18 «|

### 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.
```

## 16.19 |»

### 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.
```

## 16.20 &<|

### 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.
```

## 16.21 |&>

### 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	

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.
```

## 16.22 >^

### 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.
```

## 16.23 >^

### 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.
```

## 16.24 ?#

### 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	
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 |
```

## 16.25 ?-|

### 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.
```

## 16.26 ~=

### 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.
```

## 16.27 <>

### 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.
```

## 16.28 <

### 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.
```

## 16.29 >

### 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.
```

## 16.30 <=

### 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.
```

## 16.31 >=

### 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.
```

## 16.32 =

### 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](http://www.xugudb.com)