CBASE SQL参考指南

CBASE SQL参考指南
1. CBASE SQL简介
1.1. 支持语句
2. 基本元素
2.1. 数据类型
2.1.1. **基本数据类型**
2.1.2. **高精度数值类型**
2.2. 字符集
2.3. 转义字符
2.4. 内部表
2.5. 关键字
3. 运算符
3.1. 逻辑运算符
3.1.1. **NOT**
3.1.2. **AND**
3.1.3. **OR**
3.2. 算数运算符
3.3. 比较运算符
3.3.1. **数值比较**
3.3.2. **[NOT] BETWEEN ... AND ...**
3.3.3. **[NOT] IN**
3.3.4. **IS [NOT] NULL | TRUE| FALSE| UNKNOWN**
3.4. 拼接运算符
3.5. 优先级
4. 函数
4.1. 系统函数
4.1.1. **CAST****(expr as type)****
4.1.2. **CONCAT****(str1, str2)****
4.1.3. **CURRENT_DATE()**
4.1.4. **CURRENT_TIME****()和CURRENT_TIMESTAMP()****
4.1.5. **YEAR(****param)****
4.1.6. **MONTH(param)**
4.1.7. **DAY(****param)****
4.1.8. **DAYS(****param)****
4.1.9. **HOUR****(param)****
4.1.10. **MINUTE(param****)****
4.1.11. **SECOND(param)**
4.1.12. **DATE_ADD(date,INTERVAL expr type)**
4.1.13. **DATE_SUB(date,INTERVAL expr type)**
4.1.14. **ADDDATE(date,INTERVAL expr type****)****
4.1.15. **SUBDATE(date,INTERVAL expr type)**
4.1.16. **HEX(str)**
4.1.17. **INT2IP(INT_VALUE)和IP2INT('IP_ADDR')**
4.1.18. **LENGTH(STR)**
4.1.19. **str1 [NOT] LIKE str2**
4.1.20. **NOW()**
4.1.21. **NVL(str1, str2)**
4.1.22. **R****OUND(number,digits)****

4.1.23. **S**TRICT*CURRENT*TIMESTAMP()**
4.1.24. **SUBSTR(str,pos,len)**,SUBSTR(str,pos),SUBSTR(str FROM pos)**
4.1.25. **TRIM([{BOTH | LEADING | TRAILING} FROM] str)**
4.1.26. **UNHEX(str)**
4.1.27. **UPPER(str)**
4.1.28. **L**OWER(str)**
4.1.29. **DECODE()**
4.1.30. **FLOOR()和CEIL()**
4.2. 聚集函数
4.2.1. **AVG()**
4.2.2. **COUNT()**
4.2.3. **MAX()**
4.2.4. **MIN()**
4.2.5. **SUM()**
4.2.6. **ROW_NUMBER()**
5. CBASE SQL语句参考
5.1. 数据定义语言（DDL）
5.1.1. **CREATE DATABASE**语句**
* 格式
* 举例
5.1.2. **DROP DATABASE**语句**
* 格式
* 举例
5.1.3. **CREATE TABLE 语句**
* 格式
* 举例
5.1.4. **ALTER TABLE 语句**
* 格式
* 举例
5.1.5. **DROP TBALE 语句**
* 格式
* 举例
5.1.6. **T**RUNCATE TABLE语句**
* 格式
* 举例
5.1.7. **CREATE SEQUENCE**语句**
* 格式
* 举例
5.1.8. **ALTER SEQUENCE语句**
* 格式
* 举例
5.1.9. **DROP SEQUENCE语句**
* 格式
* 举例
5.1.10. **CREATE FUNCTION 语句**
* 格式
* 举例
5.1.11. **DROP FUNCTION 语句**
* 格式
* 举例
5.1.12. **C**REATE INDEX语句**
* 格式
* 举例

5.1.13. **DROP INDEX语句**
* 格式
* 举例
5.2. SELECT查询操作语句
5.2.1. **基本查询**
* 格式
* 举例
5.2.2. **JOIN语句**
5.2.3. **集合操作**
* UNION句法
* EXCEPT句法
* INTERSECT句法
5.2.4. **DUAL虚拟表**
* 格式
* 举例
5.2.5. **SELECT ... FOR UPDATE句法**
* 格式
* 举例
5.2.6. **IN和OR**
5.3. 数据操作语言（DML）
5.3.1. **INSERT 语句**
* 格式
* 举例
5.3.2. **REPLACE 语句**
* 格式
* 举例
5.3.3. **UPDATE 语句**
* 格式
* 举例
5.3.4. **DELETE 语句**
* 格式
* 举例
5.4. 数据库管理语言（DCL）
5.4.1. **新建用户**
* 格式
* 举例
5.4.2. **删除用户**
* 格式
* 举例
5.4.3. **修改密码**
* 格式
* 举例
5.4.4. **修改用户名**
* 格式
* 举例
5.4.5. **锁定用户**
* 格式
* 举例
5.4.6. **用户授权**
* 格式
* 举例
5.4.7. **撤销权限**
* 格式

* 举例
5.4.8. **查看权限**
* 格式
* 举例
5.4.9. **修改用户变量**
* 格式
* 举例
5.4.10. **修改系统变量**
* 格式
* 举例
5.4.11. **修改系统配置项**
* 格式
* 举例
5.4.12. **查看系统中存在的库**
* 格式
* 举例
5.4.13. **声明所**使用的库**
* 格式
* 举例
5.4.14. **查看当前**声明的库**
* 格式
* 举例
5.4.15. **查看系统中**的系统表**
* 格式
* 举例
5.4.16. **事务处理**
* 格式
* 举例
5.4.17. **实用**的SQL语句**
* SHOW 语句
* KILL 语句
* DESCRIBE 语句
* EXPLAIN 语句
* WHEN 语句
* Hint 语法
* PREPARE 语句
* EXECUTE 语句
* DEALLOCATE 语句
5.4.18. **预备执行语句**
6. CBASE SQL优化指南
6.1. 根据执行计划调优
6.2. 优化规则
6.2.1. **主键索引优化**
6.2.2. **二级索引优化**
6.2.3. **并发执行优化**
6.2.4. **复杂SQL语句优化**
6.2.5. **JOIN语句**优化**
6.2.6. **SELECT语句**优化**
6.2.7. **操作符**优化**
7. CBASE错误码

1. CBASE SQL简介

CBASE是交行基于阿里巴巴OceanBase0.4.2研发的分布式数据库，支持SQL 92协议与兼容MySQL网络协议，所以CBASE用户可以使用MySQL客户端、Java客户端和C客户端连接数据库。

1.1. 支持语句

CBASE SQL 语句中的关键字、库名、表名、列名、函数名等均大小写不敏感。库名、表名和列名都转换为小写之后存入 Schema 中，所以即使用户建库表时候列名是大写的，查询的时候获得的列名也是小写。 如果您需要保存大写字母，请使用双引号，例如："TANG"。用户使用的时候必须遵循从库再到表的原则，对库表的操作前必须获取到相应的权限。CBASE SQL 语法遵循 SQL92 标准，单引号表示字符串；双引号表示表名、列名或函数名。双引号内可以出现 SQL 保留的关键字。目前版本支持的语句有 CREATE TABLE，DROP TABLE，ALTER TABLE，CREATE FUNCTION，SELECT，INSERT，REPLACE，DELETE，UPDATE，SET，SHOW ，LIST等，具体语法请见CBASE SQL语句参考部分。

2. 基本元素

2.1. 数据类型

CBASE支持常用的基本数据类型和高精度数值类型。

2.1.1. 基本数据类型

目前CBASE支持的基本数据类型表如下：

数据类型	说明	字段
Bigint/int/ integer/mediumint/smallint/tinyint	Bigint、 int、 integer、 mediumint、smallint 和 tinyint 无论语义还是实现都是等价的，存储为 8 字节有符号整型。	MYSQL_TYPE_LONGLONG
binary/char/varchar/varbinary	字符串，使用单引号。varchar、char、 binary和 varbinary 等价，均存储为 varchar 类型。这种类型的比较使用的是字节序。此外，在数据库建表时定义的varchar列的最大长度也是不起作用的。例如varchar(32)，实际上可以插入大于 32 字节的串。	MYSQL_TYPE_VAR_STRING
bool	布尔类型，表示 True 或者 False	MYSQL_TYPE_TINY
createtime	特殊的数据类型，用于记录本行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。	-
datetime/timestamp	时间戳类型。暂不支持time、date等类型。时间戳格式必须为“YYYY-MM-DD HH:MI:SS”	MYSQL_TYPE_DATETIME
double/real	表示8字节浮点数。double和real等价，均存储为double类型	MYSQL_TYPE_DOUBLE
float	表示4字节浮点数	MYSQL_TYPE_FLOAT
modifytime	特殊的数据类型，用于记录本行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。	-

2.1.2. 高精度数值类型

CBASE 支持高精度数值类型：decimal(p,s)。

- “p”表示 precision，与“s”的差值为整数位数的限制。
  - （p-s）> 0：表示整数部分的位数不能超过“p-s”，否则将报错。
  - （p-s）≤0：表示整数部分必须为“0”，小数点后“-（p-s）”位也必须为“0”，否则将报错。
- “s”表示 scale。
  - s > 0：表示精度限制在小数点后 s 位，超过 s 位后面的小数将被截取。

- s≤0：小数部分被舍去，且小数点前 s 位，将被四舍五入。

- 最大取值范围：
  - decimal 可以表示的十进制位数最大为 38 位。

说明：scale的长度不大于37，即整数部分位数最少为1位。scale作为小数部分位数的限制，s大于0的时候，超过s位的后面的小数将会被截取，而不是四舍五入，如果用户输入的数的scale小于定义时候的scale，将会进行补0操作。

- 举例说明如下：

  ecimal(3,2) 6.789 è 6.78 超过小数部分限制，需进行截取

decimal(4,3) 3.14 è 3.140 小数位数不足，进行补0操作

- 占用空间：

要求decimal数据类型实现变长存储

- 四则运算：

Decimal算术运算结果的精度和小数位数设置参照DB2。具体设置如表1和表2：

其中，表格中各参数含义为：

p：第一个数的精度 s：第一个数的小数位数

p’：第二个数的精度 s’：第二个数的小数位数

	precision	**scale**
ADD	Min(31,max(p-s,p’-s’)+max(s,s’)+1)	Max(s,s’)
SUB	Min(31,max(p-s,p’-s’)+max(s,s’)+1)	Max(s,s’)
MUL	Min(31,p+p’)	Min(31,s+s’)
DIV	31	31-p+s-s’

表 1 DB2运算结果精度与小数位数

	precision	**scale**
ADD	运算结果的整数位数+scale	Max(s,s’)
SUB	同add	同add
MUL	Min(38,p+p’)	运算结果若超过38位，保留整数位数，截断超出的小数；若没有超过38，则为s+s’
DIV	38	会将结果补足成整个数为38位

表 2 CBASE运算结果精度与小数位数

2.2. 字符集

CBASE支持的字符集编码如下：

armscii8	ascii	big5	binary	cp1250	cp1251
cp1256	cp1257	cp850	cp852	cp866	cp932
dec8	eucjpms	euckr	gb2312	gbk	geostd8
greek	hebrew	hp8	keybcs2	koi8r	koi8u
latin1	latin2	latin5	latin7	macce	macroman
sjis	swe7	tis620	ucs2	ujis	utf8

CBASE 返回给用户的字符集的参数名为“ob_charset”，缺省值为“utf8”。

在设置该字符集时，应注意会话变量和全局变量的区别，设置字符集方法如下：

1．执行以下命令，设置 CBASE 字符集为“utf8”。

```
SET @@SESSION.ob_charset ='utf8';
```

2．执行以下命令，查看 CBASE 字符集。

```
SHOW VARIABLES LIKE 'ob_charset';
```

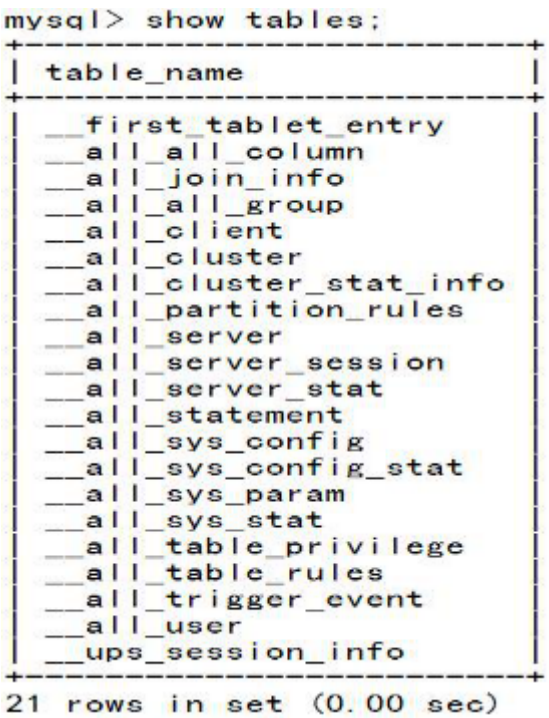
2.3. 转义字符

转义字符是在字符串中，某些序列前添加反斜线“\”，用于表达特殊含义。CBASE的转义字符表如下：

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
\”	双引号
_	_字符
\%	%字符
\0	空字符（NULL）

2.4. 内部表

CBASE内部表都以“__”开头，普通用户严禁使用此种格式的名字。可以使用show tables命令查看数据库中有哪些表。



2.5. 关键字

ADD	**AND**	**ANY**
ALL	AS	ASC
AUTO_INCREMENT	BETWEEN	BIGINT
BINARY	BOOLEAN	BY
CASE	CHARACTER	CNNOP
COLUMNS	COMPRESS_METHOD	CREATE
CREATETIME	DATE	DATETIME
DECIMAL	DEFAULT	DELETE
DESC	DESCRIBE	DISTINCT
DOUBLE	DROP	ELSE
END	END_P	ERROR
EXCEPT	EXISTS	EXPIRE_INFO
EXPLAIN	FLOAT	FROM
FULL	GLOBAL	GROUP
HAVING	IF	IN
INNER	INTEGER	INTERSECT
INSERT	INTO	IS
JOIN	JOIN_INFO	KEY
LEFT	LIMIT	LIKE
MEDIUMINT	MOD	MODIFYTIME
NOT	NUMERIC	OFFSET
ON	OR	ORDER
OUTER	PRECISION	PRIMARY
REAL	REPLACE	REPLICA_NUM
RIGHT	SCHEMA	SELECT
SERVER	SESSION	SET
SHOW	SMALLINT	STATUS
TABLE	TABLES	TABLET_MAX_SIZE
THEN	TIME	TIMESTAMP

ADD	AND	ANY
TINYINT	UNION	UPDATE
USE_BLOOM_FILTER	VALUES	VARCHAR
VARBINARY	VARIABLES	VERBOSE
WHERE	WHEN	USER
IDENTIFIED	PASSWORD	FOR
ALTER	RENAME	TO
LOCKED	UNLOCKED	GRANT
PRIVILEGES	OPTION	REVOKE

3. 运算符

本节主要介绍CBASE支持多种类型的运算符，主要包括算数运算符、比较运算符、逻辑运算符和位运算符，以及运算的优先级。

3.1. 逻辑运算符

CBASE中，逻辑操作符会把左右操作数都转成BOOL类型进行运算。逻辑运算时返回“error”表示计算错误。

CBASE各数据类型转换BOOL类型的规则如下：

- 字符串只有是“true”、“false”、“1”和“0”才能够转换到 Bool 类型，其中字符串“true”和“1”为“true”，字符串“false”和“0”为“false”。
- “int”、 “float”、 “double”和“decimal”转换 Bool 类型时，数值不为零时为“true”，数值为零时为“false”

3.1.1. NOT

逻辑非，操作类型对照表如下：

int	Float	Double	Timestamp	Varchar	Bool	NULL
True/false	True/false	True/false	Error	True/false	True/false	NULL

3.1.2. AND

逻辑与，操作类型对照表如下：

	int	Float	Double	Timestamp	Varchar	Bool	NULL
int	True/false	True/false	True/false	Error	True/false/Error	True/false	False/NULL
float		True/false	True/false	Error	True/false/Error	True/false	False/NULL
double			True/false	Error	True/false/Error	True/false	False/NULL
timestamp				Error	Error	True/false	Error
varchar					True/false/Error	True/false/Error	False/NULL
bool						True/false	False/NULL
NULL							Error

3.1.3. OR

逻辑或，操作类型对照表如下

	int	Float	Double	Timestamp	Varchar	Bool	NULL
int	True/false	True/false	True/false	Error	True/false/Error	True/false	True/NULL
float		True/false	True/false	Error	True/false/Error	True/false	True/NULL
double			True/false	Error	True/false/Error	True/false	True/NULL
timestamp				Error	Error	True/false	Error
varchar					True/false/Error	True/false/Error	True/NULL
bool						True/false	True/NULL
NULL							Error

3.2. 算数运算符

CBASE中，数值计算只允许在数值类型和varchar直接进行，其它类型直接报错。字符串在做算数运算时，如果无法转成decimal类型则报错。字符串只有在内容全为数字或者开头是“+”或者“-”,且后面跟数字的形式才能转转成decimal型。CBASE支持的运算符表如下：

表达式	含义	举例
+	加法	Select 3+7
-	减法	Select 3-7
*	乘法	Select 3*7
/	除法，返回商。除数为0，返回NULL	Select 3/7
%或者MOD	除法，返回余数。除数为0，返回NULL	Select 3%7

“+”“-”“*”的操作类型对照表如下：

	int	Float	Double	Timestamp	Varchar	Bool	NULL
int	int	double	double	Error	double/Error	Error	NULL
float		double	double	Error	double/Error	Error	NULL
double			double	Error	double/Error	Error	NULL
timestamp				Error	Error	Error	Error
varchar					double/Error	Error	NULL/Error
bool						Error	Error
NULL							NULL

“/”的操作类型对照表如下，如果除数为0则报错：

	int	Float	Double	Timestamp	Varchar	Bool	NULL
int	int	Double/Error	Double/Error	Error	double/Error	Error	NULL/Error
float		Double/Error	Double/Error	Error	double/Error	Error	NULL/Error
double			Double/Error	Error	double/Error	Error	NULL/Error
timestamp				Error	Error	Error	Error
varchar					double/Error	Error	NULL/Error
bool						Error	Error
NULL							NULL

3.3. 比较运算符

CBASE比较的策略是，先将操作数转换为相同的类型，然后进行比较。所有比较运算符的返回类型为Bool或者NULL。比较结果为真则返回“1”,为假则返回“0”,不确定则返回“NULL”。

3.3.1. 数值比较

比较运算符用于比较两个数值的大小。CBASE支持的运算符表如下：

表达式	含义	举例
=	等于	Select 1=1
>=	大于等于	Select 1>=1
>	大于	Select 1>1
<	小于	Select 1<1
<=	小于等于	Select 1<=1
!=或者<>	不等于	Select 1!=1

数值比较运算符的转换规则如下表。

	int	Float	Double	Timestamp	Varchar	Bool	NULL
int	int	float	Double	Error	int/Error	Error	NULL
float		float	Double	Error	float/Error	Error	NULL
double			Double	Error	Double/Error	Error	NULL
timestamp				timestamp	Timestamp/Error	Error	NULL
varchar					varchar	Error/bool	NULL
bool						bool	NULL
NULL							NULL

3.3.2. [NOT] BETWEEN ... AND ...

判断是否存在或者不存在于指定范围。

```
mysql> select 1 between 3 and 7;
+--------------------+
| 1 between 3 and 7 |
+--------------------+
|                  0 |
+--------------------+
1 row in set (0.00 sec)
```

3.3.3. [NOT] IN

判断是否存在于指定集合

```
mysql> select 1 in (2,3,4,5,6);
+-------------------+
| 1 in (2,3,4,5,6) |
+-------------------+
|                 0 |
+-------------------+
1 row in set (0.01 sec)
```

3.3.4. IS [NOT] NULL | TRUE| FALSE| UNKNOWN

判断是否为NULL、真、假或者未知。执行成功，结果是TRUE或者FALSE，不成功是NULL。

```
mysql> select 1 is null,null is null, null is not null;
+-----------+--------------+-------------------+
| 1 is null | null is null | null is not null |
+-----------+--------------+-------------------+
|         0 |            1 |                 0 |
+-----------+--------------+-------------------+
1 row in set (0.00 sec)
```

3.4. 拼接运算符

CBASE支持的运算符如表1-6所示。表1-6拼接运算符

表达式	含义	举例
||	将值联结到一起构成单个值	Select ‘a’||’q’

3.5. 优先级

CBASE的运算符的优先级如表1-7所示。表1-7优先级

优先级	运算符
1	* , \ , % , MOD
2	+ , -
3	= , >, >= , < , <= , <> , != , IS , LIKE , IN
4	BETWEEN
5	NOT
6	AND
7	OR

备注：可以使用小括号控制操作符的运算次序。

4. 函数

CBASE支持的函数分为系统函数和聚集函数。

4.1. 系统函数

4.1.1. CAST**(expr as type)**

将expr字段转换为type类型。数据类型如表1-1所示。

```
mysql> select cast(123 as bool);
+--------------------+
| cast(123 as bool) |
+--------------------+
|                  1 |
+--------------------+
1 row in set (0.00 sec)
```

4.1.2. CONCAT**(str1, str2)**

把两个字符串连接成一个字符串。左右参数都必须是 varchar 类型或 NULL，否则报错。如果执行成功，则返回连接后的字符串；参数中有一个值是 NULL 结果就是 NULL。如果两个字符串连接后的长度大于 OB_MAX_VARCHAR_LENGTH，返回错误。

```
mysql> select concat('hello ',' cbase!');
+---------------------------+
| concat('hello ',' cbase!') |
+---------------------------+
| hello  cbase!              |
+---------------------------+
1 row in set (0.00 sec)
```

4.1.3. CURRENT_DATE()

以'YYYY-MM-DD'格式返回当前日期值, 该函数的返回值为DATE类型，不能插入到TIME类型的列中

```
mysql> select current_date();
+----------------+
| current_date() |
+----------------+
| 2018-06-12     |
+----------------+
```

4.1.4. CURRENT_TIME**()和CURRENT_TIMESTAMP()**

用于获取系统当前时间，精确到微妙

```
mysql> select current_timestamp();
+----------------------------+
| current_timestamp()        |
+----------------------------+
| 2016-05-23 10:25:45.773774 |
+----------------------------+
1 row in set (0.00 sec)
```

4.1.5. YEAR(**param)**

输入VARCHAR(必须包含日期)，TIMESTAMP以及DATE类型的参数，返回输入参数中的年份(范围在0到9999)，不能处理TIME类型的参数

```
mysql> select year('2018-06-12');
+--------------------+
| year('2018-06-12') |
+--------------------+
|               2018 |
+--------------------+
1 row in set (0.00 sec)
```

4.1.6. MONTH(param)

输入VARCHAR(必须包含日期)，TIMESTAMP以及DATE类型的参数，返回输入参数中的月份，不能处理TIME类型的参数

```
mysql> select month('2018-06-12');
+---------------------+
| month('2018-06-12') |
+---------------------+
|                   6 |
+---------------------+
1 row in set (0.00 sec)
```

4.1.7. DAY(**param)**

输入VARCHAR(必须包含日期)，TIMESTAMP以及DATE类型的参数，返回输入参数中的日期，不能处理TIME类型的参数

```
mysql> select day('2018-06-12');
+-------------------+
| day('2018-06-12') |
+-------------------+
|                12 |
+-------------------+
1 row in set (0.00 sec)
```

4.1.8. DAYS(**param)**

输入VARCHAR(必须包含日期)，TIMESTAMP以及DATE类型的参数。输出结果：返回输入参数中的日期与基准日期('0001-01-01')数值,从1 开始计数。不能处理TIME类型的参数。

```
mysql> select days('2018-06-12');
+--------------------+
| days('2018-06-12') |
+--------------------+
|             736857 |
+--------------------+
1 row in set (0.00 sec)
```

4.1.9. HOUR**(param)**

输入VARCHAR，TIMESTAMP，DATE以及TIME类型的参数，返回参数中的小时数(范围是0到23)，当输入参数中不包含时间部分时返回零

```
mysql> select hour('08:09:10');
+------------------+
| hour('08:09:10') |
+------------------+
|                8 |
+------------------+
1 row in set (0.00 sec)
```

4.1.10. MINUTE(param**)**

输入VARCHAR，TIMESTAMP，DATE以及TIME类型的参数，返回参数中的分钟数(范围是0到59)，当输入参数中不包含时间部分时返回零

```
mysql> select minute('08:09:10');
+--------------------+
| minute('08:09:10') |
+--------------------+
|                  9 |
+--------------------+
1 row in set (0.01 sec)
```

4.1.11. SECOND(param)

输入VARCHAR，TIMESTAMP，DATE以及TIME类型的参数，返回参数中的秒数(范围是0到59)，当输入参数中不包含时间部分时返回零

```
mysql> select second('08:09:10');
+--------------------+
| second('08:09:10') |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.00 sec)
```

4.1.12. DATE_ADD(date,INTERVAL expr type)

输入参数：date :可以为DATE,TIME,TIMESTAMP类型的数值；可以为列。

expr :可以为任意表达式，最终会转为整型类型。

type ：为关键字 YEARS、MONTHS、DAYS 、HOURS 、MINUTES 、SECONDS、 MICROSECONDS的任意一个。

输出结果：返回日期与表达式相加后的结果。输出结果的类型格式与参数date的类型一致。

制 限：date参数只支持时间类型的常量及列，其余类型不支持。

注意事项：date参数类型为DATE时，只能与type 参数为YEARS、MONTHS、DAYS的进行运算；

date参数类型为TIME时，只能与type 参数为HOURS 、MINUTES 、SECONDS的进行运算；

date参数类型为TIMESTAMP时，能与type 参数为任意类型的进行运算；

```
mysql> select date_add(date '2018-06-12',interval 1 days);
+-----------------------------------------------+
| date_add(date '2018-06-12',interval 1 days) |
+-----------------------------------------------+
| 2018-06-13                                    |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

4.1.13. DATE_SUB(date,INTERVAL expr type)

输入参数：date :可以为DATE,TIME,TIMESTAMP类型的数值；可以为列。

expr :可以为任意表达式，最终会转为整型类型。

type :为关键字 YEARS、MONTHS、DAYS 、HOURS 、MINUTES 、SECONDS、 MICROSECONDS的任意一个。

输出结果：返回日期与表达式相减后的结果。输出结果的类型格式与参数date的类型一致。

制 限：date参数只支持时间类型的常量及列，其余类型不支持。

注意事项：date参数类型为DATE时，只能与type 参数为YEARS、MONTHS、DAYS的进行运算；

date参数类型为TIME时，只能与type 参数为HOURS 、MINUTES 、SECONDS的进行运算；

date参数类型为TIMESTAMP时，能与type 参数为任意类型的进行运算；

```
mysql> select date_sub(date '2018-06-12',interval 1 days);
+-----------------------------------------------+
| date_sub(date '2018-06-12',interval 1 days) |
+-----------------------------------------------+
| 2018-06-11                                    |
+-----------------------------------------------+
1 row in set (0.01 sec)
```

4.1.14. ADDDATE(date,INTERVAL expr type**)**

同DATE_ADD(date,INTERVAL expr type)

4.1.15. SUBDATE(date,INTERVAL expr type)

同DATE_SUB(date,INTERVAL expr type)

4.1.16. HEX(str)

将字符串转化为十六进制数显示。

str 必须是整数或者字符串。当输入是整数（进制不限）的时候，输出整数的十六进制表示；当输入是字符串的时候，输出是字符串的字节流的十六进制表示。

```
mysql> select hex('cbase'),hex(0),hex(10);
+--------------+--------+---------+
| hex('cbase') | hex(0) | hex(10) |
+--------------+--------+---------+
| 6362617365   | 0      | a       |
+--------------+--------+---------+
1 row in set (0.00 sec)
```

4.1.17. INT2IP(INT_VALUE)和IP2INT('IP_ADDR')

INT2IP：将一个整数转换成ip；IP2INT：将一个ip转换为整数。

```
mysql> select int2ip(126666666),ip2int('170.199.140.7');
+-------------------+-------------------------+
| int2ip(126666666) | ip2int('170.199.140.7') |
+-------------------+-------------------------+
| 170.199.140.7     |               126666666 |
+-------------------+-------------------------+
1 row in set (0.00 sec)
```

4.1.18. LENGTH(STR)

返回字符串的长度，单位为字节。参数必须是 varchar 类型或 NULL，否则报错。如果执行成功，结果是 int 型整数，表示字符串长度；当参数是 NULL 结果为 NULL。

```
mysql> select length('cbase');
+-----------------+
| length('cbase') |
+-----------------+
|               5 |
+-----------------+
1 row in set (0.00 sec)
```

4.1.19. str1 [NOT] LIKE str2

字符串匹配函数。左右参数都必须是 varchar 类型或 NULL，否则报错。如果执行成功，结果是 TRUE 或者 FALSE，或某一个参数是 NULL 结果就是 NULL。

通配符包括"%"和"_"：

Ø "%"表示匹配任何长度的任何字符，且匹配的字符可以不存在。

Ø "_"表示只匹配单个字符，且匹配的字符必须存在。

备注：CBASE的转义字符为"\"。

```
mysql> select 'ab%' like 'abc%';
+------------------+
| 'ab%' like 'abc%' |
+------------------+
|                 0 |
+------------------+
1 row in set (0.00 sec)
```

4.1.20. NOW()

用于获取系统当前时间，精确到微秒。格式为“YYYY-MM-DD HH:MI:SS.SSSSSS”。

```
mysql> select now();
+----------------------------+
| now()                      |
+----------------------------+
| 2016-05-23 10:43:34.749363 |
+----------------------------+
1 row in set (0.00 sec)
```

4.1.21. NVL(str1, str2)

如果str1为NULL，则替换成str2。

```
mysql> select nvl(null,1);
+-------------+
| nvl(null,1) |
+-------------+
|           1 |
+-------------+
1 row in set (0.00 sec)
```

4.1.22. R**OUND(number,digits)**

四舍五入函数

参数说明：

Number：要四舍五入的数；

digits：小数点后保留的位数，且digits必须为整数；

如果digits大于0，则四舍五入到指定的小数位；

如果digits等于0，则四舍五入到最接近的整数；

如果digits小于0，则在小数点左侧进行四舍五入

```
mysql> Select round(2.149,0);
+----------------+
| round(2.149,0) |
+----------------+
|              2 |
+----------------+
1 row in set (0.00 sec)
```

4.1.23. S**TRICT*CURRENT*TIMESTAMP()**

当系统运行多 MergeServer 时，由于每个 MergeServer 可能存在时间误差，将导致 NOW()的运算结果不一致，而 STRICT_CURRENT_TIMESTAMP()从UpdateServer 上获取时间，从而保证各个 MergeServer 上运行该函数时，获取的时间都是一致的。

```
mysql> select strict_current_timestamp();
+---------------------------+
| strict_current_timestamp() |
+---------------------------+
| 2016-05-23 10:47:49.610947 |
+---------------------------+
1 row in set (0.02 sec)
```

4.1.24. SUBSTR(str,pos,len)**,SUBSTR(str,pos),SUBSTR(str FROM pos)**

这三个函数均用于返回一个子字符串。 起始于位置 pos，长度为 len。使用 FROM的格式为标准 SQL 语法。

- str 必须是 varchar，pos 和 len 必须是整数。任意参数为 NULL，结果总为 NULL。
- str 中的中文字符被当做字节流看待。
- 不带有 len 参数的时，则返回的子字符串从 pos 位置开始到原字符串结尾。
- pos 值为负数时，pos 的位置从字符串的结尾的字符数起；为零 0 时，可被看做 1。
- 当 len 小于等于 0，或者 pos 指示的字符串位置不存在字符时，返回结果为空字符串。

```
mysql> select substr('cbase',3),substr('cbase',3,2),substr('cbase',-3),substr('cbase',3,-2);
+------------------+--------------------+--------------------+----------------------+
| substr('cbase',3) | substr('cbase',3,2) | substr('cbase',-3) | substr('cbase',3,-2) |
+------------------+--------------------+--------------------+----------------------+
| ase               | as                 | ase                |                      |
+------------------+--------------------+--------------------+----------------------+
1 row in set (0.00 sec)
```

4.1.25. TRIM([[{BOTH | LEADING | TRAILING}][remstr] FROM] str)

- remstr 和 str 必须为 varchar 或 NULL 类型。当参数中有 NULL 时结果总为 NULL。
- 若未指定 BOTH、LEADIN 或 TRAILING,则默认为 BOTH。
- remstr 为可选项，在未指定情况下，删除空格。

```
mysql> SELECT TRIM(' bar '),  TRIM(LEADING 'x' FROM 'xxxbarxxx'),  TRIM(BOTH 'x' FROM 'xxxbarxxx'),  TRIM(TRAILING 'x' FROM 'xxxbarxxx')\G
*************************** 1. row ***************************
                  TRIM(' bar '): bar
 TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx
    TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar
TRIM(TRAILING 'x' FROM 'xxxbarxxx'): xxxbar
1 row in set (0.00 sec)
```

4.1.26. UNHEX(str)

HEX(str)的反向操作，即将参数中的每一对十六进制数字理解为一个数字，并将其转化为该数字代表的字符。结果字符以二进制字符串的形式返回。str 必须是 varchar 或 NULL。当 str 是合法的十六进制值时将按照十六进制到字节流的转换算法进行，当 str 不是十六进制字符串的时候返回 NULL。当 str 为NULL 的时候输出是 NULL。

```
mysql> select hex('cbase'),unhex('6362617365');
+--------------+----------------------+
| hex('cbase') | unhex('6362617365') |
+--------------+----------------------+
| 6362617365   | cbase                |
+--------------+----------------------+
1 row in set (0.00 sec)
```

4.1.27. UPPER(str)

将字符串转化为大写字母的字符。参数必须是 varchar 类型。若为 NULL，结果总为 NULL。

由于中文编码的字节区间与 ASCII 大小写字符不重合，对于中文， UPPER 可以很好的兼容。

```
mysql> select upper('cbase');
+----------------+
| upper('cbase') |
+----------------+
| CBASE          |
+----------------+
1 row in set (0.00 sec)
```

4.1.28. L**OWER(str)**

将字符串转化为小写字母的字符。参数必须是 varchar 类型。若为 NULL，结果总为 NULL

```
mysql> select lower('CBASE');
+----------------+
| lower('CBASE') |
+----------------+
| cbase          |
+----------------+
1 row in set (0.00 sec)
```

4.1.29. DECODE()

DECODE(value, if1, then1, if2, then2, if3, then3, ..., else)

value：可以表示表中的任意一列（不考虑数据类型）或者任意一个计算结果（如一个日期减去另一日期、字符列的substr、一个数乘以另一个数等）。每一列都对value进行测试，如果value符合条件if1，则decode函数的结果为then1；如果value符合条件if2，则decode函数的结果为then2，等等。事实上，可以构造多对if-then。如果value与任意一个if都不符，则decode的结果为else。每个if、then和else都可以是表列或者一个函数或者计算的结果。最多可以在括号中包含255个元素。

```
mysql> select * from test;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    1 |    1 | 1    |
|    2 |    2 | 2    |
|    3 |    3 | 3    |
|    4 |    4 | 4    |
+------+------+------+
4 rows in set (0.01 sec)

mysql> select c1,c2,decode(c2,1,'a',2,'b',3,'c') c4 from test;
+------+------+------+
| c1   | c2   | c4   |
+------+------+------+
|    1 |    1 | a    |
|    2 |    2 | b    |
|    3 |    3 | c    |
|    4 |    4 | NULL |
+------+------+------+
4 rows in set (0.00 sec)

mysql> select c1,c2,decode(c2,1,'a',2,'b',3,'c','d') c4 from test;
+------+------+------+
| c1   | c2   | c4   |
+------+------+------+
|    1 |    1 | a    |
|    2 |    2 | b    |
|    3 |    3 | c    |
|    4 |    4 | d    |
+------+------+------+
4 rows in set (0.00 sec)
```

4.1.30. FLOOR()和CEIL()

Floor()：向下取整函数

Ceil()：向上取整函数

```
mysql> select floor(-1.5);
+-------------+
| floor(-1.5) |
+-------------+
|          -2 |
+-------------+
1 row in set (0.00 sec)

mysql> select floor(1.5);
+------------+
| floor(1.5) |
+------------+
|          1 |
+------------+
1 row in set (0.00 sec)

mysql> select ceil(-1.5);
+------------+
| ceil(-1.5) |
+------------+
|         -1 |
+------------+
1 row in set (0.00 sec)

mysql> select ceil(1.5);
+-----------+
| ceil(1.5) |
+-----------+
|         2 |
+-----------+
1 row in set (0.00 sec)
```

4.2. 聚集函数

CBASE的聚集函数中，表达式只能出现一个。例如：不支持COUNT(C1，C2)，仅支持COUNT(C1)。

前提：test表，数据如下：

```
mysql> select * from test where id <10;
+------+------+
| id   | age  |
+------+------+
|    4 |    4 |
|    5 |    5 |
|    6 |    6 |
|    7 |    7 |
|    8 |    8 |
|    9 |    9 |
+------+------+
6 rows in set (0.01 sec)
```

4.2.1. AVG()

返回指定组中的平均值，空值被忽略。

```
mysql> select avg(id) from test where id <10;
+---------+
| avg(id) |
+---------+
|       6 |
+---------+
1 row in set (0.01 sec)
```

4.2.2. COUNT()

返回指定组中的行数

```
mysql> select count(id) from test where id <10;
+-----------+
| count(id) |
+-----------+
|         6 |
+-----------+
1 row in set (0.01 sec)
```

4.2.3. MAX()

返回指定数据中的最大值

```
mysql> select max(id) from test where id <10;
+---------+
| max(id) |
+---------+
|       9 |
+---------+
1 row in set (0.01 sec)
```

4.2.4. MIN()

返回指定数据中的最小值

```
mysql> select min(id) from test where id <10;
+---------+
| min(id) |
+---------+
|       4 |
+---------+
1 row in set (0.00 sec)
```

4.2.5. SUM()

返回指定组中的和

```
mysql> select sum(id) from test where id <10;
+---------+
| sum(id) |
+---------+
|      39 |
+---------+
1 row in set (0.00 sec)
```

4.2.6. ROW_NUMBER()

ROW_NUMBER()是一个分析函数，它会为分组内的每一条记录按序指定一个行号。query_partition_clause指定分组规则，order_by_clause指定排序规则，序号从1开始。

ROW_NUMBER()常与子查询结合使用实现分页功能。

```
mysql> select *from test;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    1 |    1 | 1    |
|    2 |    2 | 2    |
|    3 |    3 | 3    |
|    4 |    4 | 4    |
|    5 |    1 | 1    |
|    6 |    2 | 2    |
|    7 |    3 | 3    |
|    8 |    4 | 4    |
+------+------+------+
8 rows in set (0.00 sec)

mysql> select c1,c2,c3, row_number() over ( partition by c2 order by c3) from test;
+------+------+------+-----------------------------------------------------+
| c1   | c2   | c3   | row_number() over ( partition by c2 order by c3) |
+------+------+------+-----------------------------------------------------+
|    1 |    1 | 1    |                                                   1 |
|    5 |    1 | 1    |                                                   2 |
|    2 |    2 | 2    |                                                   1 |
|    6 |    2 | 2    |                                                   2 |
|    3 |    3 | 3    |                                                   1 |
|    7 |    3 | 3    |                                                   2 |
|    4 |    4 | 4    |                                                   1 |
|    8 |    4 | 4    |                                                   2 |
+------+------+------+-----------------------------------------------------+
8 rows in set (0.01 sec)
```

5. CBASE SQL语句参考

5.1. 数据定义语言（DDL）

DDL（Data Definition Language）用来定义和管理数据库以及数据库中的各种对象的语句，这些语句包括CREATE、ALTER和DROP等语句。

CBASE支持的DDL主要有CREATE DATABASE, ALTER DATABASE, DROP DATABASE, CREATE TABLE，ALTER TABLE，DROP TABLE，CREATE FUNCTION，DROP FUNCTION，CREATE INDEX，DROP INDEX等

5.1.1. CREATE DATABASE**语句**

CREATE DATABASE语句用于在系统中创建一个新库

* 格式

CREATE DATABASE db_name；

- 这条SQL语句为系统添加一个库，库名为db_name
- 系统对库名长度有限制，不能使库名的长度超过15
- 不允许创建重名的库

* 举例

执行以下语句，创建新库。

```
mysql> CREATE DATABASE test;
Query OK, 0 rows affected (0.04 sec)
```

5.1.2. DROP DATABASE**语句**

DROP DATABASE语句用于删除系统中存在库

* 格式

DROP DATABASE *db_name*；

- 这条SQL语句为删除一个已经创建的库，库名为db_name
- 当db_name写成并不存在的库时，系统会报错
- 删除库之前必须要确保库中的表先被删除完毕，否则系统不允许删除该库，报错

* 举例

执行以下语句，删除当前系统中存在的库。

```
mysql> drop database test;
Query OK, 1 row affected (0.04 sec)
```

5.1.3. CREATE TABLE 语句

该语句用于在 CBASE 数据库中创建新表。

* 格式

CREATE TABLE [IF NOT EXISTS] table_name (column_name data_type

[NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT], ...,**

PRIMARY KEY (column_name1, column_name2...)) [table_func_list][table_options_list];**

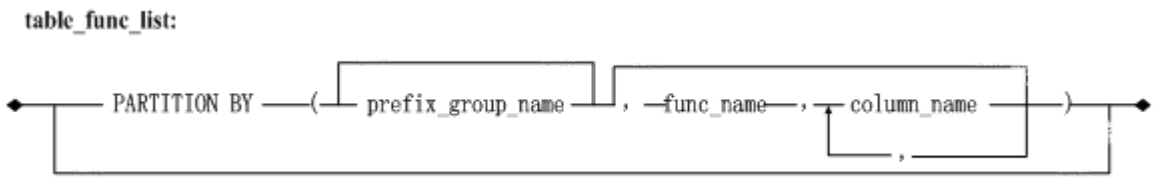
- 使用“IF NOT EXISTS”时，即使创建的表已经存在，也不会报错，如果不指定时，则会报错。
- “data_type”请参见“1.2 数据类型”章节。
- NOT NULL， DEFAULT， AUTO_INCREMENT 用于列的完整性约束。在

选项可以在 SQL 语句中出现，但目前尚未实现。

- table_func_list用于对表指定分区函数，可以指定哈希分区函数、枚举分区函数、传统RANGE分区函数、传统LIST分区函数。其中，哈希和枚举分区函数相关请参见“2.4 CREATE FUNCTION” “2.5 DROP FUNCTION”章节。

（一）创建表时指定哈希分区函数、枚举分区函数。

table_func_list的写法为：



prefix_group_name：为分区组名的前缀。func_name：为分区函数名称。

column_name：为分区列名，作为分区函数的实参传入参与计算，可以为多列但要与分区函数参数个数相等。

说明：

1. 如果指定了prefix_group_name 但没有指定func_name和column_name，则表示该表无分区，并且整表映射到用户指定的名为prefix_group_name的group中。
2. 如果没有指定prefix_group_name 但指定了func_name和column_name，则表示以该表名称作为prefix_group_name，即如果用户创建的表名为test且没有指定group前缀名，则group前缀名为该表名test。

3. 如果既指定了prefix_group_name 也指定了func_name和column_name，则表示以prefix_group_name作为group前缀，与用名为func_name的分区函数计算的结果进行拼接作为group名。
4. 如果既没有指定prefix_group_name 也没有指定func_name和column_name，但是有括号，则表示该表是无分区的，并且是整表映射到__user_default_group 这个group中。
5. 如果创建表时没有书写table_func_list这个语法部分，则表示该表是无分区的，并且是整表映射到__user_default_group 这个group中。

注意：

1. 在建立有分区的表时，必须要先确保指定的分区函数已经存在，否则报错。
2. 建表时指定的分区列的个数应与分区函数参数的个数相同，否则报错。
3. 只有主键列的子集可以作为分区函数的参数传入。
4. 使用枚举分区时，会将枚举值按照表中分区列的类型进行转换，再进行匹配。插入的值在枚举值中找不到匹配的值时，会进行报错，不允许插入该数据。

制限：

建议在使用哈希分区函数时，作为函数参数传入的列为int类型。

（二）创建表时指定RANGE分区、LIST分区。

Range partition:

使用“range”关键字表示范围划分，“values less than”关键字来定义不同的范围，范围间应该连续且不相交。对应范围分区的table_func_list 的语法为高亮部分，如下：

table_func_list:

PARTITION BY

[LINEAR] HASH(*expr*)

| [LINEAR] KEY (*column_list*)

| RANGE{(*expr*) | COLUMNS(*column_list*)}

| LIST{(*expr*) | COLUMNS(*column_list*)} }

[PARTITIONS *num*]

[(*partition_definition* [, *partition_definition*] ...)]

| WITH PREFIX (prefix_name)

partition_definition:

PARTITION *partition_name* [VALUES{

LESS THAN {(*expr* | *value_list*) | MAXVALUE}

| IN (*value_list*)}]

举例如下：

CREATE TABLE employees (

id INT PRIMARY KEY,

fname VARCHAR(30),

lname VARCHAR(30),

job_code INT NOT NULL,

store_id INT NOT NULL

)

PARTITION BY RANGE (id) (

PARTITION p0 VALUES LESS THAN (6),

PARTITION p1 VALUES LESS THAN (11),

PARTITION p2 VALUES LESS THAN (16),

PARTITION p3 VALUES LESS THAN MAXVALUE

);

注意：

1. 分区名称partition_name不能重复。
2. 分区属性须是整型列名且为单列（后期可扩充为也可以是表达式，表达式只能是对其他类型列进行运算且结果为整型）。
3. Less than括号中的值必须严格递增，并且类型必须与分区列类型匹配。
4. Less than maxvalue是为了保证分区的任何value值在划分范围内，否则，如果超过所定义的范围，会报错处理。
5. 如果指定with prefix前缀名，则以指定名称为前缀名；否则不指定时，默认前缀名是表名。

Range columns partition:

语法如下：

table_func_list:

PARTITION BY

[LINEAR] HASH(*expr*)

| [LINEAR] KEY (*column_list*)

| RANGE{(*expr*) | COLUMNS(*column_list*)}

| LIST{(*expr*) | COLUMNS(*column_list*)} }

[PARTITIONS *num*]

[(*partition_definition* [, *partition_definition*] ...)]

| WITH PREFIX (prefix_name)

partition_definition:

PARTITION *partition_name* [VALUES{

LESS THAN {(*expr* | *value_list*) | MAXVALUE}

| IN (*value_list*)}]

举例如下：

CREATE TABLE t1 (

a INT,

b INT,

c VARCHAR(3),

d INT,

PRIMARY KEY(a,c,d)

)

PARTITION BY RANGE COLUMNS(a,d,c) (

PARTITION p0 VALUES LESS THAN (5,10,'ggg'),

PARTITION p1 VALUES LESS THAN (10,20,'mmmm'),

PARTITION p2 VALUES LESS THAN (15,30,'sss'),

PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)

);

注意：

1. 它与range分区类似，区别是它的分区属性可以是多列。
2. 分区属性的列的类型可以是除了整型之外的字符类型(后期可扩充至date类型、datetime类型等日期类型)。
3. 分区属性只能为列名，不接受表达式。
4. 如用户插入到（a,d,c）列的数据要与（5,10,'ggg'）进行多维比较时，按照从前向后逐个元素进行比较，按顺序如果二者满足小于条件，则直接返回true；如果二者相等，则比较下一个元素；如果二者不满足小于条件，则直接返回false。结果为false时，继续比较下一组向量（10,20,'mmm'）。

List partition：

实际功能类似于枚举分区，通过明确列出各分区包含的元素值进行分区。语法如下：

table_func_list:

PARTITION BY

[LINEAR] HASH(*expr*)

| [LINEAR] KEY (*column_list*)

| RANGE{(*expr*) | COLUMNS(*column_list*)}

| LIST{(*expr*) | COLUMNS(*column_list*)} }

[PARTITIONS *num*]

[(*partition_definition* [, *partition_definition*] ...)]

| WITH PREFIX (prefix_name)

partition_definition:

PARTITION *partition_name* [VALUES{

LESS THAN {(*expr* | *value_list*) | MAXVALUE}

| IN (*value_list*)}]

举例如下：

CREATE TABLE employees (

id INT PRIMARY KEY,

fname VARCHAR(30),

lname VARCHAR(30),

job_code INT,

store_id INT

)

PARTITION BY LIST(id) (

PARTITION pNorth VALUES IN (3,5,6,9,17),

PARTITION pEast VALUES IN (1,2,10,11,19,20),

PARTITION pWest VALUES IN (4,12,13,14,18),

PARTITION pCentral VALUES IN (7,8,15,16)

);

注意：

1. Values in中的值不能重复，否则会报错。
2. 分区名称partition_name不能重复。
3. 分区属性可以是整型列（后期可扩充至是表达式，表达式只能是对其他类型列进行运算且结果为整型）。
4. 如果插入的值没有在list分区中列出的值中找到与之匹配的，则会报错处理。
5. 如果指定with prefix前缀名，则以指定名称为前缀名；否则不指定时，默认前缀名是表名。

List columns partition：

语法如下：

table_func_list:

PARTITION BY

[LINEAR] HASH(*expr*)

| [LINEAR] KEY (*column_list*)

| RANGE{(*expr*) | COLUMNS(*column_list*)}

| LIST{(*expr*) | COLUMNS(*column_list*)} }

[PARTITIONS *num*]

[(*partition_definition* [, *partition_definition*] ...)]

| WITH PREFIX (prefix_name)

partition_definition:

PARTITION *partition_name* [VALUES{

LESS THAN {(*expr* | *value_list*) | MAXVALUE}

| IN (*value_list*)}]

举例如下：

```
CREATE TABLE customers_1 (
first_name VARCHAR(25),
last_name VARCHAR(25),
street_1 int,
street_2 VARCHAR(30),
city VARCHAR(15),
PRIMARY KEY(street_1,city)
)
PARTITION BY LIST COLUMNS(city, street_1) (
PARTITION pRegion_1 VALUES IN(('Oskarshamn',1), ('Högsby',2)),
PARTITION pRegion_2 VALUES IN(('Vimmerby',1), ('Hultsfred',2))
);
```

注意：

1. 它与List partition分区类似，区别是它的分区属性可以是多列。
2. 分区属性的列的类型可以是除了整型之外的字符类型（后期可扩充date类型、datetime类型等日期类型）。
3. 分区属性只能为列名，不接受表达式。
4. 多维匹配必须每个元素都相同才为匹配成功。

l “table_option_list”内容请参见表 2-1，各子句间用“,”隔开。

表 2-1 表选项

参数	含义	举例
EXPIRE_INFO	在 UpdateServer 中的动态数据和 ChunkServer 中的静态数据合并时，满足表达式的行会自动删除。一般可用于自动删除过期数据。	EXPIRE_INFO = '$SYS_DATE > c1 + 246060'，表示自动删除 c1 列的值比当前时间小 24 小时的行，其中“246060”表示过期的微秒数。
TABLET_MAX_SIZE	这个表的 Tablet 最大尺寸，单位为字节，默认为256MB。	TABLET_MAX_SIZE = 268435456
REPLICA_NUM	这个表的 tablet 复本数，默认值为 3。	REPLICA_NUM = 3
COMPRESS_METHOD	存储数据时使用的压缩方法名，目前提供的方法有以下三种：· none（默认值，表示不作压缩）· lzo_1_0 · snappy_1_0	COMPRESS_METHOD = 'none'
USE_BLOOM_FILTER	对本表读取数据时，是否使用 Bloom Filter。· 0：默认值，不使用。 · 1：使用。	USE_BLOOM_FILTER = 0
COMMENT	添加注释信息。	COMMENT='create by Bruce'
table_id	指定表的 ID。如果指定的table_id 小于 1000，需要打开 RootServer 的配置项开关“ddl_system_table_switch”。	table_id=1000

注意：CBASE内部数据以b+ 树为索引，按照 Primary Key 排序，建表的时候，必须指定 Primary Key。Primary Key 有两种指定方式，详细请参见“举例”部分。另外，CBASE 不允许用户创建只包含主键列的表。

* 举例

1. 执行以下命令，创建数据库表。

CREATE TABLE test (c1 int primary key, c2 varchar)

REPLICA_NUM = 3, COMPRESS_METHOD = 'none';

或者

CREATE TABLE test (c1 int, c2 varchar, primary key(c1))

REPLICA_NUM = 3, COMPRESS_METHOD = 'none';

2. 执行命令查看表信息：

SHOW tables;

DESCRIBE test;

```
mysql> SHOW tables;
+------------+
| table_name |
+------------+
| test       |
+------------+
1 row in set (0.00 sec)

mysql> DESCRIBE test;
+-------+----------------+----------+------+---------+-------+
| field | type           | nullable | key  | default | extra |
+-------+----------------+----------+------+---------+-------+
| c1    | int            | YES      |    1 | NULL    |       |
| c2    | varchar(65536) | YES      |    0 | NULL    |       |
+-------+----------------+----------+------+---------+-------+
2 rows in set (0.00 sec)
```

3. 创建以'a'为分组前缀名,hs为分区函数，c1为分区列的表。

CREATE TABLE t1(c1 int primary key,c2 varchar(20)) partition by(a,hs,c1);

执行SHOW TABLE RULES;查看表对应的分区信息。

```
mysql> SHOW TABLE RULES;
+----------+---------------+------------+----------------+-------------+-----------+----------+
| table_id | start_version | table_name | partition_type | prefix_name | rule_name | par_list |
+----------+---------------+------------+----------------+-------------+-----------+----------+
|     3001 |             2 | t1         |              1 | a           | hs        | c1       |
+----------+---------------+------------+----------------+-------------+-----------+----------+
1 row in set (0.00 sec)
```

4. 创建以RANGE分区的表。

CREATE TABLE t2(c1 int,c2 varchar(20),c3 int,primary key(c1,c2)) partition by range(c1) (partition r0 values less than(5),partition r1 values less than (10),partition r2 values less than (MAXVALUE));

执行SHOW TABLE RULES;查看表对应的分区信息。

```
mysql> SHOW TABLE RULES;
+----------+---------------+------------+----------------+-----------------+-------------+----------+
| table_id | start_version | table_name | partition_type | prefix_name     | rule_name   | par_list |
+----------+---------------+------------+----------------+-----------------+-------------+----------+
|     3004 |             2 | t2         |              1 | t2#r0,t2#r1,t2#r2 | __range@3004 | c1       |
+----------+---------------+------------+----------------+-----------------+-------------+----------+
1 row in set (0.00 sec)
```

5.1.4. ALTER TABLE 语句

该语句用于修改已存在的表的设计。

CBASE 数据库目前只支持增加和删除列,修改表的绑定分区规则。

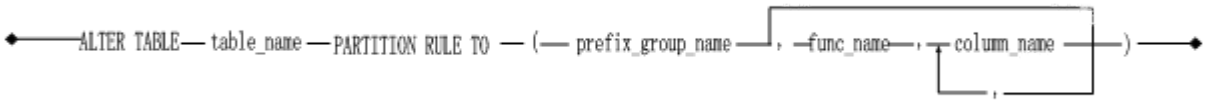
* 格式

增加列：ALTER TABLE table_name ADD [COLUMN] column_name data_type;

删除列：ALTER TABLE table_name DROP [COLUMN] column_name;

修改表绑定的分区规则：

Alter_table_partition:



注意：

- 修改表的分区规则是对下一数据版本生效，对当前版本不生效。
- 如果指定了prefix_group_name 但没有指定func_name和column_name，则表示该表无分区，并且整表映射到用户指定的名为prefix_group_name的group中。
- 如果没有指定prefix_group_name，但指定了func_name和column_name，则默认表名作为prefix_group_name。
- 如果prefix_group_name、func_name和column_name均没有指定，则表示该表全表分区，并且整表映射到__user_default_group的group中。
- 修改的分区函数名必须是已经存在的哈希函数，不能是枚举函数及以双下划线'__'开头的函数名。
- 修改的分区列必须是主键部分列，且不能重复。

* 举例

1. 增加列前，执行以下命令查看表信息：

DESCRIBE test;

```
+-------+-------------+----------+------+---------+-------+
| field | type        | nullable | key  | default | extra |
+-------+-------------+----------+------+---------+-------+
| c1    | int         |          |    1 | NULL    |       |
| c2    | varchar(-1) |          |    0 | NULL    |       |
+-------+-------------+----------+------+---------+-------+
2 rows in set (0.00 sec)
```

2. 执行以下命令增加 c3 列。

ALTER TABLE test ADD c3 int;

3. 增加列后，执行以下命令查看表信息：

DESCRIBE test;

```
+-------+-------------+----------+------+---------+-------+
| field | type        | nullable | key  | default | extra |
+-------+-------------+----------+------+---------+-------+
| c1    | int         |          |    1 | NULL    |       |
| c2    | varchar(-1) |          |    0 | NULL    |       |
| c3    | int         |          |    0 | NULL    |       |
+-------+-------------+----------+------+---------+-------+
3 rows in set (0.00 sec)
```

4. 执行以下命令删除 c3 列。

ALTER TABLE test DROP c3;

5. 删除列后，执行以下命令查看表信息：

DESCRIBE test;

```
+-------+-------------+----------+------+---------+-------+
| field | type        | nullable | key  | default | extra |
+-------+-------------+----------+------+---------+-------+
| c1    | int         |          |    1 | NULL    |       |
| c2    | varchar(-1) |          |    0 | NULL    |       |
+-------+-------------+----------+------+---------+-------+
2 rows in set (0.00 sec)
```

6. 修改表t2 的分区规则为使用'a'为分组前缀名，hs为分区函数，c1为分区列。

修改表分区规则前，t2的分区规则为：

```
mysql> SHOW TABLE RULES;
+----------+---------------+------------+----------------+-----------------+-------------+----------+
| table_id | start_version | table_name | partition_type | prefix_name     | rule_name   | par_list |
+----------+---------------+------------+----------------+-----------------+-------------+----------+
|     3004 |             2 | t2         |              1 | t2#r0,t2#r1,t2#r2 | __range@3004 | c1       |
+----------+---------------+------------+----------------+-----------------+-------------+----------+
1 row in set (0.00 sec)
```

执行修改命令： ALTER TABLE t2 partition rule to(a,hs,c1);

修改表分区规则后，查看t2的分区规则，可以看到在版本为下一版本3时，才生效：

```
mysql> SHOW TABLE RULES;
+----------+---------------+------------+----------------+-----------------+-------------+----------+
| table_id | start_version | table_name | partition_type | prefix_name     | rule_name   | par_list |
+----------+---------------+------------+----------------+-----------------+-------------+----------+
|     3004 |             2 | t2         |              1 | t2#r0,t2#r1,t2#r2 | __range@3004 | c1       |
|     3004 |             3 | t2         |              1 | a               | hs          | c1       |
+----------+---------------+------------+----------------+-----------------+-------------+----------+
2 rows in set (0.01 sec)
```

5.1.5. DROP TBALE 语句

该语句用于删除 CBASE 数据库中的表。

* 格式

DROP TABLE [IF EXISTS] table_name;

- 使用"IF EXISTS"时，即使要删除的表不存在，也不会报错，如果不指定时，则会报错。
- 同时删除多个表时，用","隔开。

* 举例

DROP TABLE IF EXISTS test;

5.1.6. T**RUNCATE TABLE语句**

该语句用于在CBASE中清空表中数据。

* 格式

TRUNCATE [TABLE][IF EXISTS] table_name1，table_name2, ..., table_nameN

- 允许truncate多张表
- 成功返回"0 rows affected"
- 失败返回错误码
- 使用限制：

- 1、truncate之后表不允许写直至下一次memtable冻结完成（不等合并完成即可恢复写）
- 2、暂定memtable冻结手动触发

*** 举例**

```
mysql> select * from test;
+------+------+------+------+------+------+
| id   | c1   | c2   | c3   | c4   | c5   |
+------+------+------+------+------+------+
|    1 |    1 |    1 |    1 |    1 |    1 |
|    2 |    2 |    2 |    2 |    2 |    2 |
|    3 |    3 |    3 |    3 |    3 |    4 |
+------+------+------+------+------+------+
3 rows in set (0.00 sec)

mysql> TRUNCATE  table test;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from test;
Empty set (0.00 sec)
```

5.1.7. CREATE SEQUENCE**语句**

SEQUENCE的功能是生成一个序列，这个序列可以递增、递减或为一个不变的常数。根据客户端请求返回相应的数值。

Sequence语句注意事项：

1 在使用SEQUENCE时，仅支持NEXTVAL与PREVVAL这一种写法，不支持NEXT VALUE与PREVIOUS VALUE这种写法，使用时请注意。

2 在使用SEQUENCE时，不能使用sequence-name.NEXTVAL代替NEXTVAL FOR sequence-name，也不能使用sequence-name.CURRVAL代替PREVVAL FOR sequence-name。

3 当前OB不支持触发器与用户自定义函数，因此，没有对在触发器与用户自定义函数中使用SEQUENCE的支持。

4 当前OB不支持用户自定义数据类型，因此OB不支持使用用户自定义类型创建或修改SEQUENCE。

5 在OB中赋予用户权限使用SEQUENCE功能时，该用户即可对所有以下SEQUENCE进行修改与删除等操作，使用时需要注意。

6 OB中未指定QUICK关键字时，调用SEQUENCE的NEXTVAL时，语句合法就生成值，值一旦生成就代表被消费，即使之后的过程失败，此次生成的值也无法再被使用（CYCLE与RESTART等情况除外）；当指定QUICK关键字时，在使用insert或者select时，如果在生成sequence过程中出现问题，则此次sequence使用过的值将不被更新，下次仍可使用。

7 有如下一个应用情景：未指定QUICK,在OB中使用NEXTVAL作为主键值插入，此时该值已经作为主键存在，则插入失败，但该NEXTVAL的值被认为已经使用，不断调用这个语句进行插入，NEXTVAL的值将会不断增加，直到NEXTVAL的值不是主键时即可插入成功。在指定QUICK时，OB中的策略是插入成功后才认为NEXTVAL的值被使用。

8 ORDER选项在目前的OB环境下是无效的。

9 CACHE选项在目前的OB环境下是无效的。

10 OB目前实现的SEQUENCE是对所有session有效，不支持单个session有效。

11 CASE表达式中不可以使用sequence。

12 聚集函数的参数列表中不可以使用sequence。

13 Join condition of a join（join连接条件中）中不可以使用sequence。

14 在select的order by语句中不可以使用sequence。

15 查询NEXTVAL的操作必须在查询PREVVAL之前。

* 格式

CREATE [OR REPLACE] SEQUENCE sequence-name [AS data-type / AS INTEGER][START WITH numeric-constant / START WITH 1] [INCREMENT BY numeric-constant / INCREMENT BY 1][MINVALUE numeric-constant / NO MINVALUE] [MAXVALUE numeric-constant / NO MAXVALUE][CYCLE / NO CYCLE] [QUICK / NO QUICK]

选项含义

OR REPLACE

此选项无默认值，若要新建的SEQUENCE的sequence-name已经存在，则调用OR REPLACE，对已经存在的SEQUENCE进行重新定义。

AS data-type

此选项用于指定SEQUENCE的数据类型，可以为：INTEGER、BIGINT、DECIMAL。其中INTEGER为32为整型，DECIMAL的范围（scale）为1~31，精度（precision）必须为0。默认为int64_t。

START WITH

此选项用于设定SEQUENCE的起始值，正数、负数、零均可但必须为整型。起始值可以不在最大最小值所限定的范围之内。若创建SEQUENCE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。默认为1。

INCREMENT BY

此选项用于设定SEQUENCE每次自增或自减的步长。该值为正数时，这里SEQUENCE为递增序列；该值为负数时，SEQUENCE为递减序列；该值为0时，SEQUENCE为常数序列。默认为1。

MINVALUE or NO MINVALUE

此选项用于设定SEQUENCE的最小值。若用户在创建SEQUENCE时未指定MINVALUE，或设定为NO MINVALUE，则采取默认值。递增序列默认为START WITH的值，递减序列默认为数据类型所能表示的最小值。

MAXVALUE or NO MAXVALUE

此选项用于设定SEQUENCE的最大值。若用户在创建SEQUENCE时未指定MAXVALUE，或设定为NO MAXVALUE，则采取默认值。递增序列默认为数据类型所能表示的最大值，递减序列默认为START WITH的值。

CYCLE or NO CYCLE

此选项用于设定SEQUENCE到达边界（即MINVALUE与MAXVALUE所确定的范围）后是否重新开始循环，若用户不指定则默认为NO CYCLE，NO CYCLE表示当SEQUENCE到达边界后无法再继续生成值。默认为NO CYCLE。

QUICK or NO QUICK

该字段为CBASE特有的一个字段，当不指定该字段时，默认为NO QUICK，该sequence的使用将保持严格的独占（排它锁）递增（递减）， 同一时刻只能有一个session独占该sequence，其他session处于等待状态。当指定该字段时，该sequence的使用将不考虑sequence数据的一致性，多session下，所有session共享该sequence，如果所有session同时使用该sequence，可能出现sequence值重复情况，但是如果用户能保证使用时，避免多session同时使用，则该字段将大大提高使用效率。

* 举例

```
mysql> CREATE OR REPLACE SEQUENCE seq1 AS integer START WITH 1 INCREMENT BY 1 MINVALUE -1 MAXVALUE 1 CYCLE;
Query OK, 1 row affected (0.00 sec)
```

5.1.8. ALTER SEQUENCE语句

ALTER命令用于修改SEQUENCE的参数，使用时必须包含其中的一个参数。在使用RESTART WITH numeric-constant参数时，将会重置SEQUENCE的当前值，而不会修改START WITH的值；在仅使用RESTART时，将会把这里SEQUENCE当前值重置为创建SEQUENCE时确定的START WITH值。

注：在系统表中存放的SEQUENCE的START WITH值只能通过OR REPLACE命令进行修改。

* 格式

ALTER SEQUENCE sequence-name [RESTART / RESTART WITH numeric-constant][INCREMENT BY numeric-constant] [MINVALUE numeric-constant / NO MINVALUE][MAXVALUE numeric-constant / NO MAXVALUE] [CYCLE / NO CYCLE][QUICK / NO QUICK]

1. 选项含义

RESTART

此选项用于使SEQUENCE重新从开始生成。若用户输入RESTART，则从START WITH设置的值开始生成；若用户输入的是 其中的参数RESTART WITH numeric-constant，则SEQUENCE将从numeric-constant的值开始重新生成。

其余选项

其余选项的含义与创建时的参数含义保持一致，在此不再赘述。需要注意的是ALTER SEQUENCE语句与UPDATE语句类似，是对SEQUENCE的修改，未输入的参数保持不变。

* 举例

```
mysql> ALTER SEQUENCE seq1 RESTART WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE  NO CYCLE;
Query OK, 1 row affected (0.02 sec)
```

5.1.9. DROP SEQUENCE语句

删除SEQUENCE。

* 格式

DROP SEQUENCE sequence-name RESTRICT;

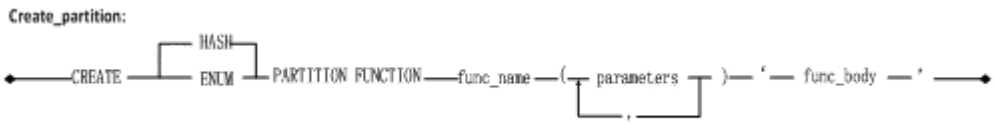
* 举例

```
mysql> DROP SEQUENCE seq1;
Query OK, 1 row affected (0.01 sec)
```

5.1.10. CREATE FUNCTION 语句

该语句用于在 CBASE 数据库中创建分区函数。

* 格式



不允许创建以双下划线开头的分区函数。不允许创建同名的分区函数。支持创建哈希函数和枚举函数，创建时需指定关键字hash或enum。会针对不同类型的分区函数对函数体进行检查。

Eg：create hash partition function hs(x) '(x+1)%2';

Enum函数是创建几组映射规则。如果有一组值输入，该组输入值映射到哪个组里，即得一个枚举值（枚举值默认从0开始递增）。输入的函数实体必须有方括号和圆括号,括号之间用逗号隔开，在方括号"[ ]"内的枚举元素所得的枚举值相同，方括号内用圆括号"()"来确定一组值。

Eg: create enum partition function em(x,y) '[(1,1),(2,3)],[(3,4)]';

使用枚举函数作为分区函数时，插入的数据不符合函数体内任意一组枚举值时，会报错。

* 举例

创建哈希分区函数：

CREATE hash partition function hs(x) '(x+1)%2';

查看创建的分区函数：

```
mysql> SHOW PARTITION FUNCTIONS LIKE 'hs';
+-----------+--------------+---------------+-----------+
| rule_name | rule_par_num | rule_par_list | rule_body |
+-----------+--------------+---------------+-----------+
| hs        |            1 | x             | x+1       |
+-----------+--------------+---------------+-----------+
1 row in set (0.01 sec)
```

创建枚举分区函数：

CREATE enum partition function em(x,y) '[(1,1),(2,3)],[(3,4)]';

查看创建的分区函数：

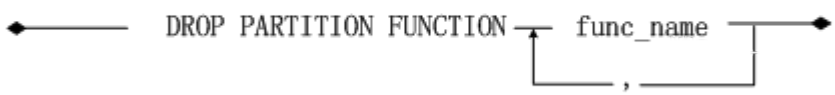
```
mysql> SHOW PARTITION FUNCTIONS LIKE 'em';
+-----------+--------------+---------------+----------------------+
| rule_name | rule_par_num | rule_par_list | rule_body            |
+-----------+--------------+---------------+----------------------+
| em        |            2 | x,y           | [(1,1),(2,3)],[(3,4)] |
+-----------+--------------+---------------+----------------------+
1 row in set (0.00 sec)
```

5.1.11. DROP FUNCTION 语句

该语句用于在 CBASE 数据库中删除分区函数。

* 格式

Drop_partition:



在执行删除分区函数时，会先检查是否有使用该分区函数的表，如果有则拒绝删除该分区函数，否则，则允许删除。不允许删除以双下划线开头的分区函数。

* 举例

DROP partition function hs;

```
mysql> desc test;
+-------+-------+----------+------+---------+-------+
| field | type  | nullable | key  | default | extra |
+-------+-------+----------+------+---------+-------+
| id    | int32 |        1 |    1 | NULL    |       |
| c1    | int32 |        1 |    0 | NULL    |       |
| c2    | int32 |        1 |    0 | NULL    |       |
| c3    | int32 |        1 |    0 | NULL    |       |
| c4    | int32 |        1 |    0 | NULL    |       |
| c5    | int32 |        1 |    0 | NULL    |       |
+-------+-------+----------+------+---------+-------+
6 rows in set (0.00 sec)
```

5.1.12. C**REATE INDEX语句**

在数据表上可以创建索引，对数据表中的一列或者多列进行排序。创建合适的索引表，可以提高查询速度，降低数据库系统的性能开销。该语句用于在CBASE的表中创建二级索引。

* 格式

CREATE INDEX idx _name ON table_name(columnname_list1) STORING (columnname_list2); 其中idx _name为INDEX的索引名，columnname_list1为索引列，其列数与源表主键列之和不超过16。columnname_list2 为冗余列，其作用是当SQL查询该数据表的冗余列时，能从索引表直接返回冗余列的数据，从而减少一次回表查询。冗余列可以指定，也可以不指定。i1是INDEX的索引名，而存储在CBASE当中的索引表，表名是按照**【数据表id**idx**索引名】的规则生成，也就是说此时索引表名为**3002**idx**i1。

使用索引时，涉及到Hint，删除索引的操作时，可以直接使用索引名。CBASE会自动处理使用到对应的索引。

对于不同表来说，可以建立相同索引名的索引表（这是因为索引表名有自己的生成规则的缘故），而对于一张表来说，索引名是不能重复的。

因为数据表的主键是用来构造索引表的复合主键的，放在storing列里没有意义。创建索引时候要注意STORING列中不能出现主键列，不然会报错。

注意事项：

- 一张数据表最多可以创建10张索引表。
- 一张数据表的所有索引表的所有列不能超过100列

合并其间不能创建、删除索引，创建索引需要在合并后生效，删除索引立即生效。

索引状态说明：0: 索引初始化完成，正在做备份迁移

1:索引状态正常，可以正常使用

2:索引出错，不能使用

3:索引正在删除

4:索引创建成功，还没有做静态数据同步，不能使用

* 举例

```
mysql> CREATE INDEX i1 ON test (c1) STORING (c2);
Query OK, 0 rows affected (0.14 sec)

mysql> show index on test;
+-----------------+--------+
| index_name      | status |
+-----------------+--------+
| __3002__idx__i1 |      4 |
+-----------------+--------+
1 row in set (0.00 sec)
```

5.1.13. DROP INDEX语句

该语句用于删除表中的二级索引。

* 格式

DROP INDEX index_list ON t1;

其中index_list是索引表的表名，可以使一张，也可以是多张。

也可以使用DROP TBALE语句根据索引表的全名删除索引表

合并时不允许删除索引

* 举例

```
mysql> DROP INDEX i1 on test;
Query OK, 0 rows affected (1.23 sec)

mysql> show index on test;
Empty set (0.00 sec)
```

5.2. SELECT查询操作语句

该语句用于查询表中的内容。

5.2.1. 基本查询

* 格式

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]] [LIMIT {[offset,] row_count | row_count **OFFSET** offset}];

表3-1 子句说明

子句	说明
ALL|DISTINCT	在数据库表中，可能会包含重复值。指定“DISTINCT”，则在查询结果中相同的行只显示一行；指定“ALL”，则列出所有的行；不指定时，默认为“ALL”
Select_list	列出要查询的列名，用“,”隔开。也可以用“*”表示所有列
AS other_name	为输出字段重新命名
FROM table_name	必选项，指名了从哪个表中读取数据
WHERE where_condition	可选项，WHERE 字句用来设置一个筛选条件，查询结果中仅包含满足条件的数据。where_conditions 为表达式
GROUP BY group_by_list	用于进行分类汇总
HAVING search_conditions	HAVING 字句与 WHERE 字句类似，但是HAVING 字句可以使用累计函数（如 SUM， AVG等）
ORDER BY order_list[ASC|DESC]	用来按升序（ASC）或者降序（DESC）显示查询结果。不指定 ASC 或者 DESC 时，默认为ASC
LIMIT{[offset,] row_count| row_count OFFSET offset}	强制 SELECT 语句返回指定的记录数。LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。 l 如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是1)。 l 如果只给定一个参数，它表示返回记录行的最大数目，偏移量为 0。

* 举例

l 略l

5.2.2. JOIN语句

JOIN 连接分为内连接和外连接。外连接又分为左连接、右连接和全连接。两个表联接后，可以使用 ON 指定条件进行筛选。

目前，CBASE 的 JOIN 不支持 USING 子句，并且 JOIN 的连接条件中必须至少有一个等值连接条件。

示例：

```
mysql> select a.id as a_id,a.age as a_age ,b.id as b_id ,b.age as b_age from test a join test b on a.id = b.id limit 10;
+------+-------+------+-------+
| a_id | a_age | b_id | b_age |
+------+-------+------+-------+
|    2 | 20000 |    2 | 20000 |
|    4 |     4 |    4 |     4 |
|    5 |     5 |    5 |     5 |
|    6 |     6 |    6 |     6 |
|    7 |     7 |    7 |     7 |
|    8 |     8 |    8 |     8 |
|    9 |     9 |    9 |     9 |
|   10 |    10 |   10 |    10 |
|   11 |    11 |   11 |    11 |
|   12 |    12 |   12 |    12 |
+------+-------+------+-------+
10 rows in set (0.02 sec)
```

5.2.3. 集合操作

CBASE中的集合操作主要包括UNION、EXPECT和INTERSECT。

* UNION句法

UNION操作符用于合并两个或多个SELECT语句的结果集。使用UNION需要注意以下几点：

- UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。
- 默认地， UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。
- UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

UNION 指令的目的是将两个或多个 SELECT 语句的结果合并起来。从这个角度来看， UNION 跟 JOIN 有些类似，因为这两个指令都可以由多个表格中检索数据。但是 UNION 只是将两个结果联结起来一起显示，并不是联结两个表。

示例：

```
mysql> select * from test where id < 10 union select * from test where id > 240;
+------+-------+
| id   | age   |
+------+-------+
|    2 | 20000 |
|    4 |     4 |
|    5 |     5 |
|    6 |     6 |
|    7 |     7 |
|    8 |     8 |
|    9 |     9 |
|  241 |    24 |
|  250 |    25 |
|  251 |    25 |
|  252 |  NULL |
+------+-------+
11 rows in set (0.01 sec)
```

* EXCEPT句法

EXCEPT 用于查询第一个集合中存在，但是不存在于第二个集合中的数据。

示例：

```
mysql> select * from test where id < 10 except select * from test where id < 10;
Empty set (0.01 sec)
```

* INTERSECT句法

INTERSECT 用于查询在两个集合中都存在的数据。

示例：

```
mysql> select * from test where id < 10 intersect select * from test where id > 5;
+------+------+
| id   | age  |
+------+------+
|    6 |    6 |
|    7 |    7 |
|    8 |    8 |
|    9 |    9 |
+------+------+
4 rows in set (0.01 sec)
```

5.2.4. DUAL虚拟表

DUAL 是一个虚拟的表，可以视为一个一行零列的表。当我们不需要从具体的表来取得表中数据，而是单纯地为了得到一些我们想得到的信息，并要通过SELECT 完成时，就要借助一个对象，这个对象就是 DUAL。一般可以使用这种特殊的 SELECT 语法获得用户变量或系统变量的值。当 SELECT 语句没有 FROM 子句的时候，语义上相当于 FROM DUAL，此时，表达式中只能是常量表达式。

* 格式

SELECT [**ALL** | **DISTINCT**] *select_list* [**FROM DUAL** [**WHERE** *where_condition*]][**LIMIT** {[*offset*,] *row_count* | *row_count* **OFFSET** *offset*}];

* 举例

```
mysql> select 100%10;
+--------+
| 100%10 |
+--------+
|      0 |
+--------+
1 row in set (0.00 sec)

mysql> select 100%10 from dual;
+--------+
| 100%10 |
+--------+
|      0 |
+--------+
1 row in set (0.00 sec)
```

5.2.5. SELECT ... FOR UPDATE句法

SELECT ... FOR UPDATE 可以用来对查询结果所有行上排他锁，以阻止其他事务的并发修改，或阻止在某些事务隔离级别时的并发读取。即使用 FOR UPDATE 语句将锁住查询结果中的元组，这些元组将不能被其他事务的 UPDATE，DELETE 和 FOR UPDATE 操作，直到本事务提交。

* 格式

SELECT * FROM test where id = 1 FOR UPDATE;

注意的是，目前 CBASE 实现有如下限制：

- 必须是单表查询。
- Where 条件中必须限定单行。

* 举例

```
mysql> select * from test for update;
ERROR 5047 (HY000): OB-5047: Primary key column 16 not specified in the WHERE clause
mysql> select * from test where id =1 for update;
+------+------+------+------+------+------+
| id   | c1   | c2   | c3   | c4   | c5   |
+------+------+------+------+------+------+
|    1 |    1 |    1 |    1 |    1 |    1 |
+------+------+------+------+------+------+
1 row in set (0.02 sec)
```

5.2.6. IN和OR

CBASE 支持逻辑运算“IN”和“OR”，其中 IN 可以直接定位到查询的数据，而OR 需要进行全表扫描，因此推荐使用 IN 语句。

5.3. 数据操作语言（DML）

DML（Data Manipulation Language）的主要作用是根据需要写入、删除、更新数据库中的数据。

CBASE支持的DML主要有INSERT，REPLACE，SELECT，UPDATE和DELETE。

5.3.1. INSERT 语句

该语句用于添加一个或多个记录到表。

CBASE是Key/value型数据库，所以INSERT的行必须包含所有主键列的值，且主键列不能为NULL。

* 格式

INSERT INTO table_name [(column_name,...)] VALUES (column_values,...);

[(column_name,...)] 用于指定插入数据的列。

同时插入多列时，用“，”隔开。

* 举例

执行以下命令，插入行。

```
INSERT INTO test VALUES (1,1000),(2,2000);
```

执行以下命令，查看插入的行。

```
mysql> select * from test where id in (1,2);
+------+------+
| id   | age  |
+------+------+
|    1 | 1000 |
|    2 | 2000 |
+------+------+
2 rows in set (0.01 sec)
```

5.3.2. REPLACE 语句

REPLACE 语句的语法和 INSERT 相同，语义有别：如果本行已经存在，则修改对应列的值为新值；如果不存在，则插入。

* 格式

REPLACE INTO *table_name* [(*column_name*,...)] **VALUES** (*column_values*,...);

- [(*column_name*,...)] 用于指定插入数据的列。
- 同时插入多列时，用“，”隔开。

* 举例

1．执行以下命令，插入行。

```
REPLACE INTO test VALUES (1,10000),(2,20000);
```

2．执行以下命令，查看插入的行。

```
mysql> select * from test where id in (1,2);
+------+-------+
| id   | age   |
+------+-------+
|    1 | 10000 |
|    2 | 20000 |
+------+-------+
2 rows in set (0.00 sec)
```

5.3.3. UPDATE 语句

该语句用于修改表中的字段值。CBASE数据库目前只支持指定Primary key条件的单行更新，且暂不支持DEFAULT。

* 格式

UPDATE *table_name* **SET** *column_name1=column_values* [*column_name2=column_values*] ... **WHERE** *where_condition;*

* 举例

1．执行以下命令，修改10000为55555。

```
UPDATE test set age = 55555 WHERE id = 1;
```

2．执行以下命令查看修改后的信息。

```
mysql> select * from test where id =1;
+------+-------+
| id   | age   |
+------+-------+
|    1 | 55555 |
+------+-------+
1 row in set (0.00 sec)
```

5.3.4. DELETE 语句

该语句用于删除表中符合条件的行

* 格式

DELETE FROM *table_name* **WHERE** *where_condition*

- Where_condition必须指定Primary key

* 举例

1．执行以下命令删除id=1的行。其中id为表test中的Primary key。

```
DELETE FROM test WHERE id = 1;
```

2．执行以下命令查看删除行后的表信息。

```
mysql> select * from test where id =1;
Empty set (0.01 sec)
```

5.4. 数据库管理语言（DCL）

数据库管理包括用户及权限管理、修改用户变量、系统变量和系统配置。

5.4.1. 新建用户

CREATE USER 用于创建新的 CBASE 用户。创建新用户后，可以使用该用户连接 CBASE。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。

* 格式

CREATE USER 'user' [**IDENTIFIED BY** [**PASSWORD**] 'password'];

- 必须拥有全局的 CREATE USER 权限或对“__all_user”表的 INSERT权限，才可以使用 CREATE USER 命令。
- 新建用户后，“__all_user”表会新增一行该用户的表项。如果同名用户已经存在，则报错。
- 使用自选的 IDENTIFIED BY 子句，可以为账户给定一个密码。
- 此处密码为明文，存入“__all_user”表后，服务器端会变为密文存储下来。
- 同时创建多个用户时，用“,”隔开

* 举例

1．执行以下命令创建“test1”和“test2”用户，密码均为“test”

```
CREATE USER 'test1' IDENTIFIED BY 'test', 'test2'  IDENTIFIED BY 'test';
```

2．执行以下命令查看创建的用户。

```
mysql> select user_name from __all_user;
+-----------+
| user_name |
+-----------+
| admin     |
| dsadmin   |
| test1     |
| test2     |
+-----------+
4 rows in set (0.01 sec)
```

5.4.2. 删除用户

DROP USER 语句用于删除一个或多个 CBASE 用户。

* 格式

DROP USER 'user';

- 必须拥有全局的 CREATE USER 权限或对“__all_user”表的 DELETE权限，才可以使用 DROP USER 命令。
- 成功删除用户后，这个用户的所有权限也会被一同删除。
- 同时删除多个用户时，用“,”隔开。

* 举例

执行以下命令，删除“test1”用户

```
DROP USER ‘test1’;
```

5.4.3. 修改密码

用于修改 CBASE 登录用户的密码。

* 格式

```
SET PASSWORD FOR 'user' = 'password';

或者

ALTER USER 'user' IDENTIFIED BY 'password';
```

- 如果没有 For user 子句，则修改当前用户的密码。任何成功登陆的用户都可以修改当前用户的密码。
- 如果有 For user 子句，或使用第二种语法，则修改指定用户的密码。必须拥有对“__all_user”表的 UPDATE 权限，才可以修改制定用户的密码。

* 举例

执行以下命令将“test2”的密码修改为“test2”。

5.4.4. 修改用户名

用于修改 CBASE 登录用户的用户名。

* 格式

RENAME USER 'old_user' **TO** 'new_user';

- 必须拥有全局 CREATE USER 权限或者对__users 表的 UPDATE 权限，才可以使用本命令。
- 同时修改多个用户名时，用“,”隔开。

* 举例

1．修改前，查看用户名

```
mysql> select user_name from __all_user;
+-----------+
| user_name |
+-----------+
| admin     |
| dsadmin   |
| test1     |
| test2     |
+-----------+
4 rows in set (0.00 sec)
```

2．执行命令修改用户名

```
mysql> RENAME USER 'test2' TO 'test_mod';
Query OK, 0 rows affected (0.02 sec)
```

3．查看修改后的用户名

```
mysql> select user_name from __all_user;
+-----------+
| user_name |
+-----------+
| admin     |
| dsadmin   |
| test1     |
| test_mod  |
+-----------+
4 rows in set (0.01 sec)
```

5.4.5. 锁定用户

锁定或者解锁用户。被锁定的用户不允许登陆。

* 格式

锁定用户：ALTER USER 'user' LOCKED;

解锁用户：ALTER USER 'user' UNLOCKED;

必须拥有对“__users”表的 UPDATE 权限，才可以执行本命令。

* 举例

锁定用户：ALTER USER 'test_mod' LOCKED;

解锁用户：ALTER USER 'test_mod' UNLOCKED

5.4.6. 用户授权

GRANT 语句用于系统管理员授予 CBASE 用户操作权限。权限体系分为3个层级：全局权限/库权限/表权限。其中，*和.表示全局权限*、*db_name.*表示库权限、db_name.tbl_name和tbl_name表示表权限

* 格式

GRANT priv_type **ON** . **TO** 'user';

- 给特定用户授予权限。如果用户不存在则报错。
- 当前用户必须拥有被授予的权限（例如，user1 把表 t1 的 SELECT 权限授予 user2，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有GRANT OPTION 权限，才能授予成功。
- 用户授权后，该用户只有重新连接 CBASE，权限才能生效。
- 用“.”表示赋予全局权限，即对数据库中的所有库表赋权。
- 同时把多个权限赋予用户时，权限类型用“,”隔开。
- 同时给多个用户授权时，用户名用“,”隔开。
- priv_type表如下。

权限	说明
ALL PRIVILEGES	除 GRANT OPTION 以外所有权限
ALTER	ALTER TABLE 的权限
CREATE	CREATE TABLE 的权限
CREATE USER	CREATE USER，DROP USER，RENAME USER 和 REVOKE ALL PRIVILEGE的权限
DELETE	DELETE 的权限
DROP	DROP 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
REPLACE	REPLACE的权限
SUPER	SET GLOBAL 修改全局系统参数的权限

本用户自动拥有自己创建的对象。例如，用户 user1 创建了表 t1 和 t2，那用户 user1 应该自动就有对 t1 和 t2 的 ALL PRIVILEGES 及GRANT OPTION 权限，不再需要额外去授权。

* 举例

执行以下命令给“test_mod”赋予所有权限。

```
GRANT ALL PRIVILEGES, GRANT OPTION ON *.* TO 'test_mod';
```

5.4.7. 撤销权限

REVOKE 语句用于系统管理员撤销 CBASE 用户的操作权限。

* 格式

REVOKE priv_type **ON** . **FROM** 'user';

- 用户必须拥有被撤销的权限（例如， user1 要撤销 user2 对表 t1 的 SELECT权限，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有 GRANT OPTION 权限。
- 撤销“ALL PRIVILEGES”和“GRANT OPTION”权限时，当前用户必须拥有同级别 GRANT OPTION 权限，或者对权限表的 UPDATE 及DELETE 权限。
- 撤销操作不会级联。例如，用户 user1 给 user2 授予了某些权限，撤回user1 的权限不会同时也撤回 user2 的相应权限。
- 同时对用户撤销多个权限时，权限类型用“,”隔开。
- 同时撤销多个用户的授权时，用户名用“,”隔开。

- priv_type 的如表 5-1 所示。

* 举例

执行以下命令撤销“test_mod”的所有权限。

```
REVOKE ALL PRIVILEGES, GRANT OPTION ON * FROM 'test_mod';
```

5.4.8. 查看权限

SHOW GRANTS 语句用于系统管理员查看 CBASE 用户的操作权限。

* 格式

SHOW GRANTS [**FOR** user];

- 如果不指定用户名，则缺省显示当前用户的权限。对于当前用户，总可以查看自己的权限。
- 如果要查看其他指定用户的权限，必须拥有对“__all_user”的 SELECT权限。

* 举例

```
SHOW GRANTS FOR 'test_mod';
```

5.4.9. 修改用户变量

用户变量用于保存一个用户自定义的值，以便于在以后引用它，这样可以将该值从一个语句传递到另一个语句。用户变量与连接有关，即一个客户端定义的用户变量不能被其它客户端看到或使用，当客户端退出时，该客户端连接的所有变量将自动释放。

* 格式

SET @var_name = expr;

- 用户变量的形式为@var_name，其中变量名 var_name 可以由当前字符集的文字数字字符、“.”、“_”和“$”组成。
- 每个变量的 expr 可以为整数、实数、字符串或者 NULL 值。
- 同时定义多个用户变量时，用“,”隔开。

* 举例

1．执行以下命令，设置用户变量

```
mysql> set @test_a =10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @test_b =10;
Query OK, 0 rows affected (0.00 sec)
```

2．执行以下命令，查看用户变量

```
mysql> select @test_a,@test_b;
+---------+---------+
| @test_a | @test_b |
+---------+---------+
| 10      | 10      |
+---------+---------+
1 row in set (0.00 sec)
```

5.4.10. 修改系统变量

系统变量和 SQL 功能相关，存放在“__all_sys_param”表中，如 autocommit，tx_isolation 等。

CBASE 维护两种变量：

- 全局变量

影响 CBASE 整体操作。当 CBASE 启动时，它将所有全局变量初始化为默认值。修改全局变量，必须具有 SUPER 权限。

- 会话变量

影响当前连接到 CBASE 的客户端。在客户端连接 CBASE 时，使用相应全局变量的当前值对该客户端的会话变量进行初始化。设置会话变量不需要特殊权限，但客户端只能更改自己的会话变量，而不能更改其它客户端的会话变量。

* 格式

设置全局变量的格式：

SET GLOBAL system_var_name = expr;或者

SET @@GLOBAL.system_var_name = expr;

设置会话变量的格式：

SET [SESSION | @@SESSION. | LOCAL | LOCAL. | @@]system_var_name= expr;

查看系统变量的格式：

如果指定 GLOBAL 或 SESSION 修饰符，则分别打印全局或当前会话的系统变量值；如果没有修饰符，则显示当前会话值。

SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'system_var_name' | WHERE expr];

* 举例

1．执行以下命令，修改会话变量中的 SQL 超时时间。

```
SET @@SESSION.ob_tx_timeout = 900000;
```

2．执行以下命令，查看 SQL 超时时间。

```
SHOW SESSION VARIABLES LIKE 'ob_tx_timeout';
```

5.4.11. 修改系统配置项

配置项存放在“__all_sys_config”表中，一般针对每一类 Server 进行配置，一般影响某个 Server 的行为，比如 MergeServer 的线程池大小， RootServer 的负载均衡策略等，当然也可以通过指定 IP 对某个 Server 单独进行配置。

* 格式

修改系统配置项的格式：

ALTER SYSTEM SET param_name = expr [**COMMENT** 'text'][**SCOPE** = conf_scope] **SERVER_TYPE** = server_type [**CLUSTER** = cluster_id | **SERVER_IP**= 'server_ip' **SERVER_PORT** = server_port];

修改系统配置项说明表如下：

子句	说明
param_name = expr	-
COMMENT ‘text’	可选，用于添加关于本次修改的注释。建议不要省略。
SCOPE = conf_scope	SCOPE 用来指定本次配置项修改的生效范围。它的值主要有以下三种：- MEMORY：表明只修改内存中的配置项，修改立即生效，且本修改在 Server 重启以后会失效（目前暂时没有配置项支持这种方式）。- SPFILE：表明只修改配置表中的配置项值，当 Server 重启以后才生效。- BOTH：表明既修改配置表，又修改内存值，修改立即生效，且 Server 重启以后配置值仍然生效。说明：SCOPE 默认值为 BOTH。对于不能立即生效的配置项，如果 SCOPE 使用 BOTH 或 MEMORY，会报错 。
SERVER_TYPE = server_type	服务器类型，ROOTSERVER\UPDATESERVER\CHUNKSERVER\MERGESERVER。
CLUSTER = cluster_id	表明本配置项的修改正对指定集群的特定Server 类型，否则，针对所有集群的特定Server类型。
SERVER_IP= 'server_ip'SERVER_PORT = server_port	只修改指定 Server 实例的某个配置项。

- 同时修改多个系统配置项时，用“,”隔开。

查看系统配置项的格式：

SHOW PARAMETERS [**LIKE** 'pattern' | **WHERE** expr];

* 举例

1．执行以下命令，修改 Tablet 副本数。

```
ALTER SYSTEM SET tablet_replicas_num=3 COMMENT 'Modify by wn' SCOPE = SPFILE SERVER_TYPE =
ROOTSERVER SERVER_IP= '10.10.10.1' SERVER_PORT= 2500;
```

2．查看“tablet_replicas_num”。

```
SHOW PARAMETERS LIKE 'tablet_replicas_num';
```

5.4.12. 查看系统中存在的库

SHOW DATABASES语句用于查看当前系统中存在的库名。

* 格式

SHOW DATABASES；

* 举例

执行以下语句，查看系统中已经存在的库。

```
mysql> show databases;
+---------+
| db_name |
+---------+
| TANG    |
| test    |
+---------+
2 rows in set (0.00 sec)
```

5.4.13. 声明所**使用的库**

USING DATABASE语句用于声明所使用的库。

* 格式

USING DATABASE db_name；

- 这条SQL语句为声明使用一个已经创建的库，库名为db_name
- 可以连续使用这条SQL语句切换当前所需要声明使用的库
- 可以在连接数据库的时候，加上-D 参数，则默认尝试连接并使用参数后面的库名。-D与库名之间没有空格

* 举例

执行以下语句，声明所使用的库。

```
mysql> using database test;
Query OK, 0 rows affected (0.00 sec)
```

5.4.14. 查看当前**声明的库**

SHOW CURRENT DATABASE语句用于查看当前声明的库名

* 格式

SHOW CURRENT DATABASE;

* 举例

执行以下语句，查看当前声明的库名。

```
mysql> show current database;
+---------+
| db_name |
+---------+
| test    |
+---------+
1 row in set (0.00 sec)
```

5.4.15. 查看系统中**的系统表**

SHOW SYSTEM TABLES语句用于查看系统中的系统表

* 格式

SHOW SYSTEM TABLES；

* 举例

执行以下语句，查看系统中的系统表。

```
mysql> show system tables;
+---------------------------+
| table_name                |
+---------------------------+
| __first_tablet_entry      |
| __all_all_column          |
| __all_join_info           |
| __all_sys_param           |
| __all_sys_stat            |
| __all_user                |
| __all_table_privilege     |
| __all_cluster             |
| __all_server              |
| __all_trigger_event       |
| __all_sys_config          |
| __all_sys_config_stat     |
| __all_client              |
| __all_ddl_operation       |
| __all_database            |
| __all_database_privilege  |
| __all_sequence            |
| __all_truncate_op         |
| __all_partition_rules     |
| __all_table_rules         |
| __all_all_group           |
| __ups_session_info        |
| __all_cluster_stat_info   |
| __all_server_stat         |
| __all_server_session      |
| __all_statement           |
| __all_ccheckSum_info      |
| __index_process_info      |
+---------------------------+
28 rows in set (0.00 sec)
```

5.4.16. 事务处理

数据库事务（Database Transaction）是指作为单个逻辑工作单元执行的一系列操作。事务处理可以用来维护数据库的完整性，保证成批的 SQL 操作全部执行或全部不执行。

* 格式

开启事务语句格式如下：

START TRANSACTION;

或者 BEGIN [WORK];

BEGIN 和 BEGIN WORK 被作为 START TRANSACTION 的别名受到支持，用于对事务进行初始化。START TRANSACTION 是标准的 SQL 语法，并且是启动一个 ad-hoc 事务的推荐方法。一旦开启事务，则随后的SQL 数据操作语句（即 INSERT， UPDATE， DELETE， REPLACE）直到显式提交时才会生效。

提交当前事务语句格式如下：

COMMIT [WORK];

回滚事务语句格式如下：

ROLLBACK [WORK];

* 举例

```
mysql> select * from test where id = 8888;
Empty set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into test values (8888,8888);
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from test where id = 8888;
+------+------+
| id   | age  |
+------+------+
| 8888 | 8888 |
+------+------+
1 row in set (0.00 sec)
```

执行事务时，要注意以下几点：

- 采用 datatime 类型的值作为主键，在进行 INSERT 操作的时候，不能直接在 SQL INSERT 语句中采用 CURRENT_TIME()函数来获取时间作为主键列的值，比如 INSERT INTO a VALUES(4, 'a', 30, CURRENT_TIME())，则会报错的。只能在 INSERT 之前，先用CURRENT_TIME()函数获取当前时间，然后在 INSERT 语句中直接用该确定的时间值。
- 由于事务有一个超时时间，所以手动输入事务中的多个 SQL 语句的时候，由于打字时间太长，会导致整个事务超时。解决方法是：修改当前 session 中的系统变量 ob_tx_timeout，使得该值进可能大，而不至于导致超时。
- 目前在 CBASE 的事务中执行 SELECT 语句，不能读取当前事务中未提交的数据。用户如果有这个需求，可以暂时用 SELECT … FOR UPDATE 语句代替。
- CBASE支持分布式事务，支持单语句、多语句级别的跨分区事务，用户可以自由使用。

5.4.17. 实用**的SQL语句**

* SHOW 语句

SHOW 语句说明表如下：

语句	说明
SHOW COLUMNS {FROM | IN} *table_name*[LIKE 'pattern' | WHERE expr];	查看指定表的列的信息。
SHOW CREATE TABLE table_name;	查看可以用来建立指定表格的建表语句。
SHOW INDEX ON table_name;	查看某个表的索引。
SHOW PARAMETERS;	查看各个配置项在各个 Server实例上的值。
SHOW PROCESSLIST;	查看所有用户的当前连接。
SHOW [ALL] TABLES [LIKE 'pattern' | WHERE expr];	查看数据库中存在哪些表。加ALL，则查询所有表，不加ALL，则只查询用户创建的表。
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern' | WHERE expr];	查看全局或会话的系统变量，默认为当前会话。
SHOW WARNINGS [LIMIT [offset,] row_count] ;SHOW COUNT(*) WARNINGS;	SHOW COUNT(*) WARNINGS;获得上一条语句执行过程中产生的“Warning”信息，不包含“Error”。
SHOW PARTITION FUNCTIONS [LIKE 'pattern' | WHERE expr];	查看数据库中所有的分区规则，可在LIKE和WHERE中指定分区规则名。
SHOW TABLE RULES [LIKE 'pattern' | WHERE expr];	查看数据库中表所对应的分区规则，可在LIKE和WHERE中指定表名。
SHOW PARTITION GROUPS [LIKE 'pattern' | WHERE expr];	查看当前所有分组和paxos_id的对应关系，可在LIKE和WHERE中指定分区组名。

* KILL 语句

KILL [CONNECTION | QUERY] *thread_id;*

说明：*每个与 CBASE 的连接都在一个独立的线程里运行，您可以使用* **SHOW PROCESSLIST;***语句查看哪些线程正在运行，并使用 KILL thread_id 语句终止一个线程。Index 列为 thread_id。

KILL CONNECTION 与不含修改符的 KILL 一样：它会终止与给定的thread_id。

KILL QUERY 会终止连接当前正在执行的语句，但是会保持连接的原状。

* DESCRIBE 语句

DESCRIBE *table_name* [*col_name* | wild];

或者

DESC *table_name* [*col_name* | wild];

这个语句等同于 SHOW COLUMNS FROM 语句。

* EXPLAIN 语句

EXPLAIN [VERBOSE] {*select_stmt* | *insert_stmt* | *update_stmt* | *delete_stmt* |

delete_stmt};

这个语句可以输出 SELECT, INSERT, UPDATE, DELETE, REPLACE 等 DML

语句内部的物理执行计划。VERBOSE 模式也会输出逻辑执行计划。DBA 和开发人员可以根据 EXPLAIN 的输出来优化 SQL 语句。

* WHEN 语句

statement WHEN *expr*|ROW_COUNT(*statement*) *op expr*;

WHEN 子句是 CBASE 扩展的 SQL 语法，其含义为：当 WHEN 子句的条件满足的时候，才执行 WHEN 前的主句。

· statement 为 SELECT、FOR UPDATE、UPDATE 、DELETE、INSERT、

REPLACE 等语句。

· WHEN 子句可以嵌套和组合。例如：

SELECT * FROM t1 WHERE c1 = 1000 FOR UPDATE WHEN ROW_COUNT(UPDATE t2 SET c1 = c1 +1 WHERE c0 = 1000) >= 0;

* Hint 语法

hint 是一种特殊的 SQL 注释，SQL 注释的一般语法和 C 语言的注释语法相同， 即/* ... */。*而 hint 的语法在此基础上加了一个加号，型如*/*+* ... */。既然是注释， 那么如果 Server 端不认识你 SQL 语句中的 hint，它可以直接忽略而不必报错。这样做的一个好处是，同一条 SQL 发给不同的数据库产品，不会因为 hint 引起语法错误。hint 只影响数据库服务器端内部优化的逻辑，而不影响 SQL 语句本身的语义。

CBASE 支持的 hint 有以下几个特点：

· 不带参数的，如/*+ KAKA */。

· 带参数的，如/*+ HAHA(param) */。

· 多个hint 可以写到同一个注释中，用逗号分隔。

· SELECT 语句的 hint 必须近接在关键字 SELECT 之后，其他词之前。如：

SELECT /*+ KAKA */ ...。

· UPDATE 语句的 hint 必须紧接在关键字 UPDATE 之后。

CBASE支持的hint如**表 7-2**所示

表 7-2

hint	参数	适用语句	含义	最早支持版本
READ_STATIC	无	SELECT	只读 ChunkServer 上的静态数据，数据不保证一致性。如果用户建表的时候加上了READ_STATIC 属性那么即使不使用本 hint，用户的 SELECT 也是只读静态数据的。	CBASE，0.4.1
HOTSPOT	无	UPDATE	在 WHEN 连接的复合语句中，表明本 UPDATE 所更新的行是"热点行"，执行计划在 UpdateServer 端会根据热点行排队，以减少锁冲突。	CBASE 0.4.2

* PREPARE 语句

PREPARE *stmt_name* FROM *preparable_stmt;*

· *preparable_stmt* 为 SQL 数据操作语句，预备好的语句在整个 SQL 会话期间可以使用 *stmt_name* 这个名字来执行。

· 数据操作语句（即 SELECT, REPLACE, INSERT, UPDATE, DELETE ）都可以被预备执行。

· 在被预备的 SQL 语句中，可以使用问号（？）表明一个之后执行时才绑定的参数。问号只能出现在 SQL 语句的常量中。一个被预备的语句也可以不包含问号。

* EXECUTE 语句

EXECUTE *stmt_name* [USING *@var_name* [, *@var_name*] ...];

· 一个使用 PREPARE 语句预备好的 SQL 语句，可以使用 EXECUTE 语句执行。

· 如果预备语句中有问号指明的绑定变量，需要使用 USING 子句指明相同个数的执行时绑定的值。USING 子句后只能使用 SET 语句定义的用户变量。

* DEALLOCATE 语句

DEALLOCATE PREPARE *stmt_name*;或者DROP PREPARE *stmt_name*;

删除一个指定的预备语句。一旦删除，以后就不能再执行。

依次使用 PREPARE 查询表 a 中 id=2 的行。

PREPARE stmt1 FROM SELECT name FROM a WHERE id=?;

SET @id = 2;

EXECUTE stmt1 USING @id;

DEALLOCATE PREPARE stmt1;

5.4.18. 预备执行语句

CBASE 实现了服务器端的真正的 Prepared statement。用户先通过客户端发送一个预备语句把要执行的 SQL 数据操作语句发给服务器，服务器端会解析这个语句，产生执行计划并返回给客户端一个句柄（名字或者 ID）。随后，用户可以使用返回的句柄和指定的参数反复执行一个预备好的语句，省去了每次执行都解析 SQL 语句的开销，可以极大地提高性能。

由于目前 CBASE SQL 引擎的优化工作还做的不够，SQL 解析并产生执行计划的过程效率不高。而使用预备执行语句，可以省掉这一过程，直接用已经预备好的执行计划和用户指定的参数执行语句，这样可以极大的提高性能。所以， 我们强烈推荐应用尽可能多地使用预备执行语句。

6. CBASE SQL优化指南

主要介绍 CBASE 的 SQL 优化，提高 CBASE 的执行效率。

6.1. 根据执行计划调优

Explain 语句，可以查看一个 DML 语句的执行计划。执行计划是一个物理运算符组成的树状结构。常见的物理运算符有 TableScan, Filter, Sort, GroupBy, Join,

Project 等。

例如，执行

EXPLAIN SELECT name, value1, value2 from all_sys_config_stat WHERE name = 'location_cache_timeout';语句，可以得到如下一个 PLAN：

Project(columns=[expr=[COL|],expr=[COL|],expr=[COL|]]) TableRpcScan(rpc_scan==[COL|varchar:location_cache_timeout|EQ|]]) Project(columns=[expr=[COL|],expr=[COL|],expr=[COL|]])

这个 plan 由两个物理运算符组成，Project 运算符负责投影和表达式计算，它的输入数据由下层的 TableRpcScan 提供。Columns 参数列出了三个表达式，表示这个投影操作会产生 3 列数据。expr=[COL|]代表一个后缀表达式，这个表达式会产生一个 TABLE_ID 为 NULL，COLUMN_ID 为 65519 的 cell。这个 cell 的值根据后缀表达式[COL|]运算后产生，这里这个表达式是一个列引用，就是取

TABLE_ID 为 12，COLUMN_ID 为 25 的数据值。

第二个物理运算符是 TableRpcScan，它实际上由 RpcScan 操作符实现。这个物理运算符实际上是在 Chunk Server 上执行。它读取 TABLE_ID 为 12 的表中通过 Project(columns=[expr=[COL|],expr=[COL|],expr=[COL|]]中指定

COLUMN_ID 为 25,27,28 的 3 列数据。然后经过一个 Filter 操作符进行过滤， 过滤条件即后缀表达式[COL|varchar:location_cache_timeout|EQ|]，它表示过滤

TABLE_ID 为 12，COLUMN_ID 为 25 的 cell，等于varchar:location_cache_timeout 的行。

6.2. 优化规则

CBASE 为SQL 语句产生执行计划的时候，实现了一些基于规则的优化策略， 这里简单描述一些目前实现的规则。

6.2.1. 主键索引优化

CBASE 表格中数据存储都是按照主键排序的。如果一个 SELECT 查询的

WHERE 条件中限定了主键的所有列，那么查询可以使用主键索引进行优化。假设有一个表 t1：CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int, primary key(c1, c2, c3))

下面举例说明优化规则：

· 使用等值条件限定所有主键的查询，会使用 MultiGet 操作进行单行查询。如“SELECT * FROM t1 WHERE c1 = 1 and c2 = 2 and c3 = 3”。注意， 这里的等值条件必须写成列在等号左边，例如 1 = c1 则不满足本规则。

· 使用 IN 表达式限定所有主键的查询，会使用 MultiGet 操作进行多行查询。如“SELECT * FROM t1 WHERE (c1, c2, c3) IN ((1,2,3), (4,5,6))”。

· 使用简单比较操作限定主键前缀列取值范围的查询，会转换为对某些特定

Tablet 的扫描。如”SELECT * FROM t1 WHERE c1 >= 1 and c1 < 10 and c2 > 100 and c2 <= 200 and c3 >300”，会转换为对主键范围(,)所有

tablet 的扫描。

不满足上面三种情况的，则需要对整个表进行全表扫描。特别的，目前执行计划生产过程没有做表达式变化。所以”SELECT * FROM t1 WHERE (c1 = 1 and c2 = 2 and c3 = 3) OR (c1 = 10 and c2 = 20 and c3 = 30)” 这个语句不会使用 MultiGet 优化，应用应该使用 IN 表达式执行这种多行查询。

6.2.2. 二级索引优化

SELECT时想要二级索引生效，需要遵循以下规则：

① 确保该SQL语句中与test有关的where条件中有一个条件是跟c1有关的，并且该条件是类似于c1=xx或者c1 in (xx,xx)

② 确保该SQL语句中与test有关的where条件中没有一个是与主键id有关的

③ 确保该SQL语句中与test有关的where条件中没有一个是包含了两列或是两列以上的, 如where c2+c3>5。

④ 当SQL语句中涉及到子查询的时候,子查询语句可以正常使用索引，但是外部SQL在回表的情况下不可以使用索引。

⑤ 当查询中涉及到when的时候不使用索引。

如果一条SQL语句满足上述5个条件，但是用户对c1这一列建立了两个索引i1、i2。按照规则的话，默认是使用最先建立的索引：i1的，但是这时你想使用i2，这个时候就可以使用hint使得该SQL语句强制使用i2.

例：select /+*index(t1 i2)*/ c4 from test where c1=10 and c3=12;

6.2.3. 并发执行优化

MergeServer 向 ChunkServer 读取基本表数据的时候，是并发查询多个ChunkServer 的。除此之外，如果满足某些条件，MergeServer 会把聚合操作分发到拥有数据的相关 ChunkServer 上执行，然后把 ChunkServer 汇总后的结果再做汇总。我们把这个优化叫做”聚合操作下压”，它需要查询满足以下一些条件：

- 单表查询。
- 没有 UNION, INTERSECT, EXCEPT 等集合操作。
- 没有 ORDER BY 子句。
- 有 GROUP BY 子句，或者有聚集函数，且任何聚集函数不能有 DISTINCT

修饰符。

- 系统变量开关 ob_group_agg_push_down_param 为 true（默认值）。

此外，limit 操作也可以下压到 ChunkServer 端执行，以减少需要网络传输的数据。适用这个优化的查询需要满足一下条件：

- 单表查询。
- 没有 UNION, INTERSECT, EXCEPT 等集合操作。
- 没有 ORDER BY 子句。
- 没有 GROUP BY 子句以及聚集函数。
- 当然需要有 LIMIT 子句。

6.2.4. 复杂SQL语句优化

在CBASE中的SQL编程应尽量不要编写复杂的SQL语句。将复杂的SQL语句分成多个简单的语句，才能够充分发挥CBASE的查询优势。

6.2.5. JOIN语句**优化**

使用JOIN连接进行表连接时，只能使用ON子句指定条件进行筛选，且目前支持多个等值连接条件，且连接条件应严格按照以下规则书写“FROM 左表以下的选项中的参数中自选 [INNER|LEFT OUTER|RIGHT OUTER| FULL] JOIN 右表 ON左表的字段=右表的字段”。

当使用多个等值连接条件时，JOIN语句的书写格式建议为

SELECT A.COL1, A.COL2, B.COL1, B.COL2

FROM A[INNER|LEFT OUTER|RIGHT OUTER|FULL] JOIN B ON A.COL1 = B.COL1

WHERE A.COL1=B.COL1 AND A.COL2=B.COL2 AND A.COL3=常量1 AND B.COL3=常量2；

根据CBASE目前的JOIN性能，不建议进行大表连接。如果不能避免大表连接，建议在where语句中加入左表和右表各自的主键过滤条件，将连接的行数降低到最小。同时考虑到过滤的效率，应该将过滤字段设置为表格的第一主键如果表连接效率不高，建议将JOIN子句上表达式两边的字段类型各自创建索引。

6.2.6. SELECT语句**优化**

1 在查询语句中如果限定主键前缀的取值范围，这样可以使用主键索引优化提高查询性能。此时建表语句primary key(c1,c2,c3)中主键的出现顺序对查询效率影响至关重要。在下面的优化规则说明中，c1为第一主键，c2为第二主键，c3为第三主键。

- 使用等值条件限定所有主键的查询。如

SELECT * FROM t1 WHERE c1 = 1 and c2 = 2 and c3 = 3

注意，这里的等值条件必须写成列在等号左边，例如1 = c1则不满足优化规则。

- 使用IN表达式限定主键前缀的查询。如

SELECT * FROM t1 WHERE (c1, c2) IN ((1,2), (4,5))

- 使用简单比较操作限定主键前缀列取值范围的查询。如

SELECT * FROM t1 WHERE c1 >= 1 and c1 < 10 and c2 > 100 and c2 <= 200 and c3 >300

SELECT * FROM t1 WHERE c1 like ‘ABB%’

这时的查询优化将根据c1的取值范围进行部分扫描。

- 不满足上面三种情况的，则需要全表扫描。

SELECT * FROM t1 WHERE c2 > 100 and c2 <= 200 and c3 >300

SELECT * FROM t1 WHERE (c1 = 1 and c2 = 2 and c3 = 3) OR (c1 = 10 and c2 = 20 and c3 = 30)

SELECT * FROM t1 WHERE c1 like '%ABB%'

这些SQL语句无法被优化会导致全表扫描。

2 避免使用select(*)，在select列表中仅仅指明需要的列，这样可以减少CPU时间和不必要的系统IO。

3 避免使用distinct关键字。因为distinct会通过排序删除重复记录，这样会大大增加IO操作次数。

4 避免使用count()*操作。开发过程中，有人会使用count()*语句来判断某张表中符合条件的记录有没有，根据这个逻辑，应该使用limit 1子句来替代count(*)。

5 限制返回结果集的大小。使用limit M offset N来限制返回结果集的大小。

6.2.7. 操作符**优化**

1 对索引优化有效的运算符有'='，'>','<'，'>=','<=' ；'<>'也是有效运算符，但是无法被优化会导致全表扫描。应该尽量转换成其他的操作符或其他查询条件。

2 当使用通配符'%'进行字符串匹配时，如果只匹配字符串前缀，建议转换成'>','<'，因为后者的执行效率比前者要高。

SELECT * FROM t1 WHERE c1 like'ABB%'

应修改为

SELECT * FROM t1 WHERE c1 >'ABB'AND c1<'ABC'

7. CBASE错误码

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_INVALID_ARGUMENT	-2	Invalid argument
OB_ARRAY_OUT_OF_RANGE	-3	Array index out of range
OB_SERVER_LISTEN_ERROR	-4	Failed to listen to the port
OB_INIT_TWICE	-5	The object is initialized twice
OB_NOT_INIT	-6	The object is not initialized
OB_NOT_SUPPORTED	-7	Not supported feature or function
OB_ITER_END	-8	End of iteration
OB_IO_ERROR	-9	IO error
OB_ERROR_FUNC_VERSION	-10	Wrong RPC command version
OB_PACKET_NOT_SENT	-11	Can not send packet
OB_RESPONSE_TIME_OUT	-12	Timeout
OB_ALLOCATE_MEMORY_FAILED	-13	No memory
OB_MEM_OVERFLOW	-14	Memory overflow
OB_ERR_SYS	-15	System error
OB_ERR_UNEXPECTED	-16	Ooooooooooooops
OB_ENTRY_EXIST	-17	Entry already exist
OB_ENTRY_NOT_EXIST	-18	Entry not exist
OB_SIZE_OVERFLOW	-19	Size overflow
OB_REF_NUM_NOT_ZERO	-20	Reference count is not zero
OB_CONFLICT_VALUE	-21	Conflict value
OB_ITEM_NOT_SETTED	-22	Item not set
OB_EAGAIN	-23	Try again
OB_BUF_NOT_ENOUGH	-24	Buffer not enough
OB_PARTIAL_FAILED	-25	Partial failed
OB_READ_NOTHING	-26	Nothing to read
OB_FILE_NOT_EXIST	-27	File not exist
OB_DISCONTINUOUS_LOG	-28	Log entry not continuous

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_SCHEMA_ERROR	-29	Schema error
OB_DATA_NOT_SERVE	-30	Required data not served by the ChunkServer
OB_UNKNOWN_OBJ	-31	Unknown object
OB_NO_MONITOR_DATA	-32	No monitor data
OB_SERIALIZE_ERROR	-33	Serialize error
OB_DESERIALIZE_ERROR	-34	Deserialize error
OB_AIO_TIMEOUT	-35	Asynchronous IO error
OB_NEED_RETRY	-36	Need retry
OB_TOO_MANY_SSTABLE	-37	Too many sstable
OB_NOT_MASTER	-38	The UpdateServer or cluster is not the master
OB_TOKEN_EXPIRED	-39	Expired token
OB_ENCRYPT_FAILED	-40	Encrypt error
OB_DECRYPT_FAILED	-41	Decrypt error
OB_USER_NOT_EXIST	-42	User not exist
OB_PASSWORD_WRONG	-43	Incorrect password
OB_SKEY_VERSION_WRONG	-44	Wrong skey version
OB_NOT_A_TOKEN	-45	Not a token
OB_NO_PERMISSION	-46	No permission
OB_COND_CHECK_FAIL	-47	Cond check error
OB_NOT_REGISTERED	-48	Not registered
OB_PROCESS_TIMEOUT	-49	Process timeout
OB_NOT_THE_OBJECT	-50	Not the object
OB_ALREADY_REGISTERED	-51	Already registered
OB_LAST_LOG_RUINNED	-52	Corrupted log entry
OB_NO_CS_SELECTED	-53	No ChunkServer selected
OB_NO_TABLETS_CREATED	-54	No tablets created
OB_INVALID_ERROR	-55	Invalid entry

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_CONN_ERROR	-56	Connection error
OB_DECIMAL_OVERFLOW_WARN	-57	Decimal overflow warning
OB_DECIMAL_UNLEGAL_ERROR	-58	Decimal overflow error
OB_OBJ_DIVIDE_BY_ZERO	-59	Devided by zero
OB_OBJ_DIVIDE_ERROR	-60	Devide error
OB_NOT_A_DECIMAL	-61	Not a decimal
OB_DECIMAL_PRECISION_NOT_EQUAL	-62	Decimal precision error
OB_EMPTY_RANGE	-63	Empty range
OB_SESSION_KILLED	-64	Session killed
OB_LOG_NOT_SYNC	-65	Log not sync
OB_DIR_NOT_EXIST	-66	Directory not exist
OB_NET_SESSION_END	-67	RPC session finished
OB_INVALID_LOG	-68	Invalid log
OB_FOR_PADDING	-69	For padding
OB_INVALID_DATA	-70	Invalid data
OB_ALREADY_DONE	-71	Already done
OB_CANCELED	-72	Operation canceled
OB_LOG_SRC_CHANGED	-73	Log source changed
OB_LOG_NOT_ALIGN	-74	Log not aligned
OB_LOG_MISSING	-75	Log entry missed
OB_NEED_WAIT	-76	Need wait
OB_NOT_IMPLEMENT	-77	Not implemented feature
OB_DIVISION_BY_ZERO	-78	Divided by zero
OB_VALUE_OUT_OF_RANGE	-79	Value out of range
OB_EXCEED_MEM_LIMIT	-80	exceed memory limit
OB_RESULT_UNKNOWN	-81	Unknown result
OB_ERR_DATA_FORMAT	-82	Data format error
OB_MAYBE_LOOP	-83	Mayby loop

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_NO_RESULT	-84	No result
OB_QUEUE_OVERFLOW	-85	Queue overflow
OB_START_LOG_CURSOR_INVALID	-99	Invalid log cursor
OB_LOCK_NOT_MATCH	-100	Lock not match
OB_DEAD_LOCK	-101	Deadlock
OB_PARTIAL_LOG	-102	Incomplete log entry
OB_CHECKSUM_ERROR	-103	Data checksum error
OB_INIT_FAIL	-104	Initialize error
OB_ASYNC_CLIENT_WAITING_RESPONSE	-108	Asynchronous client failed to get response
OB_STATE_NOT_MATCH	-109	State not match
OB_READ_ZERO_LOG	-110	Read zero log
OB_SWITCH_LOG_NOT_MATCH	-111	Switch log not match
OB_LOG_NOT_START	-112	Log not start
OB_IN_FATAL_STATE	-113	In FATAL state
OB_IN_STOP_STATE	-114	In STOP state
OB_UPS_MASTER_EXISTS	-115	Master UpdateServer already exists
OB_LOG_NOT_CLEAR	-116	Log not clear
OB_FILE_ALREADY_EXIST	-117	File already exist
OB_UNKNOWN_PACKET	-118	Unknown packet
OB_TRANS_ROLLBACKED	-119	Transaction rollbacked
OB_LOG_TOO_LARGE	-120	Log too large
OB_RPC_SEND_ERROR	-121	PRC send error
OB_RPC_POST_ERROR	-122	PRC post error
OB_LIBEASY_ERROR	-123	Libeasy error
OB_CONNECT_ERROR	-124	Connect error
OB_NOT_FREE	-125	Not free
OB_INIT_SQL_CONTEXT_ERROR	-126	Init SQL context error

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_SKIP_INVALID_ROW	-127	Skip invalid row
OB_SYS_CONFIG_TABLE_ERROR	-131	SYS_CONFIG table error
OB_READ_CONFIG_ERROR	-132	Read config error
OB_TRANS_NOT_MATCH	-140	Transaction not match
OB_TRANS_IS_READONLY	-141	Transaction is readonly
OB_ROW_MODIFIED	-142	Row modified
OB_VERSION_NOT_MATCH	-143	Version not match
OB_BAD_ADDRESS	-144	Bad address
OB_DUPLICATE_COLUMN	-145	Duplicated column
OB_ENQUEUE_FAILED	-146	Enqueue error
OB_INVALID_CONFIG	-147	Invalid config
OB_WAITING_COMMIT	-148	Waiting commit error
OB_STMT_EXPIRED	-149	Expired statement
OB_SCHEMA_SYNC_ERROR	-151	Sync schema error
OB_INDEX_NOT_EXIST	-152	no index to drop!
OB_VARCHAR_DECIMAL_INVALID	-153	VARCHAR cast to DECIMAL error
OB_TABLE_UPDATE_LOCKED	-155	table is locked
OB_UD_PARALLAL_DATA_NOT_SAFE	-156	batch update/delete data is not safe
OB_CS_CACHE_NOT_HIT	-1001	Cache not hit
OB_CS_TIMEOUT	-1002	ChunkServer timeout
OB_CS_TABLET_NOT_EXIST	-1008	Tablet not exist on Chunkserver
OB_CS_EMPTY_TABLET	-1009	Tablet is empty
OB_CS_EAGAIN	-1010	ChunkServer try again
OB_GET_NEXT_COLUMN	-1011	Get next column
OB_GET_NEXT_ROW	-1012	To get next row
OB_INVALID_ROW_KEY	-1014	Invalid row key
OB_SEARCH_MODE_NOT_IMPLEMENT	-1015	Search mode not implemented
OB_INVALID_BLOCK_INDEX	-1016	Invalid block index

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_INVALID_BLOCK_DATA	-1017	Invalid block data
OB_SEARCH_NOT_FOUND	-1018	Value not found
OB_BEYOND_THE_RANGE	-1020	Key out of range
OB_CS_COMPLETE_TRAVERSAL	-1021	ChunkServer complete tranverse block cache
OB_END_OF_ROW	-1022	End of row
OB_CS_MERGE_ERROR	-1024	ChunkServer merge error
OB_CS_SCHEMA_INCOMPATIBLE	-1025	Incomplete schema
OB_CS_SERVICE_NOT_STARTED	-1026	ChunkServer service unavailable
OB_CS_LEASE_EXPIRED	-1027	Expired lease
OB_CS_MERGE_HAS_TIMEOUT	-1028	Merge timeout on ChunkServer
OB_CS_TABLE_HAS_DELETED	-1029	Table deleted on ChunkServer
OB_CS_MERGE_CANCELED	-1030	Merge canceled ChunkServer
OB_CS_COMPRESS_LIB_ERROR	-1031	ChunkServer failed to get compress library
OB_CS_OUTOF_DISK_SPACE	-1032	ChunkServer out of disk space
OB_CS_MIGRATE_IN_EXIST	-1033	Migrate-in task already exist
OB_AIO_BUF_END_BLOCK	-1034	AIO buffer end block
OB_AIO_EOF	-1035	AIO EOF
OB_AIO_BUSY	-1036	AIO busy
OB_WRONG_SSTABLE_DATA	-1037	Wrong sstable data
OB_COLUMN_GROUP_NOT_FOUND	-1039	Column group not found
OB_NO_IMPORT_SSTABLE	-1040	No import sstable
OB_IMPORT_SSTABLE_NOT_EXIST	-1041	Imported sstable not exist
OB_UPS_TRANS_RUNNING	-2001	UpdateServer transaction in progress
OB_FREEZE_MEMTABLE_TWICE	-2002	Memtable frozen twice
OB_DROP_MEMTABLE_TWICE	-2003	Memtable drop twice
OB_INVALID_START_VERSION	-2004	Invalid memtable start version
OB_UPS_NOT_EXIST	-2005	Not exist for UpdateServer

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_UPS_ACQUIRE_TABLE_FAIL	-2006	Acquire table error on UpdateServer
OB_UPS_INVALID_MAJOR_VERSION	-2007	Invalid major version
OB_UPS_TABLE_NOT_FROZEN	-2008	Table not frozen
OB_UPS_CHANGE_MASTER_TIMEOUT	-2009	UpdateServer change master timeout
OB_FORCE_TIME_OUT	-2010	Force timeout
OB_BEGIN_TRANS_LOCKED	-2011	Begin transaction locked
OB_ERROR_TIME_STAMP	-3001	Wrong timestamp
OB_ERROR_INTRESECT	-3002	Tablet range intersect
OB_ERROR_OUT_OF_RANGE	-3003	Out of range
OB_RS_STATUS_INIT	-3004	RootServer status init
OB_IMPORT_NOT_IN_SERVER	-3005	Import not in server
OB_FIND_OUT_OF_RANGE	-3006	Search out of range
OB_CONVERT_ERROR	-3007	Convert error
OB_MS_ITER_END	-3008	MS end of iteration
OB_MS_NOT_EXIST	-3009	MS not exist
OB_BYPASS_TIMEOUT	-3010	Bypass timeout
OB_BYPASS_DELETE_DONE	-3011	Bypass delete done
OB_RS_STATE_NOT_ALLOW	-3012	RootServer state error
OB_BYPASS_NEED_REPORT	-3014	Bypass need report
OB_ROOT_TABLE_CHANGED	-3015	Table changed
OB_ROOT_REPLICATE_NO_DEST_CS_FOUND	-3016	No destination ChunkServer
OB_ROOT_TABLE_RANGE_NOT_EXIST	-3017	Tablet range not exist
OB_ROOT_MIGRATE_CONCURRENCY_FULL	-3018	Migrate concurrency full
OB_ROOT_MIGRATE_INFO_NOT_FOUND	-3019	Migrate info not found
OB_NOT_DATA_LOAD_TABLE	-3020	No data to load
OB_DATA_LOAD_TABLE_DUPLICATED	-3021	Duplicated table data to load
OB_ROOT_TABLE_ID_EXIST	-3022	Table ID exist
OB_CLUSTER_ALREADY_MASTER	-3023	Target cluster is already master

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_IP_PORT_IS_NOT_SLAVE_CLUSTER	-3024	Target cluster is not slave
OB_CLUSTER_IS_NOT_SLAVE	-3025	Cluster is not slave
OB_CLUSTER_IS_NOT_MASTER	-3026	Cluster is not master
OB_CONFIG_NOT_SYNC	-3027	Config not sync
OB_IP_PORT_IS_NOT_CLUSTER	-3028	Target cluster not exist
OB_MASTER_CLUSTER_NOT_EXIST	-3029	Master cluster not exist
OB_GET_CLUSTER_MASTER_UPS_FAILED	-3031	Fetch master cluster ups list fail
OB_MULTIPLE_MASTER_CLUSTERS_EXIST	-3032	Multiple master clusters exist
OB_MASTER_CLUSTER_ALREADY_EXISTS	-3034	Master cluster already exist
OB_BROADCAST_MASTER_CLUSTER_ADDR_FAIL	-3036	Broadcast master cluster address fail
OB_DATA_SOURCE_NOT_EXIST	-3100	Data source not exist
OB_DATA_SOURCE_TABLE_NOT_EXIST	-3101	Data source table not exist
OB_DATA_SOURCE_RANGE_NOT_EXIST	-3102	Data source range not exist
OB_DATA_SOURCE_DATA_NOT_EXIST	-3103	Data source data not exist
OB_DATA_SOURCE_SYS_ERROR	-3104	Data source sys error
OB_DATA_SOURCE_TIMEOUT	-3105	Data source timeout
OB_DATA_SOURCE_CONCURRENCY_FULL	-3106	Data source concurrency full
OB_DATA_SOURCE_WRONG_URI_FORMAT	-3107	Data source wrong URI format
OB_SSTABLE_VERSION_UNEQUAL	-3108	SSTable version not equal
OB_INNER_STAT_ERROR	-4001	Inner state error
OB_OLD_SCHEMA_VERSION	-4002	Schema version too old
OB_INPUT_PARAM_ERROR	-4003	Input parameter error
OB_NO_EMPTY_ENTRY	-4004	No empty entry
OB_RELEASE_SCHEMA_ERROR	-4005	Release schema error
OB_ITEM_COUNT_ERROR	-4006	Item count error
OB_OP_NOT_ALLOW	-4007	Operation not allowed now
OB_CHUNK_SERVER_ERROR	-4008	ChunkServer error
OB_NO_NEW_SCHEMA	-4009	No new schema

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_MS_SUB_REQ_TOO_MANY	-4010	Too many scan request
OB_TOO_MANY_BLOOM_FILTER_TASK	-4011	too many bloomfilter task
OB_ERR_PARSER_INIT	-5000	Failed to init SQL parser
OB_ERR_PARSE_SQL	-5001	Parse error
OB_ERR_RESOLVE_SQL	-5002	Resolve error
OB_ERR_GEN_PLAN	-5003	Generate plan error
OB_ERR_UNKNOWN_SYS_FUNC	-5004	Unknown system function
OB_ERR_PARSER_MALLOC_FAILED	-5005	Parser malloc error
OB_ERR_PARSER_SYNTAX	-5006	Syntax error
OB_ERR_COLUMN_SIZE	-5007	Wrong number of columns
OB_ERR_COLUMN_DUPLICATE	-5008	Duplicated column
OB_ERR_COLUMN_UNKNOWN	-5009	Unknown column
OB_ERR_OPERATOR_UNKNOWN	-5010	Unknown operator
OB_ERR_STAR_DUPLICATE	-5011	Duplicated star
OB_ERR_ILLEGAL_ID	-5012	Illegal ID
OB_ERR_WRONG_POS	-5013	Invalid position
OB_ERR_ILLEGAL_VALUE	-5014	Illegal value
OB_ERR_COLUMN_AMBIGOUS	-5015	Ambigous column
OB_ERR_LOGICAL_PLAN_FAILD	-5016	Generate logical plan error
OB_ERR_SCHEMA_UNSET	-5017	Schema not set
OB_ERR_ILLEGAL_NAME	-5018	Illegal name
OB_ERR_TABLE_UNKNOWN	-5019	Unknown table
OB_ERR_TABLE_DUPLICATE	-5020	Duplicated table
OB_ERR_NAME_TRUNCATE	-5021	Name truncated
OB_ERR_EXPR_UNKNOWN	-5022	Unknown expression
OB_ERR_ILLEGAL_TYPE	-5023	Illegal type
OB_ERR_PRIMARY_KEY_DUPLICATE	-5024	Duplicated primary key
OB_ERR_ALREADY_EXISTS	-5025	Already exist

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_ERR_CREATETIME_DUPLICATE	-5026	Duplicated createtime
OB_ERR_MODIFYTIME_DUPLICATE	-5027	Duplicated modifytime
OB_ERR_ILLEGAL_INDEX	-5028	Illegal index
OB_ERR_INVALID_SCHEMA	-5029	Invalid schema
OB_ERR_INSERT_NULL_ROWKEY	-5030	Insert null rowkey
OB_ERR_COLUMN_NOT_FOUND	-5031	Column not found
OB_ERR_DELETE_NULL_ROWKEY	-5032	Delete null rowkey
OB_ERR_INSERT_INNER_JOIN_COLUMN	-5033	Insert inner join column error
OB_ERR_USER_EMPTY	-5034	No user
OB_ERR_USER_NOT_EXIST	-5035	User not exist
OB_ERR_NO_PRIVILEGE	-5036	No privilege
OB_ERR_NO_AVAILABLE_PRIVILEGE_ENTRY	-5037	No privilege entry
OB_ERR_WRONG_PASSWORD	-5038	Incorrect password
OB_ERR_USER_IS_LOCKED	-5039	User locked
OB_ERR_UPDATE_ROWKEY_COLUMN	-5040	Can not update rowkey column
OB_ERR_UPDATE_JOIN_COLUMN	-5041	Can not update join column
OB_ERR_INVALID_COLUMN_NUM	-5042	Invalid column number
OB_ERR_PREPARE_STMT_UNKNOWN	-5043	Unknown prepared statement
OB_ERR_VARIABLE_UNKNOWN	-5044	Unknown variable
OB_ERR_SESSION_INIT	-5045	Init session error
OB_ERR_OLDER_PRIVILEGE_VERSION	-5046	Older privilege version
OB_ERR_LACK_OF_ROWKEY_COL	-5047	No rowkey column specified
OB_ERR_EXCLUSIVE_LOCK_CONFLICT	-5048	Exclusive lock conflict
OB_ERR_SHARED_LOCK_CONFLICT	-5049	Shared lock conflict
OB_ERR_USER_EXIST	-5050	User exists
OB_ERR_PASSWORD_EMPTY	-5051	Empty password
OB_ERR_GRANT_PRIVILEGES_TO_CREATE_TABLE	-5052	Failed to grant privelege
OB_ERR_WRONG_DYNAMIC_PARAM	-5053	Wrong dynamic parameters

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_ERR_PARAM_SIZE	-5054	Wrong number of parameters
OB_ERR_FUNCTION_UNKNOWN	-5055	Unknown function
OB_ERR_CREAT_MODIFY_TIME_COLUMN	-5056	CreateTime or ModifyTime column cannot be modified
OB_ERR_MODIFY_PRIMARY_KEY	-5057	Primary key cannot be modified
OB_ERR_PARAM_DUPLICATE	-5058	Duplicated parameters
OB_ERR_TOO_MANY_SESSIONS	-5059	Too many sessions
OB_ERR_TRANS_ALREADY_STARTED	-5060	Transaction already started
OB_ERR_TOO_MANY_PS	-5061	Too many prepared statements
OB_ERR_NOT_IN_TRANS	-5062	Not in transaction
OB_ERR_HINT_UNKNOWN	-5063	Unknown hint
OB_ERR_WHEN_UNSATISFIED	-5064	When condition not satisfied
OB_ERR_QUERY_INTERRUPTED	-5065	Query interrupted
OB_ERR_SESSION_INTERRUPTED	-5066	Session interrupted
OB_ERR_UNKNOWN_SESSION_ID	-5067	Unknown session ID
OB_ERR_PROTOCOL_NOT_RECOGNIZE	-5068	Incorrect protocol
OB_ERR_WRITE_AUTH_ERROR	-5069	Write auth packet error
OB_ERR_COLUMN_TYPE_NOT_COMPATIBLE	-5071	The data types of some columns are not compatible
OB_ERR_PS_TOO_MANY_PARAM	-5080	Prepared statement contains too many placeholders
OB_ERR_READ_ONLY	-5081	The server is read only now
OB_ERR_PRIMARY_KEY_CONFLICT	-5082	Update primary key conflict; please try again
OB_ERR_TABLE_NAME_LENGTH	-5090	New table name is too long
OB_ERR_SESSION_UNSET	-5100	Session not set
OB_VALUE_OUT_OF_DATES_RANGE	-5101	result value is not within the valid range of dates.
OB_ERR_TIMESTAMP_TYPE_FORMAT	-5102	Format of a TIMESTAMP value is incorrect

OB_OBJ_TYPE_ERROR	**-1**	**Object type error**
OB_ERR_DATE_TYPE_FORMAT	-5103	Format of a DATE value is incorrect
OB_ERR_TIME_TYPE_FORMAT	-5104	Format of a TIME value is incorrect
OB_ERR_VALUE_OF_DATE	-5105	Value of DATE is out of the range
OB_ERR_VALUE_OF_TIME	-5106	Value of TIME is out of the range
OB_ERR_CAN_NOT_USE_SEMI_JOIN	-5201	can not use semi join;change your join type;retry again!
OB_ERR_HAS_NO_EQUI_COND	-5202	can not use semi join;check your on expression!
OB_ERR_INDEX_NUM_IS_ZERO	-5203	filter is null; empty set!
OB_ERR_POINTER_IS_NULL	-5204	semi join:null pointer!
OB_ERR_FUNC_DEV	-5205	semi join:function is on the way!
OB_ERR_VARIABLE_NULL	-5503	The bind parameter variable can not be null
OB_ERR_SUB_QUERY_OVERFLOW	-5600	too many sub_query;not support well now
OB_ERR_SUB_QUERY_RESULTSET_EXCESSIVE	-5601	Subquery returns more than 1 row
OB_ERR_DATABASE_NOT_EXSIT	-5602	Database not exist
OB_ERR_NO_ACCESS_TO_DATABASE	-5603	Have no access to database
OB_ERR_STILL_HAS_TABLE_IN_DATABASE	-5604	Still has table in database...
OB_ERR_DROP_DEFAULT_DB	-5605	Default db can not be dropped!!!
OB_ERR_STMT_COL_NOT_FOUND	-5610	Cannot find column in the statement structure
OB_ERR_VARCHAR_TOO_LONG	-5611	Varchar is too long
OB_ERR_NULL_POINTER	-5612	Use nullptr
OB_ERR_SERVER_IN_INIT	-8001	Server is initializing