



Kingwow 分布式数据库 使用手册 V6.0

北京自然原数科技有限公司

2025 年 7 月 22 日



版权所有 © 2026 北京自然原数科技有限公司。保留一切权利。

非经北京自然原数科技有限公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

、是北京自然原数科技有限公司的商标或者注册商标。

在本文档中以及本文档描述的产品中，出现的其他商标、产品名称、服务名称以及公司名称，由其各自的所有人拥有。

免责声明

本文档信息仅供参考，不构成任何要约或承诺。本文档可能含有预测信息，由于实践中存在很多不确定因素，可能导致实际结果与预测信息有差别。本文档的内容在未经通知的情形下可能会发生改变，敬请留意。

开源声明

本文档引用的开源组件 zstd、snappy、Tongsuo、mariadb-connector-c 和 google-test 均由使用者自行下载。

文档说明

文档名称:	Kingwow 分布式数据库 SQL 使用手册		
创建者:	运维交付部成员	创建日期:	2025-07-22
复核者:	交付运维部	复核日期:	2025-07-22
审批者:	交付运维部	审批日期:	2025-07-22

修改历史:

版本号	修改内容	撰写者	发布日期
1.0	创建	交付运维部	2025-07-22

目录

1. 编写目的	1
2. 读者对象	1
3. 基本的数据库对象	1
4. SQL 基础	1
4.1. 数据类型	1
4.2. 函数	3
4.2.1. 字符串函数	3
4.2.2. 日期函数	19
4.2.3. 算术函数	34
4.2.4. 聚合函数	43
4.2.5. 空值处理函数	45
4.2.6. 条件表达式函数	46
4.2.7. 窗口函数	51
4.2.8. 自定义函数	55
4.2.9. 转换函数	55
4.2.10. JSON 相关函数	57
4.2.11. XML 相关函数	60
4.2.12. 其他函数	60
4.3. 运算符	65
4.4. 转义字符	67
4.5 字符集	68
5. SQL 语法	69
5.1. 数据定义语句	69
5.1.1. CREATE DATABASE	69
5.1.2. DROP DATABASE	70
5.1.3. RENAME DATABASE	70
5.1.4. CREATE TABLE	71
5.1.5. ALTER TABLE	80
5.1.6. DROP TABLE	82
5.1.7. FLASHBACK TABLE	82
5.1.8. CREATE VIEW	83
5.1.9. DROP VIEW	83

5.1.10. CREATE INDEX	83
5.1.11. RENAME INDEX	84
5.1.12. DROP INDEX	86
5.1.13. TRUNCATE TABLE	92
5.1.14. CREATE SEQUENCE	93
5.1.15. ALTER SEQUENCE	95
5.1.16. DROP SEQUENCE	96
5.1.17. ALTER VIEW	96
5.2. 数据操作语句	98
5.2.1. INSERT	98
5.2.2. REPLACE	100
5.2.3. UPDATE	100
5.2.4. DELETE	101
5.2.5. SELECT	101
5.2.6. JOIN	104
5.2.7. UNION	106
5.2.8. INTERSECT	106
5.2.9. EXCEPT	106
5.2.10. SELECT ... FOR UPDATE	107
5.2.11. PREPARE	107
5.2.12. EXECUTE	108
5.2.13. DEALLOCATE	108
5.2.14. ROWNUM	108
5.2.15. MERGE INTO	110
5.3 表分区	112
5.3.1 HASH 分区表	112
5.3.2 LIST 分区表	113
5.3.3 RANGE 分区表	113
5.3.4 分区管理语句	114
5.3.5 广播表	115
5.3.6 临时表	116
5.3.7 二级分区表	116

6. 数据库商密安全	118
6.1. 网络传输加密	118
6.2. 存储透明加密	121
7. 事务处理	122
7.1. START TRANSACTION	122
7.2. COMMIT	122
7.3. ROLLBACK	122
8. 触发器	123
8.1. 创建触发器	123
8.2. 查看触发器	124
8.3. 删除触发器	125
9. 数据库管理	125
9.1. 查询元数据	125
9.1.1. 查看数据库	125
9.1.2. 查看当前会话默认数据库	125
9.1.3. 设置当前会话默认数据库	126
9.1.4. 查看系统表	126
9.1.5. 查看当前会话默认库中用户数据表	126
9.1.6. 查看数据表定义	126
9.1.7. 查看建表语句	127
9.1.8. 查看列定义	127
9.1.9. 查看索引	127
9.2. 用户及权限管理	128
9.2.1. 新建用户	128
9.2.2. 删除用户	128
9.2.3. 修改用户密码	129
9.2.4. 修改用户名	129
9.2.5. 用户锁定/解锁	130
9.2.6. 授予权限	130
9.2.7. 回收权限	132
9.2.8. 查看权限	133
9.2.9. 权限检查	133
9.2.10. 登录失败控制	135

9.2.11. 客户端控制	135
9.2.12. 安全审计	136
9.2.13. 用户安全访问	138
9.2.13.1. 密码配置过期时长	138
9.2.13.2. 安全访问控制	139
9.2.13.3. 限定账户可访问时段	141
9.3. 设置用户变量	143
9.4. 修改系统变量	143
10. 其它语句	144
10.1. 查看当前连接	144
10.2. 中止用户连接	145
10.3. EXPLAIN 语句	146
10.4. 语句中的 hint	146
附录一 KINGWOW 数据库系统表	147
1. __first_tablet_entry	147
2. __all_all_column	148
3. __all_join_info	149
4. __all_cluster	150
5. __all_server	150
6. __all_database	151
7. __all_server_session	151
8. __all_server_stat	151
9. __all_sys_config_stat	152
10. __all_sys_param	152
11. __all_sys_stat	153
12. __all_table_privilege	153
13. __all_user	154
14. __all_database_privilege	155
15. __all_sequence	155
16. __all_truncate_op	156
附录二 KINGWOW 关键字列表	156

1. 编写目的

本文列出了 KINGWOW 分布式数据库 V6 版本所支持的 SQL 语句、语法规则；同时给出了针对 KINGWOW 分布式数据库编写高性能 SQL 语句的方法，为基于 KINGWOW 的应用开发提供指导性意见。

KINGWOW 数据库天然具备分布式能力，无需用户进行分库分表设计，可参照 MySQL 或 ORACLE 等传统集中式数据库的语法规则直接使用。

2. 读者对象

- 数据库管理员
- 应用开发工程师

读者应具有使用 SQL 语言的经验，通过本文可以了解 KINGWOW 数据库所支持的数据类型和语法与其它数据库系统在细节上的区别，以便顺利地使用 KINGWOW 分布式数据库。

3. 基本的数据库对象

KINGWOW 支持 DATABASE、TABLE、VIEW、INDEX、SEQUENCE、存储过程、自定义函数、触发器等数据库对象。

4. SQL 基础

KINGWOW 数据库完全兼容 MySQL 的通讯协议，所以用户可以直接使用 kingwow 客户端、MySQL JDBC Driver 连接 KINGWOW 数据库。

KINGWOW 数据库的 SQL 语法遵循 SQL92 标准，使用单引号标注字符串，使用双引号标注表名、列名或函数名，双引号内可以出现 SQL 保留的关键字。KINGWOW 数据库 SQL 语句中的关键字、表名、列名、函数名等均大小写不敏感。表名和列名都被转换为小写之后存入系统表中，所以即使建表时列名是大写的，查询时获得的列名也是小写。如果用户需要保存大写字母，请使用双引号标注库名，表名和列名，例如："Info"。

4.1. 数据类型

KINGWOW 数据库所支持的每种数据类型都有有效值范围。

表 4-1: KINGWOW 数据库支持的数据类型

数据类型	说明
bigint	按 8 字节有符号整型格式存储, 显示为 int;
int/integer/mediumint /smallint/tinyint	int、integer、mediumint 无论语义还是实现都是等价的, 按 4 字节有符号整型格式存储, 显示为 int32;
binary/char /varbinary/varchar	字符串, 使用单引号进行标注。在 KINGWOW 数据库中, 这四种类型均存储为 varchar 类型, 显示为 varchar;
bool	布尔类型, 值为 1 或者 0。 插入 bool 型数据 true 为 1, false 为 0; 插入数值型数据, 非 0 值为 1, 0 为 0; 插入 varchar 型数据, 'true'、't'、'yes'、'y' 为 1, 其它值为 0;
createtime	特殊数据类型, 用于记录本条数据第一次插入时的时间, 由系统自动维护, 用户不能直接修改; 该类型的列不能作为主键的组成部分。
modifytime	特殊数据类型, 用于记录本条数据最近一次被修改的时间, 由系统自动维护, 用户不能直接修改; 该类型的列不能作为主键的组成部分。
datetime/timestamp	时间戳类型, 支持的格式有 YYYY-MM-DD、YYYY-MM-DD HH:MI:SS、YYYY-MM-DD HH:MI:SS. SSSSSS; 不支持 time with time zone, timestamp with time zone;
date	日期类型, 格式 YYYY-MM-DD;
time	时间类型, 格式 HH:MI:SS;
double/real	8 字节浮点数, double/real 均被存储为 double 类型;
float	4 字节浮点数;
decimal(p,s)	decimal 类型, p 表示总位数, 不能超过 38, s 表示小数部分的精度, 最大不超过 37, 即整数位最少 1 位; 小数部分超过 s 的部分将被截取(非四舍五入), 不足时, 将会补 0;

数据类型	说明
ENUM	底层存储为 varchar 类型，使用时需要用单引号进行标注。
json	json 类型，最大长度 64K。输入必须是一个符合 json 数据格式的字符串，且此字符串需要用单引号进行标注。
blob	最大长度 64K。
tinyblob	最大长度 255 byte。
mediumblob	压缩前最长 16M，压缩后最长 1M， 需要同时满足。
longblob	压缩前最长 16M 压缩后最长 1M， 需要同时满足。
tinyclob	最大长度 255 byte。
text	最大长度 64K。
mediumtext	压缩前最长 16M，压缩后最长 1M， 需要同时满足。
longtext	压缩前最长 16M，压缩后最长 1M， 需要同时满足。
Sequence	是用于产生唯一整数的数据库对象。序列的值是按照一定规则自增的整数，具有唯一标识性，所以 Sequence 常被用作主键使用。
Xml	XML 类型数据具有固定的结构，需要使用 <> 标识符例如 <employee><name>John Doe</name><department>HR</department></employee>。

4.2. 函数

4.2.1. 字符串函数

LENGTH(str)

描述：用于返回字符串的长度。

str: 需要计算长度的字符串。

返回值：返回字符串 str 的长度，int 型整数，单位为字节。如果参数 str 不是字符串，则返回 NULL 值。

示例：

```
kingwow> select length('KINGWOW'),length('中国'),length(123),length(10.1),length(null);
+-----+-----+-----+-----+-----+
| length('KINGWOW') | length('中国') | length(123) | length(10.1) | length(null) |
+-----+-----+-----+-----+-----+
| 7 | 6 | 3 | 4 | NULL |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

CHR(ascii_code)

描述：用于将给定的 ASCII 码转换为对应的字符。

ascii_code:要转换的 ASCII 码。

返回值：返回对应 ASCII 码的字符，ascii_code 最大为 127。



示例：

```
kingwow> select chr(65);
+-----+
| chr(65) |
+-----+
| A       |
+-----+
1 row in set (0.00 sec)
```

CHAR|CHARACTER(ascii_code)

描述：用于将 ASCII 码转换为字符。

ascii_code:要转换的 ASCII 码。

返回值：返回对应 ASCII 码的字符，ascii_code 最大为 127。

示例：

```
kingwow> select character(124),character(64),char(124),char(64);
+-----+-----+-----+-----+
| character(124) | character(64) | char(124) | char(64) |
+-----+-----+-----+-----+
| |              | @           | |         | @         |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

TO_CHAR(value[,format])

描述：用于将传入参数转换为字符串。

value: 待转换的值，可以为任何 Kingwow 的内建数据类型。

format: 可选项，字符串转换格式。

返回值：会返回 value 对应的字符串，如果设置 format 会返回对应格式的字符串，否则默认返回原格式字符串。

示例：

```
kingwow>SELECT TO_CHAR(NOW(),'YYYY-MM-DD');
+-----+
| TO_CHAR(NOW(), 'YYYY-MM-DD') |
+-----+
| 2024-01-17                    |
+-----+
1 row in set (0.00 sec)
```

CHAR_LENGTH|CHARACTER_LENGTH(str)

描述：用于以字符为单位返回字符串的长度。

补充：类似于 length()返回字符串 str 的长度，但 length 是以字节为单位。char_length 中多字节字符算单个字符。对于包含 5 个 2 字节字符的字符串，LENGTH()返回 10，而 CHAR_LENGTH()返回 5。

返回值：str 的字符长度，如果参数 str 不是字符串，则返回 NULL 值。

示例：

```
kingwow> select char_length('中国'), char_length(null);
+-----+-----+
| char_length('中国') | char_length(null) |
+-----+-----+
| 2                  | NULL              |
+-----+-----+
```

CONCAT(str1,str2,...,strn)

描述：用于将多个字符串拼接成一个字符串。

str1,str2,...,strn: 待拼接的字符串。

返回值：拼接后的字符串。

示例：

```
kingwow> select concat('name','score','nicole');
+-----+
| concat('name','score','nicole') |
+-----+
| namescorenicole                  |
+-----+
1 row in set (0.00 sec)
```

CONCAT_WS(separator,str1,str2,...,strn)

描述：用于把多个字符串连接成一个字符串，相邻字符串间使用 separator 分隔。参数中包含 NULL 时，将忽略 NULL。

separator: 设置字符串间指定分隔符，分隔符的位置放在要连接的两个字符串之间。

分隔符可以是一个字符串，也可以是其它参数。

str1,str2,...,strn: 待拼接的字符串。

返回值：由传入字符和分隔符逐个拼接的字符串，如果分隔符为 NULL，则结果为 NULL。函数会忽略任何分隔符参数后的 NULL 值。

示例：

```
kingwow> select concat_ws(',', '11', '22', '33', null, '88');
+-----+
| concat_ws(',', '11', '22', '33', null, '88') |
+-----+
| 11,22,33,,88                                |
+-----+
1 row in set (0.00 sec)
```

POSITION(substr in str) | LOCATE(substr,str)

描述：用于查找子字符串 substr 在字符串 str 中的其实位置。

补充：在字符串 substr 里面,字符串 substr 出现的第一个位置(index)，index 是从 1 开始计算，如果没有找到就直接返回 0，如果 substr 或 str 为 NULL，则返回 NULL。

返回值：返回子字符串 substr 在字符串 str 中的位置，从 1 开始计数。如果子字符串不存在，则返回 0。

示例：

```
kingwow> select position('ab' in 'abcde'),locate('ab','abcde'),position(null in 'abcde');
+-----+-----+-----+
| position('ab' in 'abcde') | locate('ab','abcde') | position(null in 'abcde') |
+-----+-----+-----+
| 1 | 1 | NULL |
+-----+-----+-----+
1 row in set (0.00 sec)
```

INSTR(str, substr)

描述：在一个字符串中搜索指定的字符，返回发现指定的字符的位置。

补充：在字符串 str 里面,字符串 substr 出现的第一个位置(INDEX), INDEX 是从 1 开始计算，如果没有找到就直接返回 0，如果 substr 或 str 为 NULL，则返回 NULL。

返回值：返回子字符串 substr 在字符串 str 中第一次查找到的位置，从 1 开始计数。如果子字符串不存在，则返回 0。

示例：

```
kingwow> select instr('abcde' , 'de'),instr('ab', 'b'), instr('ab', null),instr(null , 'abcde');
+-----+-----+-----+-----+
| instr('abcde' , 'de') | instr('ab', 'b') | instr('ab', null) | instr(null , 'abcde') |
+-----+-----+-----+-----+
| 4 | 2 | NULL | NULL |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

LPAD|RPAD(str1,length,str2)

描述：从左边|右边对字符串 str1 使用指定的字符 str2 进行填充，直到达到指定长度 length。

str1: 要填充的原始字符串。

length: 填充后的字符串的长度。

str2: 用于填充的字符或字符串。

返回值：返回填充后的字符串，如果 str1 的长度没有 length 那么长，则使用 str2 填充；如果 str1 的长度大于 length，则截断。

示例：

```
kingwow> select lpad('hi',1,'KINGWOWKINGWOW'),lpad('hi',6,'KINGWOWKINGWOW');
+-----+-----+
| lpad('hi',1,'KINGWOWKINGWOW') | lpad('hi',6,'KINGWOWKINGWOW') |
+-----+-----+
| h | KINGhi |
+-----+-----+
1 row in set (0.01 sec)

kingwow> select rpad('tech', 8, '0');
+-----+
| rpad('tech', 8, '0') |
+-----+
| tech0000 |
+-----+
1 row in set (0.00 sec)
```

LEFT(str,length)|RIGHT(str,length)

描述：用于从字符串的左侧|右侧提取指定数量的字符。

str: 要从中提取字符的原始字符串。

length: 要提取的字符数。

返回值：返回对应长度的字符串，如果提取的字符数超过了原始字符串的长度，则返回原字符串,如果任一参数为 NULL 则返回 NULL。

示例：

```
kingwow>SELECT LEFT('2019-01-30',4),LEFT('sgwer8914',7),LEFT(NULL,4),LEFT('hhh',NULL);
+-----+-----+-----+-----+
| LEFT('2019-01-30',4) | LEFT('sgwer8914',7) | LEFT(NULL,4) | LEFT('hhh',NULL) |
+-----+-----+-----+-----+
| 2019 | sgwer89 | NULL | NULL |
+-----+-----+-----+-----+
```

```

+-----+-----+-----+-----+
1 row in set (0.00 sec)

kingwow> SELECT RIGHT('2019-01-30',4),RIGHT('sgwer8914',7),RIGHT(NULL,4),RIGHT('hhh',NULL);
+-----+-----+-----+-----+
| RIGHT('2019-01-30',4) | RIGHT('sgwer8914',7) | RIGHT(NULL,4) | RIGHT('hhh',NULL) |
+-----+-----+-----+-----+
| 1-30                  | wer8914              | NULL          | NULL              |
+-----+-----+-----+-----+
1 row in set (0.01 sec)

```

SUBSTR|SUBSTRING|MID(str, pos[, len])

描述：用于从字符串中提取子字符串。

str: 要提取子字符串的原始字符串。

pos: 开始提取的位置。

len: 要提取的字符数。如果省略，将提取从 pos 位置到字符串末尾的所有字符。

返回值：返回对应字符串，pos 值为负数时，表示会从字符串末尾开始计数提取字符串。len 小于等于 0 时，返回结果为空字符串。如果 pos 指示的位置在字符串中不存在时，返回为空字符串。

示例：

```

kingwow> select substr('01234',2,1),substr('01234',2),substr('01234',-2,1);
+-----+-----+-----+
| substr('01234',2,1) | substr('01234',2) | substr('01234',-2,1) |
+-----+-----+-----+
| 1                  | 1234              | 3                    |
+-----+-----+-----+
1 row in set (0.00 sec)

kingwow> select substring('01234',2,1),substring('01234',2),substring('01234',9,1);
+-----+-----+-----+
| substring('01234',2,1) | substring('01234',2) | substring('01234',9,1) |
+-----+-----+-----+
| 1                  | 1234              |                      |
+-----+-----+-----+
1 row in set (0.00 sec)

kingwow> select mid('01234',2,1),mid('01234',2),mid('01234',-2,1),mid('01234',9,1);
+-----+-----+-----+-----+
| mid('01234',2,1) | mid('01234',2) | mid('01234',-2,1) | mid('01234',9,1) |
+-----+-----+-----+-----+
| 1                  | 1234              | 3                    |                      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

SUBSTRING_INDEX(str, delim, count)

描述：用于返回从指定位置开始的指定长度的子字符串。

str: 要提取子字符串的原始字符串。

delim: 分隔符，用于指定子字符串的位置。

count: 索引位置，表示从分隔符的哪一侧开始提取子字符串。

返回值：返回对应字符串，如果 count 是正数，则从左侧开始提取；如果 count 是负数，则从右侧开始提取。如果不指定分隔符，则返回原始字符串；如果指定分割符不在原始字符串内，则返回原始字符串。如果原始字符串为空字符串，则返回空字符串。

示例：

```

kingwow> SELECT SUBSTRING_INDEX('a,b,c,d', ',', 1),SUBSTRING_INDEX('a,b,c,d', ',', -2);
+-----+-----+

```



```

| SUBSTRING_INDEX('a,b,c,d', ',', 1) | SUBSTRING_INDEX('a,b,c,d', ',', -2) |
+-----+-----+
| a | c,d |
+-----+-----+
1 row in set (0.00 sec)
kingwow> SELECT SUBSTRING_INDEX('a,b,c,d', '!', 2),SUBSTRING_INDEX('a,b,c,d', '!', 2);
+-----+-----+
| SUBSTRING_INDEX('a,b,c,d', '!', 2) | SUBSTRING_INDEX('a,b,c,d', '!', 2) |
+-----+-----+
| a,b,c,d | a,b,c,d |
+-----+-----+
1 row in set (0.00 sec)

```

TRIM([[[both | leading | trailing]] [remstr] from] str)

描述:用于删除一个字符串开头或结尾对应的子字符串。

both:指定同时删除字符串开头和结尾的空格或字符。

leading:指定仅删除字符串开头的空格或字符。

trailing: 指定仅删除字符串结尾的空格或字符。

remstr: 指定要删除的字符或字符串。如果未指定，默认删除空格。

from: 指定要从中删除空格或字符的字符串。如果未指定，默认为当前行的对应列。

str: 要处理的字符串。

返回值: 删除对应字符后的字符串。

示例:

```

kingwow>select trim('      x test x      ');
+-----+
| trim('      x test x      ') |
+-----+
| x test x |
+-----+
1 row in set (0.00 sec)

kingwow> select trim(both 'a' from 'a123ab'),trim(leading 'a' from 'a123ab'),trim(trailing 'b' from 'ab123ab');
+-----+-----+-----+
| trim(both 'a' from 'a123ab') | trim(leading 'a' from 'a123ab') | trim(trailing 'b' from 'ab123ab') |
+-----+-----+-----+
| 123ab | 123ab | ab123a |
+-----+-----+-----+
1 row in set (0.00 sec)

```

LTRIM(str1[,str2])|RTRIM(str1[,str2])

描述: 用于删除字符串左侧|右侧的空格或字符串。

str1: 原始字符串。

str2: 要删除的子串或字符。不加则默认删除空格。

返回值: 返回删除对应字符的字符串。

示例:

```

kingwow> SELECT ltrim('      chinese      '),ltrim('chinese','c');
+-----+-----+
| ltrim('      chinese      ') | ltrim('chinese','c') |
+-----+-----+
| chinese | hinese |
+-----+-----+
1 row in set (0.00 sec)
kingwow> SELECT rtrim('      chinese      '),rtrim('chinese','se');
+-----+-----+
| rtrim('      chinese      ') | rtrim('chinese','se') |
+-----+-----+

```

SPACE(n)

描述：用于返回一个由指定数量的空格组成的字符串。

n: 表示要返回的空格数量。

返回值：返回对应长度的空字符串。

示例：

```
kingwow> select space(1),space(50);
+-----+
| space(1) | space(50) |
+-----+
|          |           |
+-----+
1 row in set (0.00 sec)
```

STRSPLIT (str1,str2,n)

描述：用指定子字符串 str2 对源字符串 str1 进行分割，并提取出分隔后的第 n 段字符串。

str1: 待分割的源字符串。

str2: 用于分割的子字符串。

n: 输入一个整数，表示提取分割后的第 n 段字符串。

返回值：会返回一个字符串，表示分割后指定提取的字符串。如果 n 大于分割后的份数，则默认输出被分割后的第一段。如果 str2 不是 str1 的子字符串，则默认返回空串。

示例：

```
kingwow> select strsplit('csnodhabciojd','abc',2),strsplit('csnodhabciojdabchhh','abc',3);
+-----+
| strsplit('csnodhabciojd','abc',2) | strsplit('csnodhabciojdabchhh','abc',3) |
+-----+
| iojd                               | hhh                                     |
+-----+
1 row in set (0.00 sec)

kingwow> select strsplit('csnodhabciojd','abc',3),strsplit('csnodhabciojdabchhh','eee',3);
+-----+
| strsplit('csnodhabciojd','abc',3) | strsplit('csnodhabciojdabchhh','eee',3) |
+-----+
| csnodh                             |                                         |
+-----+
1 row in set (0.00 sec)
```

TRANSLATE (str,from_str,to_str)

描述：用于将一个字符串中的某些字符替换为其他字符。

str: 要替换字符的原始字符串。

from_str: 要被替换的字符或字符集。

to_str: 替换后的字符或字符集。

返回值：返回替换后的 str。

示例：

```
kingwow> select translate('ab 你好 bcdefg', 'abcdefg','1234567');
+-----+
| translate('ab 你好 bcdefg', 'abcdefg','1234567') |
+-----+
```

```

+-----+
| 12 你好 2314567 |
+-----+
1 row in set (0.00 sec)

```

REPEAT(str,count)

描述：用于重复一个字符串指定的次数。

str: 是要重复的字符串。

count: 重复的次数。

返回值：返回一个由字符串 str 重复 count 次数组成的字符串。如果 count 小于 1, 则返回一个空字符串。如果 str 或 count 为 NULL, 则返回 NULL。

示例：

```

kingwow>
SELECT repeat('KINGWOW', 3),repeat('KINGWOW', -1),repeat(NULL , 3),repeat('KINGWOW', NULL );
+-----+-----+-----+-----+
| repeat('KINGWOW', 3) | repeat('KINGWOW', -1) | repeat(NULL , 3) | repeat('KINGWOW', NULL ) |
+-----+-----+-----+-----+
| KINGWOWKINGWOWKINGWOW | NULL | NULL | NULL |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

REPLACE(str,from_str[,to_str])

描述：用于将字符表达式值中, 部分相同字符串, 替换成新的字符串。

str: 待替换的原始字符串。

from_str: 要被替换的字符或字符集。

to_str: 替换后的字符或字符集。可选项, 不写则默认为空, 表示仅删除不替换。

返回值：返回替换后的 str。

示例：

```

kingwow> SELECT REPLACE('www.kw.net', 'www', 'http://www') c1,REPLACE('www.kw.net', 'www') c2;
+-----+-----+
| c1 | c2 |
+-----+-----+
| http://www.kw.net | .kw.net |
+-----+-----+
1 row in set (0.00 sec)

```

INSERT(str1,x,len,str2)

描述：用于在字符串中替换指定位置的字符串, 并可以指定替换字符的数量。

str1: 要被替换字符的原始字符串。

x: 在 str1 中插入 str2 的起始位置。

len: 要替换的字符数。

str2: 待替换的字符串。

返回值：返回被替换后的字符串。如果 x 超过原始字符串 str1 的长度, 则直接返回 str1; 如果 len 的长度大于从 x 开始计数到 str1 最后一个字符的长度, 则直接用 str2 替换从 x 开始的全部字符; 当任何一个参数为 NULL 时, 返回值为 NULL。

示例：

```

kingwow> SELECT INSERT('abcde',2,2,'hh'),INSERT('abcde',2,20,'hh');
+-----+-----+
| INSERT('abcde',2,2,'hh') | INSERT('abcde',2,20,'hh') |
+-----+-----+

```

```

+-----+-----+
| ahhde | ahh |
+-----+-----+
1 row in set (0.00 sec)

kingwow> SELECT INSERT('abcde',null,2,'hh'),INSERT(null,7,20,'hh');
+-----+-----+
| INSERT('abcde',null,2,'hh') | INSERT(null,7,20,'hh') |
+-----+-----+
| NULL | NULL |
+-----+-----+
1 row in set (0.00 sec)

```

FIND_IN_SET(str,strlist)

描述：用于查找一个字符串在一个逗号分隔的列表中的位置。

str: 要查找的字符串。

strlist: 逗号分隔的字符串列表。

返回值：返回值是 int 类型，代表 str 在 strlist 中的位置。如果 str 在 strlist 中存在，则返回位置索引（从 1 开始），如果不存在，则返回 0。

示例：

```

kingwow> SELECT find_in_set('2','672,6843,2,85'),find_in_set('abe','jhvs,abc,jpv,abck');
+-----+-----+
| find_in_set('2','672,6843,2,85') | find_in_set('abe','jhvs,abc,jpv,abck') |
+-----+-----+
| 3 | 0 |
+-----+-----+
1 row in set (0.00 sec)

```

LOWER(str)|LCASE(str)

描述：用于将字符串中的英文转为小写。

str: 待转换的字符串。

返回值：返回转换完的字符串，兼容中文字符。

示例：

```

kingwow> select lower('欢迎使用 KINGWOW'),lcase('欢迎使用 KINGWOW'),lower('NULL'),lower(123.1);
+-----+-----+-----+-----+
| lower('欢迎使用 KINGWOW') | lcase('欢迎使用 KINGWOW') | lower('NULL') | lower(123.1) |
+-----+-----+-----+-----+
| 欢迎使用 kingwow | 欢迎使用 kingwow | null | 123.1 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

UPPER(str)|UCASE(str)

描述：用于将字符串中的英文转为大写。

str: 待转换的字符串。

返回值：返回转换完的字符串，兼容中文字符。

```

kingwow> select upper('欢迎使用 kingwow'),ucase('欢迎使用 kingwow'),upper('null'),upper(123.1);
+-----+-----+-----+-----+
| upper('欢迎使用 kingwow') | ucase('欢迎使用 kingwow') | upper('null') | upper(123.1) |
+-----+-----+-----+-----+
| 欢迎使用 KINGWOW | 欢迎使用 KINGWOW | NULL | 123.1 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

NISSORT()

描述：用于对表中数据按拼音排序。

返回值：配合 select ...order by 查询得到的数据会按拼音进行排序。

示例：

```
create table test(c1 varchar(20));
insert into test values('ABC 你好 HELLO 好吧'),('张三'),('Qbc'),('hellokitty'),('什么');
kingwow> select * from test order by nlssort(c1, 'NLS_SORT = SCHINESE_PINYIN_M');
+-----+
| c1                |
+-----+
| ABC 你好 HELLO 好吧 |
| Qbc                |
| hellokitty         |
| 什么                |
| 张三                |
+-----+
5 rows in set (0.03 sec)
```

PINYIN_INITIALS()

描述：用于返回字符串中中文字符的拼音首字母。

返回值：当字符串中有中文字符时按序返回中文字符的拼音首字母，如果没有中文字符则返回 NULL。

示例：

```
kingwow> select pinyin_initials('hh 张 hh 三 hh'),pinyin_initials('hhh');
+-----+
| pinyin_initials('hh 张 hh 三 hh') | pinyin_initials('hhh') |
+-----+
| zS                                | NULL |
+-----+
1 row in set (0.00 sec)
```

BITAND(expr1,expr2)

描述：用于对两个整数进行按位与操作。

expr1、expr2：是两个参与按位与运算的整数表达式。

返回值：返回按位与后的计算结果。

示例：

```
kingwow> select bitand(1,5),bitand(13,25);
+-----+
| bitand(1,5) | bitand(13,25) |
+-----+
| 1           | 9             |
+-----+
```

HEX(str)

描述：将数字或字符串转化为十六进制字符串。

str：是要转换的字符串或整数。

补充：KINGWOW 数据库中两个十六进制数位表示一个字符，因此需要用偶数位数来表示，例如 0x123 转换会失败，0x0123 可以转换成功。

返回值：转换后的十六进制字符串。当输入整数时（不限进制），输出结果都会是该整数的



十六进制表示结果；当输入是字符串时，输出是该字符串字节流的十六进制表示结果。

示例：

```
kingwow> select hex('KINGWOW'),hex(123),hex(null),hex(0x0123);
+-----+
| hex('KINGWOW') | hex(123) | hex(null) | hex(0x0123) |
+-----+
| 4B494E47574F57 | 7B      | NULL     | 0123        |
+-----+
1 row in set (0.00 sec)
```

UNHEX(str)

描述： HEX(str)的反向操作，用于将十六进制字符串转化为正常字符串。

str：是要转换的字符串。

返回值： 返回转换后的字符串，当字符串 str 是合法的十六进制数时，将其转换为对应的字符串，当 str 不是合法的十六进制数时返回 NULL 值，str 为 NULL 时返回 NULL。

示例：

```
kingwow> select unhex('4841505059'),unhex('7b'),unhex('123g'),unhex(null),unhex('0123');
+-----+
| unhex('4841505059') | unhex('7b') | unhex('123g') | unhex(null) | unhex('0123') |
+-----+
| HAPPY                | {           | NULL          | NULL        | #             |
+-----+
1 row in set (0.00 sec)
```

STRCMP(str1,str2)

描述： 用于比较两个字符串是否相等。

str1、str2：待比较的两个字符串。

返回值： 返回比较结果，如果 str1 等于 str2，则返回 0；如果 str1 小于 str2，则返回-1；如果 str1 大于 str2 则返回 1；如果任一参数为 null，返回结果为 NULL；

示例：

```
kingwow> SELECT strcmp(123, 123),strcmp(123, 122),strcmp(123, 124),strcmp(123, NULL);
+-----+
| strcmp(123, 123) | strcmp(123, 122) | strcmp(123, 124) | strcmp(123, NULL) |
+-----+
| 0                | 1                | -1               | NULL              |
+-----+
1 row in set (0.00 sec)
```

QUARTER(date)

描述： 用于将日期或时间戳值转换为季度。

date：要转换为季度的日期或时间戳。

返回值： 返回值结果为 1-4，对应 data 在年中的季度；如果 date 为非法值，返回结果为 NULL；

示例：

```
kingwow> select quarter('2019-11-28'),quarter(NULL),quarter(123);
+-----+
| quarter('2019-11-28') | quarter(NULL) | quarter(123) |
+-----+
| 4                      | NULL          | NULL         |
+-----+
```

VALUE(expr)

描述：用于获取 expr 的值。

返回值：返回 expr 对应的值。如果 expr 为计算表达式，则返回计算后的结果；如果 expr 为字符串或数字则返回对应的值；如果为 expr 空串，则返回 NULL；如果 expr 为 NULL，则返回 NULL。

示例：

```
kingwow> select value(1+1),value(''),value('中国'),value(NULL);
```

value(1+1)	value('')	value('中国')	value(NULL)
2	NULL	中国	NULL

1 row in set (0.00 sec)

TO_NUMBER(str)

描述：用于将 str 字符串转换为数字。

str: 待转换的字符串。

返回值：返回对应的数字。

示例：

```
kingwow>SELECT TO_NUMBER('199912'),TO_NUMBER('450.05') FROM DUAL;
```

TO_NUMBER('199912')	TO_NUMBER('450.05')
199912	450.05

1 row in set (0.00 sec)

INITCAP(str)

描述：用于将字符串中的每个单词的首字母转换为大写，其余字母转换为小写。

str: 待转换的字符串。

返回值：返回按函数规则转换后的字符串。

示例：

```
kingwow>SELECT initcap('hi THOMAS');
```

initcap('hi THOMAS')
Hi Thomas

1 row in set (0.02 sec)

ASCII(str)

描述：用于返回字符串的第一个字符的 ASCII 码值。

str: 待处理的字符串。

返回值：会返回一个整数，表示字符串中第一个字符的 ASCII 码值。

示例：

```
kingwow>SELECT ascii('xyz');
+-----+
| ascii('xyz') |
+-----+
|          120 |
+-----+
1 row in set (0.00 sec)
```

SUBSTRB(str, start_pos, len)

描述：用于从字符串中提取子字符串。

str: 源字符串。

start_pos: 提取字符串的起始位置，默认从 1 开始计数。

len:要提取的子字符串的长度。

返回值：会返回提前到的子字符串。

示例：

```
kingwow>SELECT substrb('string',2,3);
+-----+
| substrb('string',2,3) |
+-----+
| tri                   |
+-----+
1 row in set (0.01 sec)
```

REGEXP_LIKE(source_char, pattern)

描述：用于字符串中的正则表达式匹配。

source_char: 要搜索的字符串。

pattern: 正则表达式模式。

返回值：返回 1 表示正则匹配，返回 0 表示不匹配。

示例：

```
kingwow> select REGEXP_LIKE ('Nicole', '^Ste(v|ph)en$');
+-----+
| REGEXP_LIKE ('Nicole', '^Ste(v|ph)en$') |
+-----+
|                                0         |
+-----+
1 row in set (0.01 sec)
```

```
kingwow> select REGEXP_LIKE ('Stephen', '^Ste(v|ph)en$');
+-----+
| REGEXP_LIKE ('Stephen', '^Ste(v|ph)en$') |
+-----+
|                                1         |
+-----+
1 row in set (0.00 sec)
```

REGEXP_SUBSTR(str,pattern,[position[,occurrence]])

描述：用于基于正则表达式提取字符串的函数。

str: 待搜索的字符串。

pattern: 正则表达式。

position 为搜索的开始位置。必须是大于 0 的正整数, 小于等于 0 报错, 输入为 NULL



返回 NULL，默认值为 1，表示从第一个字符开始搜索。

occurrence 返回第几次匹配的值。必须是大于等于 0 的正整数，小于等于 0 报错，输入为 NULL 返回 NULL，默认值是 1，表示返回第一次匹配的结果。

返回值：会返回一个用正则表达式提取出来的字符串。

示例：

```
kingwow>
select regexp_substr('I have 2 apples and 100 bucks!', '[:blank:][:alnum:]*', 1, 1) from dual;
+-----+
| regexp_substr('I have 2 apples and 100 bucks!', '[:blank:][:alnum:]*', 1, 1) |
+-----+
| have                                     |
+-----+
1 row in set (0.00 sec)
```

REGEXP_COUNT(str, pattern [, pos [, match_occurrence [, match_option]]])

描述：用于在字符串中匹配正则表达式模式，并返回匹配的数量。

str: 待搜索的字符串。

pattern: 要匹配的正则表达式模式。

pos: 开始搜索的位置，可选项，默认为 1，即从第一个字符开始搜索。

match_occurrence :要返回的匹配项的实例。默认为 1，表示所有匹配项。

match_option: 匹配选项，如 'i' 表示不区分大小写；'c' 用于区分大小写的匹配；

'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。

返回值：会返回正则表达式匹配数量。

示例：

```
kingwow>SELECT regexp_count('foobaz', 'b(..)', 5) AS RESULT;
+-----+
| RESULT |
+-----+
|      1 |
+-----+
1 row in set (0.00 sec)

kingwow>select REGEXP_COUNT('GEORGEoGeUFTGGEikkkge000', 'GE', 2, 'i') from DUAL;
+-----+
| REGEXP_COUNT('GEORGEoGeUFTGGEikkkge000', 'GE', 2, 'i') |
+-----+
|                                     4 |
+-----+
1 row in set (0.01 sec)

kingwow>select REGEXP_COUNT('GEORGEoGeUFTGGEikkkge000', 'GE', 2, 'c') from DUAL;
+-----+
| REGEXP_COUNT('GEORGEoGeUFTGGEikkkge000', 'GE', 2, 'c') |
+-----+
|                                     2 |
+-----+
1 row in set (0.00 sec)
```

REGEXP_INSTR(str, pattern [, pos [, match_occurrence [, return_opt [, match_option]]])

描述：用于在字符串中查找正则表达式模式的匹配项，并返回匹配项的起始位置。

str: 待搜索的字符串。

pattern: 要匹配的正则表达式模式。

pos: 开始搜索的位置，可选项，默认为 1，即从第一个字符开始搜索。

match_occurrence: 要返回的匹配项的实例。默认为 1，表示所有匹配项。

return_opt: 指定返回哪一种位置索引。如果为 0，返回匹配的子串的第一个字符的位置索引；如果为 1，返回匹配的子串的后面的位置索引。默认为 0。

match_option: 匹配选项，如 'i' 表示不区分大小写；其他同 REGEXP_COUNT 函数。

返回值：会返回一个整数，代表匹配项的起始位置。

示例：

```
kingwow> SELECT REGEXP_INSTR ('BCA MCA PGDCA BBA MBA BCA', 'BCA', 1, 1, 1) AS post_index;
+-----+
| post_index |
+-----+
|          4 |
+-----+
1 row in set (0.00 sec)

kingwow> SELECT REGEXP_INSTR ('BCA MCA PGDCA BBA MBA BCA', 'BCA', 1, 1, 0) AS post_index;
+-----+
| post_index |
+-----+
|          1 |
+-----+
1 row in set (0.01 sec)
```

REGEXP_REPLACE(source_str, pattern ,replace_str[, pos [, match_occurrence [, match_option]]])

描述：用于在字符串中查找与正则表达式模式匹配的文本，并将其替换为指定的文本。

source_str: 待搜索的字符串。

pattern: 要匹配的正则表达式模式。

replace_str: 用于替换匹配项的字符串。

pos: 开始搜索的位置，可选项，默认为 1，即从第一个字符开始搜索。

match_occurrence :要返回的匹配项的实例。默认为 1，表示所有匹配项。

match_option: 匹配选项，如 'i' 表示不区分大小写；'c' 用于区分大小写的匹配；

'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。

返回值：会返回正则表达式匹配数量。

示例：

```
kingwow> SELECT REGEXP_REPLACE('123 abc 456 def', '\\s+', '-');
+-----+
```



```
| REGEXP_REPLACE('123 abc 456 def', '\\s+', '-') |  
+-----+  
| 123-abc-456-def |  
+-----+  
1 row in set (0.00 sec)
```

```
kingwow> SELECT REGEXP_REPLACE('123 abc 456 def', '\\d+', 'X', 1, 2);  
+-----+  
| REGEXP_REPLACE('123 abc 456 def', '\\d+', 'X', 1, 2) |  
+-----+  
| 123 abc X def |  
+-----+  
1 row in set (0.00 se
```

4.2.2. 日期函数

TIME(datetime)

描述：用于格式化提取 datetime 的时间信息。

返回值：以'HH:MM:SS'格式返回 datetime 的时间信息。

示例：

```
kingwow> select time('2019-11-28 11:22:23');
+-----+
| time('2019-11-28 11:22:23') |
+-----+
| 11:22:23                      |
+-----+
1 row in set (0.01 sec)
```

CURRENT_TIMESTAMP() | NOW()

描述：获取系统当前时间戳，精确到毫秒，格式为"YYYY-MM-DD HH:MM:SS.SSSSSS"。

返回值：now()的返回格式与 current_timestamp()一致。

示例：

```
kingwow> select current_timestamp(),now();
+-----+-----+
| current_timestamp() | now() |
+-----+-----+
| 2024-01-19 11:33:07.190088 | 2024-01-19 11:33:07.190088 |
+-----+-----+
1 row in set (0.00 sec)
```

STRICT_CURRENT_TIMESTAMP()

描述：当系统中运行多个 CG 时，由于不同 CG 可能存在微小的系统时钟误差，将导致 NOW() 的运算结果不一致。STRICT_CURRENT_TIMESTAMP()是从主 TG 上获取时间，从而保证所有客户端获取的时间是一致的。

返回值：以"YYYY-MM-DD HH:MM:SS.SSSSSS"的格式返回集群中主 TG 上的时间信息。

示例：

```
kingwow> select strict_current_timestamp();
+-----+
| strict_current_timestamp() |
+-----+
| 2017-08-07 10:47:21.339091 |
+-----+
1 row in set (0.01 sec)
```

CURRENT_DATE()

描述：获取系统当前时间戳。

返回值：以'YYYY-MM-DD'格式返回当前时间信息。

示例：

```
kingwow> select current_date();
+-----+
| current_date() |
+-----+
| 2024-01-19      |
+-----+
1 row in set (0.00 sec)
```

CURRENT_TIME()

描述：获取系统当前时间戳。

返回值：以'HH:MM:SS'格式返回当前时间信息。

示例：

```
kingwow> select current_time();
+-----+
| current_time() |
+-----+
| 11:35:42       |
+-----+
1 row in set (0.00 sec)
```

DATE(datetime)

描述：以'YYYY-MM-DD'格式返回 datetime 时间值。

返回值：以'YYYY-MM-DD'格式返回 datetime 的时间信息。

示例：

```
kingwow> select date('2019-11-28 11:22:33');
+-----+
| date('2019-11-28 11:22:33') |
+-----+
| 2019-11-28                  |
+-----+
1 row in set (0.00 sec)
```

LAST_DAY(expr)

描述：返回 expr 对应月份中的最后一天。

返回值：以'YYYY-MM-DD'格式返回对应月份的最后一天时间信息。

示例：

```
kingwow> select last_day('2008-02-01'),last_day('2024-08-08');
+-----+-----+
| last_day('2008-02-01') | last_day('2024-08-08') |
+-----+-----+
```

```
| last_day('2008-02-01') | last_day('2024-08-08') |
+-----+-----+
| 2008-02-29          | 2024-08-31          |
+-----+-----+
1 row in set (0.00 sec)
```

NEXT_DAY(date,num)

描述： NEXT_DAY 函数是一个用于日期计算的函数，它返回指定日期之后的下一个指定星期几的日期。这个函数有两个参数：一个日期和一个表示星期几的数字。

其中：

date 是要计算其下一个指定星期几的日期的日期值。

char 是一个表示星期几的字符串（如 'MONDAY'、'TUESDAY' 等）或数字（1 代表星期日，2 代表星期一，以此类推，7 代表星期六）。

示例：

如果你想要使用数字表示星期几，查询会是这样的：

```
SELECT NEXT_DAY(SYSDATE, 2) FROM DUAL; -- 2 代表星期一
```

这个查询会返回从今天开始算起的下一个星期一的日期。

```
kingwow> SELECT NEXT_DAY(SYSDATE, 2) FROM DUAL;
+-----+
| NEXT_DAY(SYSDATE, 2) |
+-----+
| 2025-03-31          |
+-----+
1 row in set (0.000 sec)
```

DATE_FORMAT(date, format)

描述： 用于将日期或时间格式化为指定格式的函数。

date：要格式化的日期或时间。

format：指定日期或时间的格式。

返回值： 返回对应格式的时间信息。

示例：

```
kingwow> SELECT DATE_FORMAT('2018-06-26 11:11:11.123456', '%Y-%m-%d %H:%i:%S');
+-----+
| DATE_FORMAT('2018-06-26 11:11:11.123456', '%Y-%m-%d %H:%i:%S') |
+-----+
| 2018-06-26 11:11:11                                          |
+-----+
1 row in set (0.04 sec)
```

YEAR(datetime_value)

描述：用于获取时间参数的年份信息。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```
kingwow> select year('2015-10-10 12:00:00.1'),year(current_date()),year(current_timestamp());
+-----+-----+-----+
| year('2015-10-10 12:00:00.1') | year(current_date()) | year(current_timestamp()) |
+-----+-----+-----+
| 2015 | 2024 | 2024 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

MONTH(datetime_value)

描述：用于获取时间参数的月份信息。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```
kingwow>select month('2015-10-10 12:00:00.1'),month(current_date()),month(current_timestamp());
+-----+-----+-----+
| month('2015-10-10 12:00:00.1') | month(current_date()) | month(current_timestamp()) |
+-----+-----+-----+
| 10 | 1 | 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

MONTHNAME(date_value)

描述：返回日期 date_value 对应月份的英文全名。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```
kingwow> SELECT MONTHNAME('2017-12-15'),MONTHNAME('2000-02-02');
+-----+-----+
| MONTHNAME('2017-12-15') | MONTHNAME('2000-02-02') |
+-----+-----+
| December | February |
+-----+-----+
1 row in set (0.00 sec)
```

DAY(datetime_value)

描述：用于返回时间参数的日信息。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```
kingwow select day('2015-10-10 12:00:00.1'),day(date '2016-07-05'),day(current_timestamp());
```

```

+-----+-----+-----+
| day('2015-10-10 12:00:00.1') | day(date '2016-07-05') | day(current_timestamp()) |
+-----+-----+-----+
| 10 | 5 | 19 |
+-----+-----+-----+
1 row in set (0.02 sec)

```

HOUR(datetime_value)

描述：返回时间参数的小时信息（0~23）。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```

kingwow>select hour('2015-10-10 12:00:00.1'),hour(time '12:01:02'),hour(current_time());
+-----+-----+-----+
| hour('2015-10-10 12:00:00.1') | hour(time '12:01:02') | hour(current_time()) |
+-----+-----+-----+
| 12 | 12 | 11 |
+-----+-----+-----+
1 row in set (0.00 sec)

```

MINUTE(datetime_value)

描述：返回时间参数的分钟信息。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```

kingwow> select minute('2015-10-10 12:00:00.1'),minute(time '12:01:02'),minute(current_time());
+-----+-----+-----+
| minute('2015-10-10 12:00:00.1') | minute(time '12:01:02') | minute(current_time()) |
+-----+-----+-----+
| 0 | 1 | 54 |
+-----+-----+-----+
1 row in set (0.00 sec)

```

SECOND(param)

描述：返回时间参数的秒信息。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```

kingwow> select second('2015-10-10 12:00:00.1'),second(time '12:01:02'),second(current_time());
+-----+-----+-----+
| second('2015-10-10 12:00:00.1') | second(time '12:01:02') | second(current_time()) |
+-----+-----+-----+
| 0 | 2 | 2 |
+-----+-----+-----+
1 row in set (0.00 sec)

```

DAYS(param)

描述：返回时间参数从'0001-01-01'起的天数。

返回值：输入参数类型可以为 varchar、date 和 timestamp。

示例：

```
kingwow> select days('2015-10-10 12:00:00.1'),days(date '2015-10-11'),days(current_timestamp());
+-----+-----+-----+
| days('2015-10-10 12:00:00.1') | days(date '2015-10-11') | days(current_timestamp()) |
+-----+-----+-----+
| 736246 | 736247 | 739269 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

DATE_ADD(date, INTERVAL expr type) | ADDDATE(date, INTERVAL expr type)

描述：用于在日期上添加指定的时间间隔。

date:要添加时间间隔的原始日期,可以为 DATE、TIME、TIMESTAMP 类型的数值或数据表的列。

expr:整数类型的表达式, 是要添加的时间间隔的值。

type:时间间隔的单位, 可以是以下值之一: YEARS、MONTHS、DAYS、HOURS、MINUTES、SECONDS、MICROSECONDS。

返回值：返回参数 date 与表达式 expr 相加后的结果, 结果的类型格式与参数 date 一致。

ADDDATE()与 DATE_ADD()用法一致。

示例：

```
kingwow> select date_add(date '2015-10-11',interval 1 days), date_add(time '11:10:10',interval 1
hours);
+-----+-----+
| date_add(date '2015-10-11',interval 1 days) | date_add(time '11:10:10',interval 1 hours) |
+-----+-----+
| 2015-10-12 | 12:10:10 |
+-----+-----+
1 row in set (0.00 sec)
```

DATE_SUB(date,INTERVAL expr type) | SUBDATE(date, INTERVAL expr type)

描述：用于在日期上减去指定的时间间隔。

date:要减去时间间隔的原始日期,可以为 DATE、TIME、TIMESTAMP 类型的数值或数据表的列。

expr:整数类型的表达式, 是要减去的时间间隔的值。

type:时间间隔的单位, 可以是以下值之一: YEARS、MONTHS、DAYS、HOURS、MINUTES、SECONDS、MICROSECONDS。



返回值：返回参数 date 与表达式 expr 相加后的结果，结果的类型格式与参数 date 一致。
ADDDATE()与 DATE_ADD()用法一致。

示例：

```
kingwow>
select date_sub(date '2015-10-11',interval 1 days), date_sub(time '11:10:10',interval 1 hours);
+-----+-----+
| date_sub(date '2015-10-11',interval 1 days) | date_sub(time '11:10:10',interval 1 hours) |
+-----+-----+
| 2015-10-10                                | 10:10:10                                |
+-----+-----+
1 row in set (0.00 sec)
```

INTERVAL(N,N1,N2,N3,...)

描述：INTERVAL()函数为分区函数，其中，N 是要判断的数值，N1,N2,N3,...是分段的间隔。

这个函数的返回值是段的位置：

如果 $N < N1$ ，则返回 0，

如果 $N1 \leq N < N2$ ，则返回 1，

如果 $N2 \leq N < N3$ ，则返回 2。

区间是左闭右开的。

示例：

我们想要把 time 列的时间归类为 20220601-20220610 为一个区域，
20220611-20220620 为一个区域，20220621-20220630 为一个区域，每隔 10 天为一个分区。

```
drop table if exists value_test;
```

```
create table value_test(code int,time date);
```

```
insert into value_test
```

```
values(1,20220601),(1,20220603),(1,20220604),(1,20220609),(1,20220611),(1,20220615);
```

```
insert into value_test values(2,20220618),(2,20220622),(2,20220625),(2,20220629);
```

```
select INTERVAL(time,20220611,20220621,20220631) as time from value_test;
```

```
kingwow> drop table if exists value_test;
Query OK, 0 rows affected (0.790 sec)
[DDL]DROP_TABLE SUCCESS.
```

```
kingwow> create table value_test(code int,time date);
Query OK, 0 rows affected (0.750 sec)
[DDL]CREATE_TABLE SUCCESS.
```



```
kingwow> insert into value_test
values(1,20220601),(1,20220603),(1,20220604),(1,20220609),(1,20220611),(1,20220615);
Query OK, 6 rows affected (0.010 sec)
[DML]INSERT SUCCESS.

kingwow> insert into value_test values(2,20220618),(2,20220622),(2,20220625),(2,20220629);
Query OK, 4 rows affected (0.000 sec)
[DML]INSERT SUCCESS.

kingwow> select INTERVAL(time,20220611,20220621,20220631) as time from value_test;
+-----+
| time |
+-----+
| 0 |
| 1 |
| 0 |
| 2 |
| 0 |
| 2 |
| 0 |
| 2 |
| 1 |
| 1 |
+-----+
10 rows in set (0.000 sec)
```

分完区之后一般进行统计，就可以得到对应区间的个数了：

```
select INTERVAL(time,20220611,20220621,20220631) as times ,count(*) as count from
value_test GROUP BY times;
```

```
kingwow> select INTERVAL(time,20220611,20220621,20220631) as times ,count(*) as count from
value_test GROUP BY times;
+-----+-----+
| times | count |
+-----+-----+
| 0 | 4 |
| 1 | 3 |
| 2 | 3 |
+-----+-----+
3 rows in set (0.000 sec)
```

PASSWORD()

描述：PASSWORD() 函数用于计算和返回一个字符串的哈希值，这个哈希值是 MySQL 用于存储用户密码的格式。PASSWORD() 函数生成的哈希值是一个 40 字符的字符串，这个哈希值是使用 Kingwow 内部的算法生成的。

示例：

```
kingwow> select password('111');
+-----+
| password('111') |
+-----+
| 5d23692660185493bbf7b2cde2ef377858560aac |
+-----+
1 row in set (0.000 sec)
```

SEC_TO_TIME(seconds)

描述：用于将秒数转换为时间的函数。



seconds:待转换的秒数。

返回值：返回秒数转换后的时间值，格式为“HH:MM:SS”。

示例：

```
kingwow> SELECT SEC_TO_TIME('54925');
+-----+
| SEC_TO_TIME('54925') |
+-----+
| 15:15:25             |
+-----+
1 row in set (0.01 sec)
```

TIME_TO_SEC(time_str)

描述：用于将时间转换为秒数。

time_str:要转换的时间字符串，格式为 'HH:MM:SS'。

返回值：会返回一个对应的整数，表示将给定时间 time_str 转换为的秒数。

示例：

```
kingwow> SELECT TIME_TO_SEC('22:23:00');
+-----+
| TIME_TO_SEC('22:23:00') |
+-----+
|                80580 |
+-----+
1 row in set (0.05 sec)
```

TO_DATE(str,date_format)

描述：用于将字符串转换为日期格式。

str:要转换的字符串。

date_format:指定字符串的日期格式。

返回值：会以 date_format 格式返回 str 的日期值。

示例：

```
kingwow> select to_date('2020-01-08 15:08:49','yyyy-mm-dd hh24:mi:ss');
+-----+
| to_date('2020-01-08 15:08:49','yyyy-mm-dd hh24:mi:ss') |
+-----+
| 2020-01-08                                             |
+-----+
1 row in set (0.00 sec)
```

[to_date 支持以下格式：

```
select to_date('2020-01-08','yyyy-mm-dd') FROM dual;
select to_date('2020-01-08 15:08:49','yyyy-mm-dd hh24:mi:ss') FROM dual;
select to_date('01-jun-20','dd-mon-yy') FROM dual;
select to_date('01-jan-20','dd-mon-rr') FROM dual;
select to_date('01-08-20','dd-mm-yy') FROM dual;
select to_date('01-08-2020','dd-mm-yyyy') FROM dual;
select to_date('01-08-2020 15:08:49','dd-mm-yyyy hh24:mi:ss') FROM dual;
select to_date('01/08/2020 15:08:49','dd/mm/yyyy hh24:mi:ss') FROM dual;
```

```
select to_date('01-08-2020','mm-dd-yyyy') FROM dual;
select to_date('2020/01/08 12:08:49','yyyy/mm/dd hh:mi:ss') FROM dual;
select to_date('2020/01/08 15:08:49','yyyy/mm/dd hh24:mi:ss') FROM dual;
select to_date('20200108','yyyymmdd') FROM dual;
select to_date('20200108150849','yyyymmddhh24miss') FROM dual;
select to_date('20200108 150849','yyyymmdd hh24miss') FROM dual;
select to_date('01-oct-20 15.08.49.123456','dd-mon-yy hh.mm.ss.ssssss') FROM dual;
select to_date('2020-01-08,15:08:49','yyyy-mm-dd,hh24:mi:ss') FROM dual;
select to_date('20200108-15:08:49','yyyymmdd-hh24:mi:ss') FROM dual;
select to_date('20200108-15:08:49.123456','yyyymmdd-hh24:mi:ss.ff6') FROM dual;
select to_date('2020-01-08 15:08:49.123456','yyyy-mm-dd hh24:mi:ss.ff6') FROM dual;
select to_date('1208','hhmi') FROM dual;
select to_date('1208','hhmm') FROM dual;
select to_date('1508','hh24mi') FROM dual;
select to_date('15:08:49','hh24:mi:ss') FROM dual;
]
```

STR_TO_DATE(str,date_format)

描述：用于将字符串转换为日期格式。

str:要转换的字符串。

date_format:指定字符串的日期格式。

返回值：会以 date_format 格式返回 str 的日期值。

示例：

```
kingwow> select str_to_date('2003-01-02 10:11:12', '%Y-%m-%d %H:%i:%S');
+-----+
| str_to_date('2003-01-02 10:11:12', '%Y-%m-%d %H:%i:%S') |
+-----+
| 2003-01-02 10:11:12 |
+-----+
1 row in set (0.00 sec)
```

[str_to_date 支持以下格式：

```
select str_to_date('2003-01-02 10:11:12', '%Y-%m-%d %H:%i:%S');
select str_to_date('03-01-02 8:11:2.123456', '%y-%m-%d %H:%i:%S.##');
select str_to_date('0003-01-02 8:11:2.123456', '%Y-%m-%d %H:%i:%S.##');
select str_to_date('03-01-02 8:11:2.123456', '%Y-%m-%d %H:%i:%S.##');
select str_to_date('2003-01-02 10:11:12 PM', '%Y-%m-%d %h:%i:%S %p');
select str_to_date('2003-01-02 01:11:12.12345AM', '%Y-%m-%d %h:%i:%S.%f%p');
select str_to_date('2003-01-02 02:11:12.12345AM', '%Y-%m-%d %h:%i:%S.%f %p');
select str_to_date('2003-01-02 12:11:12.12345 am', '%Y-%m-%d %h:%i:%S.%f%p');
select str_to_date('2003-01-02 11:11:12Pm', '%Y-%m-%d %h:%i:%S%p');
select str_to_date('10:20:10', '%H:%i:%s');
select str_to_date('10:20:10', '%h:%i:%s.%f');
select str_to_date('10:20:10', '%T');
select str_to_date('10:20:10AM', '%h:%i:%s%p');
select str_to_date('10:20:10AM', '%r');
select str_to_date('10:20:10.44AM', '%h:%i:%s.%f%p');
select str_to_date('15-01-2001 12:59:58', '%d-%m-%Y %H:%i:%S');
select str_to_date('15 September 2001', '%d %M %Y');
select str_to_date('15 SEPTEMB 2001', '%d %M %Y');
select str_to_date('15 MAY 2001', '%d %b %Y');
select str_to_date('15th May 2001', '%D %b %Y');
select str_to_date('Sunday 15 MAY 2001', '%W %d %b %Y');
select str_to_date('Sund 15 MAY 2001', '%W %d %b %Y');
select str_to_date('Tuesday 00 2002', '%W %U %Y');
```



```

select str_to_date('Thursday 53 1998', '%W %u %Y');
select str_to_date('Sunday 01 2001', '%W %v %x');
select str_to_date('Tuesday 52 2001', '%W %V %X');
select str_to_date('060 2004', '%j %Y');
select str_to_date('4 53 1998', '%w %u %Y');
select str_to_date('15-01-2001', '%d-%m-%Y %H:%i:%S');
select str_to_date('15-01-20', '%d-%m-%y');
select str_to_date('15-2001-1', '%d-%Y-%c');

```

TO_TIMESTAMP(str,timestamp_format)

描述：用于将日期和时间转换为时间戳。

str:要转换的字符串。 timestamp_format:指定返回时间戳的格式。

返回值：会返回对应格式的时间戳信息。

示例：

```

kingwow> select to_timestamp('2020-01-08 15:08:49','yyyy-mm-dd hh24:mi:ss');
+-----+
| to_timestamp('2020-01-08 15:08:49','yyyy-mm-dd hh24:mi:ss') |
+-----+
|                                2020-01-08 15:08:49 |
+-----+
1 row in set (0.00 sec)

```

SYSDATE()

描述：函数返回当前日期。

补充：该函数不依赖于当前会话时区，而是依赖于数据库宿主机的 Linux 操作系统的时区。

返回值：会以"YYYY-MM-DD HH:MM:SS.SSSSSS"的格式返回对应的日期。

示例：

```

kingwow> select sysdate();
+-----+
| sysdate() |
+-----+
| 2020-03-13 09:47:12.946700 |
+-----+
1 row in set (0.00 sec)

```

SYSTIMESTAMP()

描述：函数返回当前日期。

补充：该函数不依赖于当前会话时区，而是依赖于数据库宿主机的 Linux 操作系统的时区。

返回值：会以"YYYY-MM-DD HH:MM:SS.SSSSSS"的格式返回对应的日期。

示例：

```

kingwow> select systimestamp();
+-----+
| systimestamp() |
+-----+

```

MONTH_BETWEEN(date1,date2)

描述：用于计算两个日期之间的月份数。

date1、date2：用于计算的两个时间日期。

返回值：会返回 date1 到 date2 之间的月份数，如果 date1 在日历中比 date2 晚，则返回一个正数，如果 date1 在日历中比 date2 早，则返回一个负数，如果 date1 和 date2 日期一样，则返回 0。

示例：

```
kingwow>
select months_between(to_date('2014-3-21','yyyy-mm-dd'),to_date('2014-1-10','yyyy-mm-dd'));
+-----+
| months_between(to_date('2014-3-21','yyyy-mm-dd'),to_date('2014-1-10','yyyy-mm-dd')) |
+-----+
|                                                                                      2.3548387096774195 |
+-----+
1 row in set (0.00 sec)
```

TIMESTAMPDIFF(unit, date1, date2)

描述：用于返回两个日期时间之间的整数差。

unit：指定返回的差值的单位，可以是以下值之一：YEARS、MONTHS、DAYS、HOURS、MINUTES、SECONDS、MICROSECONDS。

date1、date2：是要计算差值的两个日期或时间表达式。

返回值：返回计算得到的整数差。

示例：

```
kingwow>
select timestampdiff(1,TIMESTAMP'2021-02-01 12:12:12.111111'-TIMESTAMP'2021-02-01
12:12:00.000000');
+-----+
| timestampdiff(1, TIMESTAMP '2021-02-01 12:12:12.111111' - TIMESTAMP '2021-02-01 12:12:00.000000')
+-----+
|                                                                                      12111111 |
+-----+
1 row in set (0.00 sec)
```

TO_DATETIME(str,date_format)

描述：用于将字符串转换为日期格式。

str:要转换的字符串。

date_format:指定字符串的日期格式。

返回值：会以 date_format 格式返回 str 的日期值。

示例：

```
kingwow> select to_datetime ('2019-11-15 23:50:52','yyyy-mm-dd hh24:mi:ss');
+-----+
```



```

| to_datetime('2019-11-15 23:50:52','yyyy-mm-dd hh24:mi:ss') |
+-----+
| 2019-11-15 23:50:52 |
+-----+
1 row in set (0.00 sec)

```

UNIX_TIMESTAMP(date_expr)

描述：用于返回当前的时间戳。

date_expr: 要转换的日期和时间表达式。

返回值：会返回一个整数，表示当前的时间戳。

示例：

```

kingwow> select unix_timestamp('2022-05-17');
+-----+
| unix_timestamp('2022-05-17') |
+-----+
| 1652716800 |
+-----+
1 row in set (0.00 sec)

```

QUARTER(date)

描述：用于返回年份日期的季度值。

date: 要转换为季度的日期或时间戳。

返回值：会返回 date 对应的季度值，范围为 1~4。

示例：

```

kingwow> select quarter('2020-11-11'),quarter('2010-01-03'),quarter('1999-06-07');
+-----+
| quarter('2020-11-11') | quarter('2010-01-03') | quarter('1999-06-07') |
+-----+
| 4 | 1 | 2 |
+-----+
1 row in set (0.00 sec)

```

EXTRACT (fields FROM date str)

描述：从指定的时间字段或表达式中抽取年、月、日、时、分、秒等元素。

fields : 要抽取的元素名称。

str: 待提取的时间字符串。

返回值：会返回日期/时间的单独部分，比如年、月、日、小时、分钟等等。

示例：

```

kingwow> select extract (year from date '2015-05-04') year, extract (month from date '2015-06-04')
month, extract (day from date '2011-05-22') day from dual;
+-----+
| year | month | day |
+-----+
| 2015 | 6 | 22 |
+-----+
1 row in set (0.00 sec)

```

DATEDIFF(date1, date2)

描述：用于返回 date1 和 date2 之间的天数。

返回值：返回一个整数，表示 date1 到 date2 之间相差的天数。

示例：

```
kingwow> SELECT DATEDIFF('2015-06-19','1994-12-17');
+-----+
| DATEDIFF('2015-06-19','1994-12-17') |
+-----+
| 7489 |
+-----+

1 row in set (0.04 sec)
```

TIMEDIFF(date1, date2)

描述：用于返回 date1 和 date2 之间的天数。

返回值：返回一个 Time 类型的数值，表示 date1 到 date2 之间的时间差。

示例：

```
kingwow> SELECT TIMEDIFF('2015-06-06 12:12:12', '2014-06-05 11:11:11');
+-----+
| TIMEDIFF('2015-06-06 12:12:12', '2014-06-05 11:11:11') |
+-----+
| 838:59:59 |
+-----+

1 row in set (0.00 sec)

kingwow> SELECT TIMEDIFF('2015-06-06 12:12:12', '2015-06-05 11:11:11');
+-----+
| TIMEDIFF('2015-06-06 12:12:12', '2015-06-05 11:11:11') |
+-----+
| 25:01:01 |
+-----+

1 row in set (0.00 sec)
```

SECDIFF(time1, time2)

描述：用于返回 time1 和 time2 之间的秒差值。

返回值：返回一个整型数值，表示 time1 到 time2 之间的秒差。

示例：

```
kingwow> SELECT secdiff('11:11:11', '11:11:59');
+-----+
| secdiff('11:11:11', '11:11:59') |
+-----+
| 48 |
+-----+

1 row in set (0.04 sec)
```

WEEK(date)

描述：用于获取日期 date 中的周信息。

返回值：返回一个整数，表示 date 是该年中的第几周，范围是 0-53。

示例：

```
kingwow> SELECT WEEK('2022-05-23 15:30:00');
+-----+
| WEEK('2022-05-23 15:30:00') |
+-----+
| 21 |
+-----+
1 row in set (0.00 sec)
```

ADD_MONTHS (date,n)

描述：用于在日期上添加指定的月份。

返回值：会返回一个 date 值，表示添加指定月份后的时间值。

示例：

```
kingwow> SELECT '2024-01-20 14:25:44.386201',ADD_MONTHS('2024-01-20 14:25:44.386201',3) FROM DUAL;
+-----+-----+
| '2024-01-20 14:25:44.386201' | ADD_MONTHS('2024-01-20 14:25:44.386201',3) |
+-----+-----+
| 2024-01-20 14:25:44.386201 | 2024-04-20 14:25:44.386201 |
+-----+-----+
1 row in set (0.00 sec)
```

CURDATE()/CURTIME()

描述：返回当前日期，不含时间部分。

返回值：会以“YYYY-MM-DD”格式返回当前日期。

示例：

```
kingwow> SELECT CURDATE();
+-----+
| CURDATE() |
+-----+
| 2024-02-02 |
+-----+
1 row in set (0.00 sec)
```

EXTRACT (fields FROM datetime)

描述：从指定的时间字段或表达式中抽取年、月、日、时、分、秒等元素。

返回值：返回类型为 varchar。

补充: type 可取值为: MICROSECOND、SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER、YEAR、SECOND_MICROSECOND



MINUTE_MICROSECOND、MINUTE_SECOND、HOUR_MICROSECOND

HOUR_SECOND、HOUR_MINUTE、DAY_MICROSECOND、DAY_SECOND

DAY_MINUTE、DAY_HOUR、YEAR_MONTH。

示例：

```
kingwow> SELECT EXTRACT(MICROSECOND FROM '2011-11-11 11:11:11') ;
+-----+
| EXTRACT(MICROSECOND FROM '2011-11-11 11:11:11') |
+-----+
|                                0 |
+-----+
1 row in set (0.00 sec)
```

4.2.3. 算术函数

FLOOR(expr)

描述：对参数表达式 expr 的值向下取整。

expr: 要取整的表达式。

返回值：返回向下取整后的整数。

示例：

```
kingwow> select floor(-1.5),floor(3.14),floor(5/3),floor('0.618');
+-----+
| floor(-1.5) | floor(3.14) | floor(5/3) | floor('0.618') |
+-----+
|          -2 |           3 |           1 |              0 |
+-----+
1 row in set (0.00 sec)
```

CEIL|CEILING(expr)

描述：对参数表达式 expr 的值向上取整。

expr: 要取整的表达式。

返回值：返回向上取整后的整数。

示例：

```
kingwow> select ceil(-1.5),ceil(321),ceil(3.14),ceil(5/3),ceil('0.618');
+-----+
| ceil(-1.5) | ceil(321) | ceil(3.14) | ceil(5/3) | ceil('0.618') |
+-----+
|          -1 |        321 |           4 |           2 |              1 |
+-----+
1 row in set (0.00 sec)

kingwow> select ceiling(-1.5),ceiling(321),ceiling(3.14),ceiling(5/3),ceiling('0.618');
+-----+
| ceiling(-1.5) | ceiling(321) | ceiling(3.14) | ceiling(5/3) | ceiling('0.618') |
+-----+
|          -1 |        321 |           4 |           2 |              1 |
+-----+
1 row in set (0.00 sec)
```

MOD(N, M)

描述：用于返回 N 除以 M 后的余数。

N: 被除数。

M: 除数。

返回值：返回类型与数值优先级高的参数的数据类型相同，M 不能为 0。

示例：

```
kingwow> select mod(10,3);
+-----+
| mod(10,3) |
+-----+
|          1 |
+-----+
1 row in set (0.00 sec)
```

RAND()

描述：用于生成一个介于 0-1 之间的随机浮点数。

返回值：返回一个介于 0-1 的随机浮点数。

示例：

```
kingwow> SELECT RAND( ), RAND( );
+-----+-----+
| RAND( ) | RAND( ) |
+-----+-----+
| 0.155819 | 0.0965274 |
+-----+-----+
1 row in set (0.00 sec)
```

ROUND(expr[, digits])

描述：用于将数值四舍五入到指定的小数位数。

expr:待舍入的数值或表达式。

digits: 要保留的小数位数，不指定时默认和 expr 结果保持一致。

返回值：返回四舍五入后的结果，参数 digits 必须为整数，可以大于 0、等于 0 或者小于 0。

示例：

```
kingwow> select round(3.1415,2),round(123.5,0),round(-123.4,-2),round(5.0/3,3);
+-----+-----+-----+-----+
| round(3.1415,2) | round(123.5,0) | round(-123.4,-2) | round(5.0/3,3) |
+-----+-----+-----+-----+
|          3.14 |          124 |          -100 |          1.667 |
+-----+-----+-----+-----+
1 row in set (0.05 sec)
```

ABS(X)

描述：用于返回 X 的绝对值。

expr:待舍入的数值或表达式。



digits: 要保留的小数位数, 不指定时默认和 expr 结果保持一致。

返回值: 返回绝对值结果。

示例:

```
kingwow> select abs(3-5);
+-----+
| abs(3-5) |
+-----+
| 2        |
+-----+
1 row in set (0.00 sec)
```

PI()

描述: 用于获取圆周率 π 的值。

返回值: 返回 π 的值。

示例:

```
kingwow> select pi();
+-----+
| pi()          |
+-----+
| 3.141592653589793 |
+-----+
1 row in set (0.00 sec)
```

CRC32()

描述: 计算一个字符串的 32 位 CRC(循环冗余校验)。

返回值: 返回值的范围是 0-4294967296 (2 的 32 次方减 1)。

示例:

```
kingwow> select crc32(1),crc32(2);
+-----+-----+
| crc32(1) | crc32(2) |
+-----+-----+
| 2212294583 | 450215437 |
+-----+-----+
1 row in set (0.00 sec)
```

SQRT(x)

描述: 用于非负数 x 的二次方根。

返回值: 返回二次方根计算结果, 负数没有平方根。

示例:

```
kingwow> select SQRT(16),SQRT(0),SQRT(7225);
+-----+-----+-----+
| SQRT(16) | SQRT(0) | SQRT(7225) |
+-----+-----+-----+
| 4        | 0       | 85         |
+-----+-----+-----+
```

EXP(x)

描述：函数返回 e 的 x 次幂（ e 为数学常量， $e = 2.71828183\cdots$ ）。

返回值：返回值类型与参数 x 的数据类型一致。

示例：

```
kingwow> SELECT EXP(1),EXP(3),EXP(10);
+-----+-----+-----+
| EXP(1) | EXP(3) | EXP(10) |
+-----+-----+-----+
| 2.718281828459045 | 20.085536923187668 | 22026.465794806718 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

SIGN(x)

描述：返回数字 x 的符号，大于 0 返回 1，小于 0 返回 -1，等于 0 返回 0。

返回值：返回值为 0、1 或 -1。

示例：

```
kingwow> SELECT SIGN(-4.65),SIGN(0),SIGN(4.65);
+-----+-----+-----+
| SIGN(-4.65) | SIGN(0) | SIGN(4.65) |
+-----+-----+-----+
| -1 | 0 | 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

LN(x)

描述：用于计算一个数的自然对数（以 e 为底）。

返回值： x 必须大于 0，否则将返回 NULL。

示例：

```
kingwow> select ln(2.718281828459045);
+-----+
| ln(2.718281828459045) |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

LOG(x,y)

描述：返回以 x 为底的 y 的对数， x 和 y 都必须大于 0。

返回值：不设置 x 时会返回 y 的自然对数。

示例：

```
kingwow> SELECT LOG(45),LOG(45,10);
+-----+-----+
| LOG(45) | LOG(45,10) |
+-----+-----+
```

```

| LOG(45) | LOG(45,10) |
+-----+-----+
| 3.8066624897703196 | 0.6048829123101417 |
+-----+-----+
1 row in set (0.00 sec)

```

LOG10(x)

描述：用于计算一个数以 10 为底的对数。

返回值：x 必须大于 0，否则返回 NULL。

示例：

```

kingwow> SELECT LOG10(100),LOG10(-1);
+-----+-----+
| LOG10(100) | LOG10(-1) |
+-----+-----+
|          2 |        NULL |
+-----+-----+
1 row in set (0.01 sec)

```

LOG2(x)

描述：用于计算以 2 为底的特定数字的自然对数。

返回值：x 必须大于 0，否则返回 NULL。

示例：

```

kingwow> SELECT LOG2(16),LOG2(0);
+-----+-----+
| LOG2(16) | LOG2(0) |
+-----+-----+
|          4 |        NULL |
+-----+-----+
1 row in set (0.00 sec)

```

POWER(x,y) | POW(x,y)

描述：用于计算 x 的 y 次方。

返回值：返回计算结果，当参数是非数值类型时，返回 NULL。

示例：

```

kingwow> SELECT POWER(3,4),pow(5,3),pow(NULL,2);
+-----+-----+-----+
| POWER(3,4) | pow(5,3) | pow(NULL,2) |
+-----+-----+-----+
|          81 |        125 |          NULL |
+-----+-----+-----+
1 row in set (0.05 sec)

```

RADIANS(x)

描述：用于返回 x 的弧度值，从度转换为弧度。

返回值：返回弧度值。

示例：



```
kingwow> SELECT RADIANS(90),RADIANS(0),RADIANS(-1),RADIANS(1);
```

```
+-----+-----+-----+-----+
| RADIANS(90) | RADIANS(0) | RADIANS(-1) | RADIANS(1) |
+-----+-----+-----+-----+
| 1.5707963267948966 | 0 | -0.017453292519943295 | 0.017453292519943295 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

SIN(x)

描述：用于计算正弦值。

返回值：返回 x 的正弦值。

示例：

```
kingwow> SELECT SIN(90);
```

```
+-----+
| SIN(90) |
+-----+
| 0.8939966636005579 |
+-----+
1 row in set (0.00 sec)
```

TAN(x)

描述：用于计算正切值。

返回值：返回 x 的正切值。

示例：

```
kingwow> SELECT TAN(1);
```

```
+-----+
| TAN(1) |
+-----+
| 1.5574077246549023 |
+-----+
1 row in set (0.00 sec)
```

TRUNCATE(x,d)

描述：用于数据截取。

x 表示需要处理的数字。

d 表示需要截取的位数。

返回值：如果 d 为零，则返回的数字不含小数。d 也可以是负数，这样会把整数的部分置零。

示例：

```
kingwow> SELECT truncate(199.234,2),truncate(122,-2);
```

```
+-----+-----+
| truncate(199.234,2) | truncate(122,-2) |
+-----+-----+
| 199.23 | 100 |
+-----+-----+
1 row in set (0.00 sec)
```

ACOS(x)

描述：反余弦函数。

返回值：x 值的范围必须在 -1 和 1 之间，否则返回 NULL。

示例：

```
kingwow> SELECT ACOS(2),ACOS(1),ACOS(-1);
+-----+-----+-----+
| ACOS(2) | ACOS(1) | ACOS(-1) |
+-----+-----+-----+
| NULL   | 0       | 3.141592653589793 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

ASIN(x)

描述：反正弦函数。

返回值：x 值的范围必须在 -1 和 1 之间，否则返回 NULL。

示例：

```
kingwow> SELECT ASIN(1);
+-----+
| ASIN(1) |
+-----+
| 1.5707963267948966 |
+-----+
1 row in set (0.00 sec)
```

ATAN(x)

描述：反正切函数。

返回值：x 值的范围必须在 -1 和 1 之间，否则返回 NULL。

示例：

```
kingwow> SELECT ATAN(1);
+-----+
| ATAN(1) |
+-----+
| 0.7853981633974483 |
+-----+
1 row in set (0.00 sec)
```

ATAN2(y,x)

描述：用于返回 x 和 y 的反正切，它类似于 Y/X 的反正切，所不同的是两者的符号用于查找结果的象限。

返回值：返回计算结果值。

示例：

```
kingwow> SELECT ATAN2(3,6);
+-----+
| ATAN2(3,6) |
+-----+
```



```
+-----+
| 0.4636476090008061 |
+-----+
1 row in set (0.00 sec)
```

COS(x)

描述：余弦函数。

返回值：该函数返回 x，x 值是以弧度给出的余弦值。

示例：

```
kingwow> SELECT COS(90);
+-----+
| COS(90) |
+-----+
| -0.4480736161291701 |
+-----+
1 row in set (0.00 sec)
```

COT((x)

描述：余切函数。

返回值：该函数返回 x 的余切值。

示例：

```
kingwow> SELECT COT(1);
+-----+
| COT(1) |
+-----+
| 0.6420926159343306 |
+-----+
1 row in set (0.01 sec)
```

DEGREES(x)

描述：用于将弧度转换为角度。

返回值：返回对应弧度的角度值。

示例：

```
kingwow> SELECT DEGREES(PI());
+-----+
| DEGREES(PI()) |
+-----+
| 180 |
+-----+
1 row in set (0.00 sec)
```

FORMAT(x,d)

描述：用于帮助格式化给定的数字格式，会格式化给定的数字，将其四舍五入到某个小数位，然后以字符串形式返回该数字。

x: 要格式化的数字。

d: 四舍五入到的小数位数。



返回值： 返回对应格式的数字。

示例：

```
kingwow> SELECT FORMAT(423423234.65434453,2);
+-----+
| FORMAT(423423234.65434453,2) |
+-----+
| 423,423,234.65 |
+-----+
1 row in set (0.00 sec)
```

DEC(expr, n1, n2) | DECIMAL(expr, n1, n2)

描述： 用于将 expr 转成 decimal 类型。

expr: 原表达式。

n1: 待转的 decimal 类型的位数。

n2: 待转的 decimal 类型的精度。

返回值： 返回对应 decimal 类型数据。等价于 cast(expr as decimal(p,s))的使用。

示例：

```
kingwow> select dec('123.45',8,4),dec('123.45',5,1);
+-----+-----+
| dec('123.45',8,4) | dec('123.45',5,1) |
+-----+-----+
| 123.4500 | 123.5 |
+-----+-----+
1 row in set (0.00 sec)
```

BIN(n)

描述： 用于将十进制数转换为等效的二进制数。

n: 一个长整型(BIGINT)数。

返回值： 返回一个二进制数，等同于 CONV(N,10,2)的使用。如果 n 为 NULL，则返回 NULL。

示例：

```
kingwow> SELECT BIN(12);
+-----+
| BIN(12) |
+-----+
| 1100 |
+-----+
1 row in set (0.00 sec)
```

TRUNC(number[,num_digits])

描述： 用于截断数值或日期/时间值。

number: 需要截尾取整的数字。

num_digits: 用于指定取整精度的数字,不指定时默认值为 0。

返回值： 截取时不进行四舍五入，会直接截取。如果 num_digits 为正数，则截取小数点后 num_digits 位；如果为负数，则先保留整数部分，然后从个位开始向前数，并将遇到的数字都变为 0。

示例：

```
kingwow> select trunc(123.458,1),trunc(123.458,-1);
```

```
+-----+
| trunc(123.458,1) | trunc(123.458,-1) |
+-----+
|          123.4 |          120 |
+-----+
1 row in set (0.00 sec)
```

TRUNC()函数针对日期，如：

--返回当年第一天

```
kingwow> select trunc(to_timestamp('2018-02-01 1:12:12','yyyy-mm-dd hh24:mi:ss'),'yyyy') ;
+-----+
| trunc(to_timestamp('2018-02-01 1:12:12','yyyy-mm-dd hh24:mi:ss'),'yyyy') |
+-----+
| 2018-01-01 |
+-----+
1 row in set (0.00 sec)
```

4.2.4. 聚合函数

AVG(expr)

描述：返回数值列的平均值。

expr: 数值类型或者可以转换成数值类型的表达式

返回值：返回类型与参数 expr 的数据类型相同，空值将被忽略。

示例：

```
create table Products(Price decimal(10,3));
insert into Products values(1.23),(3.45),(7.789);
SELECT AVG(Price) AS AveragePrice FROM Products;
kingwow> SELECT AVG(Price) AS AveragePrice FROM Products;
+-----+
| AveragePrice |
+-----+
|    4.1563333 |
+-----+
1 row in set (0.00 sec)
```

COUNT (expr)

描述：用于查询 expr 的行数。

expr: 数值类型或者可以转换成数值类型的表达式

返回值：返回组中表达式 expr 值的计数，空值将被忽略，但 COUNT(*)不会忽略空值。

示例：

```
create table Prod(Price decimal(10,3));
insert into Prod values(1.23),(3.45),(7.789),(2.333);
SELECT COUNT(Price) AS NumberOfProducts FROM Prod;
kingwow> SELECT COUNT(Price) AS NumberOfProducts FROM Prod;
+-----+
| NumberOfProducts |
+-----+
|          4 |
+-----+
1 row in set (0.00 sec)
```

MAX (expr)

描述：用于查询 expr 的最大值。

expr: 数值类型或者可以转换成数值类型的表达式

返回值：返回组中表达式 expr 的最大值，空值将被忽略。

示例：

```
create table test (c1 int,c2 int);
insert into test values(1,23),(3,45),(7,789);
SELECT MAX(c1) FROM test ;
kingwow> SELECT MAX(c1) FROM test ;
+-----+
| MAX(c1) |
+-----+
|      7 |
+-----+
1 row in set (0.00 sec)
```

MIN (expr)

描述：用于查询 expr 的最小值。

expr: 数值类型或者可以转换成数值类型的表达式

返回值：返回组中表达式 expr 的最小值，空值将被忽略。

示例：

```
create table test (c1 int,c2 int);
insert into test values(1,23),(3,45),(7,789);
SELECT MIN(c1) FROM test ;
kingwow> SELECT MIN(c1) FROM test ;
+-----+
| MIN(c1) |
+-----+
|      1 |
+-----+
1 row in set (0.01 sec)
```

SUM(expr)

描述：用于查询 expr 的和。

expr: 数值类型或者可以转换成数值类型的表达式。

返回值：返回组中表达式 expr 的和，空值将被忽略。

示例：

```
create table test (c1 int,c2 int);
insert into test values(1,23),(3,45),(7,789);
kingwow> select sum(c1) from test;
+-----+
| sum(c1) |
+-----+
|      11 |
+-----+
1 row in set (0.01 sec)
```

WM_CONCAT(column_name)

描述：实现行转列功能，即将查询出的某一列值使用逗号进行隔开拼接，成为一条数据。

column_name：要连接的列的名称。

返回值：返回一个由多个值组成的由逗号分隔的字符串。

示例：

```
drop table if exists t1;
create table t1(id int, c1 int, c2 varchar, c3 varchar);
insert into t1 values(1, 1, 'row1', '2012-12-12 11:11:11');
insert into t1 values(2, 1, '', '2012-12-12 12:13:11');
insert into t1 values(3, 1, 'row3', '2012-12-12 21:11:16');
insert into t1 values(4, 2, 'row4', '2012-12-12 11:21:13');
insert into t1 values(5, 2, 'row5', '2012-12-14 13:11:41');
insert into t1 values(6, 3, 'row6', '2012-12-11 11:13:11');

insert into t1 values(7, 4, 'row7', '2012-02-12 01:11:11');
insert into t1 values(8, 5, 'row8', '2012-12-12 15:16:11');
insert into t1 values(9, 6, 'row9', '2012-02-12 11:19:11');
insert into t1 values(10, 6, 'row10', '2012-12-12 11:11:11');
select wm_concat(c1) as a, wm_concat(c2) as b, wm_concat(c3) as c from t1;
select wm_concat(c1) as a, wm_concat(c2) as b, wm_concat(c3) as c from t1 group by c1;
select wm_concat(id) as a, wm_concat(c2) as b, wm_concat(c3) as c from t1 group by c1;
select wm_concat(to_char(c2)) from t1 group by c1;
select wm_concat(to_char(c1)) from t1 group by c1;
```

```
kingwow> select * from t1;
+-----+-----+-----+
| id | c1 | c2 | c3 |
+-----+-----+-----+
| 1 | 1 | row1 | 2012-12-12 11:11:11 |
| 2 | 1 |  | 2012-12-12 12:13:11 |
| 3 | 1 | row3 | 2012-12-12 21:11:16 |
| 4 | 2 | row4 | 2012-12-12 11:21:13 |
| 5 | 2 | row5 | 2012-12-14 13:11:41 |
| 6 | 3 | row6 | 2012-12-11 11:13:11 |
| 7 | 4 | row7 | 2012-02-12 01:11:11 |
| 8 | 5 | row8 | 2012-12-12 15:16:11 |
| 9 | 6 | row9 | 2012-02-12 11:19:11 |
| 10 | 6 | row10 | 2012-12-12 11:11:11 |
+-----+-----+-----+
10 rows in set (0.00 sec)
```

```
kingwow> select wm_concat(c1) as a, wm_concat(c2) as b, wm_concat(c3) as c from t1 group by c1;
+-----+-----+-----+
| a | b | c |
+-----+-----+-----+
| 1,1,1 | row1,row3 | 2012-12-12 11:11:11,2012-12-12 12:13:11,2012-12-12 21:11:16 |
| 2,2 | row4,row5 | 2012-12-12 11:21:13,2012-12-14 13:11:41 |
| 3 | row6 | 2012-12-11 11:13:11 |
| 4 | row7 | 2012-02-12 01:11:11 |
| 5 | row8 | 2012-12-12 15:16:11 |
| 6,6 | row9,row10 | 2012-02-12 11:19:11,2012-12-12 11:11:11 |
+-----+-----+-----+
6 rows in set (0.01 sec)
```

4.2.5. 空值处理函数

COALESCE(expr, expr, expr, ...)

描述：用于返回其参数中的第一个非 NULL 值。如果所有参数都是 NULL，则返回 NULL。



expr, expr, expr, ...: 要检查的值列表。

返回值: 返回非空值, 当所有都为 NULL 时, 返回 NULL。

示例:

```
kingwow> select COALESCE(null,null,3,4,5),COALESCE(null,null,null);
+-----+-----+
| COALESCE(null,null,3,4,5) | COALESCE(null,null,null) |
+-----+-----+
| 3 | NULL |
+-----+-----+
1 row in set (0.00 sec)
```

NVL(expr1,expr2)

描述: 用于从两个表达式返回一个非 NULL 值。如果 expr1 为 NULL, 则函数返回 expr2, 否则返回 expr1 本身。

expr1, expr2: 待检查的表达式, 可以是任何 Kingwow 内建的数据类型。

返回值: 返回参数中非 NULL 的值, 如果参数都为 NULL 则返回 NULL。

示例:

```
kingwow> select nvl(null,123),nvl('12',123),nvl(null,null);
+-----+-----+-----+
| nvl(null,123) | nvl('12',123) | nvl(null,null) |
+-----+-----+-----+
| 123 | 12 | NULL |
+-----+-----+-----+
1 row in set (0.00 sec)
```

NVL2(expr1,expr2,expr3)

描述: 会根据表达式是否为空, 返回不同的值。如果 expr1 不为空, 则返回 expr2 的值, 如果 expr1 为空, 则返回 expr3 的值。expr2 和 expr3 类型不同的话, expr3 会转换为 expr1 的类型。

expr1, expr2,expr3 : 待检查的表达式, 可以是任何 Kingwow 内建的数据类型。

返回值: 按函数规则, 返回参数中非 NULL 的值, 如果参数都为 NULL 则返回 NULL。

示例:

```
kingwow> select nvl2(null,123,321),nvl2('12',123,321),nvl2(null,null,null);
+-----+-----+-----+
| nvl2(null,123,321) | nvl2('12',123,321) | nvl2(null,null,null) |
+-----+-----+-----+
| 321 | 123 | NULL |
+-----+-----+-----+
1 row in set (0.00 sec)
```

4.2.6. 条件表达式函数

VALUE(expr1, expr2)

描述: 用于从两个表达式返回一个非 NULL 值。如果 expr1 为 NULL, 则函数返回 expr2,



否则返回 expr1 本身。

expr1, expr2, : 待检查的表达式, 可以是任何 Kingwow 内建的数据类型。

返回值: 返回参数中非 NULL 的值, 如果参数都为 NULL 则返回 NULL。

示例:

```
kingwow> select value(null,123),value(34,123);
+-----+-----+
| value(null,123) | value(34,123) |
+-----+-----+
|          123   |          34   |
+-----+-----+
1 row in set (0.00 sec)
```

ENCODE(str,pswd_str)

描述: ENCODE 函数可以使用加密密码 pswd_str 来加密字符串 str, 加密结果是二进制数, 需要使用 BLOB 类型的字段保存。该函数与 DECODE 是一对, 需要同样的密码才能够解密。

示例:

```
kingwow> SELECT ENCODE('123','xxoo');
+-----+
| ENCODE('123','xxoo') |
+-----+
| ;vx                   |
+-----+
1 row in set (0.010 sec)

kingwow> SELECT DECODE(';vx','xxoo');
+-----+
| DECODE(';vx','xxoo') |
+-----+
| 123                   |
+-----+
1 row in set (0.000 sec)
```

DECODE(expr, if1, then1, [if2, then2, ... ,] [,default])

描述: 用于执行条件逻辑。它类似于其他编程语言中的 switch-case 语句。所有的参数都可以是表达式, 最多支持 255 个参数。

expr: 要进行条件判断的表达式或列。

if1, then1,...: if1 表示要和 expr 比较的值, then1 表示和 if1 和 expr 匹配上后返回的值。

default: 如果前面 if 都没有匹配上, 则默认返回的结果。

返回值: 返回 if 项中和 expr 匹配上后对应的 then 值, 如果没有能匹配上的返回 default 的值。

示例:

```
kingwow> select decode(year(now()),2016,'Last Year',2017,'This Year','Long long ago') as now,
-> decode(year(now())-1,2016,'Last Year',2017,'This Year','Long long ago') as `last`,
-> decode(year(now())-10,2016,'Last Year',2017,'This Year','Long long ago') as ago
-> ;
+-----+-----+-----+
| now   | last  | ago    |
+-----+-----+-----+
```

GREATEST(expr1,expr2,...expr_n)

描述: 用于返回一个或多个表达式列表中的最大值。会从表达式 (列、常量、计算值 expr_1, expr_2, ... expr_n 等) 中找出最大的数返回。在比较时, 会自动按表达式的数据类型进行比较, 以 expr_1 的数据类型为准。

返回值: 以第一个参数类型为准, 返回对应类型的最大值, 如果参数列表有 NULL, 则直接返回 NULL 值。

示例:

```
kingwow> SELECT GREATEST(2, 5, 12, 3, 16, 8, 9) ;
+-----+
| GREATEST(2, 5, 12, 3, 16, 8, 9) |
+-----+
| 16                                |
+-----+
1 row in set (0.01 sec)
```

```
kingwow> SELECT GREATEST('A', 'B', 'C', 'D', 'E', 'GA', 'GAB');
+-----+
| GREATEST('A', 'B', 'C', 'D', 'E', 'GA', 'GAB') |
+-----+
| GAB                                             |
+-----+
1 row in set (0.00 sec)
```

```
kingwow>SELECT GREATEST(2, 5, 12, 3, 16, 8, NULL) ;
+-----+
| GREATEST(2, 5, 12, 3, 16, 8, NULL) |
+-----+
|                                NULL |
+-----+
1 row in set (0.00 sec)
```

LEAST(expr1,expr2,... exprn)

描述: 用于返回一个或多个表达式列表中的最小值。会从表达式 (列、常量、计算值 expr_1, expr_2, ... expr_n 等) 中找出最小的数返回。在比较时, 会自动按表达式的数据类型进行比较, 以 expr_1 的数据类型为准。

返回值: 以第一个参数类型为准, 返回对应类型的最小值, 如果参数列表有 NULL, 则直接返回 NULL 值。

示例:

```
kingwow> SELECT LEAST(10, 20, 3), LEAST(10, null , 30);
+-----+-----+
| LEAST(10, 20, 3) | LEAST(10, null , 30) |
+-----+-----+
| 3                | NULL                  |
+-----+-----+
1 row in set (0.00 sec)
```

NULLIF(expr1, expr2)

描述：如果 $\text{expr1} = \text{expr2}$ 成立，那么返回值为 NULL，否则返回值为 expr1 。

返回值：会根据比较规则返回 expr1 的值或 NULL 值。

示例：

```
kingwow> select nullif(1,1),nullif(1,2);
+-----+-----+
| nullif(1,1) | nullif(1,2) |
+-----+-----+
|          NULL          |          1          |
+-----+-----+
1 row in set (0.00 sec)
```

IFNULL(expr1, expr2)

描述：用于判断第一个表达式是否为 NULL，如果为 NULL 则返回第二个参数的值，如果不为 NULL 则返回第一个参数的值。

返回值：会根据比较规则得出的结果，返回 expr 的值或 NULL 值。

示例：

```
kingwow> select ifnull(null,1),ifnull(3,2);
+-----+-----+
| ifnull(null,1) | ifnull(3,2) |
+-----+-----+
| 1              | 3           |
+-----+-----+
1 row in set (0.00 sec)
```

IF(condition, true_value, false_value)

描述：一个条件函数，用于在查询中执行条件逻辑。

condition：要检查的条件。

true_value：如果条件为真，则返回的值。

false_value：如果条件为假，则返回的值。

返回值：返回条件判断后对应的值。根据条件的结果为 true 或 false，返回第一个参数，或第二个参数。

示例：

```
kingwow> SELECT IF(500<1000, 5, 10);
+-----+
| IF(500<1000, 5, 10) |
+-----+
| 5                     |
+-----+

1 row in set (0.00 sec)
```

```
kingwow> SELECT IF(2>5, 'YES', 'NO');
+-----+
| IF(2>5, 'YES', 'NO') |
+-----+
```

```
+-----+
| NO          |
+-----+
1 row in set (0.00 sec)
```

ISNULL(expr)

描述：用于判断 expr 是否为 NULL，如果为 NULL 则返回 1，非空返回 0。

返回值：根据对参数 expr 的判空结果，返回 0 或 1。

示例：

```
kingwow> select isnull(1),isnull(null);
+-----+-----+
| isnull(1) | isnull(null) |
+-----+-----+
| 0         | 1           |
+-----+-----+
1 row in set (0.00 sec)
```

CASE WHEN

描述：可以实现类似“IF…ELSE…THEN”的逻辑。

返回值：CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1，如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。

示例：

```
create table test(id int primary key, name varchar(10),sex bool,`level` int,weight int);
insert into test values(1,'xiaohong',1,1,50);
insert into test values(2,'xiaoming',0,1,90);
insert into test values(3,'xiaohuang',1,1,80);
insert into test values(4,'xiaoming',0,1,null);

kingwow>select * from test;
+-----+-----+-----+-----+-----+
| id  | name    | sex  | level | weight |
+-----+-----+-----+-----+-----+
| 1   | xiaohong | 1    | 1     | 50     |
| 2   | xiaoming | 0    | 1     | 90     |
| 3   | xiaohuang | 1    | 1     | 80     |
| 4   | xiaoming | 0    | 1     | NULL   |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

kingwow>select t.name,(case t.sex when 0 then 'woman' when 1 then 'man' else 'unknown' end) sex
from test t;
+-----+-----+
| name    | sex  |
+-----+-----+
| xiaohong | man  |
| xiaoming | woman |
| xiaohuang | man  |
| xiaoming | woman |
+-----+-----+
4 rows in set (0.01 sec)

kingwow>select t.name,(case when t.sex=0 then 'woman' when t.sex=1 then 'man' else 'unknown' end)
sex from test t;
```



```

+-----+-----+
| name   | sex   |
+-----+-----+
| xiaohong | man   |
| xiaoming | woman |
| xiaohuang | man   |
| xiaoming | woman |
+-----+-----+
4 rows in set (0.00 sec)

```

4.2.7. 窗口函数

RANK() OVER ([PARTITION BY expr_list] [ORDER BY order_list])

描述：排名函数，会基于 OVER 子句中的 ORDER BY 表达式确定一组值的排名。当有相同排序值时，将会有相同的排名，并且值相同的行数会被记录到下个排名中。

OVER：窗口函数中特有，窗口函数中会使用 over 子句定义窗口进行计算。

PARTITION BY expr_list：指定开窗口的列，expr_list 是数值类型或者可以转换成数值类型的类型。

ORDER BY order_list：指定排名依据的值，order_list 是定义排名值参考的数据列。

返回值：会返回结果集的分区内每行的排名，行的排名是相关行之前的排名数加一。简单来说 rank 函数就是对查询出来的记录进行排名。

示例：

```

create table student(id int,name varchar(20),score int,course int);
insert into student values(1,'dock',60,1);
insert into student values(2,'bob',90,2);
insert into student values(3,'clark',80,3);
insert into student values(4,'elic',100,4);
insert into student values(5,'alice',70,5);
insert into student values(6,'jacky',70,1);
insert into student values(7,'hill',50,2);
insert into student values(8,'frank',70,3);
insert into student values(9,'iris',60,4);
insert into student values(10,'grace',80,5);
select * from student;
kingwow> select name,course,rank() over(partition by course order by score desc) rank from student;
+-----+-----+
| name | course | rank |
+-----+-----+
| jacky | 1 | 1 |
| dock | 1 | 2 |
| bob | 2 | 1 |
| hill | 2 | 2 |
| clark | 3 | 1 |
| frank | 3 | 2 |
| elic | 4 | 1 |
| iris | 4 | 2 |
| grace | 5 | 1 |
| alice | 5 | 2 |
+-----+-----+
10 rows in set (0.02 sec)

```

DENSE_RANK() OVER([PARTITION BY expr_list] [ORDER BY order_list])

描述：排名函数，与 RANK 函数类似，但 DENSE_RANK 在处理并列排名时不会跳过任何数



字，即在为分区或结果集中每一行分配排名时排名值是没有间隙的。

OVER：窗口函数中特有，窗口函数中会使用 over 子句定义窗口进行计算。

PARTITION BY expr_list：指定开窗口的列，expr_list 是数值类型或者可以转换成数值类型的类型。

ORDER BY order_list：指定排名依据的值，order_list 是定义排名值参考的数据列。

返回值：会返回结果集中指定的分区内每行的排名结果。

示例：

```
create table t1(id int, val int);
insert into t1 values(4,1),(2,2),(9,4);
select DENSE_RANK() OVER(order by id) from t1;
kingwow> select DENSE_RANK() OVER(order by id) from t1;
+-----+
| DENSE_RANK() OVER(order by id) |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.00 sec)
```

PIVOT(AGGREGATE_FUNCTION(column_to_aggregate) FOR
column_to_be_pivoted IN (value1, value2, ... valueN))

描述：用于将行数据转换为列数据。

AGGREGATE_FUNCTION：用于聚合数据的函数，例如 SUM、AVG、COUNT 等。

column_to_aggregate：要聚合的列。

column_to_be_pivoted：要转换为列标题的列。

value1, value2, ... valueN：column_to_be_pivoted 列中的唯一值，它们将转换为列标题。

返回值：将某个列中的唯一值转换为列标题，并将相关行数据聚合到相应的列中。配合 select 使用可查看对应转换后的数据。

示例：

```
create table t2
(
  id int,
  sell_no int,
  name varchar(50),
  sell_count int
);
insert into t2 values(1, 1, 'a', 1);
insert into t2 values(1, 1, 'a', 2);
insert into t2 values(1, 2, 'b', 3);

insert into t2 values(1, 2, 'c', 4);
insert into t2 values(1, 3, 'd', 5);
insert into t2 values(1, 3, 'c', 6);
insert into t2 values(1, 100, 'd', 4);

kingwow> select * from t2;
+-----+-----+-----+-----+
| id | sell_no | name | sell_count |
+-----+-----+-----+-----+
| 1 | 1 | a | 1 |
| 1 | 1 | a | 2 |
```



```

| 1 | 2 | b | 3 |
| 1 | 2 | c | 4 |
| 1 | 3 | d | 5 |
| 1 | 3 | c | 6 |
| 1 | 100 | d | 4 |
+-----+
7 rows in set (0.00 sec)

```

```

kingwow>
select *from (select id, a.name, a.sell_count, sell_no from t2 a ) pivot(sum(sell_count) for sell_no
in(1 no1, 2 no2, 3 no3, 4 no4, 5 no5, 6 no6, 7 no7));
+-----+
| id | name | no1 | no2 | no3 | no4 | no5 | no6 | no7 |
+-----+
| 1 | a | 3 | NULL | NULL | NULL | NULL | NULL | NULL |
| 1 | b | NULL | 3 | NULL | NULL | NULL | NULL | NULL |
| 1 | c | NULL | 4 | 6 | NULL | NULL | NULL | NULL |
| 1 | d | NULL | NULL | 5 | NULL | NULL | NULL | NULL |
+-----+
4 rows in set (0.00 sec)

```

ROW_NUMBER() | ROWNUMBER() OVER ([PARTITION BY expr_list] ORDER BY order_list [ASC | DESC])

描述：对查询语句结果集中的每一条记录按分组排序指定一个行号，行号从 1 开始，常与子查询结合使用以实现分页功能。

OVER：窗口函数中特有，窗口函数中会使用 over 子句定义窗口进行计算。

PARTITION BY：用于将数据分成不同的分区，然后在每个分区内进行排序和计算。

ORDER BY：用于指定在每个分区内如何排序数据。

ASC | DESC：用于指定 order by 中排序使用升序还是降序，默认为 asc 升序。

返回值：会返回按指定分组和排序得出的排序行号。

示例：

```

CREATE TABLE product(name VARCHAR(8), deptno NUMBER, output NUMBER);
INSERT INTO product VALUES('Linda',100,5050);
INSERT INTO product VALUES('Tan',1001,8500);
INSERT INTO product VALUES('Tom',1001,3900);
INSERT INTO product VALUES('John',100,29500);
INSERT INTO product VALUES('Mery',1001,1500);
INSERT INTO product VALUES('Peter',100,1060);

```

```

kingwow> select * from product;
+-----+
| name | deptno | output |
+-----+
| Linda | 100 | 5050 |
| Tan | 1001 | 8500 |
| Tom | 1001 | 3900 |
| John | 100 | 29500 |
| Mery | 1001 | 1500 |
| Peter | 100 | 1060 |
+-----+
6 rows in set (0.00 sec)

```

```

kingwow> SELECT name,OUTPUT,deptno,ROW_NUMBER() OVER (partition BY deptno ORDER BY OUTPUT DESC) FROM
product;
+-----+
| name | OUTPUT | deptno | ROW_NUMBER() OVER (partition BY deptno ORDER BY OUTPUT DESC) |
+-----+
| John | 29500 | 100 | 1 |

```



Linda	5050	100	2
Peter	1060	100	3
Tan	8500	1001	1
Tom	3900	1001	2
Mery	1500	1001	3

6 rows in set (0.01 sec)

LISTAGG(col_name, split_char)[within group(order by col_name[desc|asc] [, col_name[desc|asc]]) [desc]]

描述：用于将多行数据连接为一个字符串。

column_name: 要连接的列的名称。

split_char: 为输出时的分隔符。

within group: 可选参数，用于指定排序列。

返回值：返回按设定规则拼接成的字符串。

示例：

```
CREATE TABLE orders (
  order_id INT,
  product_name VARCHAR(50)
);
```

```
INSERT INTO orders VALUES (1, 'Apple');
INSERT INTO orders VALUES (1, 'Banana');
INSERT INTO orders VALUES (1, 'Orange');
INSERT INTO orders VALUES (2, 'Grapes');
INSERT INTO orders VALUES (2, 'Mango');
```

```
kingwow> SELECT * FROM orders;
```

order_id	product_name
1	Apple
1	Banana
1	Orange
2	Grapes
2	Mango

5 rows in set (0.00 sec)

```
kingwow> select listagg(product_name,'---') from orders;
```

listagg(product_name,'---')
Apple---Banana---Orange---Grapes---Mango

1 row in set (0.00 sec)

```
kingwow> select listagg(product_name,',') within group(order by order_id) from orders;
```

listagg(product_name,',') within group(order by order_id)
Apple,Banana,Orange,Grapes,Mango

1 row in set (0.07 sec)

4.2.8. 自定义函数

创建函数：

function_name 是用户自定义的函数名，return_type 为函数返回值类型，使用时需要用 delimiter 来修改语句结束符。

```
CREATE [OR REPLACE] FUNCTION function_name RETURN return_type
[AS]IS [VARIABLE DECLARE STATEMENTS]
BEGIN
    FUNCTION_body
END;
```

示例：

```
kingwow> CREATE OR REPLACE FUNCTION is_date(@str VARCHAR2) RETURNS NUMBER
-> begin
-> declare @val DATE;
-> set @val =to_date(nvl(@str,'a'),'yyyy-mm-dd');
-> return 1;
-> EXCEPTION
-> WHEN OTHERS THEN
-> return 0;
-> end;
-> $
Query OK, 1 row affected (0.01 sec)
[DDL]FUNCTION_CREATE SUCCESS.
```

4.2.9. 转换函数

CAST(expr AS type)

描述：将表达式 expr 的值转换为 type 指定的数据类型。

expr :列名或者表达式。

AS :用于分隔两个参数，AS 之前的是要处理的数据，AS 之后是要转换的数据类型。

type :目标数据类型，数据类型为 Kingwow 的内建数据类型。

返回值：返回与 type 相同类型的值。

示例：

```
kingwow> select CAST(123 as int),CAST(123 as varchar),CAST(123 as bool);
+-----+-----+-----+
| CAST(123 as int) | CAST(123 as varchar) | CAST(123 as bool) |
+-----+-----+-----+
| 123 | 123 | 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

CONVERT(expr,expr type)

描述：用来转换为新数据类型。

expr :列名或者表达式。

expr type :表示想要转换的目标数据类型。

返回值：返回与指定的 expr type 相同类型的值。

示例：

```
kingwow> select convert(123, decimal(5,2));
+-----+
```

 **kingwow!**
 | convert(123, decimal(5,2)) |
 +-----+
 | 123.00 |
 +-----+
 1 row in set (0.00 sec)

CONVERT_TZ(dt,from_tz,to_tz)

描述: convert_tz(dt,from_tz,to_tz)

转换 datetime 值 dt, 从 from_tz 由给定的时区转到 to_tz 时给出的时区, 并返回的结果值。

返回值: 返回与指定的 expr type 相同类型的值。

示例:

```
kingwow> select CONVERT_TZ('2018-12-25 12:25:00','+00:00','+08:00') as 北京时间;
+-----+
| 北京时间 |
+-----+
| 2018-12-25 20:25:00 |
+-----+
1 row in set (0.000 sec)
```

CONV(n,from_base,to_base)

描述: 用于执行基数之间的转换, 基数范围是 2 到 36。

n:要转换的数字。

from_base:数字的原始基数。

to_base:希望得到的目标基数。

返回值: 将 N 从 from_base 转换到 to_base 转换得到的字符串, 如果任一参数为 NULL, 则返回 NULL。

示例:

```
kingwow> SELECT CONV(5,16,2);
+-----+
| CONV(5,16,2) |
+-----+
| 101 |
+-----+
1 row in set (0.00 sec)
```

HEXTORAW(str)

描述: 将用于将十六进制字符串转换为 RAW 类型的字节串。

str: 是要转换的字符串。

返回值: 返回转换后的 RAW 类型数据, 当 str 为 NULL 时返回 NULL。

示例:

```
kingwow> select hextoraw('616263646566'),hextoraw(null);
+-----+-----+
| hextoraw('616263646566') | hextoraw(null) |
+-----+-----+
| abcdef | NULL |
+-----+-----+
1 row in set (0.00 sec)
```

RAWTOHEX(str)

描述：用于将二进制数转换为一个相应的十六进制表示的字符串。

str: 是要转换的字符串。

返回值：返回转换后的字符串，当字符串 str 是合法的十六进制数时，将其转换为对应的字符串，str 为 NULL 时返回 NULL。

示例：

```
kingwow> select rawtohex('abcdef'),rawtohex(null);
+-----+
| rawtohex('abcdef') | rawtohex(null) |
+-----+
| 616263646566      | NULL          |
+-----+

1 row in set (0.00 sec)
```

REVERSE(str)

描述：提供一个 reverse 函数,可以实现将一个对象反向转换.

示例：

```
select reverse('123456') from dual;
```

```
kingwow> select reverse('123456') from dual;
+-----+
| reverse('123456') |
+-----+
| 654321            |
+-----+

1 row in set (0.010 sec)
```

4.2.10. JSON 相关函数

JSON_VALID(value)

描述：用于验证一个字符串是否包含有效的 JSON 格式。

value: 待校验的字符串。

返回值：如果 value 是有效的 JSON 格式的字符串，则返回 1，否则返回 0。另外，当输入参数为 NULL 时返回 NULL。

示例：

```
kingwow>select json_valid(-0.0001),json_valid(null);
+-----+
| json_valid(-0.0001) | json_valid(null) |
+-----+
| 0                   | NULL            |
+-----+

1 row in set (0.00 sec)
kingwow>select json_valid('{"a": "3"}'),json_valid('{"product1": "Piwo","product2": "Wino"}');
+-----+
| json_valid('{"a": "3"}') | json_valid('{"product1": "Piwo","product2": "Wino"}') |
+-----+
```

```

1 |
+-----+
1 row in set (0.00 sec)

```

JSON_VALUE(content,path)

描述：用于从 JSON 字符串中提取指定路径的值。

content：VARCHAR 类型，需要解析的 JSON 对象，使用字符串表示。

Path： VARCHAR 类型，解析 JSON 的路径表达式。

返回值：返回对应提取的字符串。

示例：

```

kingwow>
select json_value('{"a":"b","c":{"d":"e","f":"g","h":["h0","h1","h2"]},"i":"j"}', '$.c.h[1]');
+-----+
| json_value('{"a":"b","c":{"d":"e","f":"g","h":["h0","h1","h2"]},"i":"j"}', '$.c.h[1]') |
+-----+
| h1 |
+-----+
1 row in set (0.00 sec)

```

JSON_EXTRACT(json_str, path)

描述：用于从 JSON 格式的字符串中提取指定的值。

json_str：包含 JSON 数据的字符串。

path：指定要提取的 JSON 路径或键的字符串。

返回值：返回对应提取出的字符串值。

示例：

```

kingwow>select json_extract('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', '$.f4.f6');
+-----+
| json_extract('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', '$.f4.f6') |
+-----+
| "stringy" |
+-----+
1 row in set (0.03 sec)

```

JSON_QUOTE(str)

描述：用于将字符串转换为 JSON 格式的字符串，并用双引号修饰。

str：待转换的字符串。

返回值：返回一个用双引号修饰的字符串。

示例：

```

kingwow>select json_quote('kingwow');
+-----+
| json_quote('kingwow') |
+-----+
| "kingwow" |
+-----+
1 row in set (0.00 sec)

```

JSON_UNQUOTE(str)

描述：用于去除文本中的引号，对转义符有所处理，或者舍弃 JSON 值中的引号。

str: 待转换的字符串。

返回值：返回去掉修饰后的字符串。

示例：

```
kingwow>select json_unquote('dajifa\tIMIDF');
+-----+
| json_unquote('dajifa\tIMIDF') |
+-----+
| dajifa          IMIDF          |
+-----+
1 row in set (0.00 sec)
```

JSON_DEPTH(str)

描述：用于返回 JSON 格式数据的最大深度。

返回值：返回一个整型数值，表示传入 JSON 数据的最大深度。

示例：

```
kingwow>SELECT JSON_DEPTH('[10, {"a": 20}]'),JSON_DEPTH('{}');
+-----+-----+
| JSON_DEPTH('[10, {"a": 20}]') | JSON_DEPTH('{}') |
+-----+-----+
| 3 | 1 |
+-----+-----+
1 row in set (0.00 sec)
```

JSON_REMOVE(json_str, path[, path] ...)

描述：用于从 JSON 中删除数据，并返回删除后的结果。

json_str:用于删除数据的源字符串。

path:待删除数据的路径。

返回值：返回删除指定路径数据后的 json 数据，如果有参数为 NULL，则返回 NULL。

示例：

```
kingwow>select json_remove('{"name":"zhang","age":"12", "url":"https"}',$.name');
+-----+
| json_remove('{"name":"zhang","age":"12", "url":"https"}',$.name') |
+-----+
| {"age": "12", "url": "https"} |
+-----+
1 row in set (0.00 sec)
```

```
kingwow> select json_remove('{"name":"zhang","age":"12", "url":"https"}',NULL);
+-----+
| json_remove('{"name":"zhang","age":"12", "url":"https"}',NULL) |
+-----+
| NULL |
+-----+
1 row in set (0.00 sec)
```

4.2.11. XML 相关函数

EXTRACTVALUE(xml_str, path_expr)

描述：用于从 XML 字符串中提取指定路径的值。

xml_str:用于查找的 XML 字符串。

path:用于指定要提取值的路径。

补充：path 如果是//开头，则为相对路径，如果是/开头，则为绝对路径。

返回值：返回从 xml 字符串中提取出的指定值。

示例：

```
kingwow> SELECT ExtractValue('<a>X</a><a>Y</a><a>Z</a>', '//a[1]');
+-----+
| ExtractValue('<a>X</a><a>Y</a><a>Z</a>', '//a[1]') |
+-----+
| X                                             |
+-----+
1 row in set (0.00 sec)
kingwow>> SELECT EXTRACTVALUE('<a><b>x</b><b>y</b></a>', '/a/b');
+-----+
| EXTRACTVALUE('<a><b>x</b><b>y</b></a>', '/a/b') |
+-----+
| x y                                           |
+-----+
1 row in set (0.00 sec)
```

4.2.12. 其他函数

INT2IP(int_value)

描述：将一个整数转换成 IP 地址。

返回值：返回由 int_value 转换的对应的 IP 地址，当输入非整数时返回 NULL，目前只支持 IPV4 地址转换。

示例：

```
kingwow> select int2ip(16777216),int2ip(1),int2ip(4294967295),int2ip(true),int2ip(now()),
int2ip(10.1);
+-----+-----+-----+-----+-----+
| int2ip(16777216) | int2ip(1) | int2ip(4294967295) | int2ip(true) | int2ip(now()) | int2ip(10.1) |
+-----+-----+-----+-----+-----+
| 0.0.0.1          | 1.0.0.0   | 255.255.255.255    | NULL         | NULL          | NULL         |
+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

IP2INT(str)

描述：将字符串表示的 IP 地址转换为整数。

返回值：返回 IP 地址对应的整数，当输入的参数不是字符串或字符串表示的不是合法 IPV4 地址时返回 NULL。

示例：

```
kingwow> select ip2int('0.0.0.1'),ip2int('192.168.0.1'),ip2int('255.255.255.255'),
ip2int('256.0.0.0');
+-----+-----+-----+-----+
| ip2int('0.0.0.1') | ip2int('192.168.0.1') | ip2int('255.255.255.255') | ip2int('256.0.0.0') |
+-----+-----+-----+-----+
|          16777216 |          16820416 |          4294967295 |          NULL |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

CURRENT_USER()

描述：用于返回当前连接到数据库的用户。

返回值：返回一个字符串，表示当前连接到数据库的用户。

示例：

```
kingwow> select CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| admin@%        |
+-----+
1 row in set (0.04 sec)
```

USER()

描述：用于返回当前连接到数据库的用户名及 IP 地址。

返回值：返回一个字符串，表示当前连接到数据库的用户和 IP 地址。

示例：

```
kingwow> select USER();
+-----+
| USER()          |
+-----+
| admin@192.168.100.202 |
+-----+
1 row in set (0.00 sec)
```

DATABASE()

描述：用于返回当前数据库的名称。

返回值：返回一个字符串，表示当前数据库的名称。

示例：

```
kingwow> select database();
+-----+
| database() |
+-----+
| kingwow    |
+-----+
1 row in set (0.02 sec)
```

VERSION()

描述：用于返回当前数据库的版本号。

返回值：返回一个字符串，表示当前数据库的版本号。

示例：

```
kingwow> select version();
+-----+
| version() |
+-----+
| 5.7.32    |
+-----+
1 row in set (0.00 sec)
```

UUID()

描述：用于生成通用唯一标识符（UUID）。

返回值：会返回一个全局唯一的标识符。

示例：

```
kingwow> select uuid();
+-----+
| uuid()          |
+-----+
| 6547540089344360449 |
+-----+
1 row in set (0.00 sec)
```

SYS_GUID()

描述：用于生成一个全局唯一标识符（GUID）。

返回值：会返回一个全局唯一的标识符。

示例：

```
kingwow> select sys_guid();
+-----+
| sys_guid()      |
+-----+
| 6584486988827590665 |
+-----+
1 row in set (0.00 sec)
```

AES_ENCRYPT(str, key)

描述：用于使用高级加密标准（AES）算法加密字符串。

str: 要加密的字符串。

key: 用于加密的密钥。

返回值：会返回一个加密后的字符串。



示例：

```
kingwow>SELECT AES_ENCRYPT('geeksforgeeks', 'key');
+-----+
| AES_ENCRYPT('geeksforgeeks', 'key') |
+-----+
| MkcUxPmsQvm7yrr14yqgmQ==          |
+-----+
1 row in set (0.00 sec)
```

补充：

AES_ENCRYPT() 使用密钥字符串 key_str 加密字符串 str 并返回包含加密输出的二进制字符串。相关配置变量：block_encryption_mode。此变量控制基于块的算法（AES）的块加密模式。它会影响 AES_ENCRYPT() 和 AES_DECRYPT() 的加密。

block_encryption_mode 采用 aes-keylen-mode 格式的值，其中 keylen 是以位为单位的密钥长度，mode 是加密模式。该值区分大小写。允许的 keylen 值为 128,192 和 256。允许的加密模式 mode 值为：ECB，CBC，CFB1，CFB8，CFB128，OFB

例如，此语句使 AES 加密函数使用 256 位密钥长度和 CBC 模式：

SET block_encryption_mode = 'aes-256-cbc';

尝试将 block_encryption_mode 设置为包含不受支持的密钥长度的值或 SSL 库不支持的模式时发生错误。

下表列出了每种允许的块加密模式，支持它的 SSL 库以及是否需要初始化向量参数。

块加密模式	支持模式的 SSL 库	需要初始化向量
ECB	OpenSSL	否
CBC	OpenSSL	是
CFB1	OpenSSL	是
CFB8	OpenSSL	是
CFB12	OpenSSL	是
OFB	OpenSSL	是

block_encryption_mode 变量有效值：

```
aes-128-ecb
aes-192-ecb
aes-256-ecb
aes-128-cbc
aes-192-cbc
aes-256-cbc
aes-128-cfb1
aes-192-cfb1
aes-256-cfb1
aes-128-cfb8
aes-192-cfb8
aes-256-cfb8
aes-128-cfb128
```

```

aes-192-cfb128
aes-256-cfb128
aes-128-ofb
aes-192-ofb
aes-256-ofb

```

AES_DECRYPT(str, key)

描述：用于对通过 aes_encrypt 加密的数据进行解密。

str: 要解密的字符串。

key: 用于解密的密钥。

返回值：会返回一个解密后的字符串。

示例：（对应 AES_ENCRYPT 示例中的加密示例）

```

kingwow>SELECT AES_DECRYPT('MkcUxPmsQvm7yrr14yqgmQ==','key');
+-----+
| AES_DECRYPT('MkcUxPmsQvm7yrr14yqgmQ==','key') |
+-----+
| geeksforgeeks                                |
+-----+
1 row in set (0.00 sec)

```

SM3(str)

描述：使用 sm3 加密字符串。

返回值：会返回一个用 sm3 加密后的字符串。

示例：

```

kingwow>select sm3('abc');
+-----+
| sm3('abc')                                |
+-----+
| 66C7F0F462EEEDD9D1F2D46BDC10E4E24167C4875CF2F7A2297DA02B8F4BA8E0 |
+-----+
1 row in set (0.00 sec)

```

MD5(str)

描述：计算字符串的 md5 散列值。

value: varchar 类型，需要解析的 JSON 对象，使用字符串表示。

path: varchar 类型，解析 JSON 的路径表达式。

返回值：返回字符串的 MD5 值。

示例：

```

kingwow> select md5('a');
+-----+
| md5('a')                                |
+-----+
| 0cc175b9c0f1b6a831c399e269772661 |
+-----+
1 row in set (0.00 sec)

```

INET_ATON(str_ip)

描述：用于将输入的 ip 地址字符串转换为无符号整型。

返回值：返回 str_ip 对应的无符号整型。

示例：

```
kingwow> select inet_aton('192.168.100.202');
+-----+
| inet_aton('192.168.100.202') |
+-----+

|          3232261322 |
+-----+
1 row in set (0.00 sec)
```

INET_NTOA(n)

描述：用于将无符号整型转换回 ip 地址字符串。

返回值：返回 n 对应的 ip 地址字符串

示例：

```
kingwow> select inet_ntoa(3232261322);
+-----+
| inet_ntoa(3232261322) |
+-----+
| 192.168.100.202      |
+-----+
1 row in set (0.00 sec)
```

LAST_INSERT_ID()

描述：用于获取最后一个插入操作产生的自增 ID 的函数。

返回值：返回 n 对应的 ip 地址字符串

示例：

```
create table t11(c1 int auto_increment primary key,c2 int,c3 int, index idx (c3), unique index
uidx(c2));
insert into t11(c2,c3) values(1,1);
insert into t11(c2,c3) values(2,2),(3,3),(4,4),(5,5);
insert into t11 values(null,6,6),(null,7,7),(null,8,8);
select * from t11;
select last_insert_id();
insert into t11(c2,c3) values(9,9);
select * from t11;
kingwow>select last_insert_id();
+-----+
| last_insert_id() |
+-----+
|          9 |
+-----+
1 row in set (0.00 sec)
```

4.3. 运算符

KINGWOW 数据库支持四种类型的运算符，包括算数运算符、比较运算符、

表 4-2: KINGWOW 数据库中的运算符

类型	运算符	含义
算数运算符	+, -, *	加减乘;
	/	除法, 返回商, 如果除数为 0 则报错;
	%或 mod	除法, 返回余数, 如果除数为 0 则报错;
比较运算符	=, >=, >, <=, <	等于, 大于等于, 大于, 小于等于, 小于;
	!=或<>	不等于;
	between ... and ...	存在于指定范围;
	in	存在于指定集合, 最多允许指定 5120 个值;
	is null	为 null;
	is not null	不为 null;
	is true/false/unknown	为真/为假/为未知;
	is not true/false/unknown	不为真/不为假/不为未知;
	like	字符串通配符匹配, 通配符“%”表示匹配任意长度字符串, 包括空字符串; 通配符“_”表示匹配单个字符, 不包括空字符;
逻辑运算符	not	逻辑非;
	and	逻辑与;
	or	逻辑或;
拼接运算符		将值拼接在一起构成单个值, 可实现字符串拼接;

当进行混合运算时, 需要了解运算符的优先级。表 4-3 从高到低列出了各种运算符的优先级。在实际运用时, 建议用“()”将需要优先计算的操作括起, 这样

既起到优先计算的作用，又便于理解。

表 4-3: KINGWOW 数据库中运算符的优先级

优先级	运算符
1	*, /, %, mod
2	+, -
3	=, >, >=, <, <=, <>, !=, IS, LIKE, IN
4	between ... and
5	not
6	and
7	or

4.4. 转义字符

转义字符是在特定字符前加反斜线'\', 用于表示特殊的字符。KINGWOW 数据库中常用的转义字符如表 4-4 所示

表 4-4: KINGWOW 数据库中的转义字符

转义字符	含义
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	Tab 字符
\\	反斜线字符
\'	单引号

转义字符	含义
\"	双引号
_	_字符
\%	%字符
\0	空字符(NULL)

4.5 字符集

KINGWOW 数据库默认采用的字符集为 utf8,支持 gbk 字符集并兼容 gb18030、utf8mb4 字符集。

查看当前库使用的字符集，语法格式为：

```
SHOW VARIABLES LIKE 'CHARACTER%';
kingwow> SHOW VARIABLES LIKE 'CHARACTER%';
```

```
+-----+-----+
| variable_name | value |
+-----+-----+
| character_set_connection | utf8 |
| character_set_server | utf8 |
| character_set_client | utf8 |
| character_bytes | 1 |
| character_set_results | utf8 |
+-----+-----+
5 rows in set (0.03 sec)
```

设置字符集，语法格式为：

```
SET GLOBAL/SESSION CHARACTER_SET_CLIENT='GB18030';
```

创建字符集为 gbk 的测试表

```
CREATE TABLE kingwow_charset1 (
    id INT PRIMARY KEY,
    name VARCHAR(255)
)CHARACTER SET gbk;
```

```
kingwow> INSERT INTO kingwow_charset1 (id, name) VALUES (1, '测试字符');
Query OK, 1 row affected (0.025 sec)
[DML]INSERT SUCCESS.
```

```
kingwow> select * from kingwow_charset1;
+-----+-----+
| id | name |
+-----+-----+
| 1 | 测试字符 |
+-----+-----+
1 row in set (0.024 sec)
```

创建字符集为 gb18030 的测试表



```
CREATE TABLE kingwow_charset2 (  
    id INT PRIMARY KEY,  
    name VARCHAR(255)  
)CHARACTER SET gb18030;
```

```
kingwow> INSERT INTO kingwow_charset2 (id, name) VALUES (2, '测试字符');  
Query OK, 1 row affected (0.024 sec)  
[DML]INSERT SUCCESS.
```

```
kingwow> select * from kingwow_charset2;  
+-----+-----+  
| id | name |  
+-----+-----+  
| 2 | 测试字符 |  
+-----+-----+  
1 row in set (0.024 sec)
```

创建字符集为 utf8mb4 的测试表

```
CREATE TABLE kingwow_charset3 (  
    id INT PRIMARY KEY,  
    name VARCHAR(255)  
)CHARACTER SET utf8mb4;
```

```
kingwow> insert into kingwow_charset3 values(3,'龙行龘龘');  
Query OK, 1 row affected (0.025 sec)  
[DML]INSERT SUCCESS.
```

```
kingwow> select * from kingwow_charset3;  
+-----+-----+  
| id | name |  
+-----+-----+  
| 3 | 龙行龘龘 |  
+-----+-----+  
1 row in set (0.024 sec)
```

5. SQL 语法

5.1. 数据定义语句

5.1.1. CREATE DATABASE

创建新的数据库，语法格式为：

```
CREATE DATABASE db_name;
```

说明：

- 数据库名长度不能超过 19 个字符，包括字母、数字、下划线，首字符可以是字母或下划线；

```
kingwow> create database sales;  
Query OK, 0 rows affected (0.12 sec)
```

```
kingwow> show databases;  
+-----+
```



```

| Database |
+-----+
| kingwow  |
| sales    |
+-----+
2 rows in set (0.00 sec)

```

5.1.2. DROP DATABASE

删除数据库，语法格式为：

```
DROP DATABASE db_name;
```

说明：

- 当 db_name 所指定的数据库不存在时，语句会报错；
- 删除数据库之前需要先删除库中全部的数据表，否则无法删除数据库；
- 删除数据库可能会导致会话的默认数据库为空值，此时需要为会话重新指定默认数据库，否则该会话无法在默认数据库下创建表；

```

kingwow> use sales;
Database changed
kingwow> show tables;
Empty set (0.00 sec)

```

```

kingwow> drop database sales;
Query OK, 1 row affected (0.09 sec)

```

```

kingwow> select database();
+-----+
| database() |
+-----+
|            |
+-----+
1 row in set (0.00 sec)

```

```

kingwow> use test;
Database changed
kingwow> select database();
+-----+
| database() |
+-----+
| test       |
+-----+
1 row in set (0.00 sec)

```

5.1.3. RENAME DATABASE

重命名数据库，语法格式：

```
RENAME DATABASE db_name TO new_db_name;
```

```

kingwow> create database testdb;
Query OK, 0 rows affected (0.09 sec)
kingwow> show databases;
+-----+
| Database |
+-----+
| kingwow  |
| test     |
| testdb   |
+-----+
3 rows in set (0.00 sec)

```



```
kingwow> rename database testdb to test_database;
Query OK, 0 rows affected (0.48 sec)
kingwow> show databases;
+-----+
| Database |
+-----+
| kingwow  |
| test     |
| test_database |
+-----+
3 rows in set (0.00 sec)
```

5.1.4. CREATE TABLE

创建新的数据表，语法格式为：

```
CREATE TABLE [IF NOT EXISTS] tbl_name (
  col_name data_type [NOT NULL | NULL] [DEFAULT default_value] ,[COMMENT
  'string'],[AUTO_INCREMENT],
  ..... ,
  PRIMARY KEY (col_name1, col_name2...) ) [table_options_list];
```

说明：

- 支持使用 COMMENT 添加对字段或列的注释。
- 如果表有主键,必须建表时指定,不允许建表后新增或者修改主键。
- 支持全主键的表。
- 表名长度限 255 个字符，包括字母、数字、下划线，首字符必须是字母；
- 列名长度限 127 个字符，包括字母、数字、下划线，首字符可以是字母或下划线；
- 每张表的列数限制为最大 623 列；
- 如果指定 IF NOT EXISTS，即使创建的表已经存在，也不报错；不指定时，若创建的表已经存在则报错；
- KINGWOW 数据库在执行合并操作时不允许创建表；

table_options_list 用于指定创建表时的一些可选参数，详见表 5-1。

表 5-1：KINGWOW 数据库的建表参数

参数	含义	示例
EXPIRE_INFO	数据合并时删除满足条件的	EXPIRE_INFO = '\$SYS_DATE > c1 + 24

	数据，常用于自动删除过期数据；数据表创建索引后，不能再修改过期删除条件；	* 60 * 60 * 1000000' 自动删除 c1 值比当前时间小 24 小时的数据，EXPIRE_INFO 取值单位：微秒。
TABLET_MAX_SIZE	设置 tablet 大小的上限，单位为字节，默认为 64M；	TABLET_MAX_SIZE=268435456 设置 tablet 大小为 256M
REPLICA_NUM	设置 tablet 的副本数，默认为 3，最大不超过 6；	REPLICA_NUM=3
COMPRESS_METHOD	存储数据时使用的压缩方法 1. none (默认值，不压缩) 2. lzo_1.0 3. snappy_1.0 4. zstd_1.5	COMPRESS_METHOD = 'none' COMPRESS_METHOD = 'lzo_1.0' COMPRESS_METHOD = 'snappy_1.0' COMPRESS_METHOD = 'zstd_1.5'

```
kingwow> create table customer(
-> id int not null,
-> name varchar(10) not null ,
-> address varchar(50),
-> last_login datetime,
-> remark varchar(1024) default 'N/A',
-> primary key(id)
-> )compress_method='snappy_1.0',
-> replica_num=4,
-> expire_info='$SYS_DATE > last_login+38*24*60*60*1000000',
-> tablet_max_size=134217728;
Query OK, 0 rows affected (0.93 sec)
```

```
kingwow> desc customer;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id         | int32         | NULL      | 0    | 1   | NULL    |      |
| name       | varchar(10)   | NULL      | 0    | 0   | NULL    |      |
| address    | varchar(50)   | NULL      | 1    | 0   | NULL    |      |
| last_login | timestamp     | NULL      | 1    | 0   | NULL    |      |
| remark     | varchar(1024) | NULL      | 1    | 0   | N/A     |      |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

● CREATE TABLE .. LIKE ..

创建新的数据表，语法格式为：

```
CREATE TABLE tbl_name_1 LIKE tbl_name_2;
```

示例：

```
kingwow> create table t1(c1 int,c2 int);
Query OK, 0 rows affected (1.60 sec)
[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> create table t2 like t1;
Query OK, 0 rows affected (1.61 sec)
[DDL]CREATE_TABLE SUCCESS.
```



kingwow> desc t1;

```
+-----+-----+-----+-----+-----+-----+
| Field | Type | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c1    | int  | NULL      | YES  |     | NULL     |       |
| c2    | int  | NULL      | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

kingwow> desc t2;

```
+-----+-----+-----+-----+-----+-----+
| Field | Type | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c1    | int  | NULL      | YES  |     | NULL     |       |
| c2    | int  | NULL      | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.03 sec)
```

● CREATE TABLE .. AS SELECT ..

创建新的数据表，语法格式为：

```
CREATE TABLE tbl_name_1 AS SELECT ..
```

示例：

kingwow> drop table if exists t1,t2,t3;

Query OK, 0 rows affected (0.00 sec)

[DDL]DROP_TABLE SUCCESS.

kingwow> create table t1(c1 int, c2 int, c3 int);

Query OK, 0 rows affected (1.64 sec)

[DDL]CREATE_TABLE SUCCESS.

kingwow> insert into t1 values(1,1,1),(2,2,22);

Query OK, 2 rows affected (0.01 sec)

[DML]INSERT SUCCESS.

kingwow> create table t2 like t1;

Query OK, 0 rows affected (1.58 sec)

[DDL]CREATE_TABLE SUCCESS.

kingwow> desc t2;

```
+-----+-----+-----+-----+-----+-----+
| Field | Type | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c1    | int  | NULL      | YES  |     | NULL     |       |
| c2    | int  | NULL      | YES  |     | NULL     |       |
| c3    | int  | NULL      | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
```



+-----+-----+-----+-----+-----+-----+-----+

3 rows in set (0.00 sec)

```
kingwow> create table t3 as select * from t1;
```

Query OK, 0 rows affected (1.58 sec)

[DDL]CREATE_TABLE SUCCESS.

```
kingwow> select * from t2;
```

Empty set (0.00 sec)

```
kingwow> select * from t3;
```

+-----+-----+-----+

| c1 | c2 | c3 |

+-----+-----+-----+

| 1 | 1 | 1 |

| 2 | 2 | 22 |

+-----+-----+-----+

2 rows in set (0.00 sec)

过期信息 EXPIRE_INFO

一、 DDL 语法

1.建表时指定过期条件

```
CREATE TABLE table(column_name data_type,...,PRIMARY KEY(column_name,...))
```

```
EXPIRE_INFO=[expire_info_condition];
```

2.查看过期条件

```
show create table table_name;
```

3 新增过期条件

```
update __first_tablet_entry set expire_condition=[expire info_condition] where  
table_name='XXXX';
```

4.修改过期条件

```
update __first_tablet_entry set expire_condition=[expire info_condition] where
```



table_name='XXXX';

5 删除过期条件

```
update      __first_tablet_entry      set      expire_condition=""      where  
table_name='acmg_t_pecust_dept_indct_h';
```

6.示例

#创建 c4 列的过期条件

```
create table t1(c1 int primary key,c2 int,c3 varchar(30),c4 datetime)  
EXPIRE_INFO='$SYS_DATE <c4 + 1*24*60*60*1000000';  
(注：过期信息指定的列名要小写)
```

二、[expire_info_condition]格式

- expire_info_condition 仅支持布尔类型表达式
 - 时间比较格式、字符串比较格式、数值比较格式及永假和 NUL 过滤的特殊格式
 - 支持算术运算符(加、减、乘、除、取余)、比较运算符(=、>=、<=、>、<、!=)、逻辑运算符(AND、OR)
 - 不支持函数(字符串函数 substr、length 等)
- 在使用中有以下限制:
 - expire_info_condition 表达式最长支持 512 个字符。
 - expire_info_condition 里如果引用该表的某个或务个列属性,这些属性必须以索引列或冗余列的形式保存该表的所有索引表里。
 - 数据以[expire_info_condition]所指定格式解析错误的话, 错误数据保留, 不做删除处理。

下面将介绍过期条件表达式支持格式。

1.时间格式

过期条件时间格式支持：绝对时间比较、相对时间比较。

- 相对时间比较

\$SYS_DATE 为相对时间的比较提供了现实时间基准, 多用于历史数据清理场景。

```
expire_info= '$SYS_DATE < column_name + relative';
```



#relative time 以微妙为单位

➤ 列属性限制

(1)modifytime,createtime, datetime ,timestamp,varchar,char 类型

(2)varchar 列的数据格式支持 YYYYMMDD、YYYY/MM/DD 和 KINGWOW 标准时间戳格式 'YYYY-MM-DD HH:MI:SS' 。

➤ 示例

#若下一次合并的调度时刻是当天 09:15 分 (e.g. 2020/11/11 09:15:00)

#本轮过期条件计算时 SYSLDATE='2020/11/11 09:15:00'

```
create table t1(c1 int primary key,c2 int,c3 varchar(30),c4 datetime)
EXPIRE_INFO='$SYS_DATE >c4 + 1*24*60*60*1000000';
```

2.字符串格式

过期条件字符串格式允许将一列与常量字符串进行比较,根据比较结果判断是否数据过期。

➤ 基本格式

过期条件支持与常量字符串进行比较。

```
EXPIRE_INFO='c4='\compare string\''
```

➤ 列属性限制

支持 varchar 和 char 类型

➤ 示例：

#新增 t1 表的 expire_info,c3 列数据等于'test'则视为过期

```
create table t1(c1 int primary key,c2 int ,c3 varchar(30),c4 datetime)
EXPIRE_INFO='c3='\test\'';
```

3.整型比较格式

过期条件数值格式允许将一列与整型常量进行比较, 根据比较结果判断是否数据过期。

➤ 基本格式

```
expire_info='c2>3';
```

➤ 列属性限制

支持 int, big int, float, double 类型

➤ 示例

#新增 t1 表的 expire_info, 表示 c2 列值>3 的记录即视为过期

```
create table t1(c1 int primary key, c2 int, c3 varchar(30), c4 datetime) EXPIRE_INFO = 'c2 > 3';
```

4. 特殊格式

➤ 永假

```
expire_info = '1=0'
```

#该 expire_info 表示永假, 合并时所有数据全部重写。

➤ 永真

```
expire_info = '1=1'
```

#该 expire_info 表示永真, 合并时所有数据全部删除。

➤ NULL 过滤格式

```
expire_info = 'c2=NULL'
```

#c2 列值为 NULL 的记录均视为过期数据

5. 复杂过期格式示例

expire_info_condition 支持关键字 AND 和 OR 的逻辑运算。具体示例如下:

```
expire_info = 'c2>2 [and | or] c1>2 '
```

```
expire_info = 'c2>2 [and | or] c1=NULL '
```

```
expire_info = 'c2>2 [and | or] 1=0 '
```

```
expire_info = 'c2>2 [and | or] 1=1 '
```

```
expire_info = '$SYS_DATE <c4 + relative_time [and | or ] c1>2';
```

```
expire_info = '$SYS_DATE <c4 + relative_time [and | or ] c3=\'test\'';
```

```
expire_info = '$SYS_DATE <c4 + relative_time [and | or ] c3=NULL';
```



```
expire_info= '$SYS_DATE <c4 + relative_time [and | or ] 1=0';
```

```
expire_info= '$SYS_DATE <c4 + relative_time [and | or ] 1=1';
```

```
expire_info= ' c3='\test\' [and | or ] c1>1';
```

```
expire_info= ' c3='\test\' [and | or ] 1=0';
```

```
expire_info= ' c3='\test\' [and | or ] 1=1';
```

CHECK 约束的使用:

CHECK 约束用于限制列中的值的范围。如果对单个列定义 CHECK 约束,那么该列只允许特定的值。如果对一个表定义 CHECK 约束,那么此约束会基于行中其他列的值在特定的列中对值进行限制。

建表时指定 check 约束:

```
CREATE TABLE Persons
(
  Id_P int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255),
  CHECK (Id_P>0)
);

CREATE TABLE Persons1
(
  Id_P int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255),
  CONSTRAINT chk_Person CHECK (Id_P>0 AND City='Sandnes')
);
```

建表后增加 check 约束:

如果在表已存在的情况下为 "Id_P" 列创建 CHECK 约束,请使用下面的 SQL:

```
ALTER TABLE Persons ADD CHECK (Id_P>0);
```

如果需要命名 CHECK 约束,以及为多个列定义 CHECK 约束,请使用下面的 SQL 语法:

```
ALTER TABLE Persons ADD CONSTRAINT chk_Person CHECK (Id_P>0 AND
City='Sandnes');
```

删除 check 约束:

```
ALTER TABLE Persons DROP CONSTRAINT chk_Person;
```



查看表的约束信息：

```
select * from __all_constraint;
```

启用禁用约束：

启用 check 约束的命令为：

```
alter table table_name enable constraint constraint_name;
```

禁用 check 约束的命令为：

```
alter table table_name disable constraint constraint_name;
```

DEFAULT 值支持函数

目前支持的函数有：DATE_FORMAT()|UUID()|SYS_GUID()

示例使用如下：

```
kingwow> drop table if exists test;
Query OK, 0 rows affected (0.10 sec)

kingwow> create table test(c1 int primary key, c2 varchar(100) default date_format(to_char(now()),
'%Y-%m-%d %h:%i:%s'));
Query OK, 0 rows affected (0.93 sec)

kingwow> insert into test(c1) values(1);
Query OK, 1 row affected (0.04 sec)

kingwow> select * from test;
+-----+-----+
| c1   | c2                                     |
+-----+-----+
| 1    | 2020-03-13 10:08:12                 |
+-----+-----+
1 row in set (0.00 sec)
```

default 新增支持有：default sysdate

```
kingwow> create table test(c1 int,c2 date default sysdate);
```

```
Query OK, 0 rows affected (4.18 sec)
```

```
[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> insert into test(c1) values(1);
```

```
Query OK, 1 row affected (0.09 sec)
```

```
[DML]INSERT SUCCESS.
```

```
kingwow> select * from test;
```

```
+-----+-----+
| c1   | c2                                     |
+-----+-----+
```

+-----+-----+

1 row in set (0.00 sec)

UUID()|SYS_GUID()示例 SQL 如下:

```
drop table if exists test;
create table test(c1 int primary key, c2 varchar(34) default sys_guid());
insert into test(c1) values(1);
select * from test;
```

```
drop table if exists test;
create table test(c1 int primary key, c2 bigint default uuid());
insert into test(c1) values(1);
select * from test;
```

5.1.5. ALTER TABLE

修改已存在表的设计, 支持增加列、删除列、表重命名、列重命名和修改列数据类型。

增加列: **ALTER TABLE `tbl_name` ADD [COLUMN] `col_name` `data_type`;**

```
kingwow> alter table customer add phone varchar(11);
Query OK, 0 rows affected (0.28 sec)
```

```
kingwow> desc customer;
```

Field	Type	Collation	Null	Key	Default	Extra
id	int32	NULL	0	1	NULL	
name	varchar(10)	NULL	0	0	NULL	
address	varchar(50)	NULL	1	0	NULL	
last_login	timestamp	NULL	1	0	NULL	
remark	varchar(1024)	NULL	1	0	N/A	
phone	varchar(11)	NULL	1	0	NULL	

6 rows in set (0.00 sec)

删除列: **ALTER TABLE `tbl_name` DROP [COLUMN] `col_name`;**

```
kingwow> alter table customer drop phone;
Query OK, 0 rows affected (0.33 sec)
```

```
kingwow> desc customer;
```

Field	Type	Collation	Null	Key	Default	Extra
id	int32	NULL	0	1	NULL	
name	varchar(10)	NULL	0	0	NULL	
address	varchar(50)	NULL	1	0	NULL	
last_login	timestamp	NULL	1	0	NULL	
remark	varchar(1024)	NULL	1	0	N/A	

5 rows in set (0.00 sec)

表重命名：

```
ALTER TABLE tbl_name RENAME TO new_tbl_name;
```

```
RENAME TABLE tbl_name TO new_tbl_name;
```

```
kingwow> rename table customer to customer_bak;
```

```
Query OK, 0 rows affected (0.27 sec)
```

```
kingwow> alter table customer_bak rename to customer;
```

```
Query OK, 0 rows affected (0.25 sec)
```

列重命名：

```
ALTER TABLE tbl_name RENAME [COLUMN] col_name TO new_col_name;
```

```
kingwow> alter table customer RENAME last_login TO login;
```

```
Query OK, 0 rows affected (0.27 sec)
```

修改列内容(改列数据类型)，语法格式为：

```
alter table table_name modify column_name new_data_type;
```

注意：varchar 类型字段长度，只可由小变大，不能由大改小，否则会报错。

示例如下：

```
kingwow> create table type_test(c1 binary, c2 varbinary(100), c3 tinyblob, c4 blob);
```

```
Query OK, 0 rows affected (0.81 sec)
```

```
kingwow> desc type_test;
```

Field	Type	Collation	Null	Key	Default	Extra
c1	binary(1)	NULL	YES		NULL	
c2	varbinary(100)	NULL	YES		NULL	
c3	tinyblob	NULL	YES		NULL	
c4	blob	NULL	YES		NULL	

```
4 rows in set (0.00 sec)
```

```
kingwow> alter table type_test add column c5 binary;
```

```
Query OK, 0 rows affected (0.28 sec)
```

```
[DDL]ALTER_TABLE SUCCESS.
```

```
kingwow> alter table type_test add column c6 varbinary(100);
```

```
Query OK, 0 rows affected (0.25 sec)
```

```
kingwow> desc type_test;
```

Field	Type	Collation	Null	Key	Default	Extra
c1	binary(1)	NULL	YES		NULL	
c2	varbinary(100)	NULL	YES		NULL	
c3	tinyblob	NULL	YES		NULL	
c4	blob	NULL	YES		NULL	
c5	binary(1)	NULL	YES		NULL	
c6	varbinary(100)	NULL	YES		NULL	

```
kingwow> alter table type_test modify c5 varchar(20);
Query OK, 0 rows affected (0.27 sec)
```

```
kingwow> desc type_test;
+-----+-----+-----+-----+-----+-----+-----+
| Field | Type          | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| c1    | binary(1)     | NULL     | YES |     | NULL    |       |
| c2    | varbinary(100)| NULL     | YES |     | NULL    |       |
| c3    | tinyblob      | NULL     | YES |     | NULL    |       |
| c4    | blob          | NULL     | YES |     | NULL    |       |
| c5    | varchar(20)   | NULL     | YES |     | NULL    |       |
| c6    | varbinary(100)| NULL     | YES |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

说明:

- 若在列上已经创建索引，则该列不能被删除；
- 数据表的列名修改后，索引中相应的列名不会做修改；
- KINGWOW 数据库执行合并操作时不允许修改表结构；

修改 CHECK 约束

修改 check 约束内容:

```
ALTER TABLE table_name MODIFY CONSTRAINT constraint_name CHECK (check_expr);
```

修改 check 约束名:

```
ALTER TABLE table_name RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

5.1.6. DROP TABLE

删除 KINGWOW 数据库中的表及其索引，语法格式为:

```
DROP TABLE [IF EXISTS] tbl_name1 [, tbl_name2];
```

说明:

- 指定 IF EXISTS 选项时，即使要删除的表不存在也不报错；不指定时，若要删除的表不存在则报错；
- 需要同时删除多个表时，多个表名用逗号分隔；
- KINGWOW 数据库执行合并操作时不允许删除表；

```
kingwow> drop table customer,testtbl;
Query OK, 0 rows affected (1.68 sec)
```

5.1.7. FLASHBACK TABLE

当参数 `enable_recyclebin` 为 `true` 时，删除表操作会将表放入回收站中（`__all_recyclebin`）中，可通过该语法将回收站中的表恢复成指定名称，语法格式为：

```
FLASHBACK TABLE tbl_name TO new_tbl_name;
```

说明：

- `tbl_name` 是 `__all_recyclebin` 表 `recycle_table_name` 列值；
- `new_tbl_name` 可以为用户指定，也可指定为 `__all_recyclebin` 表 `origin_table_name` 列的值，不能与同库中已有表重名；

5.1.8. CREATE VIEW

创建视图，语法格式为：

```
CREATE [OR REPLACE] VIEW relation_factor [opt_view_expr_list] AS  
select_stmt
```

注：创建 `view` 时，以上选项均为可选项，对于用户未输入的参数均采取默认值选项，具体的默认选项见选项含义。

OPT_VIEW_EXPR_LIST

此选项无默认值，该选项指定视图列名，其中列名个数应与 `select_stmt` 中查询结果集的列数相同。若该选项未输入参数，视图列名同 `select_stmt` 中查询结果集的列名。

5.1.9. DROP VIEW

删除视图，语法格式为：

- `DROP VIEW view_name;`

```
kingwow> drop view v1;
```

```
Query OK, 0 rows affected (1.61 sec)
```

```
[DDL]DROP_VIEW SUCCESS.
```

5.1.10. CREATE INDEX

为数据表创建索引，语法格式为：

```
CREATE INDEX idx_name ON tbl_name(col_list1) [STORING (col_list2)];
```

说明:

- idx_name 为索引名，系统会在用户指定的索引名前添加前缀 ‘_xxxx_idx_’，其中 xxxx 为索引的表 ID；
- col_list1 为索引列，其列数与源表主键列数之和不能超过 16，col_list2 为冗余列；
- 一张数据表最多可以创建 24 个索引；
- 一张数据表所有索引的列数之和不能超过 100；
- KINGWOW 数据库执行合并操作时不允许创建索引；
- 新创建的索引需要等待一次数据合并后才可以生效使用；
- 索引的状态
 - ✧ NOT_AVAILABLE: 索引初始化完成，正在做备份迁移；
 - ✧ AVAILABLE: 索引状态正常，可以正常使用；
 - ✧ ERROR: 索引状态不正常，不能使用；
 - ✧ WRITE_ONLY: 索引正在被删除；
 - ✧ INDEX_INIT: 索引创建成功，等待数据合并，不能使用；

```
kingwow> create index last_login on customer(last_login) storing (name);
Query OK, 0 rows affected (0.41 sec)
```

```
kingwow> show index from customer;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation |
+-----+-----+-----+-----+-----+-----+-----+
| customer | 0 | PRIMARY | 1 | id | A | 0 |
| NULL | NULL | BTREE | | | | |
+-----+-----+-----+-----+-----+-----+-----+
| customer | 1 | 3045_idx_last_login | 1 | last_login | A |
| 0 | NULL | NULL | BTREE | INDEX_INIT | |
+-----+-----+-----+-----+-----+-----+-----+
| customer | 1 | 3045_idx_last_login | 2 | id | A | 0 |
| NULL | NULL | BTREE | | INDEX_INIT | |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

5.1.11. RENAME INDEX

为数据表修改索引名，语法格式为：

```
ALTER TABLE tbl_name RENAME INDEX idx_name_1 TO idx_name_2;
```

示例:

```
kingwow> drop table if exists sbtest;

X idx TO idx111;

show index in sbtest;

drop index idx111 on sbtest;

Query OK, 0 rows affected (2.62 sec)

[DDL]DROP_TABLE SUCCESS.
```

```
kingwow> create table sbtest(
    -> id INTEGER NOT NULL,
    -> k INTEGER NOT NULL,
    -> c VARCHAR(120) NOT NULL,
    -> pad VARCHAR(60) NOT NULL,
    -> primary key(id));

Query OK, 0 rows affected (1.59 sec)

[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> CREATE INDEX idx on sbtest(k) storing(pad);

Query OK, 0 rows affected (0.63 sec)

[DDL]CREATE_INDEX SUCCESS.
```

```
kingwow> show index in sbtest;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
Packed	Null	Index_type	Comment	Index_comment			
sbtest	0	PRIMARY	1	id	A	0	NULL
BTREE							NULL
sbtest	1	idx	1	k	A	0	NULL
BTREE		INDEX_INIT					NULL
sbtest	1	idx	2	id	A	0	NULL
BTREE		INDEX_INIT					NULL

3 rows in set (0.00 sec)

```
kingwow> alter table sbtest RENAME INDEX idx TO idx111;

Query OK, 0 rows affected (0.61 sec)

[DDL]ALTER_TABLE SUCCESS.
```

```
kingwow> show index in sbtest;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
Packed	Null	Index_type	Comment	Index_comment			
sbtest	0	PRIMARY	1	id	A	0	NULL
BTREE							
sbtest	1	idx111	1	k	A	0	NULL
BTREE		INDEX_INIT					
sbtest	1	idx111	2	id	A	0	NULL
BTREE		INDEX_INIT					

```
3 rows in set (0.00 sec)
```

```
kingwow> drop index idx111 on sbtest;
```

```
Query OK, 0 rows affected (2.22 sec)
```

```
[DDL]DROP_INDEX SUCCESS.
```

5.1.12. DROP INDEX

删除数据表的索引，语法格式为：

```
DROP INDEX idx1 [, idx2] ON tbl_name;
```

说明：

- KINGWOW 数据库执行合并操作时不允许删除索引；

```
kingwow> drop index last_login on customer;
```

```
Query OK, 0 rows affected (1.10 sec)
```

```
kingwow> show index from customer;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
Packed	Null	Index_type	Comment	Index_comment			
customer	0	PRIMARY	1	id	A	0	NULL
NULL		BTREE					

```
1 row in set (0.00 sec)
```

- 普通单键索引



```
kingwow> drop table if exists sbtest;
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
[DDL]DROP_TABLE SUCCESS.
```

```
kingwow> create table sbtest(
```

```
    -> id INTEGER NOT NULL,
```

```
    -> k INTEGER NOT NULL,
```

```
    -> c VARCHAR(120) NOT NULL,
```

```
    -> pad VARCHAR(60) NOT NULL,
```

```
    -> primary key(id));
```

```
X idx TO idx111;
```

```
show index in sbtest;
```

```
drop index idx111 on sbtest;
```

```
Query OK, 0 rows affected (1.59 sec)
```

```
[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> CREATE INDEX idx on sbtest(k) storing(pad);
```

```
Query OK, 0 rows affected (0.63 sec)
```

```
[DDL]CREATE_INDEX SUCCESS.
```

```
kingwow> show index in sbtest;
```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+										
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+										
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type Comment Index_comment
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+										
sbtest	0	PRIMARY	1	id	A	0			NULL	NULL
BTREE										
sbtest	1	idx	1	k	A	0			NULL	NULL
BTREE		INDEX_INIT								
sbtest	1	idx	2	id	A	0			NULL	NULL
BTREE		INDEX_INIT								
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+										
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+										

```
3 rows in set (0.00 sec)
```

```
kingwow> alter table sbtest RENAME INDEX idx TO idx111;
```

```
Query OK, 0 rows affected (0.62 sec)
```

```
[DDL]ALTER_TABLE SUCCESS.
```

```
kingwow> show index in sbtest;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
Null	Index_type	Comment	Index_comment					
sbtest	0	PRIMARY	1	id	A	0	NULL	NULL
BTREE								
sbtest	1	idx111	1	k	A	0	NULL	NULL
BTREE		INDEX_INIT						
sbtest	1	idx111	2	id	A	0	NULL	NULL
BTREE		INDEX_INIT						

3 rows in set (0.00 sec)

kingwow> drop index idx111 on sbtest;

Query OK, 0 rows affected (2.23 sec)

[DDL]DROP_INDEX SUCCESS.

● 多键联合索引

kingwow> create table sbtest(

-> id INTEGER NOT NULL,

-> k INTEGER NOT NULL,

-> c VARCHAR(120) NOT NULL,

-> pad VARCHAR(60) NOT NULL,

-> primary key(id));

Query OK, 0 rows affected (1.59 sec)

[DDL]CREATE_TABLE SUCCESS.

kingwow> CREATE INDEX idx on sbtest(k,c) storing(pad);

Query OK, 0 rows affected (0.63 sec)

[DDL]CREATE_INDEX SUCCESS.

kingwow> show index in sbtest;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
Null	Index_type	Comment	Index_comment					
sbtest	0	PRIMARY	1	id	A	0	NULL	NULL
BTREE								



```
| sbtest |      1 | idx |      1 | k | A |      0 | NULL | NULL |
| BTREE |      | INDEX_INIT |

| sbtest |      1 | idx |      2 | c | A |      0 | NULL | NULL |
| BTREE |      | INDEX_INIT |

| sbtest |      1 | idx |      3 | id | A |      0 | NULL | NULL |
| BTREE |      | INDEX_INIT |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

4 rows in set (0.00 sec)

```
kingwow> alter table sbtest RENAME INDEX idx TO idx111;
```

Query OK, 0 rows affected (0.61 sec)

[DDL]ALTER_TABLE SUCCESS.

```
kingwow> show index in sbtest;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
Null	Index_type	Comment	Index_comment					

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

sbtest	0	PRIMARY		1	id	A		0		NULL		NULL	
BTREE													
sbtest	1	idx111		1	k	A		0		NULL		NULL	
BTREE		INDEX_INIT											
sbtest	1	idx111		2	c	A		0		NULL		NULL	
BTREE		INDEX_INIT											
sbtest	1	idx111		3	id	A		0		NULL		NULL	
BTREE		INDEX_INIT											

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

4 rows in set (0.00 sec)

```
kingwow> drop index idx111 on sbtest;
```

Query OK, 0 rows affected (2.23 sec)

[DDL]DROP_INDEX SUCCESS.

● 唯一索引

```
kingwow> drop table if exists t1;
```

Query OK, 0 rows affected (2.60 sec)

[DDL]DROP_TABLE SUCCESS.

```
kingwow> create table t1 (a int primary key, b int, unique index unqid1(b));
```



Query OK, 0 rows affected (2.55 sec)

[DDL]CREATE_TABLE SUCCESS.

kingwow> show index in t1;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
Null	Index_type	Comment	Index_comment					
t1	0	PRIMARY	1	a	A	0	NULL	NULL
BTREE								
t1	0	unqidx1	1	b	A	0	NULL	NULL
BTREE		UNIQUE_AVAILABLE						

2 rows in set (0.00 sec)

kingwow> alter table t1 RENAME INDEX unqidx1 TO unqidx2;

Query OK, 0 rows affected (0.61 sec)

[DDL]ALTER_TABLE SUCCESS.

kingwow> show index in t1;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
Null	Index_type	Comment	Index_comment					
t1	0	PRIMARY	1	a	A	0	NULL	NULL
BTREE								
t1	0	unqidx2	1	b	A	0	NULL	NULL
BTREE		UNIQUE_AVAILABLE						

2 rows in set (0.00 sec)

kingwow> drop index unqidx2 on t1;

Query OK, 0 rows affected (2.21 sec)

[DDL]DROP_INDEX SUCCESS.

kingwow> show index in t1;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
-------	------------	----------	--------------	-------------	-----------	-------------	----------	--------



| Null | Index_type | Comment | Index_comment |

t1	0	PRIMARY	1	a	A	0	NULL	NULL
BTREE								

1 row in set (0.00 sec)

kingwow> show create table t1;

Table	Create	Table
-------	--------	-------

```
linf.t1 | CREATE TABLE `t1` (  
  `a` INT  
  , `b` INT  
  , PRIMARY KEY(`a`)  
 ) TABLET_MAX_SIZE = 134217728, TABLET_BLOCK_SIZE = 16384, USE_BLOOM_FILTER = FALSE |
```

1 row in set (0.00 sec)

● 主键索引

kingwow> drop table if exists t1;

Query OK, 0 rows affected (2.60 sec)

[DDL]DROP_TABLE SUCCESS.

kingwow> create table t1 (a int primary key, b int);

Query OK, 0 rows affected (2.55 sec)

[DDL]CREATE_TABLE SUCCESS.

kingwow> show index in t1;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
Null	Index_type	Comment	Index_comment					

t1	0	PRIMARY	1	a	A	0	NULL	NULL
BTREE								

```
2 rows in set (0.00 sec)
```

● 函数索引

```
kingwow> drop table if exists t1;
```

```
Query OK, 0 rows affected (1.60 sec)
```

```
[DDL]DROP_TABLE SUCCESS.
```

```
kingwow> create table t1(c1 int, c2 int, c3 int);
```

```
Query OK, 0 rows affected (1.61 sec)
```

```
[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> create index func_idx1 on t1(substr(c2, 1, 3));
```

```
Query OK, 0 rows affected (0.65 sec)
```

```
[DDL]CREATE_INDEX SUCCESS.
```

```
kingwow> show index in t1;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
Packed	Null	Index_type	Comment	Index_comment			
t1	1	func_idx1	1	substr(c2, 1, 3)	A	0	NULL
	BTREE		INDEX_INIT				

```
1 row in set (0.00 sec)
```

5.1.13. TRUNCATE TABLE

清空数据表中的数据，语法格式为：

```
TRUNCATE [TABLE] [IF EXISTS] tbl_name1 [, tbl_name2, ...];
```

说明：

- 允许同时 truncate 多张表，执行成功返回“0 rows affected”，执行失败返回错误码；

- KINGWOW 数据库执行合并操作时不允许 truncate 数据表;

```
kingwow> select * from customer;
+-----+-----+-----+-----+-----+
| id   | name | address | last_login | remark |
+-----+-----+-----+-----+-----+
| 10   | tom  | NULL    | NULL      | N/A    |
| 11   | tom  | NULL    | NULL      | N/A    |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
kingwow> truncate table customer;
Query OK, 0 rows affected (0.02 sec)
```

```
kingwow> select * from customer;
Empty set (0.00 sec)
```

5.1.14. CREATE SEQUENCE

创建序列，语法格式为：

CREATE [OR REPLACE] SEQUENCE seq_name

[AS data_type / AS INTEGER]

[START WITH num_constant / START WITH 1]

[INCREMENT BY num_constant / INCREMENT BY 1]

[MINVALUE num_constant / NO MINVALUE]

[MAXVALUE num_constant / NO MAXVALUE]

[CYCLE / NO CYCLE]

[QUICK / NO QUICK]

选项含义

- OR REPLACE

若新建序列的 seq_name 已经存在，则对该序列进行重新定义；

- AS data_type

指定序列的数据类型，可以为 INTEGER 或 DECIMAL。其中 INTEGER 为 64 位整型，DECIMAL 的范围（precision）为 1~31，精度（scale）必须为 0，默认为 INTEGER；

- START WITH

设定序列的起始值，正数、负数和零均可，但必须为整数。设定的起始值可

以不在最大值和最小值所限定的范围之内。若创建序列时没有指定最大值或最小值，则对于递增序列来说，其最小值等于起始值；对于递减序列来说，其最大值等于起始值；默认值为 1；

- INCREMENT BY

设定序列每次自增或自减的步长。步长为正数时是递增序列，步长为负数时是递减序列，步长为 0 时是常数序列，默认值为 1；

- MINVALUE 或 NO MINVALUE

设定序列的最小值。若创建序列时未指定 MINVALUE 或设定为 NO MINVALUE，则采用默认最小值，递增序列的默认最小值为 START WITH 选项所设定的值，递减序列的默认最小值为数据类型所能表示的最小值；

- MAXVALUE 或 NO MAXVALUE

设定序列的最大值。若创建序列时未指定 MAXVALUE，或设定为 NO MAXVALUE，则采用默认最大值，递增序列的默认最大值为数据类型所能表示的最大值，递减序列的默认最大值为 START WITH 选项所设定的值；

- CYCLE 或 NO CYCLE

设定序列到达边界（即 MINVALUE 与 MAXVALUE 所确定的范围）后是否重新开始循环，默认为 NO CYCLE，表示当序列到达边界后无法再继续生成新的值；

- QUICK 或 NO QUICK

设定会话使用序列的方式，独占（NO QUICK）或非独占（QUICK），默认为 NO QUICK。在 NO QUICK 方式下，同一时刻只能有一个会话独占使用该序列，其它会话处于等待状态；在 QUICK 方式下，所有会话可以同时使用该序列，但不同会话获取的序列值可能出现重复情况；QUICK 方式能够提供更好的性能；

说明：

- 仅支持 NEXTVAL FOR seq_name 和 PREVVAL FOR seq_name 方式使用；
- 在对用户授予 SEQUENCE 功能权限时，该用户即可对所有序列进行修改与删除等操作；
- 如果创建序列时未指定 QUICK 选项，调用 NEXTVAL 后就代表序列生成的值已经被消费，不会再次生成该值（CYCLE 与 RESTART 场景除外）；如果创建序列时指定 QUICK 选项，在 insert 或 select 语句中调用 NEXTVAL 所获取的值，在该语句执行失败的场景下不会被计入序列，即再次调用 NEXTVAL 时仍然会生成该值；

✧ 例如：未指定 QUICK 选项时，使用序列的 NEXTVAL 作为主键值向数据表中插入记录，如果该值已经作为主键存在，则插入语句失败，但序列认为该值已经被使用，再次执行这条插入语句，序列会生成新的 NEXTVAL 值；在指定 QUICK 选项时，只有插入语句成功后，序列才认为 NEXTVAL 的值被使用。

- 目前 KINGWOW 数据库的序列不支持 ORDER 选项和 CACHE 选项；
- 目前 KINGWOW 数据库的序列是面向所有会话有效，不支持面向单个会话的序列；
- 不支持在 CASE 表达式中使用序列；
- 不支持在聚集函数的参数列表中使用序列；
- 不支持在 JOIN 操作的连接条件中使用序列；
- 不支持在 SELECT 语句的 ORDER BY 子句中使用序列；
- 使用序列时，查询 NEXTVAL 的操作必须在查询 PREVVAL 之前；

```
kingwow> create or replace sequence seq1 as integer start with 1 increment by 1 minvalue -1 maxvalue
2 cycle;
```

```
Query OK, 1 row affected (0.01 sec)
```

```
kingwow> select prevval for seq1, nextval for seq1;
```

```
ERROR 65535 (HY000): kw--1: THE PREVVAL expression of sequence [seq1] can't be used before using
NEXTVAL
```

```
kingwow> select nextval for seq1;
```

```
+-----+
```

```
| nextval for seq1 |
```

```
+-----+
```

```
| 2 |
```

```
+-----+
```

```
1 row in set (0.03 sec)
```

5.1.15. ALTER SEQUENCE

修改序列，语法格式为：

```
ALTER SEQUENCE seq_name
```

```
[RESTART / RESTART WITH num_constant] [INCREMENT BY num_constant]
```

```
[MINVALUE num_constant / NO MINVALUE]
```

```
[MAXVALUE num_constant / NO MAXVALUE]
```

```
[CYCLE / NO CYCLE] [QUICK / NO QUICK]
```

选项含义：

- RESTART

重新设定序列生成新值的开始位置，若指定 RESTART 选项，则从创建序列时 START WITH 选项设置的值开始；若指定 RESTART WITH num_constant 选项，则从 num_constant 开始；

- 其它选项

其它选项的含义与创建序列语句的选项含义保持一致。在修改序列时，未指定的选项保持不变，与创建时的设定一致。

说明：

- 在指定 RESTART WITH num_constant 选项时，将会重置序列的当前值，而不会修改创建序列时指定的 START WITH 的值；如需改变 START WITH 的设定值，只能通过 CREATE OR REPLACE SEQUENCE 命令进行修改。

5.1.16. DROP SEQUENCE

删除序列，语法格式为：

```
DROP SEQUENCE seq_name RESTRICT
```

选项含义：

- RESTRICT

检查是否有其它数据库对象依赖于待删除的序列，若有则禁止删除。未指定该选项则可以直接删除序列。

```
kingwow> drop sequence seq1 ;
Query OK, 1 row affected (0.02 sec)

kingwow> select nextval for seq1;
ERROR 65535 (HY000): kw--1: the sequence::[seq1] you input is not exist,please check!
```

5.1.17. ALTER VIEW

修改视图，语法格式为：

```
ALTER VIEW relation_factor [opt_view_expr_list] AS select_stmt
```

注：修改 view 时，以上选项均为可选项，对于用户未输入的参数均采取默认值选项，具体的默认选项见选项含义。

OPT_VIEW_EXPR_LIST

此选项无默认值，该选项指定视图列名，其中列名个数应与 `select_stmt` 中查询结果集的列数相同。若该选项未输入参数，视图列名同 `select_stmt` 中查询结果集的列名。

示例：

```
kingwow> drop table if exists t1;
```

```
Query OK, 0 rows affected (1.60 sec)
```

```
[DDL]DROP_TABLE SUCCESS.
```

```
kingwow> create table t1(c1 int, c2 int, c3 int);
```

```
Query OK, 0 rows affected (1.60 sec)
```

```
[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> insert into t1 values(1,1,1),(2,2,22);
```

```
Query OK, 2 rows affected (0.01 sec)
```

```
[DML]INSERT SUCCESS.
```

```
kingwow> create view v1 as select * from t1;
```

```
Query OK, 0 rows affected (0.63 sec)
```

```
[DDL]CREATE_VIEW SUCCESS.
```

```
kingwow> select * from v1;
```

```
+-----+-----+-----+
```

```
| c1 | c2 | c3 |
```

```
+-----+-----+-----+
```

```
| 1 | 1 | 1 |
```

```
| 2 | 2 | 22 |
```

```
+-----+-----+-----+
```

```
2 rows in set (0.01 sec)
```

```
kingwow> create view v11 as select * from t1;
```

```
Query OK, 0 rows affected (0.63 sec)
```

```
[DDL]CREATE_VIEW SUCCESS.
```

```
kingwow> select * from v11;
```

```
+-----+-----+-----+
```

```
| c1 | c2 | c3 |
```

```
+-----+-----+-----+
```



```
| 1 | 1 | 1 |  
| 2 | 2 | 22 |  
+-----+-----+-----+  
2 rows in set (0.01 sec)
```

```
kingwow> alter view v11 as select c2,c3 from t1;  
Query OK, 0 rows affected (0.62 sec)  
[DDL]ALTER VIEW SUCCESS.
```

```
kingwow> select * from v11;  
+-----+-----+  
| c2 | c3 |  
+-----+-----+  
| 1 | 1 |  
| 2 | 22 |  
+-----+-----+  
2 rows in set (0.01 sec)
```

```
kingwow> drop view v1,v11;  
Query OK, 0 rows affected (2.60 sec)  
[DDL]DROP_VIEW SUCCESS.
```

```
kingwow> show tables;  
+-----+  
| tables |  
+-----+  
| __3008__idx__unqidx2 |  
| sbtest |  
| t1 |  
| t2 |  
| t3 |  
+-----+  
5 rows in set (0.00 sec)
```

5.2. 数据操作语句

5.2.1. INSERT

向数据表中添加一条或多条数据记录，新插入的数据记录必须包含所有主键列的值。语法格式为：

- 插入一条或多条记录

```
INSERT INTO tbl_name[(col_name, ...)] VALUES (col_value, ...)[, (col_value, ...), ...];
```

- 插入子查询的结果集

```
INSERT INTO tbl_a[(a1, a2, ..., am)] SELECT b1, b2, ..., bm FROM tbl_b WHERE ...;
```

- 分批次插入子查询的结果集（不能保证语句执行的原子性）

```
INSERT INTO tbl_a[(a1, a2, ..., am)]
```

```
SELECT /*+I_MULTI_BATCH*/ b1, b2, ..., bm FROM table_b WHERE ...;
```

- 在插入语句中引用序列的值

```
INSERT INTO tbl_name(col1, ...) VALUES(NEXTVAL/PREVVAL FOR seq_name, ...)
```

说明：

- 在插入语句中使用子查询时，子查询的输出列数应当等于插入语句所指定的待插入列数，并且对应列的数据类型必须兼容，对应于主键列的值不能为空；
- 目前 KINGWOW 默认每个事务所产生的事务日志大小上限也是 32M（可参数配置）。这个限制决定了执行批量插入时，如果插入数据量超过 32M，则需要使用多批次插入策略，每个批次作为一个事务执行。这种策略会把一条插入语句拆分成多个事务执行，不保证插入语句执行的原子性；因此 KINGWOW 数据库提供了两种 INSERT ... SELECT ... 语句模式，一种模式可以保证语句执行的原子性，但只允许插入 32M 以内的数据量，如果超出则报错；另一种模式通过 hint `/*+I_MULTI_BATCH*/` 指定采用多批次插入策略，允许插入超过 32M 的数据量，但不保证语句执行的原子性；如果某一批次插入不成功，则这一批次前面的数据都会插入成功，并会返回成功插入了几批数据以及一共插入多少条数据；
- 在插入语句中可以使用序列产生的值。需要注意的是，在一条插入语句中同一个序列的所有的 PREVVAL 返回的都是同一个值。

```
kingwow> create or replace sequence cust_id as integer start with 1 increment by 1;
Query OK, 1 row affected (0.02 sec)
kingwow> insert into customer values(nextval for cust_id,'Jerry','Disney Avenue',now());
Query OK, 1 row affected (0.08 sec)
kingwow> insert into customer values(nextval for cust_id,'Jerry','Disney Avenue',now()),
-> (nextval for cust_id,'Alice','Disney Avenue',now()),
-> (nextval for cust_id,'Bob','Disney Avenue',now());
Query OK, 3 rows affected (0.09 sec)
kingwow> select * from customer;
```

id	name	address	last_login
1	Tom	Disney Avenue	2018-05-13 13:29:42.200672
2	Jerry	Disney Avenue	2018-05-13 13:29:42.200672



```

| 3 | Alice | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 4 | Bob   | Disney Avenue | 2018-05-13 13:29:42.200672 |
+-----+-----+-----+-----+
4 rows in set (0.01 sec)

```

5.2.2. REPLACE

向数据表中添加一条数据记录，与 INSERT 语句的区别在于，如果数据记录的主键已经存在于数据表中，则修改对应列的值为新值，如不存在则插入该条数据记录。语法格式为：

```
REPLACE INTO tbl_name [(col_name, ...)] VALUES (col_value, ...);
```

说明：

- 目前 REPLACE 语句中不支持使用序列；
- 与 INSERT 语句不同，REPLACE 语句在执行时直接将数据从 CG 发送给 TG，不会向 DG 请求数据来判断是否存在主键冲突，因此 REPLACE 语句的性能要优于 INSERT 语句；

```

kingwow> select * from customer;
+-----+-----+-----+-----+
| id | name | address      | last_login      |
+-----+-----+-----+-----+
| 1 | Tom  | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 2 | Jerry | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 3 | Alice | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 4 | Bob   | Disney Avenue | 2018-05-13 13:29:42.200672 |
+-----+-----+-----+-----+
4 rows in set (0.01 sec)

```

```

kingwow> select * from customer;
+-----+-----+-----+-----+
| id | name | address      | last_login      |
+-----+-----+-----+-----+
| 1 | Micky | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 2 | Jerry | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 3 | Alice | Disney Avenue | 2018-05-13 13:29:42.200672 |
| 4 | Bob   | Disney Avenue | 2018-05-13 13:29:42.200672 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

5.2.3. UPDATE

修改数据表中符合条件数据记录的列值，语法格式为：

```
UPDATE [/*+UD_MULTI_BATCH*/] tbl_name
```

```
SET col1 = col_value1 [col2=col_value2] WHERE where_condition;
```

说明：

- 通过 hint `/*+UD_MULTI_BATCH*/` 可以指定采用多批次更新策略，允许更新超过 32M 的数据量，但不保证语句执行的原子性；更新大量数据时，需要根据机器性能调整超时时间，避免语句因执行超时而失败；

- KINGWOW 数据库只支持对单表进行更新操作，在 where_condition 中可以使用序列的 PREVVAL 值，但不能使用序列的 NEXTVAL 值；

```
kingwow> update customer set address='New York',last_login=now() where id=2;
Query OK, 1 row affected (0.07 sec)
```

5.2.4. DELETE

删除数据表中符合条件的数据记录，语法格式为：

```
DELETE [/*+UD_MULTI_BATCH*/] FROM tbl_name [WHERE where_condition];
```

说明：

- 通过 hint `/*+UD_MULTI_BATCH*/` 可以指定采用多批次删除策略，允许删除超过 32M 的数据量，但不保证语句执行的原子性；删除大量数据时，需要根据机器性能调整超时时间，避免语句因执行超时而失败；
- KINGWOW 数据库只支持对单表进行删除操作，在 where_condition 中可以使用序列的 PREVVAL 值，但不能使用序列的 NEXTVAL 值；

```
kingwow> delete from customer where id=2;
Query OK, 1 row affected (0.07 sec)
```

5.2.5. SELECT

查询数据表中的内容，语法格式为：

```
SELECT [ ALL | DISTINCT ] select_list [AS other_name]
FROM table_name
[WHERE where_conditions ]
[GROUP BY group_by_list ]
[HAVING search_confitions ]
[ORDER BY order_list [ASC | DESC ] ]
[Connect By REGEXP_SUBSTR(str, '^[^,]+' ,1, Level)]
[LIMIT {[offset,] row_count | row_count OFFSET offset};
```

说明：

- 目前 KINGWOW 不支持在 SELECT 查询语句的 where 子句中使用列别名，但是支持在 GROUP BY ,ORDER BY, HAVING 子句中使用列别名

- SELECT 查询语句的 from 子句中出现多张表或子查询结果集时，每张表（或子查询结果集）都至少根据一个等值连接条件与其他表（或子查询结果集）连接。
- LIMIT 子句强制 SELECT 语句返回指定的记录数, 参数 row_count 和 offset 必须是整数常量, row_count 指定返回的记录数, offset 指定返回记录的偏移量, 初始记录的偏移量是 0（不是 1）；

```
kingwow> select id,name from customer order by id desc;
```

```
+-----+-----+
| id | name |
+-----+-----+
|  4 | Bob  |
|  3 | Alice|
|  2 | Jerry|
|  1 | Micky|
+-----+-----+
4 rows in set (0.01 sec)
```

```
kingwow> select id,name from customer limit 2;
```

```
+-----+-----+
| id | name |
+-----+-----+
|  1 | Micky|
|  2 | Jerry|
+-----+-----+
2 rows in set (0.01 sec)
```

```
kingwow> select id,name from customer limit 1,2;
```

```
+-----+-----+
| id | name |
+-----+-----+
|  2 | Jerry|
|  3 | Alice|
+-----+-----+
2 rows in set (0.00 sec)
```

Connect By REGEXP_SUBSTR() 用法:

```
drop table if exists t1;
create table t1(id int primary key, c1 VARCHAR2(100));
insert into t1 values (1, '601,602,603,6055');
Select REGEXP_SUBSTR(c1, '^[^,]+' ,1,Level) From t1 c Connect By REGEXP_SUBSTR(c1, '^[^,]+' ,1, Level)
Is Not Null;
```

```
kingwow> select REGEXP_SUBSTR(c1, '^[^,]+' ,1,Level) From t1 c Connect By REGEXP_SUBSTR(c1, '^[^,]+' ,1,
Level) Is Not Null;
```

```
+-----+-----+
| REGEXP_SUBSTR(c1, '^[^,]+' ,1,Level) |
+-----+-----+
| 601 |
| 602 |
| 603 |
| 6055|
+-----+-----+
```

```
4 rows in set (0.01 sec)
```



● select(左关联、右关联、全关联、UNION 查询、group by+limit、聚集函数+having+order by、order by+limit、多层 FROM 嵌套子查询、标量子查询、any/some/all 子查询、row 子查询、绑定变量查询、多表 join 复杂查询)

```
drop table if exists t1;
```

```
create table t1 (id int not null,name varchar(20));
```

```
insert into t1 values (1,'t1a'),(2,'t1b'),(3,'t1c'),(4,'t1d'),(5,'t1f'),(666,'t1'),(888,'t2');
```

```
drop table if exists t2;
```

```
create table t2 (id int not null,name varchar(20));
```

```
insert into t2 values (2,'t2b'),(3,'t2c'),(4,'t2d'),(5,'t2f'),(6,'t2a'),(666,'t1');
```

#左关联

```
select * from t1 left join t2 on t1.id = t2.id;
```

#右关联

```
select * from t1 right join t2 on t1.id = t2.id;
```

#内关联

```
select * from t1 inner join t2 on t1.id = t2.id;
```

#左外连接

```
select * from t1 left join t2 on t1.id = t2.id where t2.id is null;
```

#右外连接

```
select * from t1 right join t2 on t1.id = t2.id where t1.id is null;
```

#全关联

```
select * from t1 full join t2 on t1.id = t2.id;
```

#全表连接 等价 mysql 写法 union 联合

```
select * from t1 left join t2 on t1.id = t2.id
```

```
union
```

```
select * from t1 right join t2 on t1.id = t2.id;
```

#group by+limit

```
select * from t1 group by name limit 2;
```

#聚集函数+having+order by

```
select id,name from t1 where name='t1' group by id having min(id) < max(id) order by name;
```

#order by+limit

```
select id,name from t1 order by id limit 3;
```



#多层 FROM 嵌套子查询

```
select * from t1 WHERE name=(select name from t1 where name='t1');
```

#标量子查询

```
select max(id) from t1;
```

#row 子查询

```
select * from t1 WHERE name=(select name from t1 where name='t2');
```

#绑定变量查询

详情见 prepare 章节

#多表 join 复杂查询

```
select * from t1 left join t2 on t1.id = t2.id where t2.id is null
```

```
union
```

```
select * from t1 right join t2 on t1.id = t2.id where t1.id is null;
```

#any/some/all 子查询

```
drop table if exists t1;
```

```
create table t1(id int, value int, primary key(id));
```

```
insert into t1 values(1,10),(2,20),(3,30);
```

```
select * from t1 where value > any(select value from t1 where id >=2);
```

```
select * from t1 where value > all(select value from t1 where id >=2);
```

```
select * from t1 where value < any(select value from t1 where id >=2);
```

```
select * from t1 where value < all(select value from t1 where id >=2);
```

```
select * from t1 where value >= any(select value from t1 where id >=2);
```

```
select * from t1 where value >= all(select value from t1 where id >=2);
```

```
select * from t1 where value <= any(select value from t1 where id >=2);
```

```
select * from t1 where value <= all(select value from t1 where id >=2);
```

5.2.6. JOIN

KINGWOW 数据库支持在查询语句中显式指定关系表连接操作的方式，包括内连接（INNER JOIN）、左外连接（LEFT JOIN）、右外连接（RIGHT JOIN）和全外连接。语法格式为：

```
SELECT expr_list FROM tbl1 INNER JOIN tbl2 ON join_condition;
```

```
kingwow> select * from customer c inner join emp e on c.id = e.empno;
```

id comm	name deptno	address	last_login	empno	ename	job	mgr	hiredate	sal
7369 NULL	SMITH 20	New York	NULL	7369	SMITH	CLERK	7902	1980-12-17	800.00
7654 1400.00	MARTIN 30	NULL	NULL	7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00
7788 NULL	SCOTT 20	New York	NULL	7788	SCOTT	ANALYST	7566	1982-12-09	3000.00

3 rows in set (0.00 sec)

```
SELECT expr_list FROM tbl1 LEFT OUTER JOIN tbl2 ON join_condition;
```

```
kingwow> select * from customer c left outer join emp e on c.id = e.empno;
```

id comm	name deptno	address	last_login	empno	ename	job	mgr	hiredate	sal
7369 NULL	SMITH 20	New York	NULL	7369	SMITH	CLERK	7902	1980-12-17	800.00
7654 1400.00	MARTIN 30	NULL	NULL	7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00
7788 NULL	SCOTT 20	New York	NULL	7788	SCOTT	ANALYST	7566	1982-12-09	3000.00

3 rows in set (0.00 sec)

```
SELECT expr_list FROM tbl1 OUTER JOIN tbl2 ON join_condition;
```

```
kingwow> select * from customer c left outer join emp e on c.id = e.empno;
```

id comm	name deptno	address	last_login	empno	ename	job	mgr	hiredate	sal
7369 NULL	SMITH 20	New York	NULL	7369	SMITH	CLERK	7902	1980-12-17	800.00
7654 1400.00	MARTIN 30	NULL	NULL	7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00
7788 NULL	SCOTT 20	New York	NULL	7788	SCOTT	ANALYST	7566	1982-12-09	3000.00

3 rows in set (0.00 sec)

```
SELECT expr_list FROM tbl1 FULL OUTER JOIN tbl2 ON join_condition;
```

```
kingwow> select * from customer c left outer join emp e on c.id = e.empno;
```

id comm	name deptno	address	last_login	empno	ename	job	mgr	hiredate	sal
------------	----------------	---------	------------	-------	-------	-----	-----	----------	-----

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 7369 | SMITH | New York | NULL | 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 |
| NULL | 20 | | | | | | | | |
| 7654 | MARTIN | NULL | NULL | 7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00 |
| 1400.00 | 30 | | | | | | | | |
| 7788 | SCOTT | New York | NULL | 7788 | SCOTT | ANALYST | 7566 | 1982-12-09 | 3000.00 |
| NULL | 20 | | | | | | | | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

说明：

- 在连接条件 `join_condition` 中，必须包含一个涉及两表的等值连接条件，并且左表应出现在等值表达式的左侧，如 `tbl1.col1=tbl2.col2`；

5.2.7. UNION

对两条 SELECT 语句的执行结果进行集合并运算。语法格式为：

```
select_statement1 UNION [ALL] select_statement2;
```

说明：

- 两条 SELECT 语句的执行结果应当具有相同的列数，并且对应列的数据类型应当相互兼容；
- UNION 操作缺省将会去除结果集中重复记录，通过使用 UNION ALL 可以保留重复记录；

5.2.8. INTERSECT

对两条 SELECT 语句的执行结果进行集合交运算。语法格式为：

```
select_statement1 INTERSECT [ALL] select_statement2;
```

说明：

- 两条 SELECT 语句的执行结果应当具有相同的列数，并且对应列的数据类型应当相互兼容；
- INTERSECT 操作缺省将会去除结果集中重复记录，通过使用 INTERSECT ALL 可以保留重复记录；

5.2.9. EXCEPT

对两条 SELECT 语句的执行结果进行集合差运算。语法格式为：

```
select_statement1 EXCEPT [ALL] select_statement2;
```

说明：

- 两条 SELECT 语句的执行结果应当具有相同的列数，并且对应列的数据类型应当相互兼容；
- EXCEPT 操作缺省将会去除结果集中重复记录，通过使用 EXCEPT ALL 可以保留重复记录；

5.2.10. SELECT ... FOR UPDATE

加排它锁读取数据表中的数据记录，语法格式为：

```
select_statement1 FOR UPDATE;
```

说明：

- 在事务中使用加锁读时，所读取的数据记录不能被其它事务修改，直到本事务结束；
- 加锁读只能针对单表查询，并且查询语句的结果集只能包含一条记录；

```
kingwow> begin;
Query OK, 0 rows affected (0.00 sec)

kingwow> select * from customer where id = 7369 for update;
+-----+-----+-----+-----+
| id   | name  | address | last_login |
+-----+-----+-----+-----+
| 7369 | SMITH | New York | NULL      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

5.2.11. PREPARE

预备执行语句，语法格式为：

```
PREPARE stmt_name FROM prepare_statement;
```

说明：

- prepare_statement 是预备执行的数据操作语句，在整个会话期间都可以 stmt_name 来执行预备好的语句；
- 在预备执行的数据操作语句中可以使用占位符'?'来标识执行时需要绑定的参数；

5.2.12. EXECUTE

执行已经预备好的数据操作语句，语法格式为：

```
EXECUTE stmt_name [USING @var_name1 [,@var_name2,...]];
```

说明：

- 如果预备好的语句中有占位符标识的参数，需要使用 USING 子句提供相同个数的执行时绑定值，所绑定的值必须为 SET 语句定义的用户变量；

5.2.13. DEALLOCATE

删除预备好的数据操作语句，语法格式为：

```
{DEALLOCATE | DROP} PREPARE stmt_name;
```

```
kingwow> prepare stmt1 from select id,name from customer where id=?;  
Query OK, 0 rows affected (0.00 sec)
```

```
kingwow> set @id=7369;  
Query OK, 0 rows affected (0.00 sec)
```

```
kingwow> execute stmt1 using @id;
```

```
+-----+-----+  
| id | name |  
+-----+-----+  
| 7369 | SMITH |  
+-----+-----+  
1 row in set (0.00 sec)
```

```
kingwow> deallocate prepare stmt1;  
Query OK, 0 rows affected (0.00 sec)
```

5.2.14. ROWNUM

kingwow 数据库支持 ROWNUM 语法。

```
kingwow> drop table if exists test1;
```

```
Query OK, 0 rows affected (0.07 sec)
```

```
[DDL]DROP_TABLE SUCCESS.
```

```
kingwow> create table test1(c1 int primary key,c2 int);
```

```
Query OK, 0 rows affected (1.21 sec)
```

```
[DDL]CREATE_TABLE SUCCESS.
```

```
kingwow> insert into test1 values(1,1);
```

```
Query OK, 1 row affected (0.01 sec)
```

```
[DML]INSERT SUCCESS.
```

```
kingwow> insert into test1 values(2,2);
```



Query OK, 1 row affected (0.03 sec)

[DML]INSERT SUCCESS.

kingwow> insert into test1 values(3,2);

Query OK, 1 row affected (0.01 sec)

[DML]INSERT SUCCESS.

kingwow> insert into test1 values(4,2);

Query OK, 1 row affected (0.03 sec)

[DML]INSERT SUCCESS.

kingwow> select rownum, c1 from test1;

+-----+-----+

rownum	c1
--------	----

+-----+-----+

1	1
---	---

2	2
---	---

3	3
---	---

4	4
---	---

+-----+-----+

4 rows in set (0.01 sec)

kingwow> select rownum as rn, c1,c2 from test1;

+-----+-----+-----+

rn	c1	c2
----	----	----

+-----+-----+-----+

1	1	1
---	---	---

2	2	2
---	---	---

3	3	2
---	---	---

4	4	2
---	---	---

+-----+-----+-----+

4 rows in set (0.00 sec)

kingwow> select rownum,c1 from test1 where c2 = 2 and rownum + 1 < 6;

+-----+-----+

rownum	c1
--------	----

+-----+-----+

1	2
---	---

2	3
---	---

3	4
---	---

+-----+-----+

3 rows in set (0.03 sec)

```
kingwow> delete from test1 where rownum < 3;

Query OK, 2 rows affected (0.06 sec)

[DML]DELETE SUCCESS.

kingwow> select rownum, c1 from test1;

+-----+-----+
| rownum | c1   |
+-----+-----+
|      1 |    3 |
|      2 |    4 |
+-----+-----+

2 rows in set (0.00 sec)
```

5.2.15. MERGE INTO

判断表中有没有符合 on () 条件中的数据, 有了就更新数据, 没有就插入数据。语法格式为:

```
MERGE INTO [目标表] [别名 A]
USING (
    -- 条件为真的查询语句
    SELECT ...
    FROM ...
    WHERE ...
) [别名 B]
ON (
    --判断条件
    [别名 A].[字段 1] = [别名 B].[字段 1]
    AND [别名 A].[字段 2] = [别名 B].[字段 2]
    AND [别名 A].[字段 3] = [别名 B].[字段 3]
)
WHEN MATCHED THEN
    --符合条件时
    UPDATE
```

```
SET [别名 A].[字段 4] = [别名 B].[字段 4]
```

```
,[别名 A].[字段 5] = [别名 B].[字段 5] DELETE WHERE ...
```

```
WHEN NOT MATCHED THEN
```

```
--不符合条件时
```

```
INSERT INTO (
```

```
[别名 A].[字段 1]
```

```
,[别名 A].[字段 2]
```

```
,[别名 A].[字段 3]
```

```
,[别名 A].[字段 4]
```

```
,[别名 A].[字段 5]
```

```
) VALUES (
```

```
[别名 B].[字段 1]
```

```
,[别名 B].[字段 2]
```

```
,[别名 B].[字段 3]
```

```
,[别名 B].[字段 4]
```

```
,[别名 B].[字段 5]
```

```
);
```

说明:

- 其中的"WHEN MATCHED" 和 "WHEN NOT MATCHED"是可选的，并非都必须存在的
- 因为第一句已经将要操作的目标表已经定义好了，因此，在后面的 INSERT/UPDATE/DELETE 操作中，不需要再写表名
- 在写 DELETE 的时候，必须前面写 UPDATE SET 并且后面必须要有 WHERE 条件，否则会报错"缺少关键字"的错误

```
kingwow> merge into test_merge1 t1
-> using (
-> select 1 as id, 2 as c1, 3 as c2, 4 as c3, 5 as c4 from dual
-> union
-> select 11 as id, 12 as c1, 13 as c2, 14 as c3, 15 as c4 from dual
-> union
-> select 21 as id, 22 as c1, 23 as c2, 24 as c3, 25 as c4 from dual
-> union
-> select 31 as id, 32 as c1, 33 as c2, 34 as c3, 35 as c4 from dual
-> union
```



```
-> select 41 as id, 42 as c1, 43 as c2, 44 as c3, 45 as c4 from dual
-> union
-> select 51 as id, 52 as c1, 53 as c2, 54 as c3, 55 as c4 from dual
-> union
-> select 61 as id, 62 as c1, 63 as c2, 64 as c3, 65 as c4 from dual ) t2
-> on (t1.id = t2.id and t1.c1 = t2.c1 and t1.c2 = t2.c2)
-> when matched then
-> update set t1.c3 = t1.c3 + t2.c3, t1.c4 = t1.c4 + t2.c4
-> when not matched then
-> insert(id, c1, c2, c3, c4) values(t2.id, t2.c1, t2.c2, t2.c3, t2.c4);
Query OK, 7 rows affected (0.05 sec)
[DML]MERGE SUCCESS.
```

5.3 表分区

5.3.1 HASH 分区表

创建哈希分区表的格式为：

```
CREATE TABLE 表名称(
    列名 1 数据类型,
    列名 2 数据类型,
    ...
    PRIMARY KEY(列名 1)
) PARTITION BY HASH(列名 1) PARTITIONS n;
```

说明：

- 表名称：要创建的表名。
- 列名 1：用于哈希分区的列。
- n：要创建的分区数。

例：

```
kingwow> CREATE TABLE `test` (
-> `c1` INT
-> , `c2` INT
-> , PRIMARY KEY(`c1`)
-> ) partition by hash(c1)
-> (partition p0,
-> partition p1,
-> partition p2,
-> partition p3,
-> partition p4);
Query OK, 0 rows affected (5.40 sec)
[DDL]CREATE_TABLE SUCCESS.partition p4);
```

5.3.2 LIST 分区表

创建列表分区表的格式为：

```
CREATE TABLE 表名称(  
    列名 1 数据类型,  
    列名 2 数据类型,  
    ...  
    PRIMARY KEY(列名 1)  
) PARTITION BY LIST(列名 1) (  
    PARTITION 分区名 1 VALUES IN (列 1 值 1, 列 1 值 2, ...),  
    PARTITION 分区名 2 VALUES IN (列 1 值 3, 列 1 值 4, ...),  
    ...  
);
```

说明：

- 表名称：要创建的表名。
- 列名 1：用于列表分区的列。
- 分区名 1、分区名 2…：要创建的分区名。
- 列 1 值 1、列 1 值 2…：分区所包含的值。

例：

```
kingwow> CREATE TABLE `test1` (  
    -> `c1` INT  
    -> , `c2` VARCHAR(50)  
    -> , `c3` date  
    -> , PRIMARY KEY(`c1`)  
    -> ) partition by list(c1)  
    -> (partition p1 values in (1, 2, 3),  
    -> partition p2 values in (4, 5, 6),  
    -> partition p3 values in (7, 8, 9));  
Query OK, 0 rows affected (3.50 sec)  
[DDL]CREATE_TABLE SUCCESS.
```

5.3.3 RANGE 分区表

创建范围分区表的格式为：

```
CREATE TABLE 表名称(  
    列名 1 数据类型,
```

列名 2 数据类型,

...

```
PRIMARY KEY(列名 1)
) PARTITION BY RANGE(列名 1) (
PARTITION 分区名 1 VALUES LESS THAN (值 1),
PARTITION 分区名 2 VALUES LESS THAN (值 2),
...
PARTITION 分区名 n VALUES LESS THAN (MAXVALUE)
);
```

说明:

- 表名称: 要创建的表名。
- 列名 1: 用于范围分区的列。
- 分区名 1、分区名 2...: 要创建的分区名。
- 值 1、值 2、值 n: 用于区分分区的值范围。

例:

```
kingwow> CREATE TABLE `test2` (
-> `c1` INT
-> , `c2` VARCHAR(50)
-> , `c3` date
-> , PRIMARY KEY(`c1`)
-> ) partition by range(c1)
-> (partition p1 values less than (100),
-> partition p2 values less than (200),
-> partition p3 values less than (MAXVALUE));
Query OK, 0 rows affected (3.51 sec)
[DDL]CREATE_TABLE SUCCESS.
```

5.3.4 分区管理语句

1. SELECT 分区

SELECT 语句可以选择要查询的分区或子分区中的数据。语法如下:

```
SELECT * FROM 表名 PARTITION (分区名);
```

例:

```
SELECT * FROM tbl_name PARTITION (p1);
```

2. ADD 分区

ADD 分区语句可以向已有的分区表中添加新分区。语法如下:

```
ALTER TABLE 表名 ADD PARTITION (分区定义);
```

例：

```
ALTER TABLE TBL_NAME ADD PARTITION (PARTITION p4 VALUES IN ('2022-01-01'));
```

说明：

其中，p4 为分区名，2022-01-01 为要插入的值。

3. DROP 分区

DROP 分区可以删除指定的分区。语法如下：

```
ALTER TABLE 表名 DROP PARTITION (分区定义);
```

例：

```
ALTER TABLE TBL_NAME DROP PARTITION p4;
```

说明：

其中，p4 为要删除的分区名。

5. DELETE 分区

DELETE 分区可以清空指定的分区的数据。语法如下：

```
DELETE FROM 表名 PARTITION(分区名) WHERE COL_NAME = " ;
```

例：

```
DELETE FROM TBL_NAME PARTITION(p4) WHERE C1 = 1 ;
```

6. TRUNCATE 分区

TRUNCATE 分区可以清空指定的分区的所有数据。语法如下：

```
ALTER TABLE 表名 TRUNCATE PARTITION p4 ;
```

例：

```
ALTER TABLE TBL_NAME TRUNCATE PARTITION p4 ;
```

5.3.5 广播表

1. 创建广播表：

创建临时表使用 broadcast，语法如下：



```
CREATE TABLE 表名 (  
列名 1 数据类型 1,  
列名 2 数据类型 2,  
) BROADCAST;
```

说明:

- 使用 print_mg_table 查看该表是否在所有机器上都有一个副本

例:

```
kingwow> create table t_broadcast(c1 int, c2 int) broadcast;  
Query OK, 0 rows affected (1.61 sec)  
[DDL]CREATE_TABLE SUCCESS
```

5.3.6 临时表

1. 创建临时表

创建临时表使用 CREATE TEMPORARY TABLE 语句，语法如下:

```
CREATE TEMPORARY TABLE 表名 (  
列名 1 数据类型 1,  
列名 2 数据类型 2,  
...  
);
```

例:

```
kingwow> create TEMPORARY TABLE temp_table(id int);  
Query OK, 0 rows affected (0.04 sec)  
[DDL]CREATE_TABLE SUCCESS.
```

5.3.7 二级分区表

创建表及进行分区:

```
CREATE TABLE t2_m_rr(col1 INT PRIMARY KEY,col2 INT) PARTITION BY RANGE(col1)  
SUBPARTITION BY RANGE(col2) SUBPARTITION TEMPLATE (SUBPARTITION mp0 VALUES  
LESS THAN(2020), SUBPARTITION mp1 VALUES LESS THAN(2021), SUBPARTITION mp2  
VALUES LESS THAN(2022) ) (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES  
LESS THAN(200) );
```

```
insert into t2_m_rr values(1,1);
```

```
CREATE TABLE t2_m_rl(col1 INT PRIMARY KEY,col2 VARCHAR2(50)) PARTITION BY  
RANGE(col1) SUBPARTITION BY LIST(col2) SUBPARTITION TEMPLATE (SUBPARTITION mp0
```

```
VALUES IN('01'), SUBPARTITION mp1 VALUES IN('02'), SUBPARTITION mp2 VALUES IN('03') )
(PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(200) );
```

```
insert into t2_m_rl values(1,'01');
```

```
CREATE TABLE t2_m_rh(col1 INT PRIMARY KEY,col2 VARCHAR2(50)) PARTITION BY
RANGE(col1) SUBPARTITION BY HASH(col2) SUBPARTITION TEMPLATE SUBPARTITIONS 5
(PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(200) );
```

```
insert into t2_m_rh values(1,'01');
```

```
CREATE TABLE t2_m_lr(col1 INT PRIMARY KEY,col2 INT) PARTITION BY LIST(col1)
SUBPARTITION BY RANGE(col2) SUBPARTITION TEMPLATE (SUBPARTITION mp0 VALUES
LESS THAN(100), SUBPARTITION mp1 VALUES LESS THAN(200), SUBPARTITION mp2
VALUES LESS THAN(300) ) (PARTITION p0 VALUES IN(1), PARTITION p1 VALUES IN(2) );
```

```
insert into t2_m_lr values(1,99);
```

```
CREATE TABLE t2_m_ll(col1 INT PRIMARY KEY,col2 varchar2(50)) PARTITION BY
LIST(col1) SUBPARTITION BY LIST(col2) SUBPARTITION TEMPLATE (SUBPARTITION mp0
VALUES IN('A'), SUBPARTITION mp1 VALUES IN('B'), SUBPARTITION mp2 VALUES IN('C') )
(PARTITION p0 VALUES IN(1), PARTITION p1 VALUES IN(2) );
```

```
insert into t2_m_ll values(1,'A');
```

```
CREATE TABLE t2_m_lh(col1 INT PRIMARY KEY,col2 VARCHAR2(50)) PARTITION BY
LIST(col1) SUBPARTITION BY HASH(col2) SUBPARTITION TEMPLATE SUBPARTITIONS 5
(PARTITION p0 VALUES IN(1), PARTITION p1 VALUES IN(2) );
```

```
insert into t2_m_lh values(1,'A');
```

```
CREATE TABLE tbl2_m_hr(col1 INT PRIMARY KEY,col2 INT,col3 INT) PARTITION BY
HASH(col1) SUBPARTITION BY RANGE(col2) SUBPARTITION TEMPLATE (SUBPARTITION sp0
VALUES LESS THAN(100), SUBPARTITION sp1 VALUES LESS THAN(200), SUBPARTITION sp2
VALUES LESS THAN(300) ) PARTITIONS 5;
```

```
insert into tbl2_m_hr values(1,55,55);
```

```
CREATE TABLE tbl2_m_hl(col1 INT PRIMARY KEY,col2 INT,col3 INT) PARTITION BY
HASH(col1) SUBPARTITION BY LIST(col2) SUBPARTITION TEMPLATE (SUBPARTITION sp0
```

```
VALUES IN(100), SUBPARTITION sp1 VALUES IN(200), SUBPARTITION sp2 VALUES IN(300) )  
PARTITIONS 5;
```

```
insert into tbl2_m_hl values(1,55,55);
```

```
CREATE TABLE tbl2_m_hh(col1 INT PRIMARY KEY,col2 INT,col3 INT) PARTITION BY  
HASH(col1) SUBPARTITION BY HASH(col2) SUBPARTITION TEMPLATE SUBPARTITIONS 3  
PARTITIONS 5;
```

```
insert into tbl2_m_hh values(1,55,55);
```

```
show create table t2_m_lh;
```

```
show create table t2_m_ll;
```

```
show create table t2_m_lr;
```

```
show create table t2_m_rh;
```

```
show create table t2_m_rl;
```

```
show create table t2_m_rr;
```

```
show create table tbl2_m_hh;
```

```
show create table tbl2_m_hl;
```

```
show create table tbl2_m_hr;
```

二级分区对系统的兼容性影响说明：

- 1、在__all_part_info 表增加`b\_sub\_part\_expr`字段，__all_part_info 为核心表，无法添加字段
- 2、改动了 KwPartition，在 schema 中涉及到序列化反序列化操作，影响升级

因此兼容升级步骤如下：

- 1、删除集群内所有分区表
- 2、合并集群
- 3、集群合并结束后，替换 bin 包
- 4、拉起集群

6. 数据库商密安全

6.1. 网络传输加密

SSL (Secure Sockets Layer) 加密是一种用于在数据库客户端和金乌数据库服务器之间建立安全连接的加密协议。它通过使用 SSL/TLS1.2 协议对通信进行加密和认证，确保数据



在传输过程中的保密性和完整性。

金乌数据库的 SSL 加密连接提供以下主要功能：

加密通信：SSL 加密使得客户端和服务端之间的通信变得安全，防止敏感数据在传输过程中被窃听或篡改。

身份验证：SSL 加密可以确保连接的双方的身份。服务器可以通过使用数字证书来验证自己的身份，而客户端也可以验证服务器的身份。这可以防止中间人攻击（Man-in-the-Middle Attack）。

完整性保护：SSL 加密通过使用消息认证码（Message Authentication Code, MAC）来验证数据的完整性，以防止数据在传输过程中被篡改。

要在金乌数据库中启用 SSL 加密，需要执行以下步骤：

生成证书和密钥：需要使用 SSL 工具（如 OpenSSL）生成服务器的数字证书和私钥。数字证书用于验证服务器的身份，私钥用于对通信进行加密和解密。在数据库安装目录 etc/pem 目录有一套默认的密钥，运维人员也可自行生成。

配置金乌服务器：金乌数据库默认未开启客户端 SSL 加密通信，在数据库服务器启动后，可动态开启和关闭通信加密功能。连接数据库后，开关项的 SQL 语法如下：

```
启用 SSL: alter system set enable_ssl=true server_type=calEng;
关闭 SSL: alter system set enable_ssl=false server_type=calEng;
```

配置客户端：在客户端连接时，需要使用 SSL 选项来指定客户端使用 SSL 加密连接。这可以通过在连接字符串中设置相关选项或者在客户端配置文件中指定。

1. /tmp/pem 目录存放是 SSL 使用的证书，来自于 kingwow 安装目录 etc/pem
2. 使用 mysql 客户端：

使用 MySQL5.5 客户端：

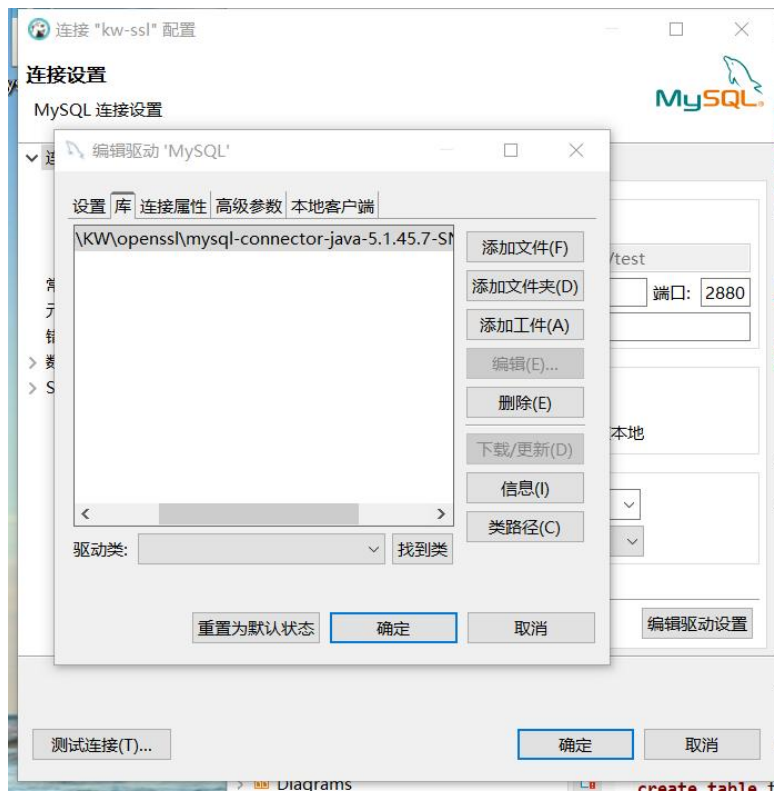
```
mysql -h 127.0.0.1 -P2880 -uadmin -padmin -Dkingwow --ssl=true \
--ssl-ca=/tmp/pem/ca.pem --ssl-cert=/tmp/pem/client-cert.pem --ssl-key=/tmp/pem/client-key.pem
```

使用 MySQL5.7 客户端：

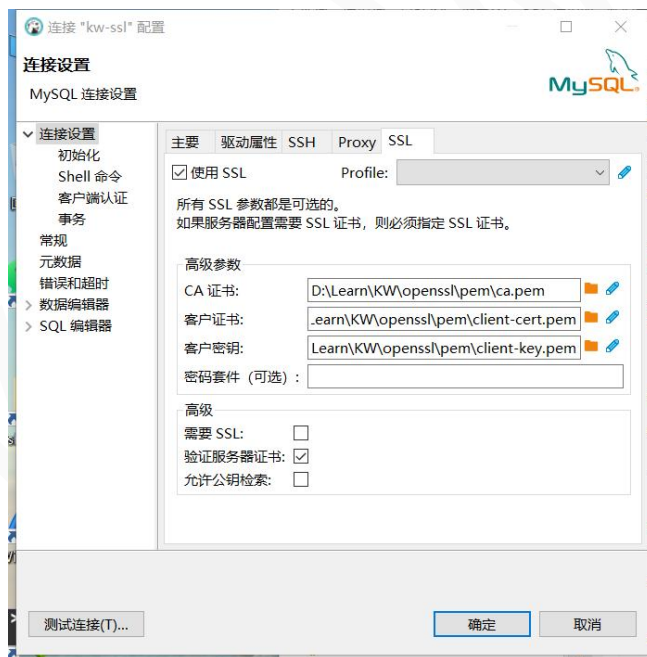
```
mysql -h 127.0.0.1 -P2880 -uadmin -padmin -Dkingwow --ssl-mode=VERIFY_CA --ssl-ca=/tmp/pem/ca.pem
--ssl-cert=/tmp/pem/client-cert.pem --ssl-key=/tmp/pem/client-key.pem
```

3. JDBC 使用 SSL，以 DBeaver 为例：

- 1) 修改连接配置，在驱动属性中打开 SSL，配置 requireSSL=true, useSSL=true，如下：



2) 证书配置, 把 Kingwow 安装目录中 etc/pem 目录里的 ca.pem, client-cert.pem, client-key.pem 取到客户端本地, 配置证书路径如图:



3) 测试连接 登录成功, 正常连数据库可以数据, 通过 tcpdump 抓包能看到, 通信是加密的:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.212.134.17	192.168.100.141	TCP	74	61326 → 4880 [SYN] Seq=0 Win=0 Len=0 MSS=1314 SACK_PERM TSval=273175297 TSecr=0 MS=512
2	0.000000	192.168.100.141	10.212.134.17	TCP	74	4880 → 61326 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460 SACK_PERM TSval=1306120631 TSecr=273175297 MS=512
3	0.005041	10.212.134.17	192.168.100.141	TCP	66	61326 → 4880 [ACK] Seq=1 Ack=1 Win=2624 Len=0 TSval=273175304 TSecr=1306120631
4	0.010141	192.168.100.141	10.212.134.17	MySQL	122	Server Greeting proto=10 version=5.7.32
5	0.021845	10.212.134.17	192.168.100.141	TCP	66	61326 → 4880 [ACK] Seq=1 Ack=57 Win=2624 Len=0 TSval=273175318 TSecr=1306120642
6	0.022223	10.212.134.17	192.168.100.141	MySQL	182	Login Request user=
7	0.022256	192.168.100.141	10.212.134.17	TCP	66	4880 → 61326 [ACK] Seq=57 Ack=37 Win=29184 Len=0 TSval=1306120653 TSecr=273175318
8	0.022425	10.212.134.17	192.168.100.141	TLSv1.2	293	Client Hello
9	0.022470	192.168.100.141	10.212.134.17	TCP	66	4880 → 61326 [ACK] Seq=57 Ack=264 Win=30208 Len=0 TSval=1306120654 TSecr=273175319
10	0.024916	192.168.100.141	10.212.134.17	TLSv1.2	2637	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
11	0.034856	10.212.134.17	192.168.100.141	TCP	66	61326 → 4880 [ACK] Seq=264 Ack=2628 Win=31744 Len=0 TSval=273175330 TSecr=1306120656
12	0.035054	10.212.134.17	192.168.100.141	TCP	1368	61326 → 4880 [ACK] Seq=264 Ack=2628 Win=31744 Len=0 TSval=273175332 TSecr=1306120656 [TCP segment of a reassembled PDU]
13	0.035171	10.212.134.17	192.168.100.141	TLSv1.2	1380	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
14	0.035193	192.168.100.141	10.212.134.17	TCP	66	4880 → 61326 [ACK] Seq=2028 Ack=2606 Win=35328 Len=0 TSval=1306120666 TSecr=273175332
15	0.035926	192.168.100.141	10.212.134.17	TLSv1.2	1284	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
16	0.041897	10.212.134.17	192.168.100.141	TLSv1.2	346	Application Data
17	0.046521	192.168.100.141	10.212.134.17	TLSv1.2	386	Application Data
18	0.081105	10.212.134.17	192.168.100.141	TLSv1.2	114	Application Data
19	0.084109	192.168.100.141	10.212.134.17	TLSv1.2	204	Application Data
20	0.090926	10.212.134.17	192.168.100.141	TLSv1.2	111	Application Data
21	0.091528	192.168.100.141	10.212.134.17	TLSv1.2	189	Application Data
22	0.095908	10.212.134.17	192.168.100.141	TLSv1.2	183	Application Data
23	0.100141	192.168.100.141	10.212.134.17	TLSv1.2	128	Application Data
24	0.100868	10.212.134.17	192.168.100.141	TLSv1.2	112	Application Data
25	0.100814	192.168.100.141	10.212.134.17	TLSv1.2	228	Application Data
26	0.155214	10.212.134.17	192.168.100.141	TCP	66	61326 → 4880 [ACK] Seq=2902 Ack=4321 Win=39424 Len=0 TSval=273175454 TSecr=1306120739
27	0.472632	10.212.134.17	192.168.100.141	TLSv1.2	114	Application Data
28	0.675706	192.168.100.141	10.212.134.17	TLSv1.2	204	Application Data
29	1.081479	10.212.134.17	192.168.100.141	TCP	66	61326 → 4880 [ACK] Seq=2950 Ack=4459 Win=41804 Len=0 TSval=273180979 TSecr=1306126307
30	6.711664	10.212.134.17	192.168.100.141	TLSv1.2	180	Application Data

6.2. 存储透明加密

数据库存储透明加密（Transparent Data Encryption）是一种将数据库中的数据在存储过程中进行加密的安全机制。金乌数据库通过存储引擎层对数据进行加密和解密，使得数据在磁盘上存储时是加密的状态，对应用程序和数据库用户来说是透明的。

金乌数据库存储透明加密提供以下功能和特点：

数据加密：数据库存储透明加密将数据在存储过程中进行加密，确保数据在磁盘上的安全性。加密基于对称密钥加密算法，支持算法如 AES、国密 SM4。

透明性：数据库存储透明加密对应用程序和数据库用户透明，他们可以像处理明文数据一样对待加密的数据，无需修改现有应用程序代码。

密钥管理：数据库存储透明加密需要对密钥进行安全管理，支持数据库管理员配置密钥文件。

性能影响：数据库存储透明加密可能会引入一定的性能开销，包括加密和解密的计算开销以及存储空间的额外开销。金乌数据库通过缓存解密数据块的方案，优化存储加解密对性能的影响，对于热数据几乎无影响，对于首次访问的冷数据，通过并行读取的方式提高加密数据的读取性能。

金乌数据库的存储透明加密提供以下主要功能：

1. 存储透明加密密钥管理：

- 1) 配置加密密钥文件地址：`alter system set tde_key_file='etc/tde/key.pem' server_type=datEng;`
- 2) 首次使用存储加密时，如未配置加密密钥，使用系统内置随机密钥；
- 3) 使用过程中密钥不可修改；

2. 数据库级别开启存储透明加密：

- 1) `create database test tde_method='sm4_ecb_mode';`
- 2) 数据库 test 库内，新建表默认采用国密 sm4 加密；

3. 数据表级别开启存储透明加密：

- 1) `create table table1(c1 int, c2 int) tde_method='sm4_ecb_mode';;`
- 2) table1 表在存储时将采用存储加密；

4. 验证透明加密的效果：

在配置存储加密密钥的情况下，对于 3 中的新建表 table1，如果删除/修改对应的密钥文件，重启集群后首次访问 table1 表会报错 KW-200，表数据存储解密失败。

7. 事务处理

KINGWOW 数据库系统变量 autocommit 的值决定了每个会话中事务提交的方式，默认值为 1，表示每条语句执行完自动提交；当将 autocommit 的值设为 0 时，则需要对事务进行显式提交。

KINGWOW 数据库通过系统变量控制查询和事务的超时时间，事务在超时时间范围内没有提交，则会被系统自动回滚。如果需要执行一个长时间的事务，可以通过修改当前会话系统变量 kw_tx_timeout 的值来延长超时时间范围。修改系统变量的方法参见 7.3 和 7.4 节内容。

7.1. START TRANSACTION

开启事务，别名为 BEGIN，语法格式为：

```
START TRANSACTION;
```

```
BEGIN;
```

说明：

- 通过 START TRANSACTION 或 BEGIN 语句开启的事务，必须进行显式提交；

7.2. COMMIT

显式提交当前事务，语法格式为：

```
COMMIT;
```

7.3. ROLLBACK

显式回滚当前事务，语法格式为： `ROLLBACK;`

```
kingwow> start transaction;  
Query OK, 0 rows affected (0.00 sec)  
  
kingwow> insert into customer(id,name) values(12,'tom');
```

```
kingwow> commit;  
Query OK, 0 rows affected (0.02 sec)
```

8. 触发器

触发器是对表进行插入、更新、删除操作时自动执行的存储过程;触发器通常用于强制业务规则;触发器是一种高级约束, 可以定义比 check 等约束更为复杂的约束,可执行复杂的 sql 语句(if/while/case), 可引用其它表中的列。触发器定义在特定的表上, 与表相关, 自动触发执行, 不能直接调用。

触发器的类型:

delete 触发器 当删除表中记录时触发, 自动执行触发器所定义的 SQL 语句

insert 触发器 当向表中插入数据时触发, 自动执行触发器所定义的 SQL 语句

update 触发器 当更新表中某列、多列时触发, 自动执行触发器所定义的 SQL 语句

8.1. 创建触发器

语法:

```
CREATE [OR REPLACE] TRIGGER trigger_name  
{AFTER | [NO CASCADE] BEFORE | INSTEAD OF} trigger_event ON table_name  
{FOR EACH (ROW | STATEMENT)} trigger_when triggered-action
```

trigger_event:

```
{INSERT [OR] | DELETE [OR] | UPDATE [OF column_names] [OR]}  
T_TRIGGER_ACTION_TIME: ([NO CASCADE] BEFORE )| AFTER | INSTEAD OF  
T_TRIGGER_EVENTS: INSERT | DELETE | (UPDATE [OF column_list])
```

trigger_when: WHEN (INSERTING | UPDATING | DELETING)

triggered-action:T_PROC_BEGIN_END_BODY

触发器示例:

```
create table test_r1(id int primary key, c int);  
create table test_r2(id int, c int);  
create table test_r3(id int, c int);  
create table test_r1_update(id1 int, c1 int, id2 int, c2 int);  
  
DELIMITER $$  
create or replace trigger tri_insert_r1 after insert on test_r1  
REFERENCING new as new_row  
for each row  
begin  
    insert into test_r2 values(new_row.id, new_row.c);  
end;  
$$
```

```
create or replace trigger tri_insert_r2 after insert on test_r1
for each statement
begin
    insert into test_r2 values(-1, -1);
end;
$$

create or replace trigger tri_delete_r1 after delete on test_r1
REFERENCING old as old_row
for each row
begin
    insert into test_r3 values(old_row.id, old_row.c);
end;
$$

create or replace trigger tri_delete_r2 after delete on test_r1
for each statement
begin
    insert into test_r3 values(-2, -2);
end;
$$

create or replace trigger tri_update_r1 after update on test_r1
REFERENCING old as old_row new as new_row
for each row
begin
    insert into test_r1_update values(old_row.id, old_row.c, new_row.id, new_row.c);
end;
$$

create or replace trigger tri_update_r2 after update on test_r1
for each statement
begin
    insert into test_r1_update values(-3,-3,-3,-3);
end;
$$

DELIMITER ;
```

8.2. 查看触发器

查看触发器语法:

```
show triggers;
```



查看触发器详细信息:

```
select *from __all_trigger;
```

8.3. 删除触发器

```
drop trigger trigger_name;
```

9. 数据库管理

KINGWOW 通过数据库(database)来管理系统中的所有用户数据表, 类似于 MySQL 中的 database 或者 DB2 中的 schema 对象。不同数据库下管理的数据表相对独立, 允许不同数据库下的数据表重名。在 KINGWOW 数据库系统初次启动并初始化时会创建一个名为 kingwow 的默认数据库。用户在连接 KINGWOW 系统时如果没有指定所使用的数据库名, 则系统默认用户使用 kingwow 数据库。

在 KINGWOW 数据库系统中使用数据表时, 如果不指定数据表所属的数据库, 则认为使用的是当前会话所连接数据库中的数据表。可以通过在数据表名前添加数据库名前缀的方式显式指定所使用的数据库, 如“db_name.tbl_name”。

KINGWOW 将系统中的库表定义、配置参数和环境变量等系统元数据以数据表的形式进行存储, 这些数据表被称为系统表。与用户创建的数据表不同, 系统表不属于任何一个数据库(包括默认库 kingwow), 所有的系统表名都以双下划线开头, 用户可以通过查询系统表获取系统的各种元数据信息。

9.1. 查询元数据

9.1.1. 查看数据库

查看系统中所有的数据库, 语法格式为:

```
SHOW DATABASES;
```

```
kingwow> show databases;
+-----+
| Database |
+-----+
| kingwow  |
| test     |
| testdb   |
+-----+
3 rows in set (0.00 sec)
```

9.1.2. 查看当前会话默认数据库

查看当前会话的默认数据库, 语法格式为: `SHOW CURRENT DATABASE;`

```
kingwow> SHOW CURRENT DATABASE;
+-----+
| Database |
+-----+
```

```

kingwow!
/ test /
+-----+
1 row in set (0.00 sec)

```

9.1.3. 设置当前会话默认数据库

指定当前会话所使用的默认数据库，语法格式为：

```
USE db_name;
```

或

```
USING DATABASE db_name;
```

说明：

- 指定了当前会话所使用的默认数据库后，在使用数据表时如果不加库名前缀，则认为是使用当前默认库下的数据表；
- 可以通过连续使用 USING DATABASE 语句来切换当前会话所使用的默认数据库；
- 在使用 KINGWOW 客户端连接 KINGWOW 数据库时，可以通过 -D 参数指定当前会话所使用的默认数据库，如果不加 -D 参数，则当前会话的默认数据库为系统的默认库 kingwow；

9.1.4. 查看系统表

查看系统表，语法格式为：

```
SHOW SYSTEM TABLES ;
```

9.1.5. 查看当前会话默认库中用户数据表

查看当前会话默认数据库中的所有数据表，语法格式为：

```
SHOW TABLES [LIKE 'pattern' | WHERE expr];
```

```

kingwow> show tables like '%cus%';
+-----+
/ Tables /
+-----+
/ customer /
+-----+
1 row in set (0.00 sec)

```

9.1.6. 查看数据表定义

查看数据表的定义，语法格式为：

```
DESCRIBE tbl_name;
```

```
kingwow> describe customer;
```

```

+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id         | int32     | NULL     | 0    | 1   | NULL    |      |
| name      | varchar(10) | NULL    | 0    | 0   | NULL    |      |
| address   | varchar(50) | NULL    | 1    | 0   | NULL    |      |
| last_login | timestamp | NULL    | 1    | 0   | NULL    |      |
| remark    | varchar(1024) | NULL   | 1    | 0   | N/A     |      |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

9.1.7. 查看建表语句

查看指定表的建表语句，语法格式为：

```
SHOW CREATE TABLE tbl_name;
```

```
kingwow> show create table customer;
```

```

+-----+-----+-----+-----+-----+-----+
| Table      | Create Table
+-----+-----+-----+-----+-----+-----+
| test.customer | CREATE TABLE test.customer (
id INT32
, name VARCHAR
, address VARCHAR
, last_login DATETIME
, remark VARCHAR
, PRIMARY KEY(id)
) TABLET MAX SIZE = 134217728, TABLET BLOCK SIZE = 16384, EXPIRE_INFO = '$SYS_DATE >
last_login+38*24*60*60*1000000', USE_BLOOM_FILTER = FALSE |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

9.1.8. 查看列定义

查看指定表中指定列的定义，语法格式为：

```
SHOW COLUMNS {FROM | IN} tbl_name [LIKE 'pattern' | WHERE expr];
```

```
kingwow> show columns from customer like '%i%';
```

```

+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id         | int32     | NULL     | 0    | 1   | NULL    |      |
| last_login | timestamp | NULL     | 1    | 0   | NULL    |      |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

9.1.9. 查看索引

查看数据表上定义的索引，语法格式为：

```
SHOW INDEX {FROM|IN} tbl_name;
```

```
kingwow> show index from customer;
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non unique | Key name | Seq in index | Column name | Collation |
| Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-----+-----+-----+-----+-----+-----+-----+-----+
| customer | 0 | PRIMARY | 1 | id | A | 0
| NULL | NULL | BTREE | | | |
| customer | 1 | 3045_idx_Last_Login | 1 | Last_Login | A |
| NULL | NULL | BTREE | | | |
| customer | 1 | 3045_idx_Last_Login | 2 | id | A | 0
| NULL | NULL | BTREE | | | |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

9.2. 用户及权限管理

9.2.1. 新建用户

创建一个或多个用户，语法格式为：

```
CREATE USER 'user1' IDENTIFIED BY 'password' [, 'user2' IDENTIFIED BY 'password'];
```

说明：

- 用户名和密码不能为空字符串，限制最大长度均为 31 个字符；
- 新建用户仅拥有系统表“__all_server”和“__all_cluster”的 SELECT 权限，以及“__all_client”的 REPLACE 和 SELECT 权限；
- 必须拥有全局的 CREATE USER 权限或对“__all_user”表的 INSERT 权限才可以创建用户；
- 新建用户后，“__all_user”表中会新增一行对应该用户的表项，如果同名用户已经存在则会报错；
- 新建用户时指定的密码会以密文方式存入“__all_user”表；
- 同时创建多个用户时用“,”分隔；

```

kingwow> select user_id,user_name,pass_word from __all_user;
+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | user_name | pass_word |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | dsadmin | f314cc4760637e4c425528d7af105a6bb07b6654 |
| 2 | test_user | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

9.2.2. 删除用户

删除一个或多个用户，语法格式为：

```
DROP USER 'user_name1' [, 'user_name2', ...];
```

说明：

- 必须拥有全局的 CREATE USER 权限或对“__all_user”表的 DELETE 权限才可以删除用户；
- 成功删除用户后，该用户的所有权限也会被一同删除；
- 同时删除多个用户时用“,”分隔；

```
kingwow> drop user 'test_user';
Query OK, 0 rows affected (0.07 sec)
```

```
kingwow> select user_id,user_name,pass_word from __all_user;
+-----+-----+-----+
| user_id | user_name | pass_word |
+-----+-----+-----+
| 1 | admin | ed32f974c5e622767438d999101e5d56f6ac5da9 |
+-----+-----+-----+
1 rows in set (0.00 sec)
```

9.2.3. 修改用户密码

修改用户的登录密码，语法格式为：

```
SET PASSWORD FOR 'user' = 'password';
ALTER USER 'user' IDENTIFIED BY 'password';
```

说明：

- 必须拥有对“__all_user”表的 UPDATE 权限才可以修改用户的密码；

9.2.4. 修改用户名

修改登录用户的用户名，语法格式为：

```
RENAME USER 'old_name' TO 'new_name';
```

说明：

- 必须拥有全局 CREATE USER 权限或者对“__all_user”表的 UPDATE 权限，才可以使用本命令；
- 同时修改多个用户名时用“,”分隔

```
kingwow> select user_id,user_name,pass_word from __all_user;
+-----+-----+-----+
| user_id | user_name | pass_word |
+-----+-----+-----+
| 1 | admin | ed32f974c5e622767438d999101e5d56f6ac5da9 |
| 2 | test_user | 83915e5dadb2180d3e574a7699ffb52f01a6f60a |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
kingwow> rename user 'test_user' to 'test_user_renamed';
Query OK, 0 rows affected (0.05 sec)
```

```
kingwow> select user_id,user_name,pass_word from __all_user;
+-----+-----+-----+
| user_id | user_name | pass_word |
+-----+-----+-----+
```

```
+-----+
|      1 | admin          | ed32f974c5e622767438d999101e5d56f6ac5da9 |
|      2 | test_user_renamed | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |
+-----+
2 rows in set (0.00 sec)
```

9.2.5. 用户锁定/解锁

锁定/解锁用户，被锁定的用户不允许登陆。语法格式为：

```
ALTER USER 'user' LOCKED;
```

```
ALTER USER 'user' UNLOCKED;
```

说明：

- 必须拥有对“__all_user”表的 UPDATE 权限，才可以执行本命令；

```
kingwow> select user_id,user_name,pass_word,is_locked from __all_user;
+-----+
| user_id | user_name | pass_word | is_locked |
+-----+
|      1 | admin    | ed32f974c5e622767438d999101e5d56f6ac5da9 |      0 |
|      2 | dsadmin  | f314cc4760637e4c425528d7af105a6bb07b6654 |      0 |
|      3 | test     | 216e3bd722d99a9659fd9936c2a5eed90e671ae1 |      0 |
|      5 | test_user | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |      0 |
+-----+
4 rows in set (0.00 sec)
```

```
kingwow> alter user 'test_user' Locked;
Query OK, 0 rows affected (0.04 sec)
```

```
kingwow> select user_id,user_name,pass_word,is_locked from __all_user;
+-----+
| user_id | user_name | pass_word | is_locked |
+-----+
|      1 | admin    | ed32f974c5e622767438d999101e5d56f6ac5da9 |      0 |
|      2 | test_user | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |      1 |
+-----+
2 rows in set (0.00 sec)
```

9.2.6. 授予权限

为用户授予操作权限。KINGWOW 数据库中的权限分为全局权限、库权限和表权限三个级别，全局权限存储在__all_user 系统表中，适用于数据库中的所有对象；库权限存储在__all_database_privilege 系统表中，适用于某个库及该库下的所有表；表权限存储在__all_table_privilege 系统表中，适用于某个库下的表。

语法格式为：

```
GRANT priv_type [,priv_type,...] ON priv_level TO 'user1' [,user2,...];
```

说明：

- priv_level 可以设为*、**、db_name.*、db_name.tbl_name 和 tbl_name，*和**都代表全局权限，db_name.*代表库级权限，db_name.tbl_name 和

tbl_name 代表表级权限，不加 db_name 前缀的表是指当前会话默认库下的表；

- 当前用户必须拥有被授予的权限和 GRANT OPTION 权限才能为其他用户授权；
- 同时把多个权限授予用户时，权限类型不能重复，注意 ALL PRIVILEGES 与除 GRANT OPTION 外的其它权限都重复；
- 被授权用户只有重新连接 KINGWOW 数据库后，权限才能生效；
- 用户自动拥有自己创建的对象的所有权限，不再需要额外去授权；
- 用“*”代替 tbl_name，表示赋予全局权限，即对数据库中所有表的权限；
- 同时把多个权限授予用户时，权限类型用“,”分隔；同时给多个用户授权时，用户名用“,”分隔；

表 7-4: KINGWOW 数据库中的权限

权限	说明
ALL PRIVILEGES	除 GRANT OPTION 以外的所有权限；
ALTER	ALTER TABLE 的权限；
CREATE	CREATE DATABASE 和 CREATE TABLE 的权限；
CREATE USER	CREATE USER, DROP USER, RENAME USER 和 REVOKE ALL PRIVILEGES 的权限；
DELETE	DELETE 的权限；
DROP	DROP DATABASE 和 DROP TABLE 的权限；
GRANT OPTION	GRANT OPTION 的权限；
INSERT	INSERT 的权限；
SELECT	SELECT 的权限；
UPDATE	UPDATE 的权限；

权限	说明
REPLACE	REPLACE 的权限;
SUPER	SET GLOBAL 修改全局系统参数的权限;
CREATE SESSION	CREATE SESSION 的权限

示例:

```
kingwow> create database db_user1;
Query OK, 0 rows affected (0.61 sec)
[DDL]CREATE_DATABASE SUCCESS.

kingwow> CREATE USER 'user1' IDENTIFIED BY 'user1_666';
Query OK, 0 rows affected (0.03 sec)
[DCL]CREATE_USER SUCCESS.

kingwow> CREATE USER 'user2' IDENTIFIED BY 'user2_888';
Query OK, 0 rows affected (0.02 sec)
[DCL]CREATE_USER SUCCESS.

kingwow> GRANT all ON db_user1.* TO 'user1';
Query OK, 0 rows affected (0.02 sec)
[DCL]GRANT SUCCESS.

kingwow> GRANT CREATE ON db_user1.* TO 'user2';
Query OK, 0 rows affected (0.02 sec)
[DCL]GRANT SUCCESS.

kingwow> show grants for 'user1';
+-----+
| grants                                     |
+-----+
| GRANT ALL PRIVILEGES ON db_user1.* TO 'user1' |
+-----+
1 row in set (0.00 sec)

kingwow> show grants for 'user2';
+-----+
| grants                                     |
+-----+
| GRANT CREATE ON db_user1.* TO 'user2' |
+-----+
1 row in set (0.00 sec)
```

9.2.7. 回收权限

回收用户的操作权限，语法格式为：

```
REVOKE priv_type [,priv_type,...] ON tbl_name FROM 'user1' [,user2,...];
```

说明：

- 用户必须拥有被撤销的权限 GRANT OPTION 权限才可以回收其他用户的权限；

- 回收“ALL PRIVILEGES”和“GRANT OPTION”权限时，当前用户必须拥有全局 GRANT OPTION 权限，或者对权限表的 UPDATE 及 DELETE 权限；
- 回收权限操作不会级联；
- 用“*”代替 tbl_name 表示回收全局权限，即回收对数据库中所有表的操作权限；
- 同时对用户回收多个权限时，权限类型用“,”分隔；同时回收多个用户的权限时，用户名用“,”分隔；

示例：

```
kingwow> show grants for 'user1';
+-----+
| grants |
+-----+
| GRANT ALL PRIVILEGES ON db_user1.* TO 'user1' |
+-----+
1 row in set (0.00 sec)

kingwow> revoke all ON db_user1.* from 'user1';
Query OK, 0 rows affected (0.04 sec)
[DCL]REVOKE SUCCESS.

kingwow> show grants for 'user1';
+-----+
| grants |
+-----+
| GRANT ALL PRIVILEGES ON db_user1.* TO 'user1' |
+-----+
1 row in set (0.00 sec)
```

9.2.8. 查看权限

查看用户的操作权限，语法格式为：

```
SHOW GRANTS [FOR 'user'];
```

说明：

- 如果不指定用户名则显示当前用户的权限；
- 如果要查看其他用户的权限，必须拥有对“__all_user”的 SELECT 权限；

9.2.9. 权限检查

KINGWOW 数据库的权限管理有两阶段的验证，第一阶段首先会检查连接数据库的用户名和密码是否正确，以及是否拥有指定数据库的连接权限；第二阶段

会检查每个请求是否有足够的权限实施。KINGWOW 数据库中各种操作需要的权限如表 7-5 所示。

表 7-5: KINGWOW 数据库中各种操作所需的权限

操作	权限
SELECT	SQL 中的所有表的 SELECT 权限
INSERT	SQL 中的表的 INSERT 权限
REPLACE	SQL 中的表的 REPLACE 权限
UPDATE	SQL 中的表的 UPDATE 权限
DELETE	SQL 中的表的 DELETE 权限
CREATE USER DROP USER	全局 CREATE_USER 权限
CREATE DATABASE	全局 CREATE 权限
DROP DATABASE	库级 DROP 权限
CREATE TABLE	库级 CREATE 权限
DROP TABLE	表级 DROP 权限
ALTER TABLE	表级 ALTER 权限
修改用户名/密码 锁定用户	对__all_user 表的 UPDATE 权限
GRANT	GRANT 语句中的所有权限以及 GRANT OPTION 权限
REVOKE	REVOKE 语句中的所有权限以及 GRANT OPTION 权限
SHOW GRANTS	__all_user/__all_database_privilege/__all_table_privilege 三张系统表的 SELECT 权限
USING DATABASE	指定库下任何对象的任何权限
CONNECT DATABASE	指定库下任何对象的任何权限

9.2.10. 登录失败控制

客户端连接数据库, 如果认证失败次数超过一定限制, 则数据库自动锁定该用户, 该用户在一定时间内不允许再被连接; 超过锁定时间后, 该用户自动解锁, 可以再次连接。

配置参数:

- FAILED_LOGIN_ATTEMPTS 用户登录失败次数限制, 超过该值, 会导致用户被锁定
- PASSWORD_LOCK_TIME 用户锁定时长, 达到这个时间之后, 下次登录时系统自动解除对这个账户的锁定。

FAILED_LOGIN_ATTEMPTS 默认值 10 次

USER_LOCK_TIME 默认值 0, 单位 s 0 表示不会锁定用户

参数修改:

```
alter system set failed_login_attempts=3 server_type=calEng; alter system set user_lock_time=60 server_type=calEng;
```

9.2.11. 客户端控制

数据库可以设置允许访问的 ip 信息和不允许的 ip 信息。白名单中对应的是允许访问数据库的客户端 ip 信息, 黑名单中对应的是不允许访问数据库的客户端 ip 信息。

配置参数

- INVITED_NODES 允许访问的 IP (白名单)
- EXCLUDED_NODES 不允许访问的 IP (黑名单)
- VALIDNODE_CHECKING 是否校验客户端 ip 合法性

VALIDNODE_CHECKING 默认值为 false, 即不对客户端 ip 进行校验; 如果要校验, 必须设置为 true; INVITED_NODES 默认值为 "all", 所有 ip 都为合法的; 如果设置该参数, 必须设置本地 ip (127.0.0.1), 否则可能会导致无法登陆 EXCLUDED_NODES 默认值为 "none", 没有黑名单; INVITED_NODES/EXCLUDED_NODES 参数值格式: ip1;ip2;...;ipn

ip 校验时, EXCLUDED_NODES 的优先级大于 INVITED_NODES, 即如果同一个 ip 即包含在黑名单中, 也包含在白名单中, 先匹配到黑名单, 则该 ip 为非

法 ip，不能连接数据库。

- 数据库的字符串参数不能为空串，否则启动的时候会解析错误，导致进程无法启动； 因此，INVITED_NODES 默认值设置为特殊字符串 “all”； EXCLUDED_NODES 默认值为 “none”
- INVITED_NODES/EXCLUDED_NODES 参数值，多个 ip 之间使用分号分隔，（逗号在数据库中有特殊处理，使用逗号分隔会导致进程启动时参数解析出错）

参数修改：

```
alter system set VALIDNODE_CHECKING=true server_type=calEng;
```

```
alter system set invited_nodes='127.0.0.1;192.168.100.204;192.168.100.202'  
server_type=calEng;
```

```
alter system set excluded_nodes='192.168.100.202' server_type=calEng;
```

9.2.12. 安全审计

- 1) 支持数据库操作审计，至少包含用户登录、数据库启停、用户锁定解锁、数据库对象增删改等
- 2) 支持审计用户设定和审计操作类型设定，允许针对给定用户和给定操作进行审计
- 3) 安全审计模块应该封装成一个公共服务组件，通过 cache+异步更新触发逻辑保证性能

● 安全审计相关语法

- 1) 开始数据库审计

```
alter system set enable_audit=true server_type=calEng;
```

- 2) 启用指定配置的数据库审计

— 审计 SQL 语句|权限

```
AUDIT {SQL 语句 | 权限} [, ...] [BY <用户> [, ...]] [ BY { SESSION | ACCESS }  
[ WHENEVER [ NOT ] SUCCESSFUL ] ] ;
```

-- 审计对象（目前不支持）

```
AUDIT {操作} [, ...] ON {[方案.]<对象>|DIRECTORY<目录>|DEFAULT} [ BY  
{ SESSION | ACCESS } [ WHENEVER [ NOT ] SUCCESSFUL ] ] ;
```

案例：

①audit alter,delete,insert on cmy.t by scott;

后续由 scott 用户发起对 cmy.t 表的 alter delete drop insert 操作会被审计

②audit create on default by access;

后续所有 create 操作都会被审计

3) 停用指定配置的数据库审计

-- 停止 SQL 语句|权限审计

```
NOAUDIT {SQL 语句 | 权限} [, ...] BY <用户> [, ...] [ WHENEVER [NOT]  
SUCCESSFUL ] ;
```

-- 停止对象审计

```
NOAUDIT {操作} [, ...] ON {[方案.]<对象>|DIRECTORY<目录>|DEFAULT}  
[ WHENEVER [ NOT ] SUCCESSFUL ] ;
```

4) 查看审计日志

```
select * from __all_audit_record;
```

● 安全审计支持如下：

支持系统表(__all_audit_config/__all_audit_record)

支持语法

支持启用/停用审计模块

支持将审计规则写入__all_audit_config 系统表

支持根据审计规则审计 SQL/权限/对象，将对应审计日志写入
__all_audit_record 系统表/日志文件

支持用户名合法性检查

强制审计完善(用户登录登出/数据库启停/用户锁定解锁)

映射表类型:

audit_type	类型	说明
1	AUDIT_SYSTEM	系统级别(强制)审计
2	AUDIT_STMT_ALL_USER	不需要指定 USER 的全用户 SQL 审计
3	AUDIT_STMT	指定 USER 的 SQL 审计

operation_type	类型	说明
1	AUDIT_SYS_LOGIN	用户登录
2	AUDIT_SYS_LOGIN_OUT	用户登出
3	AUDIT_SYS_SERVER_START	节点上线
4	AUDIT_SYS_SERVER_STOP	节点下线
5	AUDIT_SYS_USER_LOCK	用户锁定
6	AUDIT_SYS_USER_UNLOCK	用户解锁
12	AUDIT_OP_ALTER_SYSTEM	修改系统参数
37	AUDIT_OP_ALTER_TABLE	修改表结构
45	AUDIT_OP_CREATE_TABLE	建表
50	AUDIT_OP_DROP_TABLE	删表

9.2.13. 用户安全访问

9.2.13.1. 密码配置过期时长

语法简介

创建用户带密码过期时间:

```
create user user ... password expire [default|never|interval N second|interval N day]
```



修改用户密码过期时间：

```
alter user user ... password expire [default|never|interval N second|interval N day]
```

其中：

- 1.default 表示默认密码过期时间，由系统参数 password_lifetime 控制，默认 300s；
- 2.never 表示永不过期；
- 3.interval N second 表示 N 秒后过期，N 的取值范围为 $\geq 30s$ ；
- 4.interval N day 表示 N 天后过期。
- 5.不指定 password expire 则默认没有过期时间，即永不过期。

测试案例

1.使用 admin 账户登录&创建新用户并授权

```
create user 'u1' identified by 'pass@word1';
```

```
grant all on *.* to 'u1';
```

2.修改用户 u1 密码有效期为 180s

```
alter user 'u1' identified by 'pass@wordnew1' password expire interval 180 second;
```

```
MySQL [kingwow]> alter user 'u1' identified by 'pass@wordnew1' password expire interval 180 second;
Query OK, 0 rows affected (0.04 sec)
[DCL]SET_PASSWORD SUCCESS.
```

3.使用 u1 用户登录

```
kingwow -h192.168.100.156 -P27782 -uu1 -ppass@wordnew1 -Dkingwow -c
```

登录成功

4.180s 后使用 u1 用户再次登录

```
kingwow -h192.168.100.156 -P27782 -uu1 -ppass@wordnew1 -Dkingwow -c
```

登录失败

9.2.13.2. 安全访问控制

用户登录 ip 控制：

基本语法为：create user 'user_name'@'host' identified by "pass";

其中 host 支持不写，或者为%，或者具体 ip，或者形如'192.168.100.203/255.255.255.0'这样的网段（client_ip & netmask == ip 则可登录）

1.使用 admin 用户登录&创建用户 u2



```
create user 'u2'@'192.168.100.203' identified by 'pass@word2';  
grant all on *.* to 'u2';
```

2.在 192.168.100.156 上使用 u2 用户登录

```
kingwow -h 192.168.100.156 -P27782 -uu2 -ppass@word2 -Dkingwow -c  
登录失败
```

3.在 192.168.100.203 上使用 u2 用户登录

```
kingwow -h192.168.100.156 -P27782 -uu2 -ppass@word2 -Dkingwow -c  
登陆成功
```

Ip 黑白名单功能 新增三个系统参数:

validnode_checking :表示是否开启 ip 检查。默认值为 false

invited_nodes: ip 白名单, 默认值为"all"

excluded_nodes, : ip 黑名单, 默认值为"none"

黑白名单的判定逻辑:

- 1.黑名单不为 none。如果命中黑名单, 则 ip 不合法; 否则校验白名单, 如果命中白名单则合法, 未命中则 ip 非法。
- 2.黑名单为 none, 则仅校验白名单。如果命中白名单, 则合法; 否则, ip 非法。

测试案例:

1.开启 sql 黑白名单

```
alter system set validnode_checking = true server_type = calEng;
```

2.设置白名单

```
alter system set invited_nodes='192.168.100.203;172.16.10.14' server_type=calEng;
```

3.从 192.168.100.156 使用 admin 用户登录

```
kingwow -h192.168.100.156 -P27782 -uadmin -padmin -Dkingwow -c  
登录失败
```

4.从 192.168.100.203 使用 admin 用户登录

```
kingwow -h192.168.100.156 -P27782 -uadmin -padmin -Dkingwow -c  
登录成功
```

5.从 172.16.10.14 使用 admin 用户登录

```
kingwow -h192.168.100.156 -P27782 -uadmin -padmin -Dkingwow -c  
登录成功
```

密码错误次数超限锁定用户：

两个相关的系统参数：

failed_login_attempts: 登录失败次数阈值，默认值为 10 次

user_lock_time: 用户锁定时长

备注：锁定只对单个节点生效。每个节点会记录各自的登录失败次数

测试案例：

1.设置登录失败阈值次数为 3 次

```
alter system set failed_login_attempts=3 server_type=calEng;
```

2.设置用户锁定时长

```
alter system set user_lock_time=60 server_type=calEng;
```

3.使用用户 admin 和正确的密码登录

```
kingwow -h192.168.100.156 -P27782 -uadmin -padmin -Dkingwow -c
```

登录成功

4.使用用户 admin 和错误的密码登录四次

```
kingwow -h192.168.100.156 -P27782 -uadmin -padmin1 -Dkingwow -c
```

前三次提示密码错误，第四次锁住

4.等待 60s 后使用正确密码登录

```
kingwow -h192.168.100.156 -P27782 -uadmin -padmin -Dkingwow -c
```

登录成功

9.2.13.3. 限定账户可访问时段

注意需要在 `enable_visit_time_interval` 为 true 前为 admin 设置一个可访问时间段！要不然 admin 退出后也登不上去

基本语法：

新增可访问时间段： `add user user_name time_interval interval_list;`

删除可访问时间段： `drop user user_name time_interval interval_list;`

`interval_list` 支持四种区间：

`(time1, time2)`

`(time1, time2]`

`[time1, time2)`

`[time1, time2]`



interval_list 支持两种时间格式:

(time '00:12:30', time '02:14:25')

[timestamp '2024-03-03 00:12:30', timestamp '2023-03-04 02:14:25']

测试案例:

1.使用 admin 用户登录创建用户 u1

```
create user 'u1' identified by 'pass@word1';
```

```
grant all on *.* to 'u1';
```

2.使用用户 u1 登录（登录成功）

```
kingwow -h192.168.100.156 -P27782 -uu1 -ppass@word1 -Dkingwow -c
```

3.使用 admin 用户登录设置用户 u1 可访问时间段

```
alter system set enable_visit_time_interval = true server_type=calEng;
```

```
add user 'u1' time_interval (time '00:12:30', time '02:14:25');
```

```
add user 'u1' time_interval (time '05:12:30', time '06:14:33');
```

```
add user 'u1' time_interval (time '09:15:00', time '15:00:25');
```

```
add user 'u1' time_interval [time '09:15:00', time '17:00:25'];
```

4.使用用户 u1 登录（当前时间不在上述时段内，预期报错）

```
kingwow -h192.168.100.156 -P27782 -uu1 -ppass@word1 -Dkingwow -c
```

5.使用 admin 用户登录添加用户 u1 可访问时段

```
add user 'u1' time_interval [timestamp '2024-03-03 00:12:30', timestamp '2024-03-04 02:14:25'];
```

6.再次使用用户 u1 登录（当前时间在上述时段内，可成功登录）

```
kingwow -h192.168.100.156 -P27782 -uu1 -ppass@word1 -Dkingwow -c
```

9.3. 设置用户变量

```
SET @var_name1 = expr1 [, @var_name2 = expr2, ...];
```

说明:

- @var_name1 和 @var_name2 为用户变量名，用户变量名由字母、数字和下划线组成，首字符不能为数字；
- 变量的值可以设为整数、实数、字符串或者 NULL 值；
- 同时定义多个用户变量时，用','分隔；

```
kingwow> set @var1=2+3,@var2=100,@var3=@var1+@var2;
Query OK, 0 rows affected (0.00 sec)
```

```
kingwow> select @var1,@var2,@var3;
+-----+-----+-----+
| @var1 | @var2 | @var3 |
+-----+-----+-----+
| 5      | 100    | 105    |
+-----+-----+-----+
1 row in set (0.00 sec)
```

9.4. 修改系统变量

KINGWOW 数据系统中定义了多个系统变量，用于控制会话的环境，如事务隔离级别、查询超时时间等。通过系统表 `_all_sys_param` 可以查询所有的系统变量。系统变量分为全局变量和局部变量两种，前者影响所有新建会话的环境，后者仅影响当前会话。语法格式如下：

设置全局系统变量

```
SET GLOBAL sys_var_name = expr;
```

```
SET @@GLOBAL.sys_var_name = expr;
```

设置局部系统变量

```
SET SESSION sys_var_name = expr;
```

```
SET @@SESSION.sys_var_name = expr;
```

查询系统变量

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern' | WHERE expr];
```

说明:

- 全局变量的值仅用于对新建会话的局部变量进行初始化，修改全局变量的值不影响目前已连接会话的局部变量；
- 修改全局变量必须具有 SUPER 权限，修改会话的局部变量不需要特殊权限，但只能修改当前会话的局部变量，不能修改其它会话的局部变量；
- 查询系统变量时如指定 GLOBAL 或 SESSION 选项，则分别输出全局或当前会话的系统变量值，如不指定则显示当前会话的系统变量值；

表 7-6: KINGWOW 数据库的系统变量

系统变量	数据类型	说明
autocommit	int	是否自动提交事务
kw_charset	varchar	返回结果的字符集
max_allowed_packet	int	网络包大小上限
kw_app_name	varchar	应用名称
kw_group_agg_push_down_param	bool	聚合操作是否下推到 DG 执行
kw_tx_idle_timeout	int	事务开始后无操作的超时时间
kw_read_consistency	int	读一致性级别
kw_tx_timeout	int	事务超时时间
tx_isolation	varchar	事务隔离级别
wait_timeout	int	等待超时时间
kw_query_timeout	int	查询超时时间

10. 其它语句

10.1. 查看当前连接

每个与 KINGWOW 数据库的连接都在一个独立的线程里运行，可以通过 SHOW PROCESSLIST 语句查看当前系统中所有的用户连接，Index 列为线程 ID。语法格式为：SHOW PROCESSLIST;

```
kingwow> show processlist;
+-----+-----+-----+-----+-----+-----+-----+
| Id      | User | Host      | db  | Command | Time | State | Info |
+-----+-----+-----+-----+-----+-----+-----+
| CalEng  |      |           |     |         |      |      |      |
```

```
| 218143959232 | admin | 192.168.100.201:49596 | kingwow | NULL | 0 | SLEEP | NULL |
| 192.168.100.202 | 50 |
| 377057749184 | admin | 192.168.1.2:40309 | sbtest | NULL | 0 | SLEEP | NULL |
| 192.168.100.202 | 87 |
| 398532585664 | admin | 192.168.1.2:41218 | sbtest | show processlist | 418 | ACTIVE | NULL |
| 192.168.100.202 | 92 |
+-----+-----+-----+-----+-----+-----+-----+
+-----+
3 rows in set (0.00 sec)
```

10.2. 中止用户连接

中止用户的连接线程或语句，语法格式为：

```
KILL [CONNECTION | QUERY] ID;
```

说明：

- KILL CONNECTION 与不含选项的 KILL 一样，会终止指定的连接线程；
- KILL QUERY 会终止指定连接当前正在执行的语句，但不会中止连接；

```
kingwow> show processlist;
+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Id      | User | Host                | db   | Command | Time | State | Info |
| CalEng | Index |                      |      |          |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+
+-----+
| 604691015872 | admin | 192.168.100.202:41982 | sbtest | show processlist | 697 | ACTIVE | NULL |
| 192.168.100.202 | 140 |
+-----+-----+-----+-----+-----+-----+-----+
+-----+
1 row in set (0.00 sec)
```

```
kingwow> kill 604691015872;
Query OK, 0 rows affected (0.00 sec)
```

```
kingwow> show processlist;
ERROR 2013 (HY000): Lost connection to Kingwow server during queryNo connection. Trying to reconnect...
Connection id: 168
Current database: sbtest
+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Id      | User | Host                | db   | Command | Time | State | Info |
| CalEng | Index |                      |      |          |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+
+-----+
| 724950100160 | admin | 192.168.100.202:41991 | sbtest | show processlist | 295 | ACTIVE | NULL |
| 192.168.100.202 | 168 |
+-----+-----+-----+-----+-----+-----+-----+
+-----+
1 row in set (0.01 sec)
```

10.3. EXPLAIN 语句

输出数据操作语句的物理执行计划和逻辑执行计划，便于数据库管理员和应用开发人员理解 KINGWOW 数据库中语句的执行策略。

```
EXPLAIN [VERBOSE] {select_stmt | insert_stmt | update_stmt | delete_stmt |
replace_stmt};
```

```
kingwow> explain select * from emp where deptno=20;

+| Query Plan                                     +
| Explain()
Project(rowkey_count=0,
columns=[expr<NULL,65519>=[COL<3041,16>|],expr<NULL,65518>=[COL<3041,17>|],expr<NULL,65517>=[CO
L<3041,18>|],expr<NULL,65516>=[COL<3041,19>|],expr<NULL,65515>=[COL<3041,20>|],expr<NULL,65514>
=[COL<3041,21>|],expr<NULL,65513>=[COL<3041,22>|],expr<NULL,65512>=[COL<3041,23>|]])
TableRpcScan(read_method=SCAN, is use index for storing=0,index tid=-1; is use index without
storing=0,index tid=-1;is_indexed_group=0; limit=<Limit(limit=expr<0,0>=*, offset=expr<0,0>=*,
rpc_scan=<RpcScan(Hint=<max_parallel_count=1, is_get_skip_empty_row=true, read_method=1,
read_consistency=WEAK>, tid=NULL, is_indexed_group=0, limit(Limit=expr<0,0>=*,
offset=expr<0,0>=*)Filter(filters=[expr<NULL,65511>=[COL<3041,23>|int:20|EQ<2>|],)]Project(rowk
ey_count=0,
columns=[expr<3041,16>=[COL<3041,16>|],expr<3041,17>=[COL<3041,17>|],expr<3041,18>=[COL<3041,18
>|],expr<3041,19>=[COL<3041,19>|],expr<3041,20>=[COL<3041,20>|],expr<3041,21>=[COL<3041,21>|],e
xpr<3041,22>=[COL<3041,22>|],expr<3041,23>=[COL<3041,23>|]])TransID(sd=0,tg=0.0.0.0@0,start=0>
)
/
```

10.4. 语句中的 hint

嵌入在 SQL 语句的提示 (hint) 是一种特殊的注释，语法格式为 /*+ ... */，其作用是影响数据库引擎对 SQL 语句的执行策略，不会影响 SQL 语句自身的语义。多个 hint 写在同一个注释中时，用“,”分隔。在使用 KINGWOW 客户端连接 KINGWOW 数据库时，只有设置了 -c 参数，语句中的 hint 才会生效，如：

```
kingwow -h 192.168.1.4 -P 2504 -uadmin -pxxxxx -Dsales -c
```

表 8-1: KINGWOW 数据库支持的 hint 语法

hint	参数	语句	含义
READ_STATIC	无	SELECT	指定查询语句只读 取 DG 上的静态数据，不保证数据一致性；
INDEX(tbl idx)	tbl: 表名 idx: 索引名	SELECT	限制查询执行时使用指定的索引，如该索引不可用，则直接查询源表，且不使用任何索引；
JOIN(join_alg)	si: semi join merge_join bloomfilter_join	SELECT	指定查询语句中所使用的连接算法，semi join 或 merge join
I_MULTI_BATCH	无	INSERT	指定语句采用多批次插入策略，允许插入超过 32M 的数据量，但不保证语句

hint	参数	语句	含义
			执行的原子性；
UD_MULTI_BATCH	无	UPDATE DELETE	指定语句采用多批次更新/删除策略，允许更新/删除超过 32M 的数据量，但不保证语句执行的原子性；

附录一 KINGWOW 数据库系统表

KINGWOW 数据库中所有的系统表表名均以双下划线“__”开头。

1. __first_tablet_entry

记录系统中所有数据表的属性信息，主键为(table_name, db_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
table_name	varchar(256)	数据表名
create_time_column_id	bigint	gm_create 列的列 ID
modify_time_column_id	bigint	gm_modify 列的列 ID
table_id	bigint	数据表 ID
table_type	bigint	数据表类型
load_type	bigint	(未启用)
table_def_type	bigint	1-系统表，2-用户数据表
rowkey_column_num	bigint	主键列数
column_num	bigint	数据表全部列数（包括主键）
max_used_column_id	bigint	该表使用过的最大列 ID（列 ID 不重用）
replica_num	bigint	Tablet 的副本个数（1~6）
create_mem_version	bigint	创建时系统内存表版本（暂留）
tablet_max_size	bigint	Tablet 对应 SSTable 文件大小上限(字节)

属性	类型	说明
max_rowkey_length	bigint	主键长度上限（字节）
compress_func_name	varchar(256)	存储时所使用压缩方法名称
is_use_bloomfilter	bigint	指定是否使用 Bloom Filter
merge_write_sstable_version	bigint	合并时写哪 SSTable 的版本
is_pure_update_table	bigint	是否属于内存更新表（未启用）
rowkey_split	bigint	数据合并过程中 Tablet 的分裂位置
expire_condition	varchar(512)	数据自动过期删除的条件
tablet_block_size	bigint	Tablet 文件数据块的大小（字节）
is_read_static	bigint	是否仅读取静态数据
schema_version	bigint	Schema 的版本
data_table_id	bigint	索引所对应数据表的 ID
index_status	bigint	索引表的状态
db_name	varchar(16)32	数据表所属的数据库名
comment	varchar(512)	

2. __all_all_column

记录系统中所有数据表的字段定义信息，主键为(table_id, column_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
table_id	bigint	所属数据表 ID
column_name	varchar(128)	字段名
table_name	varchar(256)	数据表名
column_id	bigint	字段 ID
column_group_id	bigint	（未启用）
rowkey_id	bigint	0: 非主键属性, >0: 在主键中的排序
length_in_rowkey	bigint	在主键二进制串中占用的字节数

属性	类型	说明
order_in_rowkey	bigint	字段的排序顺序
join_table_id	bigint	-1: 无连接表, >0: 连接表的 ID
join_column_id	bigint	-1: 无连接表, >0: 连接表的字段 ID
data_type	bigint	数据类型
data_length	bigint	整数的字节数或字符串的最大长度
data_precision	bigint	Decimal 类型的有效位数
data_scale	bigint	Decimal 类型的精度
nullable	bigint	是否可以为空
auto_increment	bigint	
special_default_type	bigint	
column_default	varchar(128)	
comment	varchar(256)	
col_type	varchar(10240)	

3. __all_join_info

存储数据表之间内部的连接关系, 主键为(left_table_id, left_column_id, right_table_id, right_column_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
left_table_id	bigint	左表 ID
left_column_id	bigint	左表字段 ID
right_table_id	bigint	右表 ID
right_column_id	bigint	右表字段 ID
left_table_name	varchar(256)	左表表名
left_column_name	varchar(256)	左表字段名
right_table_name	varchar(256)	右表表名

属性	类型	说明
right_column_name	varchar(256)	右表字段名

4. __all_cluster

存储系统中所有集群的信息，主键为(cluster_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	bigint	集群 ID
cluster_vip	varchar(32)	集群主 MG 的 IP 地址
cluster_port	bigint	集群端口号（未启用）
cluster_role	bigint	1: 主, 2: 备
cluster_name	varchar(128)	系群名称（未启用）
cluster_info	varchar(128)	集群说明信息（未启用）
cluster_flow_percent	bigint	流量配比
read_strategy	bigint	客户端负载均衡策略, 0: 轮询, 1: 一致性哈希, 2: 随机
maneng_port	bigint	主 MG 的端口

5. __all_server

存储系统中所有服务器的信息，主键为(cluster_id, svr_type, svr_ip, svr_port)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	bigint	集群 ID
svr_type	varchar(16)	服务器类型
svr_ip	varchar(32)	服务器 IP 地址
svr_port	bigint	服务器端口号
inner_port	bigint	内部交互端口号

属性	类型	说明
svr_role	bigint	0: 无关主备, 1: 主, 2: 备
svr_version	varchar(64)	服务器程序版本信息

6. __all_database

存储系统中所有数据库的基本信息，主键为(db_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
db_name	varchar(32)	数据库名
db_id	bigint	数据库 ID
stat	bigint	数据库状态

7. __all_server_session

记录当前连接到 KINGWOW 数据库的所有会话，主键为(id)。

属性	类型	说明
id	bigint	会话 ID
username	varchar(512)	会话所使用的用户名
host	varchar(128)	会话来源主机 IP 地址和端口
db	varchar(128)	会话连接的数据库名
command	varchar(1024)	会话中正在执行的命令
timeelapsed	bigint	命令耗时
state	varchar(128)	ACTIVE: 正在使用, SLEEP: 暂时未用
info	varchar(128)	会话说明信息
calEng	varchar(128)	会话所连接的 CG
index	bigint	会话所对应的线程 ID

8. __all_server_stat

记录系统中所有服务器的监控信息，仅维护在内存中，不实际存储，主键为

(svr_type, svt_ip, svr_port, name)。

属性	类型	说明
svr_type	varchar(16)	服务器类型
svr_ip	varchar(32)	服务器 IP 地址
svr_port	bigint	服务器端口
name	varchar(64)	监控项名称
value	bigint	监控项的值

9. __all_sys_config_stat

记录当前系统中各服务器已生效配置项的设置值, 主键为(cluster_id, svr_type, svr_ip, svr_port, name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	bigint	所属集群 ID
svr_type	varchar(16)	服务器类型
svr_ip	varchar(32)	服务器 IP 地址
svr_port	bigint	服务器端口
name	varchar(256)	配置项名称
section	varchar(256)	(未启用)
data_type	varchar(256)	配置项值的类型
value	varchar(256)	配置项的值
value_strict	varchar(512)	(未启用)
info	varchar(512)	配置项的说明

10. __all_sys_param

存储了系统所需的环境变量等参数, 主键为(cluster_id, name)。

属性	类型	说明
gm_create	createtime	创建时间

属性	类型	说明
gm_modify	modifytime	修改时间
cluster_id	bigint	集群 ID
name	varchar(256)	参数名称
data_type	bigint	参数值的数据类型
value	varchar(256)	参数的值
info	varchar(256)	参数的说明

11. __all_sys_stat

存储了系统中的各种状态值，主键为(cluster_id, name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	bigint	集群 ID
name	varchar(256)	状态名称
data_type	bigint	状态值的数据类型
value	varchar(256)	状态的值
info	varchar(256)	状态的说明

12. __all_table_privilege

存储了系统中每张数据表的授权信息，主键为(user_id, db_id, table_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
user_id	bigint	用户 ID
table_id	bigint	数据表 ID，0 表示所有表
priv_all	bigint	所有权限
priv_alter	bigint	ALTER TABLE 权限
priv_create	bigint	CREATE TABLE 权限

属性	类型	说明
priv_create_user	bigint	CREATE USER 权限
priv_delete	bigint	DELETE 权限
priv_drop	bigint	DROP TABLE 权限
priv_grant_option	bigint	授权权限
priv_insert	bigint	INSERT 权限
priv_update	bigint	UPDATE 权限
priv_select	bigint	SELECT 权限
priv_replace	bigint	REPLACE 权限
db_id	bigint	数据库 ID

13. __all_user

存储了系统中的用户信息及其全局权限，主键为(user_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
user_name	varchar(1024)	用户名
user_id	bigint	用户 ID
pass_word	varchar(1024)	用户密码（密文存储）
info	varchar(1024)	用户信息
priv_all	bigint	所有权限
priv_alter	bigint	ALTER 权限
priv_create	bigint	CREATE TABLE 权限
priv_create_user	bigint	CREATE USER 权限
priv_delete	bigint	DELETE 权限
priv_drop	bigint	DROP TABLE 权限
priv_grant_option	bigint	授权权限
priv_insert	bigint	INSERT 权限

属性	类型	说明
priv_update	bigint	UPDATE 权限
priv_select	bigint	SELECT 权限
priv_replace	bigint	REPLACE 权限
is_locked	bigint	是否被锁定

14. __all_database_privilege

存储了系统中每个数据库的授权信息，主键为(user_id, db_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
user_id	bigint	用户 ID
db_id	bigint	数据库 ID
priv_all	bigint	所有权限
priv_alter	bigint	ALTER TABLE 权限
priv_create	bigint	CREATE TABLE 权限
priv_create_user	bigint	CREATE USER 权限
priv_delete	bigint	DELETE 权限
priv_drop	bigint	DROP TABLE 权限
priv_grant_option	bigint	授权权限
priv_insert	bigint	INSERT 权限
priv_update	bigint	UPDATE 权限
priv_select	bigint	SELECT 权限
priv_replace	bigint	REPLACE 权限
is_locked	bigint	是否被锁定

15. __all_sequence

存储系统中所有序列的信息，主键为(sequence_name)。

属性	类型	说明
----	----	----

属性	类型	说明
sequence_name	varchar(128)	序列名
data_type	bigint	数据类型, 0: 整型, >0: 精度
current_value	decimal(31,0)	序列的当前值 (prevval)
increment_by	decimal(31,0)	序列的步长
min_value	decimal(31,0)	序列的最小值
max_value	decimal(31,0)	序列的最大值
is_cycle	bigint	0: 不循环, 1: 循环
cache_num	bigint	序列值缓存的数量
is_order	bigint	并发访问是否严格递增(减)
is_valid	bigint	0: 不可用, 1: 可用
const_start_with	decimal(31,0)	RESTART 设置的起始值
can_use_prevval	bigint	0: 不可用, 1: 可用
use_quick_path	bigint	0: 无效, 1: 有效

16. __all_truncate_op

记录系统中清空表操作的执行情况, 主键为(rs_trun_time, table_id)。

属性	类型	说明
rs_trun_time	timestamp	MG 上执行的时间戳
table_id	bigint	数据表 ID
table_name	varchar(289)	数据库名.数据表名
user_name	varchar(32)	执行操作的用户名
info	varchar(512)	说明信息

附录二 KINGWOW 关键字列表

ABSOLUTE	FLASHBACK	PRIOR
ADD	FLOAT	PROCEDURE
ADDDATE	FLOAT4	PROCESSLIST
AES_DECRYPT	FLOAT8	QUICK

AES_ENCRYPT	FOR	RANGE
AFTER	FORCE	READ
ALL	FOREIGN	READS
ALTER	FROM	REAL
AND	FROZEN	REDUNDANT
ANTI	FULL	REFERENCES
ANY	FULLTEXT	REFERENCING
ARRAY	FUNCTION	REGEXP
AS	GATHER	RELATIVE
ASC	@@global	RELEASE
BEFORE	GLOBAL	RENAME
BEGIN	GRANT	REPEATABLE
BETWEEN	GROUP	REPLACE
BIGINT	HASH	RESTART
BIT	HASH_JOIN_SINGLE	RESTRICT
BLOB	HAVING	REVERSE
BLOOMFILTER_JOIN	HOURS	REVOKE
BOOL(EAN)?	IDENTIFIED	RIGHT
BOTH	IF	ROLLBACK
BTREE	IMMEDIATE	ROLLUP
BY	I_MULTI_BATCH	ROW
CACHE	IN	%ROWCOUNT
CALL	INCREMENT	ROWNUM
CASCADE	INDEX(ES)?	ROW(S)?
CASCADEED	INNER	SAVEPOINT
CASE	INOUT	SCHEMA
CHANGE_OBI	INSERT	SCOPE
CHANGE_VALUE_SIZE	INSERTING	SECONDS
CHAR(ACTER)?	INSTEAD	SELECT
CHARSET	INSTR	SEMI
CHECK	INT	SEQUENCE
CLOSE	INTEGER	SERIALIZABLE
CLUSTER	INTERSECT	@@session
COLUMN	INTERVAL	SESSION
COMMIT	INTO	SET
COMMITTED	IS	SET_SLAVE_CLUSTER
COMPACT	ISOLATION	SHOW
COMPILE	JOIN	SI
COMPRESSED	JSON	SIB
CONCAT	KILL	SIBLINGS
CONNECT	LAST	SIMPLE
CONNECTION	LAST_INSERT_ID	SLAVE
CONSISTENT	LEADING	SMALLINT

CONSTRAINT	LEFT	SNAPSHOT
CONTAINS	LEVEL	SPFILE
CONTINUE	LIKE	SQL
CONVERT	LIMIT	START
CREATE	LOCAL	STATIC
CROSS	LOCKED	STORING
CURRENT	LONGBLOB	STRONG
CURRENT_DATE	LONGTEXT	SUBDATE
CURRENT_TIME	LOOP	SWITCH_CLUSTER
CURRENT_TIMESTAMP	MASTER	SYNONYM
CURRENT_USER	MATCH	SYSDATE
CURSOR	MAXVALUE	SYSTEM
%CURSOR_ROW	MEDIUMBLOB	SYSTIMESTAMP
CYCLE	MEDIUMINT	THEN
DATA	MEDIUMTEXT	TIMESTAMP
DATABASE	MEMORY	TINYBLOB
DATABASES	MERGE_JOIN	TINYCLOB
DATE	MICROSECONDS	TINYINT
DATE_ADD	MINUTES	TINYTEXT
DATE_SUB	MINVALUE	TO
DAYS	MOD	TRAILING
DEALLOCATE	MODIFIES	TRANSACTION
DEC(IMAL)?	MODIFY	TRIGGER
DECLARE	MONTHS	TRUE
DEFAULT	NEW	TRUNCATE
DELETE	NEXT	UD_ALL_ROWKEY
DELETING	NEXTVAL	UD_MULTI_BATCH
DESC	NLSSORT	UD_NOT_PARALLAL
DETERMINISTIC	NO	UNCOMMITTED
DISTINCT	NOCYCLE	UNION
DIV	NOT	UNIQUE
DO	%NOTFOUND	UNKNOWN
DOUBLE	NULL	UNSIGNED
DROP	NULLS	UPDATE
DUAL	OF	UPDATING
DYNAMIC	OFFSET	USE
EACH	OLD	USE_OPTIMIZE
EDITIONABLE	ON	USING
ELSE	ONLY	_UTF8
ELSEIF	OPEN	_UTF8MB4
ENUM	OPTION	VALUES
ERROR	OR	VARBINARY
ESCAPE	ORDER	VARCHAR2

EXCEPT	OUT	VARCHAR(ACTER)?
EXCEPTION	OUTER	VIEW
EXECUTE	OVER	WEAK
EXISTS	PARAMETERS	WHEN
EXIT	PARTIAL	WHERE
EXPLAIN	PARTITION	WHILE
EXTRACT	PIVOT	WITH
FALSE	PRECISION	WITHIN
FETCH	PREPARE	WORK
FIRST	PREVVAL	XOR
FIXED	PRIMARY	