云数据库 OceanBase

SQL语法参考

阿里云
aliyun.com

SQL语法参考

OceanBase SQL 快速概览

OceanBase支持SQL 92与兼容MySQL， 原则上只要是MySQL的语法，OceanBase都能运行，本节列出一些尚不支持的功能以及需要注意的地方。

尚不支持的功能

- 不支持可更新视图、存储过程、触发器、游标；
- 不支持用户自定义数据类型、自定义函数；
- 不支持某些数据类型，比如不支持blob、text、enum、set。
- 不支持临时表。
- 分区表不支持range分区，不支持子分区，不支持分区管理。
- 不支持load data。
- 不支持select ... into。
- 不支类似IF，WHILE等流程控制类语句；不支持类似BEGIN...END，LOOP...END LOOP，REPEAT...UNTIL...END REPEAT， WHILE...DO...END WHILE等复合语句。
- 不支持INSERT/REPLACE语句中的带SELECT子句的修改，不支持DELETE语句多表删除，不支持UPDATE多表更新。
- SELECT...FOR UPDATE只支持单表查询。
- 某些函数不支持，具体支持函数请参考函数章节。
- 不支持alter add column before|after。
- 不支持某些列类型的变更，比如从varchar变更为int。

需要注意的点

- 如果你的表使用了分区，则有可能被分布到多台单独的物理机上(目前公测规格不会)。
- insert/update/delete单条语句不能跨分区，如果有跨分区的需求，需要写成事务。
- 一般来说，select需要带上分区键。
- 不支持prepare， OceanBase不需要你使用prepare。
- 字符集目前只支持utf8和utf8mb4。

OceanBase SQL简介

语言结构

OceanBase SQL语句中涉及以下元素：

- 字符串和数字等文字值；
- 识别符，例如表名和列名；
- 用户和系统变量；
- 注释和保留字等。

文字值

文字值包括字符串、数值、日期、时间、十六进制、布尔值和NULL。

字符串使用单引号（'）或双引号（"）引起来的字符序列。如果OceanBase SQL模式启用了ANSI_QUOTES，表示只用单引号引用字符串，用双引号引用的字符串被解释为一个识别符。

在字符串中，某些序列具有特殊含义。这些序列均用反斜线（'\'）开始，即所谓的转义字符。转义字符对大小写敏感。

OceanBase识别的转义字符如下表所示。

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)
\Z	ASCII 26(控制(Ctrl)-Z)

数值可以分为精确数值（整数和定点数值）和浮点数值。数值可以使用.作为十进制间隔符。数值也可以在前面加一个-来表示负值。

日期值可以有多种形式，例如单引号字符串或数值。例如：'2015-07-21'，'20150721'和20150721。

十六进制值。在数字上下文中，十六进制数如同整数（64位精度）。在字符串上下文，如果有二进制字符串，每对二进制数字被转换为一个字符。

布尔值，常量TRUE等于1，常量FALSE等于0。常量名可以写成大写或小写。

NULL值表示"没有数据"。NULL可以写成大写或小写。请注意NULL值不同于数值类型的0或字符串类型的空字符串。

识别符

OceanBase1.0 SQL语句中的租户、数据库、表、视图、索引、表格组、列名和别名等Schema对象名称之为识别符。

OceanBase识别符的最大长度和允许的字符，如下表所示。

识别符	最大长度(字节)	允许的字符
用户名	16	大小写英文字母，数字和下划线，而且必须以字母或下划线开头，并且不能OceanBase的关键字。
租户	64	大小写英文字母，数字和下划线，而且必须以字母或下划线开头，并且不能OceanBase的关键字。
数据库	64	大小写英文字母，数字和下划线，'$'组成。
表组	64	大小写英文字母，数字和下划线，而且必须以字母或下划线开头，并且不能OceanBase的关键字。
表	64	大小写英文字母，数字和下划线，'$'组成。
列	64	大小写英文字母，数字和下划线，'$'组成。
索引	64	大小写英文字母，数字和下划线，'$'组成。
别名	255	大小写英文字母，数字和下划线，'$'组成。

变量	64	文字数字字符、'.'、'_'和'$'组成。

注：除了表内注明的限制，识别符不可以包含ASCII 0或值为255的字节。数据库、表和列名不应以空格结尾。在识别符中尽管可以使用引号识别符，应尽可能避免这样使用。

识别符可以用引用符引起来也可以不引起来。如果识别符是一个保留字或包含特殊字符，无论何时使用，必须将它引起来。OceanBase识别符的引用符是反引号（`）。如果服务器模式包括ANSI_QUOTES模式选项，还可以用双引号将识别符引起来。

例如：

```
Oceanbase>create table `add`(id int, `select` int);
Query OK, 0 rows affected (0.11 sec)

Oceanbase> create table "delete"(id int, "select" int);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '"delete"(id int, "select" int)' at line 1

Oceanbase> set sql_mode='ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

Oceanbase> create table "delete"(id int, "select" int);
Query OK, 0 rows affected (0.12 sec)
```

识别符引用

OceanBase允许使用由单个识别符或多个识别符组成的名字。多个识别符之间以句点(.)间隔开。在OceanBase中可以引用下面形式的列：

列参考	含义
col_name	列**col_name**，查询中使用的表包含有此名字的列。
tbl_name.col_name	默认数据库中的表**tbl_name**的列**col_name**。
db_name.tbl_name.col_name	数据库**db_name**中的表**tbl_name**的列**col_name**。

如果多部分名的组件需要引用，应分别将它们引起来而不要将整个名引起来。例如，**'my-tables'.'my-column'**有效，而**'my-tables.my-column'**无效。

不需要在语句中为列指定**tbl_name**或**db_name.tbl_name**前缀，除非列不确定。如表t1和t2各包含一个列c，你使用SELECT语句在t1和t2中搜索c。在这种情况下，c不确定，因为它在语句中使用的表内不唯一。你必须用表名t1.c或t2.c限定它，表示指哪个表。同样，要想用同一语句搜索数据库db1中的表t和数据库db2中的表t，你必须将那些表中的列指为db1.t.col_name和db2.t.col_name。

限定名中句点后面的字必须为一个识别符，因此不需要将它引起来，即使是一个保留字。

语法.tbl_name表示当前数据库中的tbl_name。该语法与ODBC兼容，因为某些ODBC程序在表名前面加前缀.字符。

识别符大小写敏感性

在OceanBase中识别符大小写不敏感，但为了保持与MySQL兼容，同样设计了与MySQL一样的lower_case_tables_name系统参数，以决定表名和库名这两种识别符大小写是否敏感。

值	含义
0	表名和库名比较对大小写敏感
1	表名和库名比较对大小写不敏感

用户变量

可以先在用户变量中保存值然后在以后引用，这样可以将值从一个语句传递到另一个语句。用户变量与连接有关，一个客户端定义的用户变量不能被其他客户端看到或使用。当客户端退出时，该客户端连接的所有变量将自动释放。

用户变量的形式如@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。

系统变量

OceanBase可以访问系统和连接变量。当服务器运行时许多变量可以动态更改。

注释

OceanBase支持3种注释语法：

从'#'字符到行尾。

从 ' __'序列到行尾。请注意' __'(双破折号)注释要求第2个破折号后面至少跟一个空格符。

从/* 序列到后面的 */序列。结束序列不一定在同一行中，因此该语法允许注释跨越多行。

保留字

在OceanBase中，保留字如下表所示。

保留字				
NAME_OB	STRING_VALUE	INTNUM	DATE_VALUE	TIMESTAMP_VALUE

HINT_VALUE	BOOL_VALUE	APPROXNUM	DECIMAL_VAL	NULLX
QUESTIONMARK	SYSTEM_VARIABLE	TEMP_VARIABLE	CLIENT_VERSION	MYSQL_DRIVER
HEX_STRING_VALUE	EXCEPT	UNION	INTERSECT	INNER
RIGHT	FULL	LEFT	CROSS	JOIN
SET_VAR	OR_OP	OR	XOR	AND_OP
AND	ELSE	THEN	WHEN	CASE
BETWEEN	LOWER_THAN_COMP	REGEXP	IN	LIKE
IS	COMP_NE	COMP_LT	COMP_LE	COMP_GT
COMP_GE	COMP_NSEQ	COMP_EQ	ESCAPE	SHIFT_RIGHT
SHIFT_LEFT	POW	DIV	MOD	LOWER_THAN_NEG
CNNOP	NEG	NOT2	NOT	COLLATE
BINARY	INTERVAL	LOWER_KEY	KEY	ERROR
READ_STATIC	INDEX_HINT	USE_NL	FROZEN_VERSION	TOPK
QUERY_TIMEOUT	READ_CONSISTENCY	HOTSPOT	LOG_LEVEL	LEADING_HINT
ORDERED	FULL_HINT	USE_MERGE	USE_HASH	USE_PLAN_CACHE
STRONG	FROZEN	STATIC	WEAK	_BINARY
_UTF8	_UTF8MB4	HINT_BEGIN	HINT_END	END_P
ACCESSIBLE	ADD	ALTER	ANALYZE	ALL
AS	ASENSITIVE	ASC	BEFORE	BIGINT
BLOB	BOTH	BY	CALL	CASCADE
CHANGE	CHAR	CHARACTER	CHECK	CONDITION
CONSTRAINT	CONTINUE	CONVERT	COLUMN	CREATE
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DAY_HOUR	DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND	DATABASE
DATABASES	DEC	DECIMAL	DECLARE	DEFAULT
DELAYED	DELETE	DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DOUBLE	DROP	DUAL
EACH	ENCLOSED	ELSEIF	ESCAPED	EXISTS

EXIT	EXPLAIN	FETCH	FOREIGN	FLOAT
FLOAT4	FLOAT8	FOR	FORCE	FROM
FULLTEXT	GET	GRANT	GROUP	HAVING
HIGH_PRIORITY	HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND	IF
IFIGNORE	INDEX	INFILE	INOUT	INSENSITIVE
INT	INT1	INT2	INT3	INT4
INT8	INTEGER	INSERT	INTO	IO_AFTER_GTIDS
IO_BEFORE_GTIDS	ISNULL	ITERATE	KEYS	KILL
LEADING	LEAVE	LIMIT	LINEAR	LINES
LOAD	LOCALTIME	LOCALTIMESTAMP	LOCK_	LONG
LONGBLOB	LONGTEXT	LOOP	LOW_PRIORITY	MASTER_BIND
MASTER_SSL_VERIFY_SERVER_CERT	MATCH	MAXVALUE	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND	MINUTE_SECOND	MODIFIES
NATURAL	NO_WRITE_TO_BINLOG	NUMERIC	ON	OPTION
OPTIMIZE	OPTIONALLY	ORDER	OUT	OUTER
OUTFILE	PROCEDURE	PURGE	PARTITION	PRECISION
PRIMARY	RANGE	READ	READ_WRITE	READS
REAL	RELEASE	REFERENCES	RENAME	REPLACE
REPEAT	REQUIRE	RESIGNAL	RESTRICT	RETURN
REVOKE	RLIKE	SECOND_MICROSECOND	SELECT	SCHEMA
SCHEMAS	SEPARATOR	SET	SENSITIVE	SHOW
SIGNAL	SMALLINT	SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SSL_	STARTING	STRAIGHT_JOIN	TERMINATED
TINYBLOB	TINYINTTINYTEXT	TABLE	TABLEGROUP	TO
TRAILING	TRIGGER	UNDO	UNIQUE	UNLOCK

UNSIGNED	UPDATE	USAGE	USE	USING
UTC_DATE	UTC_TIME	UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING	WHERE	WHILE
WITH	WRITE	YEAR_MONTH	ZEROFILL	GLOBAL_ALIAS
SESSION_ALIAS	ACCOUNT	ACTION	ACTIVE	ADDDATE
AFTER	AGAINST	AGGREGATE	ALGORITHM	ANALYSE
ANY	ASCII	AT	AUTHORS	AUTOEXTEND_SIZE
AUTO_INCREMENT	AVG	AVG_ROW_LENGTH	BACKUP	BASIC
BEGI	BINLOG	BIT	BLOCK	BLOCK_SIZE
BOOL	BOOLEAN	BOOTSTRAP	BTREE	BYTE
CACHE	CANCEL	CASCADED	CAST	CATALOG_NAME
CHAIN	CHANGED	CHARSET	CHECKSUM	CIPHER
CLASS_ORIGIN	CLEAN	CLEAR	CLIENT	CLOSE
COALESCE	CODE	COLLATION	COLUMN_FORMAT	COLUMN_NAME
COLUMNS	COMMENT	COMMIT	COMMITTED	COMPACT
COMPLETION	COMPRESSED	COMPRESSION	CONCURRENT	CONNECTION
CONSISTENT	CONSISTENT_MODE	CONSTRAINT_CATALOG	CONSTRAINT_NAME	CONSTRAINT_SCHEMA
CONTAINS	CONTEXT	CONTRIBUTORS	COPY	COUNT
CPU	CREATE_TIMESTAMP	CUBE	CURDATE	CURRENT
CURTIME	CURSOR_NAME	DATA	DATAFILE	DATE
DATE_ADD	DATE_SUB	DATETIME	DAY	DEALLOCATE
DEFAULT_AUTH	DEFINER	DELAY	DELAY_KEY_WRITE	DES_KEY_FILE
DESTINATION	DIAGNOSTICS	DIRECTORY	DISABLE	DISCARD
DISK	DO	DUMP	DUMPFILE	DUPLICATE
DYNAMIC	EFFECTIVE	ENABLE	END	ENDS
ENGINE_	ENGINES	ENUM	ERROR_P	ERRORS
EVENT	EVENTS	EVERY	EXCHANGE	EXECUTE

EXPANSION	EXPIRE	EXPIRE_INFO	EXPORT	EXTENDED
EXTENT_SIZE	EXTRACT	FAST	FAULTS	FIELDS
FILEX	FINAL_COUNT	FIRST	FIXED	FLUSH
FOLLOWER	FORMAT	FOUND	FREEZE	FUNCTION
GENERAL	GEOMETRY	GEOMETRYCOLLECTION	GET_FORMAT	GLOBAL
GRANTS	GROUP_CONCAT	HANDLER	HASH	HELP
HOST	HOSTS	HOUR	IDENTIFIED	IGNORE
IGNORE_SERVER_IDS	IMPORT	INDEXES	INITIAL_SIZE	INSERT_METHOD
INSTALL	INVOKER	IO	IO_THREAD	IPC
ISOLATION	ISSUER	JSON	KEY_BLOCK_SIZE	KEY_VERSION
LANGUAGE	LAST	LEADER	LEAVES	LESS
LEVEL	LINESTRING	LIST_	LOCAL	LOCKED
LOCKS	LOGFILE	LOGS	MAJOR	MASTER
MASTER_AUTO_POSITION	MASTER_CONNECT_RETRY	MASTER_DELAY	MASTER_HEARTBEAT_PERIOD	MASTER_HOST
MASTER_LOG_FILE	MASTER_LOG_POS	MASTER_PASSWORD	MASTER_PORT	MASTER_RETRY_COUNT
MASTER_SERVER_ID	MASTER_SSL	MASTER_SSL_CA	MASTER_SSL_CAPATH	MASTER_SSL_CERT
MASTER_SSL_CIPHER	MASTER_SSL_CRL	MASTER_SSL_CRLPATH	MASTER_SSL_KEY	MASTER_USER
MAX	MAX_CONNECTIONS_PER_HOUR	MAX_CPU	MAX_DISK_SIZE	MAX_IOPS
MAX_MEMORY	MAX_QUERIES_PER_HOUR	MAX_ROWS	MAX_SESSION_NUM	MAX_SIZE
MAX_UPDATES_PER_HOUR	MAX_USER_CONNECTIONS	MEDIUM	MEMORY	MEMTABLE
MERGE	MESSAGE_TEXT	META	MICROSECOND	MIGRATE
MIN	MIN_CPU	MIN_IOPS	MIN_MEMORY	MINOR
MIN_ROWS	MINUTE	MODE	MODIFY	MONTH
MOVE	MULTILINESTRING	MULTIPOINT	MULTIPOLYGON	MUTEX
MYSQL_ERRNO	NAME	NAMES	NATIONAL	NCHAR
NDB	NDBCLUSTER	NEW	NEXT	NO

NODEGROUP	NONE	NORMAL	NOW	NOWAIT
NO_WAIT	NUMBER	NVARCHAR	OFF	OFFSET
OLD_PASSWORD	ONE	ONE_SHOT	ONLY	OPEN
OPTIONS	OWNER	PACK_KEYS	PAGE	PARAMETERS
PARSER	PARTIAL	PARTITION_ID	PARTITIONING	PARTITIONS
PASSWORD	PAUSE	PHASE	PLUGIN	PLUGIN_DIR
PLUGINS	POINT	POLYGON	POOL	PORT
PREPARE	PRESERVE	PREV	PRIMARY_ZONE	PRIVILEGES
PROCESSLIST	PROFILE	PROFILES	PROXY	QUARTER
QUERY	QUICK	READ_ONLY	REBUILD	RECOVER
RECYCLE	REDO_BUFFER_SIZE	REDOFILE	REDUNDANT	REFRESH
RELAY	RELAYLOG	RELAY_LOG_FILE	RELAY_LOG_POS	RELAY_THREAD
RELOAD	REMOVE	REORGANIZE	REPAIR	REPEATABLE
REPLICA	REPLICA_NUM	REPLICATION	REPORT	RESET
RESOURCE	RESOURCE_POOL_LIST	RESTART	RESTORE	RESUME
RETURNED_SQLSTATE	RETURNS	REVERSE	ROLLBACK	ROLLUP
ROOT	ROOTTABLE	ROUTINE	ROW	ROW_COUNT
ROW_FORMAT	ROWS	RTREE	SAVEPOINT	SCHEDULE
SCHEMA_NAME	SCOPE	SECOND	SECURITY	SERIAL
SERIALIZABLE	SERVER	SERVER_IP	SERVER_PORT	SERVER_TYPE
SESSION	SESSION_USER	SET_MASTER_CLUSTER	SET_SLAVE_CLUSTER	SHARE
SHUTDOWN	SIGNED	SIMPLE	SLAVE	SLOW
SNAPSHOT	SOCKET	SOME	SONAME	SOUNDS
SOURCE	SPFILE	SQL_AFTER_GTIDS	SQL_AFTER_MTS_GAPS	SQL_BEFORE_GTIDS
SQL_BUFFER_RESULT	SQL_CACHE	SQL_NO_CACHE	SQL_THREAD	SQL_TSI_DAY
SQL_TSI_HOUR	SQL_TSI_MINUTE	SQL_TSI_MONTH	SQL_TSI_QUARTER	SQL_TSI_SECOND
SQL_TSI_WEEK	SQL_TSI_YEAR	START	STARTS	STATS_AUTO_R

				ECALC
STATS_PERSISTENT	STATS_SAMPLE_PAGES	STATUS	STEP_MERGE_NUM	STOP
STORAGE	STORING	STRING	SUBCLASS_ORIGIN	SUBDATE
SUBJECT	SUBPARTITION	SUBPARTITIONS	SUBSTR	SUBSTRING
SUM	SUPER	SUSPEND	SWAPS	SWITCH
SWITCHES	SYSTEM	SYSTEM_USER	TABLE_CHECKSUM	TABLE_ID
TABLE_NAME	TABLEGROUPS	TABLES	TABLESPACE	TABLET
TABLET_MAX_SIZE	TEMPORARY	TEMPTABLE	TENANT	TEXT
THAN	TIME	TIMESTAMP	TIMESTAMPADD	TIMESTAMPDIFF
TINYINT	TRADITIONAL	TRANSACTION	TRIGGERS	TRIM
TRUNCATE	TYPE	TYPES	UNCOMMITTED	UNDEFINED
UNDO_BUFFER_SIZE	UNDOFILE	UNICODE	UNINSTALL	UNIT
UNIT_NUM	UNLOCKED	UNTIL	UNUSUAL	UPGRADE
USE_BLOOM_FILTER	UNKNOWN	USE_FRM	USER	USER_RESOURCES
VALUE	VARIABLES	VERBOSE	VIEW	WAIT
WARNINGS	WEEK	WEIGHT_STRING	WORK	WRAPPER
X509_	XA	XML	YEAR	ZONE
ZONE_LIST	LOCATION	PLAN	HIGHER_THAN_NEG	

支持语句

OceanBase1.0支持的语句

数据定义语句

CREATE DATABASE， ALTER DATABASE， DROP DATABASE， CREATE TABLE，ALTER TABLE，DROP TABLE，CREATE INDEX，DROP INDEX，CREATE VIEW，DROP VIEW，ALTER

VIEW，TRUNCATE TABLE；

数据操作语句

INSERT，REPLACE，DELETE，UPDATE, SELECT查询及查询子句；

事务类语句

START TRANSACTION，COMMIT，ROLLBACK；

数据库管理语句

CREATE TABLEGROUP，DROP TABLEGROUP， CREATE USER，DROP USER，SET，SET GLOBAL，SET PASSWORD，RENAME USER，ALTER USER，GRANT，REVOKE， ALTER SYSTEM；

实用的SQL语句

SHOW，KILL，USE，DESCRIBE，EXPLAIN，WHEN，HINT，HELP等。

SQL限制与约束:

OceanBase的SQL限制与约束：

暂不支持用户自定义数据类型、自定义函数；

暂不支持可更新视图、存储过程、触发器、游标；

暂不支持临时表；

暂不支持类似BEGIN...END，LOOP...END LOOP，REPEAT...UNTIL...END REPEAT，WHILE...DO...END WHILE等的复合语句；

暂不支类似IF，WHILE等流程控制类语句；

暂不支持INSERT/REPLACE语句中的带SELECT子句的修改，暂不支持DELETE语句多表删除，暂不支持UPDATE多表更新等；

最大长度或个数限制如下：

a) 建表或索引的时候，主键长度之和小于等于16K,单行长度小于等于1.5M;

b) 单个VARCHAR列长度小于等于262，143字节（最大长度是256K）；

c) 主键个数小于等于64个

d) 单表的最大列数为512

e) 单表的最大索引数为128

SELECT...FOR UPDATE只支持单表查询；

- OceanBase 1.0一期TRUNCATE TABLE语句暂时支持单个Partition的表，暂不支持多Parition的表；

分区

OceanBase分区

OceanBase是分布式数据库，如果想把一个表的数据分散到多台机器上，要求创建分区表，OceanBase自动按照分区把数据分散到各OceanBase Server上。也允许用户创建非分区表，非分区表只会落在一台 OceanBase Server上。

OceanBase 分区表的特点：

- OceanBase 1.0实现的是hash partition和key partition。
- 根据 Create Table 时指定的分区字段进行分区。
- 一个分区表的分区数量在Create Table 时指定。
- 允许创建非分区表，比如业务上只有几条记录的元数据表。
- OceanBase 1.0 一期暂不支持非分区表和分区表之间的转换；暂不支持分区管理。
- 因OceanBase 1.0是按partition进行数据打散的，从性能上考虑，INSERT、REPLACE 、SELECT 、UPDATE和DELETE都的WHERE条件要求带上"PARTITION(partition_list)"；不带 parition 字段将报错。
- OceanBase 1.0 一期分区数量最大值为8192。

分区介绍

在OceanBase中，每个表格可以按照规则划分为多个分区，这些分区可以分布到不同的OceanBase Server。

用户所选择的、实现数据分割的规则被称之为分区函数。OceanBase暂不支持用户自定义函数，分区函数只能

是系统函数。

它可以是模数，或者是简单的匹配一个连续的数值区间或数值列表，或是一个内部hash函数，或一个线性hash函数。

函数根据用户指定的分区类型来选择，把用户提供的表达式的值作为参数。该表达式可以是一个整数列值，或一个作用在一个或多个列值上并返回一个整数的函数。 这个表达式的值传递给分区函数，分区函数返回一个表示那个特定记录应该保存在哪 个分区的序号。

这个函数不能是常数，也不能是任意数。

它可以使用任何可用的 SQL 表达式，只要该表达式返回一个小于 MAXVALUE（最大可能的正整数）的正数值。

分区适用于一个表的所有数据和索引；不能只对数据分区而不对索引分区，反之亦然，同时也不能只对表的一部分进行分区。

使用分区的优点

对于那些已经失去保存意义的数据，通常可以通过删除与那些数据有关的分区，很容易地删除那些数据。相反地，在某些情况下，添加新数据的过程又可以通过为那些新数据专门增加一个新的分区，来很方便地实现。

一些查询可以得到极大的优化，这主要是借助于满足一个给定WHERE语句的数据可以只保存在一个或多个分区内，这样在查找时就不用查找其他剩余的分区。因为分区可以在创建了分区表后进行修改，所以在第一次配置分区方案还不曾这么 做时，可以重新组织数据，来提高那些常用查询的效率。

涉及到例如SUM()和COUNT()这样聚合函数的查询，可以很容易地进行并行处理。这种查询的一个简单例子如 "SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id；"。通过"并行"，这意味着该查询可以在每个分区上同时进行，最终结果只需通过总计所有分区得到的结果。

通过跨多个磁盘来分散数据查询，来获得更大的查询吞吐量。

分区类型

RANGE分区 基于属于一个给定连续区间的列值，把多行分配给分区。

HASH分区 基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含 OceanBase 中有效的、产生非负整数 值的任何表达式。

无论使用何种类型的分区，分区总是在创建时就自动的顺序编号，且从 0 开始记 录。当有一新行插入到一个分区表中时，就是使用这些分区编号来识别正确的分区。

例如，如果你的表使用 4 个分区，那么这些分区就编号为 0, 1, 2, 和 3。对于RANGE 分区类型，确保每个分区编号都定义了一个分区。对HASH分区，使用的用户函数必须返回一个大于 0 的整数值。分区的名字是不区分大小写的。

KEY分区 按照 KEY 分区类似与按照 HASH 分区，除了HASH 分区使用的用户定义的表达式，而KEY分区的哈希函数是由OceanBase服务器提供，且按key分区的列值不仅仅可以是整数值，还可以是其他类型。

RANGE分区（暂不支持）

语法

```
...
 PARTITION BY RANGE {(expr) | COLUMNS(column_list)}
    (partition_definition [, partition_definition] ...)

 partition_definition:
    PARTITION partitionname
    VALUES {LESS THAN {(expr | value_list) | MAXVALUE}
```

举例

按照 RANGE 分区的表是通过如下一种方式进行分区的，每个分区包含那些分区表达式的值位于一个给定的连续区间内的行。在下面的几个例子中，假定你创建了一个如下的一个表，该表保存有 20 家音像店的职员记录，这20家音像店的编号从1到20。

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
 ) ;
```

根据你的需要，这个表可以有多种方式来按照区间进行分区。一种方式是使用store_id 列。例如，你可能决定通过添加一个PARTITION BY RANGE子句把这个表分割成4个区间，如下所示:

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
```

```
 store_id INT NOT NULL
 )
 PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
 ) ;
```

按照这种分区方案，在商店 1 到 5 工作的雇员相对应的所有行被保存在分区 P0中，商店 6 到 10雇员保存在 P1 中，依次类推。注意，每个分区都是按顺序进行定义，从最低到最高。这是 PARTITION BY RANGE 语法的要求；在这点上，它类似于C或Java中的"switch ... case"语句。

对于包含数据(72, 'Michael', 'Widenius', '1998-06-25', NULL, 13)的一个新行，可以很容易地确定它将插入到 p2 分区中，但是如果增加了一个编号为第 21的商店，将会发生什么呢？在这种方案下，由于没有规则把 store_id 大于 20 的商店包含在内，服务器将不知道把该行保存在何处，将会导致错误。 要避免这种错误，可以通过在 CREATE TABLE 语句中使用一个"catchall" VALUES LESS THAN 子句，该子句提供给所有大于明确指定的最高值的值：

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
 )
 PARTITION BY RANGE (store_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
 ) ;
```

MAXVALUE 表示最大的可能的整数值。现在，store_id 列值大于或等于 16（定义了的最高值）的所有行都将保存在分区 p3 中。在将来的某个时候，当商店数已经增长到 25, 30, 或更多，可以使用ALTER TABLE 语句为商店 21-25, 26-30,等等增加新的分区。

在几乎一样的结构中，你还可以基于雇员的工作代码来分割表，也就是说，基于job_code 列值的连续区间。例如,假定 2 位数字的工作代码用来表示普通（店内的）工人，三个数字代码表示办公室和支持人员，四个数字代码表示管理层，你可以使用下面的语句创建该分区表：

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
```

```
)
PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
) ;
```

在这个例子中, 店内工人相关的所有行将保存在分区 p0 中，办公室和支持人员相关的所有行保存在分区 p1 中，管理层相关的所有行保存在分区 p2 中。

在 VALUES LESS THAN 子句中使用一个表达式也是可能的。这里最值得注意的限制是必须能够计算表达式的返回值作为LESS THAN (<)比较的一部分；因此，表达式的值不能为NULL 。由于这个原因，雇员表的hired, separated, job_code和store_id列已经被定义为非空（NOT NULL）。

除了可以根据商店编号分割表数据外，你还可以使用一个基于两个 DATE（日期）中的一个的表达式来分割表数据。例如，假定你想基于每个雇员离开公司的年份来分割表，也就是说，YEAR(separated)的值。实现这种分区模式的 CREATE TABLE 语句的一个例子如下所示：

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT,
 store_id INT
 )
 PARTITION BY RANGE (YEAR(separated)) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
 ) ;
```

在这个方案中，在 1991 年前雇佣的所有雇员的记录保存在分区 p0 中，1991 年到 1995 年期间雇佣的所有雇员的记录保存在分区p1中，1996年到2000年期间雇佣的所有雇员的记录保存在分区 p2 中，2000年后雇佣的所有工人的信息保存在p3中。

使用场景

1. 当需要删除"旧的"数据时。
2. 想要使用一个包含有日期或时间值，或包含有从一些其他级数开始增长的值的列。
3. 经常运行直接依赖于用于分割表的列的查询。

HASH分区

语法

HASH 分区将要被哈希的列指定一个列值或表达式，以及指定被分区的表将要被分割成的分区数量

，OceanBase自动完成分区。

```
...
PARTITION BY HASH (expr)
    PARTITIONS num
```

要使用 HASH 分区来分割一个表，要在 CREATE TABLE 语句上添加一个"PARTITION BY HASH (expr)"子句，其中"expr"是一个返回一个整数的表达式。它可以仅仅是字段类型为 OceanBase 整型的一列的名字。此外，你很可能需要在后面再添加一个"PARTITIONS num"子句，其中 num 是一个非负的整数，它表示表将要被分割成分区的数量。

举例

也许你想基于雇用雇员的年份来进行分区。这可以通过下面的语句来实现：

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT,
 store_id INT
 )
 PARTITION BY HASH(YEAR(hired))
    PARTITIONS 4 ;
```

使用场景

HASH分区主要用来确保数据在预先确定数目的分区中平均分布。

KEY分区

语法

```
...
PARTITION BY KEY ( column_list )
 PARTITIONS num
```

KEY 分区只采用一个或多个列名的一个列表，它是按系统内部的哈希函数实现分区。

举例

创建表k1, 按id字段进行key分区。

```
Oceanbase> create table k2(id int primary key, name varchar(20)) partition by key() partitions 2;
Query OK, 0 rows affected (0.78 sec)
```

```
Oceanbase>show create table k2;
+-------+----------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+----------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------+
| k2    | CREATE TABLE `k2` (
  `id` int(11) NOT NULL,
  `name` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE partition by
key() partitions 2 |
+-------+----------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------+
1 row in set (0.03 sec)
```

此例中，id是主键字段，partition by key()和partition by key(id)是等效的。

子分区（暂不支持）

子分区是分区表中每个分区的再次分割。

语法

```
    ...
 PARTITON BY RANGE(expr)
     SUBPARTITION BY HASH(expr)
    ...
```

举例

```
CREATE TABLE ts (id INT, purchased DATE)
 PARTITION BY RANGE(YEAR(purchased))
 SUBPARTITION BY HASH(TO_DAYS(purchased))
 SUBPARTITIONS 2
 (
     PARTITION p0 VALUES LESS THAN (1990),
     PARTITION p1 VALUES LESS THAN (2000),
     PARTITION p2 VALUES LESS THAN MAXVALUE
 ) ;
```

表 ts 有 3 个 RANGE 分区。这 3 个分区中的每一个分区 p0, p1 和 p2 又被进一步分成了 2 个子分区。实际上，整个表被分成了 3 * 2 = 6 个分区。但是，由于 PARTITION BY RANGE 子句的作用，这些分区的头 2 个只保存"purchased"列中值小于 1990 的那些记录。等价于：

```
CREATE TABLE ts (id INT, purchased DATE)
 PARTITION BY RANGE(YEAR(purchased))
 SUBPARTITION BY HASH(TO_DAYS(purchased))
```

```
(
    PARTITION p0 VALUES LESS THAN (1990)
        (
          SUBPARTITION s0,
          SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000)
        (
          SUBPARTITION s2,
          SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE
        (
          SUBPARTITION s4,
          SUBPARTITION s5
        )
);
```

几点要注意的语法项：

- 每个分区必须有相同数量的子分区。
- 如果在一个分区表上的任何分区上使用 SUBPARTITION 来明确定义任何子分区，那么就必须定义所有的子分区。换句话说，下面的语句将执行失败：

```
CREATE TABLE ts (id INT, purchased DATE)
 PARTITION BY RANGE(YEAR(purchased))
 SUBPARTITION BY HASH(TO_DAYS(purchased))
 (
    PARTITION p0 VALUES LESS THAN (1990)
    (
      SUBPARTITION s0,
      SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
    (
      SUBPARTITION s2,
      SUBPARTITION s3
    )
 ) ;
```

即便这个语句包含了一个 SUBPARTITIONS 2 子句，但是它仍然会执行失败。

每个 SUBPARTITION 子句必须包括 (至少)子分区的一个名字。否则，你可能要对该子分区设置任何你所需要的选项，或者允许该子分区对那些选项采用其默认的设置。

在每个分区内，子分区的名字必须是唯一的，但是在整个表中，没有必要保持唯一。例如，下面的 CREATE TABLE 语句是有效的：

```
CREATE TABLE ts (id INT, purchased DATE)
 PARTITION BY RANGE(YEAR(purchased))
 SUBPARTITION BY HASH(TO_DAYS(purchased))
```

```
(
    PARTITION p0 VALUES LESS THAN (1990)
    (
        SUBPARTITION s0,
        SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000)
    (
        SUBPARTITION s0,
        SUBPARTITION s1
    ),
    PARTITION p2 VALUES LESS THAN MAXVALUE
    (
        SUBPARTITION s0,
        SUBPARTITION s1
    )
) ;
```

分区处理NULL值的方式

NULL值视为0

对于分区禁止在空值（NULL）上没有进行处理，无论它是一个列值还是一个用户定义表达式的值。一般而言，在这种情况下把 NULL 视为 0。如果你希望回避这种做法，你应该在设计表时不允许空值；最可能的方法是，通过声明列"NOT NULL"来实现这一点。

说明：此处的 NULL 值，是指表达式计算结果为 NULL,当作 0 处理。

举例

如果插入一行到按照 RANGE 分区的表，该行用来确定分区的列值为 NULL，分区将把该 NULL 值视为 0。例如，考虑下面的两个表，表的创建和插入记录如下：

```
mysql> CREATE TABLE tnrange (
    -> id INT,
    -> name VARCHAR(5)
    -> )
    -> PARTITION BY RANGE(id) (
    -> PARTITION p1 VALUES LESS THAN (1),
    -> PARTITION p2 VALUES LESS THAN MAXVALUE
    -> );
mysql> INSERT INTO tnrange VALUES (NULL, 'jim');
mysql> SELECT * FROM tnrange;
+------+------+
| id | name |
+------+------+
| NULL | jim |
+------+------+
1 row in set (0.00 sec)
```

在表 tnrange 中，id 列没有声明为"NOT NULL"，这意味着它们允许 Null 值。可以通过删除这些分区，然后重新运行 SELECT 语句，来验证这些行被保存在每个表的 p1 分区中：

```
mysql> ALTER TABLE tnrange DROP PARTITION p1;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM tnrange;
Empty set (0.00 sec)
```

在按 HASH 分区的情况下，任何产生 NULL 值的表达式都视同好像它的返回值为 0。我们可以通过先创建一个按 HASH 分区的表，然后插入一个包含有适当值的记录，再检查对文件系统的作用，来验证这一点。假定有使用下面的语句在测试数据库中创建了一个表 tnhash：

```
CREATE TABLE tnhash (
 id INT,
 name VARCHAR(5)
 )
 PARTITION BY HASH(id)
    PARTITIONS 2;
```

现在在表 tnhash 中插入一行id列值为 NULL 的行，然后验证该行已经被插入

```
mysql> INSERT INTO tnhash VALUES (NULL, 'sam');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM tnhash;
+------+------+
| id | name |
+------+------+
| NULL | sam |
+------+------+
1 row in set (0.01 sec)
```

对于任意的整数 N,NULL MOD N 的值总是等于 NULL。这个结果在确定正确的分区方面被认为是 0。回到系统 shell(仍然假定 bash 用于这个目的），通过再次列出数据文件，可以看出值被成功地插入到第一个分区（默认名称为 p0）中：

```
mysql> SELECT * FROM tnhash partition(p0);
+------+------+
| id | name |
+------+------+
| NULL | sam |
+------+------+
1 row in set (0.00 sec)
```

分区管理（暂不支持）

分区管理主要是指添加、删除、重新定义、合并或拆分已经存在的分区。所有这些操作都可以通过使用 ALTER

TABLE 命令的分区扩展来实现。

RANGE分区

删除分区以及分区数据

语法

```
ALTER TABLE table_name DROP PARTITION partiton_name;
```

举例

使用下面的 CREATE TABLE 和 INSERT 语句创建了一个按照 RANGE 分区的表，并且已经插入了 10 条记录：

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
    -> PARTITION BY RANGE(YEAR(purchased))
    -> (
    -> PARTITION p0 VALUES LESS THAN (1990),
    -> PARTITION p1 VALUES LESS THAN (1995),
    -> PARTITION p2 VALUES LESS THAN (2000),
    -> PARTITION p3 VALUES LESS THAN (2005)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO tr VALUES
    -> (1, 'desk organiser', '2003-10-15'),
    -> (2, 'CD player', '1993-11-05'),
    -> (3, 'TV set', '1996-03-10'),
    -> (4, 'bookcase', '1982-01-10'),
    -> (5, 'exercise bike', '2004-05-09'),
    -> (6, 'sofa', '1987-06-05'),
    -> (7, 'popcorn maker', '2001-11-22'),
    -> (8, 'aquarium', '1992-08-04'),
    -> (9, 'study desk', '1984-09-16'),
    -> (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.01 sec)
```

可以通过使用下面的命令查看那些记录已经插入到了分区 p2 中：

```
mysql> SELECT * FROM tr
   -> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+------+-----------+------------+
| id | name | purchased |
+------+-----------+------------+
| 3 | TV set | 1996-03-10 |
| 10 | lava lamp | 1998-12-25 |
+------+-----------+------------+
2 rows in set (0.00 sec)
```

要删除名字为 p2 的分区，执行下面的命令：

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

记住下面一点非常重要：当删除了一个分区，也同时删除了该分区中所有的数据。可以通过重新运行前面的 SELECT 查询来验证这一点：

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

删除数据：保留表的结构，包括分区

语法

```
TRUNCATE TABLE table_name
```

举例

接上面的例子，删除 tr 表的所有内部。

```
mysql> TRUNCATE TABLE tr;
Query OK, 0 rows affected (0.25 sec)
```

查看表的内容为空

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

查看表的数据结构

```
mysql> show create table tr;
 +-------+-----------------------------------------------------------------
| Table | Create Table
 +-------+-----------------------------------------------------------------
| tr | CREATE TABLE `tr` (
`id` int(11) DEFAULT NULL,
`name` VARCHAR(50) DEFAULT NULL,
`purchased` date DEFAULT NULL
)
/*!50100 PARTITION BY RANGE (YEAR(purchased))
    (PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,
    PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,
    PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB) */ |
 +-------+-----------------------------------------------------------------
1 row in set (0.00 sec)
```

添加分区：为已经分区的表添加后续分区

语法

```
ALTER TABLE ... ADD PARTITION
```

举例

例如，假设有一个包含你所在组织的全体成员数据的分区表，该表的定义如下：

```
CREATE TABLE members (
 id INT,
 fname VARCHAR(25),
 lname VARCHAR(25),
 dob DATE
 )
 PARTITION BY RANGE(YEAR(dob)) (
    PARTITION p0 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990)
 );
```

进一步假设成员的最小年纪是 16 岁。随着日历接近 2005 年年底，你会认识到不久将要接纳 1990 年（以及以后年份）出生的成员。可以按照下面的方式，修改成员表来容纳出生在 1990 - 1999 年之间的成员：

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

约束

对于通过 RANGE 分区的表，只可以使用 ADD PARTITION 添加新的分区到分区列表的高端。设法通过这种方式在现有分区的前面或之间增加一个新的分区，将会导致下面的一个错误：

```
mysql> ALTER TABLE members ADD PARTITION (PARTITION p4 VALUES LESS THAN (1960));
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition
```

分区扩展

分区可以通过 REOGANIZE PARTITION 来实现拆分或合并分区等。

语法

```
ALTER TABLE tblname REORGANIZE PARTITION partition_list INTO (partition_definitions)；
```

其中，tbl_name 是分区表的名称，partition_list 是通过逗号分开的、一个或多个将要被改变的现有分区的列表。partition_definitions 是一个是通过逗号分开的、新分区定义的列表，它遵循与用在"CREATE TABLE"中的partition_definitions相同的规则。把多少个分区合并到一个分区或把一个分区拆分成多少个分区方面，没有限制。

举例

```
mysql> show create table members;
+---------+-------------------------------------------------------------
| Table | Create Table
+---------+-------------------------------------------------------------
| members | CREATE TABLE `members` (
 `id` int(11) DEFAULT NULL,
 `fname` VARCHAR(25) DEFAULT NULL,
 `lname` VARCHAR(25) DEFAULT NULL,
 `dob` date DEFAULT NULL
 )
 /*!50100 PARTITION BY RANGE (YEAR(dob))
    (PARTITION p0 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990),
    PARTITION p3 VALUES LESS THAN (2000) */ |
 +---------+-------------------------------------------------------------
 1 row in set (0.01 sec)
```

假定想要把表示出生在 1960 年前成员的所有行移入到一个分开的分区中。正如我们前面看到的，不能通过使用"ALTER TABLE ... ADD PARTITION"来实现这一点。但是，要实现这一点，可以使用 ALTER TABLE 上的另外一个与分区有关的扩展，具体实现如下：

```
mysql> ALTER TABLE members REORGANIZE PARTITION p0 INTO (
    -> PARTITION s0 VALUES LESS THAN (1960),
    -> PARTITION s1 VALUES LESS THAN (1970)
    -> );
 Query OK, 0 rows affected (0.53 sec)
 Records: 0 Duplicates: 0 Warnings: 0
 mysql> show create table members; //查看结果
 +---------+-------------------------------------------------------------
 | Table   | Create Table
 +---------+-------------------------------------------------------------
 | members | CREATE TABLE `members` (
    `id` int(11) DEFAULT NULL,
    `fname` VARCHAR(25) DEFAULT NULL,
    `lname` VARCHAR(25) DEFAULT NULL,
    `dob` date DEFAULT NULL
    )
 /*!50100 PARTITION BY RANGE (YEAR(dob))
    (PARTITION s0 VALUES LESS THAN (1960),
    PARTITION s1 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990),
    PARTITION p3 VALUES LESS THAN (2000) */ |
 +---------+-------------------------------------------------------------
 1 row in set (0.02 sec)
```

可以重新组织成员表的五个分区成两个分区，具体实现如下：

```
mysql> ALTER TABLE members REORGANIZE PARTITION s0,s1,p1,p2,p3 INTO (
```

```
    -> PARTITION m0 VALUES LESS THAN (1980),
   -> PARTITION m1 VALUES LESS THAN (2000));
Query OK, 0 rows affected (0.73 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> show create table members; //查看结果
+---------+-------------------------------------------------------------
| Table   | Create Table
+---------+-------------------------------------------------------------
| members | CREATE TABLE `members` (
   `id` int(11) DEFAULT NULL,
   `fname` VARCHAR(25) DEFAULT NULL,
   `lname` VARCHAR(25) DEFAULT NULL,
   `dob` date DEFAULT NULL
)
/*!50100 PARTITION BY RANGE (YEAR(dob))
   (PARTITION m0 VALUES LESS THAN (1980) ENGINE = InnoDB,
   PARTITION m1 VALUES LESS THAN (2000) ENGINE = InnoDB) */ |
+---------+-------------------------------------------------------------
1 row in set (0.01 sec)
```

HASH分区

合并分区

语法

```
ALTER TABLE ... COALESCE PARTITION
```

举例

例如，假定有一个包含顾客信息数据的表，它被分成了 12 个分区。该顾客表的定义如下：

```
CREATE TABLE clients (
 id INT,
 fname VARCHAR(30),
 lname VARCHAR(30),
 signed DATE
 )
 PARTITION BY HASH( MONTH(signed) )
    PARTITIONS 12 ;
```

要减少分区的数量从 12 到 6，执行下面的 ALTER TABLE 命令：

```
mysql> ALTER TABLE clients COALESCE PARTITION 6 ;
Query OK, 0 rows affected (0.02 sec)
```

约束

COALESCE 不能用来增加分区的数量，如果你尝试这么做，结果会出现类似于下面的错误：

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1508 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

增加分区

语法

```
ALTER TABLE ... ADD PARTITION
```

举例

在表 clients 中，把原有 6 个分区，增加 18 个分区，更新为 24 个分区。

```
mysql> alter table clients add partition partitions 18;
Query OK, 0 rows affected (3.33 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> show create table clients; //查看结果
+----------+-------------------------------------------------------------|
|Table     | Create Table
+----------+-------------------------------------------------------------|
clients2 | CREATE TABLE `clients` (
 `id` int(11) DEFAULT NULL,
 `fname` VARCHAR(30) DEFAULT NULL,
 `lname` VARCHAR(30) DEFAULT NULL,
 `signed` date DEFAULT NULL
) /*!50100 PARTITION BY HASH ( MONTH(signed))
    PARTITIONS 24 */ |
+----------+-------------------------------------------------------------1
row in set (0.00 sec)
```

注释："ALTER TABLE ... REORGANIZE PARTITION"不能用于按照HASH或HASH分区的表。

数据类型

数据类型

OceanBase1.0 支持的数据类型，主要分为三类：数值类型、字符串类型以及日期、时间类型，如下表所示。

数据类型分类	数据类型
数值类型	TINYINT BOOL,BOOLEAN SMALLINT MEDIUMINT INT, INTEGER

	BIGINT FLOAT DOUBLE DECIMAL, DEC NUMERIC
字符串类型	CHAR VARCHAR BINARY VARBINARY
日期和时间类型	DATE DATETIME TIMESTAMP TIME YEAR

数值类型

数值类型按精确度可以划分为两类：

- 精确数据类型
  - 整数数据类型 TINYINT, SAMLLINT, MEDIUMINT, INTEGER, BIGINT
  - 定点数据类型 DECIMAL, NUMERIC
- 近似数据类型 FLOAT, REAL, DOUBLE PRECISIONs

整数数据类型如下表：

类型	大小（字节）	范围	最小值	最大值
		(带符号的/无符号的)	(带符号的/无符号的)	(带符号的/无符号的)
TINYINT	1	27-1	-128	127
		28-1	0	255
SMALLINT	2	215-1	-32768	32767
		216-1	0	65535
MEDIUMINT	3	223-1	-8388608	8388607
		224-1	0	16777215
INT	4	231-1	-2147483648	2147483647
		232-1	0	4294967295
BIGINT	8	263-1	-9223372036854775808	9223372036854775807
		264-1	0	18446744073709551615

定点数据类型如下表：

类型	范围	用途
DECIMAL (M,D) NUMERIC (M,D)	DECIMAL、NUMERIC等价； 变长数据类型； 精度（M）最大位数38，标度（D）最大位数30，具体取值范围是受精度和标度的约束。（与MySQL不一致）	定点数值

浮点数据类型如下：

类型	大小（字节）	用途
FLOAT	4	单精度浮点数值
FLOAT(*p*)	如果0 <= *p* <= 24为4个字节， 如果25 <= *p* <= 53为8个字节	0到24的精度对应FLOAT列的4字节单精度。 25到53的精度对应DOUBLE列的8字节双精度。
DOUBLE [PRECISION]	8	双精度浮点数值

说明：

在TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT类型中都可以指定显示宽度，格式为"数据类型（M）"，比如INT(20)，这里的M指示最大显示宽度。最大有效显示宽度是255。显示宽度与存储大小或类型包含的值的范围无关。

UNSIGNED 修饰符规定字段只保存正值。

ZEROFILL修饰符规定0（不是空格）可以用于填补输出值。如果为一个数值列指定ZEROFILL，系统自动为该列添加UNSIGNED属性。

SERIAL是BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE的一个别名。在整数列定义中，SERIAL DEFAULT VALUE是NOT NULL AUTO_INCREMENT UNIQUE的一个别名。

BOOL（BOOLEAN）类型等价于TINYINT类型。

TINYINT

TINYINT[(*M*)] [UNSIGNED] [ZEROFILL]

很小的整数。带符号的范围是-128到127。无符号的范围是0到255。

BOOL,BOOLEAN

是TINYINT(1)的同义词。zero值被视为假。非zero值视为真。

SMALLINT

SMALLINT[(*M*)] [UNSIGNED] [ZEROFILL]

小的整数。带符号的范围是-32768到32767。无符号的范围是0到65535。

MEDIUMINT

MEDIUMINT[(*M*)] [UNSIGNED] [ZEROFILL]

中等大小的整数。带符号的范围是-8388608到8388607。无符号的范围是0到16777215。

INT

INT[(*M*] [UNSIGNED] [ZEROFILL]

普通大小的整数。带符号的范围是-2147483648到2147483647。无符号的范围是0到4294967295。

将非法的int值插入表之前会自动改为0。

INTEGER

INTEGER[(*M*)] [UNSIGNED] [ZEROFILL]

INT的同义词。

BIGINT

BIGINT[(*M*)] [UNSIGNED] [ZEROFILL]

大整数。带符号的范围是-9223372036854775808到9223372036854775807。无符号的范围是0到18446744073709551615。

使用带符号的BIGINT或DOUBLE值进行所有算法，因此除了位函数，不应使用大于9223372036854775807(63位)的无符号的大整数！如果这样做，结果中的最后几位可能出错，这是由于将BIGINT值转换为DOUBLE进行四舍五入时造成的错误。

可以在以下情况下处理BIGINT：

当使用整数在一个BIGINT列保存大的无符号的值时。

在MIN(col_name)或MAX(col_name)中，其中col_name指BIGINT列。

使用操作符(+，-，*等等)并且两个操作数均为整数时。

总是可以使用一个字符串在BIGINT列中保存严格整数值。在这种情况下，执行字符串-数字转换，其间不存在双精度表示。

当两个操作数均为整数值时，-、+和* 操作符使用BIGINT算法。这说明如果乘两个大整数(或来自返回整数的函数)，当结果大于9223372036854775807时，会得到意想不到的结果。

FLOAT

FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

小(单精度)浮点数。允许的值范围为-2128 ~ +2128，也即-3.402823466E+38到-1.175494351E-38、0和1.175494351E-38到3.402823466E+38。这些是理论限制，基于IEEE标准。实际的范围根据硬件或操作系统的不同可能稍微小些。

*M*是小数总位数，*D*是小数点后面的位数。如果*M*和*D*被省略，根据硬件允许的限制来保存值。单精度浮点数精确到大约7位小数位。

如果指定UNSIGNED，不允许负值。

使用浮点数可能会遇到意想不到的问题，因此所有计算用双精度完成。

DOUBLE

DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

普通大小(双精度)浮点数。允许的值范围为：-21024 ~ +21024，也即是-1.7976931348623157E+308到-2.2250738585072014E-308、0和2.2250738585072014E-308到1.7976931348623157E+308。这些是理论限制，基于IEEE标准。实际的范围根据硬件或操作系统的不同可能稍微小些。

*M*是小数总位数，*D*是小数点后面的位数。如果*M*和*D*被省略，根据硬件允许的限制来保存值。双精度浮点数精确到大约15位小数位。

如果指定UNSIGNED，不允许负值。

DOUBLE PRECISION

DOUBLE PRECISION [(*M*,*D*)] [UNSIGNED] [ZEROFILL], REAL[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

为DOUBLE的同义词。

FLOAT(*p*)

FLOAT(*p*) [UNSIGNED] [ZEROFILL]

浮点数。*p*表示精度（以位数表示），只使用该值来确定是否结果列的数据类型为FLOAT或DOUBLE。如果p为从0到24，数据类型变为没有M或D值的FLOAT。如果p为从25到53，数据类型变为没有M或D值的DOUBLE。结果列范围与本节前面描述的单精度FLOAT或双精度DOUBLE数据类型相同。

DECIMAL(与MySQL不完全一致)

DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

压缩的"严格"定点数。*M*是小数位数(精度)的总数，*D*是小数点(标度)后面的位数。小数点和'-'(负数)符号不包括在*M*中。如果*D*是0，则值没有小数点或分数部分。DECIMAL整数最大位数(*M*)为38（MySQL 为65）。支持的十进制数的最大位数(*D*)是30。如果*D*被省略， 默认是0。如果*M*被省略， 默认是10。

如果指定UNSIGNED，不允许负值。

所有DECIMAL列的基本计算(+，-，*，/)用38（MySQL为65）位精度完成。

DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]是DECIMAL的同义词。FIXED同义词适用于与其它服务器的兼容性。

NUMERIC

NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]是DECIMAL的同义词。

字符串类型

字符类型	字节	说明
CHAR	0-255	CHAR列的长度固定为创建表时声明的长度。 长度可以为从0到255的任何值。
VARCHAR	0-262143	VARCHAR的最大实际长度由最长的行的大小和使用的字符集确定。 **最大有效长度是256K,区别于MySQL的64K。**
BINARY	0-255	类似于CHAR类型，但保存二进

		制字节字符串而不是非二进制字符串。
VARBINARY	0-262143	类似于VARCHAR类型，但保存二进制字节字符串而不是非二进制字符串。

CHAR

CHAR[(M)] [CHARACTER SET charsetname] [COLLATE collationname]

CHAR列的长度固定为创建表时声明的长度。长度可以为从 0 到 255 的任何值。 当保存 CHAR 值时，在它们的右边填充空格以达到指定的长度。当检索到 CHAR 值时， 尾部的空格被删除掉。在存储或检索过程中不进行大小写转换。

若没有指定长度，则是CHAR(1)的同义词。

OceanBase 1.0中Char和Varchar数据类型在内部都是变长存储。

CHAR是CHARACTER的简写。

VARCHAR

VARCHAR(M) [CHARACTER SET charsetname] [COLLATE collationname]

变长字符串。*M*表示最大列长度。VARCHAR的最大实际长度由最长的行的大小和使用的字符集确定。最大有效长度是256K（262,144字节）。假设系统使用UTF8MB4字符集，每个字符占用4字节，*M*最大值为:（1024*256）/4 = 65536 (字符)。则*M*的范围是0到**65536**。

如果分配给CHAR或VARCHAR列的值超过列的最大长度，则对值进行裁剪以使其适合。如果被裁掉的字符是空格，则可以成功插入。如果裁剪非空格字符，则会造成错误(而不是警告)并通过使用严格SQL模式禁用值的插入。

VARCHAR是字符CHAR VARYING的简写。

varchar类型最大长度限制：

建表的时候，检查varchar类型主键长度之和小于等于16K，且总长度小于1.5M，否则报错;

建立索引的时候，索引表中varchar类型主键之和小于等于16K，且总长度小于1.5M，否则不允许建立索引;

单个varchar列长度小于256K。

CHAR和VARCHAR的区别:

同CHAR对比，VARCHAR值保存时只保存需要的字符数，另加一到三个字节来记录长度。

下面的表显示了将各种字符串值保存到CHAR(4)和VARCHAR(4)列后的结果，说明 了CHAR和VARCHAR之间的差别：

值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
''	' '	4 个字节	''	1 个字节
'ab'	'ab '	4 个字节	'ab '	3 个字节
'abcd'	'abcd'	4 个字节	'abcd'	5 个字节
'abcdefgh'	'abcd'	4 个字节	'abcd'	5 个字节

上表最后一行仅限于没有使用严格SQL模式的情况下，若使用了严格SQL模式，会直接报错，并非存储截断的字符串。

BINARY, VARBINARY

BINARY和VARBINARY类似CHAR和VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

BINARY和VARBINARY允许的最大长度一样，如同CHAR和VARCHAR，不同的是BINARY和VARBINARY的长度是字节长度而不是字符长度。

日期和时间类型

日期和时间类型	格式	范围	大小（字节）
DATE	YYYY-MM-DD	'1000-01-01'到 '9999-12-31'	3
DATETIME	YYYY-MM-DD HH:MM:SS	'1000-01-01 00:00:00'到 '9999-12-31 23:59:59'	5+秒精度存储（实际存储：8字节）
TIMESTAMP	YYYY-MM-DD HH:MM:SS	'1970-01-01 00:00:00'到 '2037-12-31 23:59:59'	4+秒精度存储（实际存储：8字节
TIME	HH:MM:SS	'-838:59:59'到 '838:59:59'	3+秒精度存储（实际存储：6字节）
YEAR	YYYY(默认格式)	'1901到2155'和 '0000'	1
	YY	'70到69',表示从1970年到2069年。	

在日期和时间函数中，DATETIME、TIMESTAMP、TIME类型，默认是精确到秒，还支持指定秒精度参数，秒精度存储根据存储需求最大支持微妙（3字节），秒精度存储根据精度不同，占用空间取0-3字节。

形如：*type_name*(*fsp*)。

type_name 表示DATETIME、TIMESTAMP、TIME等类型。

*fsp*参数是表示秒精度，取值范围为：0-6。默认值取0；最大值为6，表示精确到微妙。

DATE

日期。支持的范围为'1000-01-01'到'9999-12-31'。以'YYYY-MM-DD'格式显示DATE值，但允许使用字符串或数字为DATE列分配值。

DATETIME

DATETIME[(*fsp*)]

日期和时间的组合。支持的范围是'1000-01-01 00:00:00.000000'到'9999-12-31 23:59:59.000000'。以'YYYY-MM-DD HH:MM:SS[.*fraction*] '格式显示DATETIME值，但允许使用字符串或数字为DATETIME列分配值。*fsp*参数是表示秒精度，取值范围为：0-6。默认值取0；最大值为6，表示精确到微妙。

DATETIME 数据类型和 TIMESTAMP 数据类型都支持 DEFAULT CURRENT_TIMESTAMP 和 ON UPDATE CURRENT_TIMESTAMP 子句。

TIMESTAMP

TIMESTAMP[(*fsp*)]

这个数据类型与MySQL不完全一致，主要是在OceanBase 1.0里TIMESTAMP类型的格式是按严格模式来判断，不允许"0000-00-00 00:00:00"等非法值。

下列字符串类型是不合法的：

```
'2012^12^32'
'20070523'
'070523'
'071332'
```

下列整数类型同样是不合法的：

```
19830905
830905
```

DEFAULT CURRENT_TIMESTAMP，ON UPDATE CURRENT_TIMESTAMP 子句

用DEFAULT CURRENT_TIMESTAMP和ON UPDATE CURRENT_TIMESTAMP子句，列默认使用当前的时间戳，并且自动更新；如果默认值设定为其他值（如0或NULL），则不会自动更新时间戳。

TIME

TIME[(*fsp*)]

时间。范围是 '-838:59:59.000000'到'838:59:59.000000'。以'HH:MM:SS[.*fraction*]'格式显示TIME值，但允许使用字符串或数字为TIME列分配值。*fsp*参数是表示秒精度，取值范围为：0-6。默认值取0；最大值为6，表示精确到微妙。

'D HH:MM:SS[.fraction]'格式的字符串。还可以使用下面任何一种"非严格"语法：'H:MM:SS[.*fraction*]'、'HH:MM:SS'、'HH:MM'、'D HH:MM:SS'、'D HH:MM'、'D HH'或'SS'。这里D表示日，可以取0到34之间的值。

'HHMMSS'格式的没有间割符的字符串，假定是有意义的时间。例如，'101112'被理解为'10:11:12'，但'109712'是不合法的(它有一个没有意义的分钟部分)，将变为'00:00:00'。

HHMMSS格式的数值，假定是有意义的时间。例如，101112被理解为'10:11:12'。下面格式也可以理解：SS、MMSS、HHMMSS、HHMMSS.fraction。

函数返回的结果，其值适合TIME上下文，例如CURRENT_TIME。

对于指定为包括时间部分间割符的字符串的TIME值，如果时、分或者秒值小于10，则不需要指定两位数。'8:3:2'与'08:03:02'相同。

为TIME列分配简写值时应注意。没有冒号，解释值时假定最右边的两位表示秒。(解释TIME值为过去的时间而不是当天的时间）。例如，你可能认为'1112'和1112表示'11:12:00'(11点过12分),系统中将它们解释为'00:11:12'(11分，12 秒)。同样，'12'和12 被解释为 '00:00:12'。相反，TIME值中使用冒号则肯定被看作当天的时间。也就是说，'11:12'表示'11:12:00'，而不是'00:11:12'。

超出TIME范围但合法的值被裁为范围最接近的端点。例如，'-850:00:00'和'850:00:00'被转换为'-838:59:59'和'838:59:59'。

无效TIME值被转换为'00:00:00'。请注意由于'00:00:00'本身是一个合法TIME值，只从表内保存的一个'00:00:00'值还不能说出原来的值是 '00:00:00'还是不合法的值。

YEAR

两位或四位格式的年。默认是四位格式。在四位格式中，允许的值是1901到2155和0000。在两位格式中，允许的值是70到69，表示从1970年到2069年。以YYYY 格式显示YEAR值，但允许使用字符串或数字为YEAR列分配值。

可以指定各种格式的YEAR值：

四位字符串，范围为'1901'到'2155'。

四位数字，范围为1901到2155。

两位字符串，范围为'00'到'99'。'00'到'69'和'70'到'99'范围的值被转换为2000到2069和1970到1999范围的YEAR值。

两位整数，范围为1到99。1到69和70到99范围的值被转换为2001到2069和1970到1999范围的YEAR值。请注意两位整数范围与两位字符串范围稍有不同，因为你不能直接将零指定为数字并将它解释为2000。你必须将它指定为一个字符串'0'或'00'或它被解释为0000。

函数返回的结果，其值适合YEAR上下文，例如NOW()。

非法YEAR值被转换为0000。

对于DATETIME、DATE、TIMESTAMP和YEAR类型，OceanBase使用以下规则解释含模糊年值的日期：

00-69范围的年值转换为2000-2069。

70-99范围的年值转换为1970-1999。

ORDER BY可以正确排序有两位年的TIMESTAMP或YEAR值。

部分函数如MIN()和MAX()将TIMESTAMP或YEAR转换为一个数字。这说明使用有两位年值的值，这些函数不能工作正确。在这种情况下的修复方法是将TIMESTAMP或YEAR转换为四位年格式或使用MIN(DATE_ADD(TIMESTAMP,INTERVAL 0 DAYS))。

运算符

运算符

OceanBase支持多种类型的运算符，主要包括算数运算符、比较运算符、向量比较运算符、逻辑运算符和位运算符。本节介绍OceanBase支持的运算符以及运算符的优先级。

逻辑运算符

在OceanBase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。

OceanBase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"。

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/E	True/False	NULL

```
Oceanbase>SELECT NOT 0, NOT 1, NOT NULL;
+-------+-------+----------+
| NOT 0 | NOT 1 | NOT NULL |
+-------+-------+----------+
|     1 |     0 |     NULL |
+-------+-------+----------+
1 row in set (0.00 sec)
```

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							NULL

```
Oceanbase>SELECT (0 AND 0), (0 AND 1), (1 AND 1), (1 AND NULL);
+-----------+-----------+-----------+--------------+
| (0 AND 0) | (0 AND 1) | (1 AND 1) | (1 AND NULL) |
+-----------+-----------+-----------+--------------+
|         0 |         0 |         1 |         NULL |
+-----------+-----------+-----------+--------------+
1 row in set (0.00 sec)
```

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	Error	Error
VARCHAR					True/False/Error	True/False/Error	True/NULL
BOOL						True/False	True/NULL
NULL							NULL

```
Oceanbase>SELECT (0 OR 0), (0 OR 1), (1 OR 1), (1 AND NULL);
+----------+----------+----------+--------------+
| (0 OR 0) | (0 OR 1) | (1 OR 1) | (1 AND NULL) |
+----------+----------+----------+--------------+
|        0 |        1 |        1 |         NULL |
+----------+----------+----------+--------------+
1 row in set (0.01 sec)
```

XOR(暂不支持)

逻辑异或。当任意一个操作数为NULL时，返回值为NULL。对于非NULL的操作数，假如有奇数个操作数为非零值，则计算所得结果为1，否则为0。

```
Mysql>SELECT 1 XOR TRUE, 1 XOR 1, 1 XOR 2, 1 XOR NULL, 1 XOR 1 XOR 1;
+------------+---------+---------+------------+---------------+
| 1 XOR TRUE | 1 XOR 1 | 1 XOR 2 | 1 XOR NULL | 1 XOR 1 XOR 1 |
+------------+---------+---------+------------+---------------+
|          0 |       0 |       0 |       NULL |             1 |
+------------+---------+---------+------------+---------------+
1 row in set (0.01 sec)
```

算数运算符

OceanBase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。字符串在做算术运算时，如果无法转成DOUBLE类型则报错，比如"'3.4he' + 3"。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成DOUBLE型。

OceanBase支持的运算符如下表所示。

表达式	含义	举例
+	加法。	SELECT 2+3;
-	减法。	SELECT 2-3;
*	乘法。	SELECT 2*3;
/	除法，返回商。如果除数为"0"，则返回结果为"NULL"。	SELECT 2/3;
%或MOD	除法，返回余数。如果除数为"0"，则返回结果为"NULL"	SELECT 2%3, 2 MOD 3;
^	返回指定数值的指定幂运算结果。	SELECT 2^2

"+"、"-"、"*"的操作类型对照如下表所示：

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL
INT	INT	DOUBLE	DOUBLE	Error	DOUBLE/Error	Error
FLOAT		DOUBLE	DOUBLE	Error	DOUBLE/Error	Error
DOUBLE			DOUBLE	Error	DOUBLE/Error	Error
TIMESTAMP				Error	Error	Error
VARCHA					DOUBLE/	Error

R					Error	
BOOL						Error

"/"的操作类型对照如下表，如果除数为零则报错。

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	DOUBLE/Error	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

"%"和"MOD"的操作类型对照如下表。

	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

比较运算符

OceanBase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"。

数值比较

比较运算符用于比较两个数值的大小。OceanBase支持的比较运算符如下表所示。

表达式	含义	举例
=	等于。	SELECT 1=0, 1=1, 1=NULL;
>=	大于等于。	SELECT 1>=0, 1>=1, 1>=2, 1>=NULL;
>	大于。	SELECT 1>0, 1>1, 1>2, 1>NULL;
<=	小于等于。	SELECT 1<=0, 1<=1, 1<=2, 1<=NULL;
<	小于。	SELECT 1<0, 1<1, 1<2, 1<NULL;
!=或<>	不等于。	SELECT 1!=0, 1!=1, 1<>0 , 1<>1, 1!=NULL, 1<>NULL;

数值比较运算符的类型转换规则如下表所示。

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	INT	FLOAT	DOUBLE	Error	?INT	Error	NULL
FLOAT		FLOAT	DOUBLE	Error	?FLOAT	Error	NULL
DOUBLE			DOUBLE	Error	?DOUBLE	Error	NULL
TIMESTAMP				TIMESTAMP	?TIMESTAMP	Error	NULL
VARCHAR					VARCHAR	?BOOL	NULL
BOOL						BOOL	NULL
NULL							NULL

注意：

- "?"开头的类型表示会在执行的时候尝试转换到指定类型，如果转换失败则报错。
- BOOL类型比较时，False小于True。

[NOT] BETWEEN ... AND ...

判断是否存在或者不存在于指定范围。

```
Oceanbase>SELECT 2 BETWEEN 1 AND 2, 3 NOT BETWEEN 1 AND 2, 1 BETWEEN null AND 0, 1 NOT BETWEEN null AND 0\G
*************************** 1. row ***************************
      2 BETWEEN 1 AND 2: 1
  3 NOT BETWEEN 1 AND 2: 1
   1 BETWEEN null AND 0: 0
1 NOT BETWEEN null AND 0: 1
1 row in set (0.01 sec)
```

示例

有两张表，emp雇员信息表（员工姓名、工资等信息）和salgrade工资等级表（工资范围和工资等级）；现要查询员工姓名、工资以及工资等级信息。

```
Oceanbase>select * from emp;  //雇员信息表
+--------+----------+
| ename  | sal      |
+--------+----------+
| Jerry  | 25000.00 |
| Larry  | 40000.00 |
| Maggie | 46000.00 |
| Micky  | 15000.00 |
+--------+----------+
4 rows in set (0.00 sec)
Oceanbase>select * from salgrade; //工资等级表
+-------+----------+----------+
| grade | losal    | hisal    |
+-------+----------+----------+
|     1 | 10000.00 | 20000.00 |
|     2 | 20001.00 | 30000.00 |
|     3 | 30001.00 | 40000.00 |
|     4 | 40001.00 | 50000.00 |
|     5 | 50001.00 | 90000.00 |
+-------+----------+----------+
5 rows in set (0.00 sec)
Oceanbase>select a1.ename, a1.sal, a2.grade from emp a1, salgrade a2 where a1.sal between a2.losal and a2.hisal;
//通过between...and...设定查询条件
+--------+----------+-------+
| ename  | sal      | grade |
+--------+----------+-------+
| Micky  | 15000.00 |     1 |
| Jerry  | 25000.00 |     2 |
| Larry  | 40000.00 |     3 |
| Maggie | 46000.00 |     4 |
+--------+----------+-------+
4 rows in set (0.00 sec)
```

[NOT] IN

判断是否存在于指定集合。

```
Oceanbase>SELECT 2 IN (1, 2), 3 IN (1, 2)\G;
*************************** 1. row ***************************
2 IN (1, 2): 1
3 IN (1, 2): 0
1 row in set (0.00 sec)
```

IS [NOT] NULL | TRUE | FALSE | UNKNOWN

判断是否为NULL、真、假或未知。如果执行成功，结果是TRUE或者FALSE，不会出现NULL。

左参数必须是BOOL类型，或者是NULL，否则报错。

```
Oceanbase>SELECT 0 IS NULL,
->     NULL IS NULL, NULL IS TRUE,
->     (0>1) IS FALSE,
->     NULL IS UNKNOWN,
->     0 IS NOT NULL,
->     NULL IS NOT NULL,
->     NULL IS NOT TRUE,
->     (0>1) IS NOT FALSE,
->     NULL IS NOT UNKNOWN\G;
*************************** 1. row ***************************
         0 IS NULL: 0
      NULL IS NULL: 1
      NULL IS TRUE: 0
     (0>1) IS FALSE: 1
    NULL IS UNKNOWN: 1
      0 IS NOT NULL: 1
   NULL IS NOT NULL: 0
   NULL IS NOT TRUE: 1
 (0>1) IS NOT FALSE: 0
NULL IS NOT UNKNOWN: 0
1 row in set (0.00 sec)
```

向量比较运算符

向量比较运算符对两个向量（ROW）进行比较，支持"<"、" >"、"="、"<="、">="、 "!="、"<=>"、"in"和"not in"等操作符。这几个操作符都是二元操作符，被比较的两个向量的维度要求相同。

"<=>"表示NULL-safe equal，这个操作符和=操作符执行相同的比较操作，不过在两个操作码均为NULL时，其所得值为1而不为NULL，而当一个操作码为NULL时，其所得值为0而不为NULL。

表达式（1，2）和ROW（1，2）有时被称为行构造符。两者是等同的，在其它的语境中，也是合法的。例如，以下两个语句在语义上是等同的（但是目前只有第二个语句可以被优化）：

```
SELECT * FROM t1 WHERE (column1, column2) = (1, 1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

向量比较操作符和普通操作符相比主要有以下不同：

- 如果两个操作数的第i个标量的比较就决定了比较结果，则不再继续比较后续的数值。

向量比较操作符需要注意以下几点：

多元向量比较，关键字ROW可以省略。例如，ROW(1,2,3) < ROW(1,3,5)等价于(1,2,3) < (1,3,4)。

in/not in操作一定是向量操作，表示左参数（不）在右集合内，集合用"（）"括起。例如， 1 in (2, 3, 1)。

in/not in操作需要左右操作数对应位置的标量都是可以比较的，否则返回错误。例如，ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4))成功，ROW(1,2) in (ROW(2,1), ROW(2,3), ROW(1,3,4))失败。

```
Oceanbase>SELECT ROW(1,2) < ROW(1, 3),
    -> ROW(1,2,10) < ROW(1, 3, 0),
    -> ROW(1,null) < ROW(1,0),
    -> ROW(null, 1) < ROW(null, 2),
    -> ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)),
    -> 1 in (1,2,3),
    -> 1 not in (2,3,4),
    -> ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)),
    -> NULL = NULL,
    -> NULL <=> NULL,
    -> NULL <=> 1,
    -> 1 <=> 0 \G;
*************************** 1. row ***************************
                          ROW(1,2) < ROW(1, 3): 1
                       ROW(1,2,10) < ROW(1, 3, 0): 1
                         ROW(1,null) < ROW(1,0): NULL
                      ROW(null, 1) < ROW(null, 2): NULL
ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)): 1
                                1 in (1,2,3): 1
                              1 not in (2,3,4): 1
      ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)): 1
                                  NULL = NULL: NULL
                                NULL <=> NULL: 1
                                  NULL <=> 1: 0
                                    1 <=> 0: 0
1 row in set (0.00 sec)
```

位运算符

对于比特运算，OceanBase使用BITINT（64比特）算法，这些操作符的最大范围是64比特。

表达式	含义	举例
BIT_COUNT(N)	返回参数N中设置的比特数。	SELECT BIT_COUNT(29); -> 4
&	位运算符与。	SELECT 29 & 15; -> 13

		结果为一个64比特无符号整数。
~	反转所有比特。	SELECT 29 & ~15; -> 16 结果为一个64比特无符号整数。
|	位运算或。	SELECT 29 | ~15; -> 31 结果为一个64比特无符号整数。
^	位运算异或。	SELECT 1 ^ 1; -> 0 结果为一个64比特无符号整数。
<<	把一个BIGINT数左移两位。	SELECT 1 << 2; -> 4 其结果为一个64比特无符号整数。
>>	把一个BIGINT数右移两位。	SELECT 4 << 2; -> 1 其结果为一个64比特无符号整数。

运算符优先级

当我们需要对OceanBase的操作符进行混合运算时，我们需要了解这些操作符的优先级。

OceanBase中操作符的优先级由高到低，如下所示。

优先级	运算符
15	!
14	-(负号), ~
13	^
12	*，/，%，MOD
11	+，-
10	<<, >>
9	&
8	|
7	=(比较运算符等于)，<=>,>，>=，<，<=，<>，!=，IS，LIKE，REGEXP，IN
6	BETWEEN

5	NOT
4	AND, &&
3	XOR
2	OR, ||
1	=（赋值运算符）

函数

函数

OceanBase支持的函数可以分为日期时间函数、字符串函数、转换函数、聚合函数、控制流程函数、数学函数、比较函数、信息函数以及其他函数。

在SQL语句中，表达式可用于一些诸如SELECT语句的ORDER BY或 HAVING子句、SELECT、DELETE或UPDATE语句的WHERE子句或SET语句之类的地方。使用文本值、cloumn值、NULL值、函数、操作符来书写表达式。

一个包含NULL的表达式通常产生一个NULL值。

OceanBase支持的函数如下表所示。

分类	函数特性	函数名称
日期时间函数	日期时间函数主要用来显示有关日期和时间的信息。	CURRENT_TIME CURTIME CURRENT_TIMESTAMP CURDATE DATE_ADD DATE_FORMAT DATE_SUB EXTRACT NOW STR_TO_DATE TIME_TO_USEC USEC_TO_TIME UNIX_TIMESTAMP DATEDIFF TIMEDIFF TIMESTAMPDIFF PERIOD_DIFF TO_DAYS FROM_DAYS FROM_UNIXTIME
字符串函数	字符串函数对N进制数据、字符	CONCAT

	串和表达式执行不同的运算，如返回字符串的起始位置，返回字符串的个数等。	SUBSTRING SUBSTR TRIM LENGTH UPPER LOWER HEX UNHEX INT2IP IP2INT LIKE REPEAT SUBSTRING_INDEX LOCATE INSTR REPLACE FIELD ELT
转换函数	指定的数据类型转换为另一种数据类型。	CAST
聚合函数	聚合函数对一组值进行计算并返回单一的值。 通常聚合函数会与select语句的group by子句一同使用； 在与group by子句使用时，聚合函数会为每一个组产生一个单一值，而不会为整个表产生一个单一值。	AVG COUNT MAX MIN SUM GROUP_CONCAT
数学函数	数学函数能够对数字表达式进行数学计算。	ROUND CEIL FLOOR ABS NEG SIGN CONV MOD POW,POWER
比较函数	比较参数大小。	GREATEST LEAST ISNULL
控制流程函数	流程控制。	CASE IF IFNULL NULLIF
信息函数	返回动态的数据库信息。	FOUND_ROWS LAST_INSERT_ID
其他函数	未归类函数。	COALESCE NVL DECODE

日期时间函数

日期时间函数主要用来显示有关日期和时间的信息。

CURRENT_TIME()和CURRENT_TIMESTAMP()

注意：

1. CURRENT_TIME()和 CURRENT_TIMESTAMP()和 OceanBase 0.5 已有实现不一致，请格外留意。
2. current_time()和 current_timestamp()可以传入 0-6 的数字参数，表示秒后的小数点精度。

CURRENT_TIME()和CURRENT_TIMESTAMP()这两个函数用于获取系统当前时间，但返回格式有区别。

CURRENT_TIME()是取当前时间，但不包括日期，返回格式为"HH:MI:SS"。

CURRENT_TIMESTAMP()则取当前日期+时间，返回格式为"YYYY-MM-DD HH:MI:SS"。

current_time()和current_timestamp()可以传入0-6的数字参数，表示秒后的小数点精度。

```
Oceanbase>SELECT CURRENT_TIME(), CURRENT_TIMESTAMP();
+----------------+---------------------+
| CURRENT_TIME() | CURRENT_TIMESTAMP() |
+----------------+---------------------+
| 13:53:28       | 2016-03-14 13:53:28 |
+----------------+---------------------+
1 row in set (0.01 sec)

#可以支持传入秒后的小数位精度参数
Oceanbase>select CURRENT_TIME(1);
+-----------------+
| CURRENT_TIME(1) |
+-----------------+
| 13:54:00.3      |
+-----------------+
1 row in set (0.01 sec)

#传入的精度范围是0-6，否则报错，错误码1426 (42000)
Oceanbase>select CURRENT_TIME(7);
ERROR 1426 (42000): Too big precision 7 specified for column 'current_time'. Maximum is 6.
```

CURTIME()

CURTIME() 是CURRENT_TIME()，CURRENT_TIME的同义词。

```
Oceanbase>select curtime(), current_time(), current_time;
+-----------+----------------+--------------+
| curtime() | current_time() | current_time |
```

```
+-----------+---------------+--------------+
| 14:45:37  | 14:45:37      | 14:45:37     |
+-----------+---------------+--------------+
1 row in set (0.00 sec)
```

CURRENT_DATE()

CURRENT_DATE() 返回当前日期，以'YYYY-MM-DD'或者 YYYYMMDD的形式显示。以何种形式显示取决于函数中的内容是字符串还是数值。

例如：

```
Oceanbase>select current_date, current_date+5;
+--------------+----------------+
| current_date | current_date+5 |
+--------------+----------------+
| 2016-03-14   |       20160319 |
+--------------+----------------+
1 row in set (0.01 sec)
```

CURDATE()

CURDATE()是CURRENT_DATE(), CURRENT_DATE的同义词。

```
Oceanbase>select curdate(), current_date(), current_date;
+------------+----------------+--------------+
| curdate()  | current_date() | current_date |
+------------+----------------+--------------+
| 2016-03-14 | 2016-03-14     | 2016-03-14   |
+------------+----------------+--------------+
1 row in set (0.01 sec)
```

DATE_ADD(*date*, INTERVAL *expr unit*)

这个函数用来执行时间的算术计算。将*date*值作为基数，对*expr*进行相加计算，*expr*的值允许为负数。DATE_ADD()计算时间值是否使用夏令时，由操作系统系统根据其内部配置和相应时区设定来决定。

> *date*参数类型只能为Time类型(DATETIME, TIMESTAMP等)或者代表时间的一个字符串，不接受其它类型。
>
> *date*参数的期望日期类型为"YYYY-MM-DD HH:MM:SS.SSSSSS"格式。MySQL允许解析日期类型字符串时允许"不严格"语法，如果一个字符串中包含数字和非数字，MySQL将解析出被非数字隔断的数字序列作为时间序列，依次赋给年月日。例如"Ywwe1990d07 09,12:45-08&900"该字符串和"1990-07-09 12:45:08.900"在表示时间值上是等价的。在OceanBase 1.0实现"严格语法"，对非法日期类型进行报错处理，比如用户插入"abc"这种非法日期时做报错处理。

在*date*字符串中，日期部分是必须的，而时间部分是可以缺省的。 例如"1990-07-09"是合法的，这种情况下后面的时间部分将默认填充为0，其等价为"1990-07-09 00:00:00.000000"；而"1990-07"和"1990"这样的格式都是非法的。

目前date_add在解析date字符串的时候还不支持例如"990309"这种TIMESTAMP类型字符串。

OceanBase中的其它系统函数调用结果可以作为date参数进行计算。

OceanBase不支持对两位数的年份进行模糊匹配，例如12年在MySQL中匹配为2012年，而在OceanBase中就代表12年。

*expr*的值允许为负，对一个负值相加功能等同于对一个正值相减。允许系统函数的调用结果作为该参数，但是所有结果都将作为字符串结果。

*unit*为单位，支持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和YEAR_MONTH。其中QUARTER代表季度。

*unit*为复合单位时，*expr*必须加单引号。

在MySQL命令行客户端中，当单行显示过长，造成阅读困难时，可在SELECT结尾使用"\G"，将查询结果垂直排列。

```
Oceanbase>SELECT DATE_ADD(now(), INTERVAL 5 DAY),
->     DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL 5 SECOND),
->     DATE_ADD('2014-01-10', INTERVAL 5 MINUTE),
->     DATE_ADD('2014-01-10', INTERVAL 5 HOUR),
->     DATE_ADD('2014-01-10', INTERVAL 5 DAY),
->     DATE_ADD('2014-01-10', INTERVAL 5 WEEK),
->     DATE_ADD('2014-01-10', INTERVAL 5 MONTH),
->     DATE_ADD('2014-01-10', INTERVAL 5 QUARTER),
->     DATE_ADD('2014-01-10', INTERVAL 5 YEAR),
->     DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE),
->     DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND),
->     DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE),
->     DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR),
->     DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH) \G
```

```
*************************** 1. row ***************************
                                DATE_ADD(now(), INTERVAL 5 DAY): 2016-03-19 13:56:45
                     DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND): 2014-01-10 00:00:00.000005
                        DATE_ADD('2014-01-10', INTERVAL 5 SECOND): 2014-01-10 00:00:05
                        DATE_ADD('2014-01-10', INTERVAL 5 MINUTE): 2014-01-10 00:05:00
                          DATE_ADD('2014-01-10', INTERVAL 5 HOUR): 2014-01-10 05:00:00
                           DATE_ADD('2014-01-10', INTERVAL 5 DAY): 2014-01-15
                          DATE_ADD('2014-01-10', INTERVAL 5 WEEK): 2014-02-14 00:00:00
                         DATE_ADD('2014-01-10', INTERVAL 5 MONTH): 2014-06-10
                       DATE_ADD('2014-01-10', INTERVAL 5 QUARTER): 2015-04-10 00:00:00
                          DATE_ADD('2014-01-10', INTERVAL 5 YEAR): 2019-01-10
      DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND): 2014-01-10 00:00:05.000005
  DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND): 2014-01-10 00:05:05.000005
             DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND): 2014-01-10 00:05:05
  DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-10 05:05:05.000005
             DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND): 2014-01-10 05:05:05
               DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE): 2014-01-10 05:05:00
DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND): 2014-01-11 05:05:05.000005
           DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND): 2014-01-11 05:05:05
             DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE): 2014-01-11 05:05:00
                DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR): 2014-01-11 05:00:00
               DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH): 2015-02-10
1 row in set (0.01 sec)
```

日期计算还支持 INTERVARL 后面直接跟上加减（ +/- ）运算符形式。

```
date + INTERVAL expr unit
date - INTERVAL expr unit


Oceanbase>SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND,'2005-01-01' - INTERVAL 1 SECOND\G;
*************************** 1. row ***************************
'2008-12-31 23:59:59' + INTERVAL 1 SECOND: 2009-01-01 00:00:00
       '2005-01-01' - INTERVAL 1 SECOND: 2004-12-31 23:59:59
1 row in set (0.01 sec)
```

DATE_FORMAT(*date*, *format*)

DATE_FORMAT()是STR_TO_DATE()的逆函数，DATE_FORMAT()接受一个时间值*date*，将其按*format*的格式格式化成一个时间字符串。

> *date*参数给出了被格式化的时间值，date只接受time类型和时间字符串作为参数，其具体描述参考DATE_ADD()的date参数描述。
>
> *format*的格式如下表所示。

格式	含义	返回格式
%a	星期。	Sun..Sat
%b	月份的缩写名称。	Jan，...，Dec

%c	月份，数字形式。	1，...，12
%D	带有英语后缀的日期。	1st，2nd，...，31st
%d	日期，数字形式。	01，...，31
%e	日期，数字形式。	1，...，31
%f	微秒。	000000，...，999999
%H	小时。	00，...，23
%h	小时。	01，...，12
%I	小时。	01，...，12
%i	分钟。	00，...，59
%j	一年中的第几天。	000，...，366
%k	小时。	0，...，23
%l	小时。	01，...，12
%M	月份名称。	January，...，December
%m	月份，数字形式。	01，...，12
%p	上午或下午。	AM，PM
%r	12小时制时间。	hh:mm:ss AM/PM
%S	秒。	00，...，59
%s	秒。	00，...，59
%T	24小时制时间。	hh:mm:ss
%U	一年中的第几周，其中周日为每周的第一天。	00，...，53
%u	一年中的第几周，其中周一为每周的第一天。	00，...，53
%V	一年中的第几周，其中周日为每周的第一天， 和%X同时使用。 说明：在一年中的第一周或者最后一周产生跨年时 （以2014-01-01 星期三为例 ）， %U和%u时，该天为2014年的第00周， %V和%v时为2013年的第52周。	01，...，53
%v	一年中的第几周，其中周一为每周的第一天， 和%x同时使用。	01，...，53
%W	星期。	Sunday，...，Saturday
%w	一周中的第几天。	0=Sunday，...，6=Saturday

%X	某一周所属的年份，其中周日为每周的第一天， 数字形式，4位数，和%V同时使用。	-
%x	某一周所属的年份，其中周一为每周的第一天， 数字形式，4位数，和%v同时使用。	-
%Y	年，用四位数字表示。	-
%y	年，用两位数字表示。	-
%%	文字字符，输出一个%。	-

注："-"表示无。

```
Oceanbase>SELECT DATE_FORMAT('2014-01-01', '%Y-%M-%d'),
DATE_FORMAT('2014-01-01', '%X-%V'),DATE_FORMAT('2014-01-01', '%U') \G
*************************** 1. row ***************************
DATE_FORMAT('2014-01-01', '%Y-%M-%d'): 2014-January-01
DATE_FORMAT('2014-01-01', '%X-%V'): 2013-52
DATE_FORMAT('2014-01-01', '%U'): 00
```

DATE_SUB(*date*, INTERVAL *expr unit*)

对时间进行算数计算。将*date*作为基数，对*expr*进行相减计算，*expr*允许为负，结果相当于做取反做加法。

参数说明参考DATE_ADD()。

```
Oceanbase>SELECT DATE_SUB('2014-01-10', INTERVAL 5 HOUR),
DATE_SUB('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND)\G;
*************************** 1. row ***************************
DATE_SUB('2014-01-10', INTERVAL 5 HOUR): 2014-01-09 19:00:00
DATE_SUB('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-09 18:54:54.999995
1 row in set (0.00 sec)
```

EXTRACT(*unit* FROM *date*)

提取*date*表达式中被*unit*指定的时间组成单元的值。

参数参考DATE_ADD()。

EXTRACT函数返回的结果为BIGINT类型。

对于MICROSECOND~YEAR这种单一单元（Single unit），将直接返回对应的int值。

*unit*为WEEK返回的是*date*表达式中指定的日期在该年所对应的周数， 而 OceanBase将一年的第一个星期日作为该年第一周的开始，如果某年的第一个星期日不是1月1日，那么该星期日之前的日期处于第0周。例如，2013年第一个星期日是1月6日，所以SELECT EXTRACT(WEEK FROM '2013-01-01')返回的结果为0，而SELECT EXTRACT(WEEK FROM '2013-01-06')返回的结果是1。

对于SECOND_MICROSECOND这种复合单元（combinative unit），OceanBase将各个值拼接在一起作为返回值。例如，SELECT EXTRACT(YEAR_MONTH FROM '2012-03-09')返回的结果将是"201203"。

```
Oceanbase>SELECT EXTRACT(WEEK FROM '2013-01-01'),
EXTRACT(WEEK FROM '2013-01-06'),
EXTRACT(YEAR_MONTH FROM '2012-03-09'),
EXTRACT(DAY FROM NOW())\G;
*************************** 1. row ***************************
EXTRACT(WEEK FROM '2013-01-01'): 0
EXTRACT(WEEK FROM '2013-01-06'): 1
EXTRACT(YEAR_MONTH FROM '2012-03-09'): 201203
EXTRACT(DAY FROM NOW()): 18
1 row in set (0.00 sec)
```

NOW([*fsp*])

和CURRENT_TIMESTAMP()函数同义。用于获取系统当前时间，精确到秒，格式为"YYYY-MM-DD HH:MI:SS"。

now()函数的括号里可以传入0-6数字参数，表示秒后面的小数点精度，默认now()相当于now(0) 。

> **注意：** now()函数的括号里可以传入 0-6 数字参数，表示秒后面的小数点精度，默认now()相当于now(0)，OB 1.0 版本先实现 now()带参数的功能。
>
> 为了兼容 MySQL，NOW()函数功能在OB 1.0里将与OB 0.5 中不兼容，OB 0.5 版本中的 NOW()相当于OB 1.0里的 NOW(6)

例子：

```
Oceanbase>SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2016-03-14 14:00:02 |
+---------------------+
1 row in set (0.01 sec)

Oceanbase>select now(0);
+---------------------+
| now(0)              |
+---------------------+
```

```
| 2016-03-14 14:00:35 |
+---------------------+
1 row in set (0.01 sec)

Oceanbase>select now(1);
+-----------------------+
| now(1)                |
+-----------------------+
| 2016-03-14 14:00:37.8 |
+-----------------------+
1 row in set (0.01 sec)

Oceanbase>select now(2);
+------------------------+
| now(2)                 |
+------------------------+
| 2016-03-14 14:00:39.67 |
+------------------------+
1 row in set (0.00 sec)

Oceanbase>select now(3);
+-------------------------+
| now(3)                  |
+-------------------------+
| 2016-03-14 14:00:41.671 |
+-------------------------+
1 row in set (0.01 sec)

Oceanbase>select now(7);
ERROR 1426 (42000): Too big precision 7 specified for column 'now'. Maximum is 6.
```

STR_TO_DATE(*str*,*format*)

STR_TO_DATE函数获取一个字符串*str*和一个格式字符串*format*。若格式字符串包含日期和时间部分，则STR_TO_DATE()返回一个DATETIME值，若该字符串只包含日期部分或时间部分，则返回一个DATE或TIME值。

*str*所包含的日期、时间或日期时间值应该在*format*指示的格式中被给定。若*str*包含一个非法日期、时间或日期时间值，则 STR_TO_DATE()返回NULL。同时，一个非法值会引起警告。

*format*的格式如下表所示。

格式	含义	返回格式
%b	月份的缩写名称。	Jan，...，Dec
%c	月份，数字形式。	1，...，12
%D	带有英语后缀的日期。	1st，2nd，...，31st
%d	日期，数字形式。	01，...，31
%e	日期，数字形式。	1，...，31

%f	微秒。	000000，...，999999
%H	小时。	00，...，23
%h	小时。	01，...，12
%I	小时。	01，...，12
%i	分钟。	00，...，59
%k	小时。	0，...，23
%l	小时。	01，...，12
%M	月份名称。	January，...，December
%m	月份，数字形式。	01，...，12
%p	上午或下午。	AM，PM
%r	12小时制时间。	hh:mm:ss AM/PM
%S	秒。	00，...，59
%s	秒。	00，...，59
%T	24小时制时间。	hh:mm:ss
%Y	年，用四位数字表示。	-

注："-"*表示无。*

```
Oceanbase>SELECT STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D %r');
+-----------------------------------------------------+
| STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D %r') |
+-----------------------------------------------------+
| 2014-01-01 17:05:05                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

TIME_TO_USEC(*date*)

将OceanBase的内部时间类型转换成一个微秒数计数。表示*date*所指的时刻距离"1970-01-01 00:00:00"的微秒数，这是一个UTC时间，不带时区信息。

date为被计算的时刻，且这个时刻附带时区信息，而时区信息是用户当前系统设置的时区信息。该参数为TIMESTAMP类型或者时间格式的字符串。

TIME_TO_USEC能够接受其它函数的调用结果作为参数，但是其的结果类型必须为TIMESTAMP或者时间格式的字符串。

该函数返回值为微秒计数，返回类型为INT。

```
Oceanbase>SELECT TIME_TO_USEC('2014-03-25'), TIME_TO_USEC(now());
  +----------------------------+--------------------+
| TIME_TO_USEC('2014-03-25') | TIME_TO_USEC(now()) |
  +----------------------------+--------------------+
|          1395676800000000 |    1395735415207794 |
  +----------------------------+--------------------+
1 row in set (0.00 sec)
```

USEC_TO_TIME(*usec*)

该函数为TIME_TO_USEC(*date*)的逆函数，表示"1970-01-01 00:00:00"增加*usec*后的时间，且附带了时区信息。例如在东八区调用该函数"USEC_TO_TIME(1)"，返回值为"1970-01-01 08:00:01"。

*usec*为一个微秒计数值。

返回值为TIMESTAMP类型。

```
Oceanbase>SELECT USEC_TO_TIME(1);
+----------------------------+
| USEC_TO_TIME(1)            |
+----------------------------+
| 1970-01-01 08:00:00.000001 |
+----------------------------+
1 row in set (0.00 sec)
```

UNIX_TIMESTAMP(),UNIX_TIMESTAMP(*date*)

若无参数调用，则返回一个Unix timestamp ('1970-01-01 00:00:00' GMT 之后的秒数) 作为无符号整数。若用*date*来调用UNIX_TIMESTAMP()，它会将参数值以'1970-01-01 00:00:00' GMT后的秒数的形式返回。*date*可以是一个DATE 字符串、一个 DATETIME字符串、一个 TIMESTAMP或一个当地时间的YYMMDD 或YYYMMDD格式的数字。

```
Oceanbase>SELECT UNIX_TIMESTAMP();
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|      1427176668 |
+------------------+
1 row in set (0.00 sec)

Oceanbase>SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00')
+---------------------------------------+
| UNIX_TIMESTAMP('1997-10-04 22:23:00') |
+---------------------------------------+
|                          875974980 |
+---------------------------------------+
1 row in set (0.00 sec)
```

DATEDIFF(*expr1,expr2*)

DATEDIFF()返回起始时间*expr1*和结束时间*expr2*之间的天数。*expr1*和*expr2*为日期或日期时间表达式。计算中只用到这些值的日期部分。

此函数必须跟两个参数，多于或少于两个参数，系统执行都将参数个数不正确错误。

```
Oceanbase>select datediff('2015-06-19','1994-12-17'), datediff('2015-06-19','1998-06-27 10:10:10'), datediff(now(), '2014-01-02')\G;
*************************** 1. row ***************************
       datediff('2015-06-19','1994-12-17'): 7489
datediff('2015-06-19','1998-06-27 10:10:10'): 6201
             datediff(now(), '2014-01-02'): 533
1 row in set (0.00 sec)
```

TIMEDIFF(*expr1,expr2*)

TIMEDIFF()返回起始时间*expr1*和结束时间*expr2*之间的时间。*expr1*和*expr2*为时间或日期时间表达式，两个的类型必须一样。

TIMEDIFF()返回结果限于时间值允许的范围。另外，你也可以使用TIMESTAMPDIFF()和UNIX_TIMESTAMP()函数，这两个函数的范围值为整数类型。

```
   Oceanbase>select timediff(now(), '2015-06-06 11:11:22'), timediff('2015-06-06 12:12:12', '2014-06-05 11:11:11')\G;
*************************** 1. row ***************************
             timediff(now(), '2015-06-06 11:11:22'): 838:59:59
timediff('2015-06-06 12:12:12', '2014-06-05 11:11:11'): 838:59:59
1 row in set (0.01 sec)
```

TIMESTAMPDIFF(*unit,datetime_expr1,datetime_expr2*)

返回日期或日期时间表达式*datetime_expr1*和*datetime_expr2*之间的整数差。其结果的单位由*unit*参数给出。*unit*的值为：MICROSECOND（mircoseconds）、SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER，以及YEAR.

```
Oceanbase>select timestampdiff(second,now(), '2011-01-01 11:11:11'),  timestampdiff(second, '2011-01-01 11:11:11', now())\G;
*************************** 1. row ***************************
 timestampdiff(second,now(), '2011-01-01 11:11:11'): -140843995
timestampdiff(second, '2011-01-01 11:11:11', now()): 140843995
1 row in set (0.00 sec)
```

PERIOD_DIFF(*p1,p2*)

返回周期*p1*和*p2*之间的月数。*p1*和*p2*的格式应该为 YYMM或YYYYMM。注意周期参数*p1*和*p2*不是日期值。

```
Oceanbase>select period_diff(20150702, 20790503), period_diff(150702, 790503);
+---------------------------------+----------------------------+
| period_diff(20150702, 20790503) | period_diff(150702, 790503) |
+---------------------------------+----------------------------+
|                        -76777 |                  -76777 |
+---------------------------------+----------------------------+
1 row in set (0.00 sec)
```

TO_DAYS(date)

给定一个日期 date,返回一个天数（从年份为 0 开始的天数）。

```
Oceanbase>SELECT TO_DAYS('2015-11-04'), TO_DAYS('20151104');
+----------------------+--------------------+
| TO_DAYS('2015-11-04') | TO_DAYS('20151104') |
+----------------------+--------------------+
|               736271 |             736271 |
+----------------------+--------------------+
1 row in set (0.01 sec)
```

TO_DAYS()不用于阳历出现（1582）前的值，原因是当日历改变时，遗失的日期不会被考虑在内。

日期中的二位数年份值转化为四位。例如，'2015-11-04'和'15-11-04'被视为同样的日期。

对于0日期'0000-00-00'在日期中被认为是不合法的。

FROM_DAYS(N)

给定一个天数N，返回一个DATE值。

```
Oceanbase>SELECT FROM_DAYS(736271), FROM_DAYS(700000);
+--------------------+--------------------+
| FROM_DAYS(736271) | FROM_DAYS(700000) |
+--------------------+--------------------+
| 2015-11-04         | 1916-07-15         |
+--------------------+--------------------+
1 row in set (0.00 sec)
```

使用 FROM_DAYS()处理古老日期时，务必谨慎，它不用于处理阳历日历出现(1582)前的日期。

FROM_UNIXTIME(unix_timestamp[,format])

```
FROM_UNIXTIME(unix_timestamp) , FROM_UNIXTIME(unix_timestamp,format)
```

返回'YYYY-MM-DD HH:MM:SS'或 YYYYMMDDHHMMSS 格式值的 unix_timestamp 参数表示，具体格式取决于该函数是否用在字符串中或是数字语境中。

若 format 已经给出，则结果的格式是根据 format 字符串而定。 format可以包含同 DATE_FORMAT() 函数输入项列表中相同的说明符。

格式	说明
%a	工作日的缩写名称 (Sun..Sat)
%b	月份的缩写名称 (Jan..Dec)
%c	月份，数字形式(0..12)
%D	带有英语后缀的该月日期 (0th, 1st, 2nd, 3rd, ...)
%d	该月日期, 数字形式 (00..31)
%e	该月日期, 数字形式(0..31)
%f	微秒 (000000..999999)
%H	小时(00..23)
%h	小时(01..12)
%I	小时 (01..12)
%i	分钟,数字形式 (00..59)
%j	一年中的天数 (001..366)
%k	小时 (0..23)
%l	小时 (1..12)
%M	月份名称 (January..December)
%m	月份, 数字形式 (00..12)
%p	上午（AM）或下午（ PM）
%r	时间，12 小时制 (小时 hh:分钟 mm:秒数 ss 后加 AM 或 PM)
%S	秒 (00..59)
%s	秒 (00..59)
%T	时间，24 小时制 (小时 hh:分钟 mm:秒数 ss)
%U	周 (00..53), 其中周日为每周的第一天
%u	周 (00..53), 其中周一为每周的第一天
%V	周 (01..53), 其中周日为每周的第一天；和 %X 同时使用
%v	周 (01..53), 其中周一为每周的第一天；和 %x 同时使用
%W	工作日名称 (周日..周六)
%w	一周中的每日 (0=周日..6=周六)
%X	该周的年份，其中周日为每周的第一天, 数字形式,4 位数;和%V 同时使用

%x	该周的年份，其中周一为每周的第一天, 数字形式,4 位数;和%v 同时使用
%Y	年份, 数字形式,4 位数
%y	年份, 数字形式 (2 位数)
%%	'%'文字字符

```
Oceanbase>SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
+-----------------------------------------------------------+
| FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x') |
+-----------------------------------------------------------+
| 2016 8th March 08:11:26 2016                              |
+-----------------------------------------------------------+
1 row in set (0.01 sec)
```

字符串函数

OceanBase 1.0字符集为utf8mb4，暂不支持其它字符集。utf8mb4字符集对应的collation支持utf8mb4_bin、utf8mb4_general_ci这两种，默认为utf8mb4_general_ci。

CONCAT(*str1,...,strN*)

把一个或多个字符串连接成一个字符串。左右参数都必须是字符串类型或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。

说明：str参数可以是数值类型，系统能隐式转换为字符串处理。

```
Oceanbase>select concat('test'), concat('test','OceanBase'), concat('test', 'OceanBase', '1.0'),
concat('test','OceanBase','1.0', NULL)\G;
*************************** 1. row ***************************
                    concat('test'): test
          concat('test','OceanBase'): testOceanBase
   concat('test', 'OceanBase', '1.0'): testOceanBase1.0
concat('test','OceanBase','1.0', NULL): NULL
1 row in set (0.00 sec)
```

SUBSTRING

```
SUBSTRING(str,pos)

SUBSTRING(str FROM pos)

SUBSTRING(str,pos,len)

SUBSTRING(str FROM pos FOR len)
```

和SUBSTR同语义。

SUBSTR

```
SUBSTR(str,pos,len)

SUBSTR(str,pos)

SUBSTR(str FROM pos)

SUBSTR (str FROM pos FOR len)
```

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*必须是字符串，*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。

```
Oceanbase>SELECT SUBSTR('abcdefg',3), SUBSTR('abcdefg',3,2), SUBSTR('abcdefg',-3), SUBSTR('abcdefg',3,-2),
SUBSTR('abcdefg' from -4 for 2)\G;
*************************** 1. row ***************************
          SUBSTR('abcdefg',3): cdefg
        SUBSTR('abcdefg',3,2): cd
         SUBSTR('abcdefg',-3): efg
       SUBSTR('abcdefg',3,-2):
SUBSTR('abcdefg' from -4 for 2): de
1 row in set (0.00 sec)
```

TRIM

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

删除字符串所有前缀和（或）后缀。

*remstr*和*str*必须为字符串或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。

```
Oceanbase>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 (0.01 sec)
```

LENGTH(*str*)

返回字符串的长度，单位为字节。参数必须是字符串类型或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。

*str*参数为数值类型时，系统能隐式转换为字符串类型。

```
Oceanbase>SELECT LENGTH('text');
+----------------+
| LENGTH('text') |
+----------------+
|              4 |
+----------------+
1 row in set (0.00 sec)

Oceanbase>select length(-1.23);
+---------------+
| length(-1.23) |
+---------------+
|             5 |
+---------------+
1 row in set (0.00 sec)

Oceanbase>select length(1233e);
ERROR 1054 (42S22): Unknown column '1233e' in 'field list'
```

UPPER(*str*)

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

*str*参数为数值类型时，能隐式转换为字符串类型。

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。

```
Oceanbase>SELECT UPPER('OceanBase您好！');
+-----------------------------+
| UPPER('OceanBase您好！')    |
+-----------------------------+
| OCEANBASE您好！             |
+-----------------------------+
1 row in set (0.00 sec)

Oceanbase>select upper(e);
ERROR 1054 (42S22): Unknown column 'e' in 'field list'

Oceanbase>select upper(1.235.);
ERROR 1064 (42000): You have an error in your SQL syntax;
```

LOWER(*str*)

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

*str*参数为数值类型时，能隐式转换为字符串类型。

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，LOWER可以很好的兼容。

```
Oceanbase>SELECT LOWER('OceanBase您好！');
+-----------------------------+
| LOWER('OceanBase您好！')    |
+-----------------------------+
| oceanbase您好！             |
+-----------------------------+
1 row in set (0.00 sec)

Oceanbase>select lower(1.23) ;
+-------------+
| lower(1.23) |
+-------------+
| 1.23        |
+-------------+
1 row in set (0.00 sec)

Oceanbase>select lower(1.23h);
ERROR 1583 (42000): Incorrect parameters in the call to native function 'lower'

Oceanbase>select lower(1.23e);
ERROR 1582 (42000): Incorrect parameter count in the call to native function 'lower'
```

HEX(*str*)

将字符串转化为十六进制数显示。输入为NULL时，输出也为NULL。

当*str*是数值时，输出整数的十六进制表示；

当*str*输入是字符串的时候，返回值为*str*的十六进制字符串表示，其中每个*str*里的每个字符被转化为两个十六进

制数字。

```
Oceanbase>SELECT HEX(255);
       -> FF
Oceanbase>SELECT HEX('abc');
       -> 616263

Oceanbase>SELECT HEX('OceanBase'),
   ->     HEX(123),
   ->     HEX(0x0123);
+--------------------+----------+-------------+
| HEX('OceanBase')   | HEX(123) | HEX(0x0123) |
+--------------------+----------+-------------+
| 4F6365616E42617365 | 7B       | 0123        |
+--------------------+----------+-------------+
1 row in set (0.01 sec)

Oceanbase>select hex(0x012);
+------------+
| hex(0x012) |
+------------+
| 0012       |
+------------+
1 row in set (0.00 sec)
```

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*必须是字符串类型或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。

```
Oceanbase>SELECT HEX('OceanBase'),
->     UNHEX('4f6365616e42617365'),
->     UNHEX(HEX('OceanBase')),
->     UNHEX(NULL)\G;
*************************** 1. row ***************************
          HEX('OceanBase'): 4F6365616E42617365
UNHEX('4f6365616e42617365'): OceanBase
    UNHEX(HEX('OceanBase')): OceanBase
              UNHEX(NULL): NULL
1 row in set (0.00 sec)

Oceanbase>select unhex(abc);
ERROR 1054 (42S22): Unknown column 'abc' in 'field list';
```

INT2IP(*int_value*)

注：INT2IP为OceanBase特有函数。

将一个整数转换成IP地址。

- 输入数据类型必须为INT。若输入为NULL，则输出为NULL。若输入的数字大于MAX_INT32或小于0则输出为NULL。

```
Oceanbase>SELECT INT2IP(16777216),
    -> HEX(16777216),
    -> INT2IP(1);
+------------------+---------------+-----------+
| INT2IP(16777216) | HEX(16777216) | INT2IP(1) |
+------------------+---------------+-----------+
| 1.0.0.0          | 1000000       | 0.0.0.1   |
+------------------+---------------+-----------+
1 row in set (0.00 sec)
```

IP2INT('*ip_addr*')

注：IP2INT为OceanBase特有函数。

将字符串表示的IP地址转换成整数内码表示。

输入数据类型必须为字符串类型。若输入为NULL，则输出为NULL。若输入的IP地址不是一个正确的IP地址(包含非数字字符，每一个ip segment的数值大小超过256等），则输出为NULL。

仅支持ipv4地址，暂不支持ipv6地址。

```
Oceanbase>SELECT IP2INT('0.0.0.1'),
    -> HEX(IP2INT('0.0.0.1')),
    -> HEX(IP2INT('1.0.0.0')),
    -> IP2INT('1.0.0.257')\G;
*************************** 1. row ***************************
     IP2INT('0.0.0.1'): 1
HEX(IP2INT('0.0.0.1')): 1
HEX(IP2INT('1.0.0.0')): 1000000
   IP2INT('1.0.0.257'): NULL
1 row in set (0.01 sec)
```

[NOT] LIKE *str2* [ESCAPE *str3*]

字符串通配符匹配。左右参数都必须是字符串类型或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"_"：

"%"表示匹配任何长度的任何字符，且匹配的字符可以不存在。

"_"表示只匹配单个字符，且匹配的字符必须存在。

如果你需要查找"a_c"，而不是"abc"时，可以使用OceanBase的转义字符"\\"，即可以表示为"a_c"。

ESCAPE用于定义转义符，即表示如果*str2*中包含*str3*，那么在匹配时，*str3*后的字符为普通字符处理，例如：LIKE 'abc%' ESCAPE 'c'，此时"c"为转义符，而"%"为普通字符，不再作为转义字符，本语句匹配的字符串为"ab%"。

```
Oceanbase>SELECT 'ab%' LIKE 'abc%' ESCAPE 'c';
+------------------------------+
| 'ab%' LIKE 'abc%' ESCAPE 'c' |
+------------------------------+
|                            1 |
+------------------------------+
1 row in set (0.00 sec)
```

用ESCAPE定义转义符时，长度为1的字符串；且 % 和 _ 不能用来作为转义字符。

```
Oceanbase>select 'a_c' like 'a\\_c';
+---------------------+
| 'a_c' like 'a\\_c' |
+---------------------+
|                   1 |
+---------------------+
1 row in set (0.00 sec)

Oceanbase>select 'abc_' like 'abcdd_' escape 'dd';
ERROR 1064 (42000): Incorrect arguments to ESCAPE
```

expr [NOT] REGEXP | RLIKE *pat*

执行字符串表达式*expr*和模式*pat*的模式匹配。若*expr*匹配 *pat*，则返回 1; 否则返回0。若 *expr* 或 *pat* 任意一个为 NULL, 则结果为 NULL。 RLIKE 是REGEXP的同义词。

*expr*和*pat*参数都必须为字符串或NULL，支持隐式转换成字符串类型。数据类型不匹配的，报错。PATTERN必须为合法的正则表达式，否则报错。

```
Oceanbase>select 1234 regexp 1;
+----------------+
| 1234 regexp 1 |
+----------------+
|              1 |
+----------------+
1 row in set (0.00 sec)

Oceanbase>select 'hello'  rlike 'h%';
```

```
+---------------------+
| 'hello'  rlike 'h%' |
+---------------------+
|                   0 |
+---------------------+
1 row in set (0.00 sec)

Oceanbase>select 1234 regexp ^y;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual thatcorresponds to your MySQL server version for the right syntax to use near '^y' at line 1

Oceanbase>select yunzhi regexp '^y';
ERROR 1054 (42S22): Unknown column 'yunzhi' in 'field list'

Oceanbase>select 'hello' not  rlike '*h*';
ERROR 1139 (42000): Got error 'empty (sub)expression' from regexp
```

REPEAT(*str, count*)

返回一个由重复*count*次数的字符串*str*组成的字符串 。 若 *count* <= 0,则返回一个空字符串。

若*str* 或 *count* 为 NULL，则返回 NULL。

*str*为数值类型时，系统隐式转换为字符串类型。

*count*支持隐式转换成数值类型，如果转换失败，则相当于*count*为0。

```
Oceanbase>select repeat('1',-1),  repeat(null,null),repeat('test',4);
+----------------+-------------------+------------------+
| repeat('1',-1) | repeat(null,null) | repeat('test',4) |
+----------------+-------------------+------------------+
|                | NULL              | testtesttesttest |
+----------------+-------------------+------------------+
1 row in set (0.00 sec)

Oceanbase>select repeat(11111,'2');
+--------------------+
| repeat(11111,'2') |
+--------------------+
| 1111111111         |
+--------------------+
1 row in set (0.00 sec)
```

SUBSTRING_INDEX(*str, delim, count*)

在定界符 *delim* 以及*count* 出现前，从字符串*str*返回字符串。若*count*为正值,则返回最终定界符(从左边开始)左边的一切内容。若*count*为负值，则返回定界符（从右边开始）右边的一切内容。任意一个参数为NULL，返回NULL；若*str*或*delim*为空字符串，则返回空串；若*count*=0,返回空串。

*str, delim, count*参数支持数值与字符串的隐式转换。

```
Oceanbase>select substring_index('abcdabc', 'abc', 0), substring_index('abcdabc', 'abc', 1),
substring_index('abcdabc', 'abc', 2), substring_index('abcdabc', 'abc', 3), substring_index('abcdabc', 'abc', -1),
substring_index('abcdabc', 'abc', -2), substring_index('abcdabc', 'abc', -3)\G;
*************************** 1. row ***************************
 substring_index('abcdabc', 'abc', 0):
 substring_index('abcdabc', 'abc', 1):
 substring_index('abcdabc', 'abc', 2): abcd
 substring_index('abcdabc', 'abc', 3): abcdabc
substring_index('abcdabc', 'abc', -1):
substring_index('abcdabc', 'abc', -2): dabc
substring_index('abcdabc', 'abc', -3): abcdabc
1 row in set (0.00 sec)
```

LOCATE(*substr,str*)，LOCATE(*substr,str,pos*)

第一个语法返回字符串 *str*中子字符串*substr*的第一个出现位置。第二个语法返回字符串*str*中子字符串*substr*的第一个出现位置, 起始位置在*pos*。如若*substr* 不在*str*中，则返回值为0。

```
Oceanbase>SELECT LOCATE('bar', 'foobarbar');
       -> 4
Oceanbase>SELECT LOCATE('xbar', 'foobar');
       -> 0
Oceanbase>SELECT LOCATE('bar', 'foobarbar',5);
       -> 7
```

INSTR(*str,substr*)

返回字符串*str*中子字符串的第一个出现位置。这和LOCATE()的双参数形式相同，除非参数的顺序被颠倒。

```
Oceanbase>SELECT INSTR('foobarbar', 'bar');
       -> 4
Oceanbase>SELECT INSTR('xbar', 'foobar');
       -> 0
```

REPLACE(*str, from_str, to_str*)

返回字符串*str*以及所有被字符*to_str*替代的字符串*from_str*。

```
Oceanbase>SELECT REPLACE('abc.efg.gpg.nowdew.abc.dabc.e', 'abc.', 'www');
+-----------------------------------------------------------+
| REPLACE('abc.efg.gpg.nowdew.abc.dabc.e', 'abc.', 'www') |
+-----------------------------------------------------------+
| wwwefg.gpg.nowdew.wwwdwwwe                                |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

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

返回参数*str*在*str1,str2,str3,...*列表中的索引位置（从1开始的位置）。在找不到*str*的情况下，返回值为0。

如果所有的对于FIELD()的参数均为字符串，则所有参数均按照字符串进行比较。如果所有的参数均为数字，则按照数字进行比较。否则，参数均按照double类型进行比较。

如果*str*为NULL，则返回值为0，原因是NULL不能同任何值进行同等比较。FILED()是ELT()的补数。

```
Oceanbase>select field('abc','abc1','abc2','abc','abc4','abc'), field(NULL, 'null1', NULL);
+-----------------------------------------------+---------------------------+
| field('abc','abc1','abc2','abc','abc4','abc') | field(NULL, 'null1', NULL) |
+-----------------------------------------------+---------------------------+
|                                             3 |                         0 |
+-----------------------------------------------+---------------------------+
1 row in set (0.00 sec)
```

ELT(N, *str1, str2, str3,...*)

若N=1，则返回值为*str1*, 若N=2，则返回值为*str2*，以此类推。若N小于1或大于参数的数目，则返回值为NULL。ELT()是FIELD()的补数。

```
Oceanbase>select elt(3, 'abc1', 'abc2', 'abc', 'abc4', 'abc'), elt(0, 'null1', NULL);
+-----------------------------------------------+-----------------------+
| elt(3, 'abc1', 'abc2', 'abc', 'abc4', 'abc') | elt(0, 'null1', NULL) |
+-----------------------------------------------+-----------------------+
| abc                                           | NULL                  |
+-----------------------------------------------+-----------------------+
1 row in set (0.00 sec)
```

INSERT (str1,pos,len,str2)

返回字符串str1, 字符串中起始于pos位置，长度为len的子字符串将被str2取代。如果 pos 超过字符串长度，则返回值为原始字符串。 假如 len 的长度大于其它字符串的长度，则从位置pos开始替换。若任何一个参数为null，则返回值为NULL。这个函数支持多字节字元。

- str1和str2必须是字符串，pos和len必须是整数。任意参数为 NULL，结果总为 NULL。
- str1和str2中的文字符被当做字节流看待。
- pos 值为负数或者大于 str1 长度时，返回 str1；
- 当len小于0，或大于 str1 的长度时，返回结果为str1从开头到pos位置的串，和str2的组合串。

```
Oceanbase>select insert('Quadratic',-2,100,'What'), insert('Quadratic',7,3,'What'),
    -> insert('Quadratic',-1,3,'What'), insert('Quadratic',10,3,'What'), insert('Quadratic',5,-1,''),
    -> insert('Quadratic',7,-1,'What')\G;
*************************** 1. row ***************************
insert('Quadratic',-2,100,'What'): Quadratic
  insert('Quadratic',7,3,'What'): QuadraWhat
```

```
  insert('Quadratic',-1,3,'What'): Quadratic
  insert('Quadratic',10,3,'What'): Quadratic
     insert('Quadratic',5,-1,''): Quad
  insert('Quadratic',7,-1,'What'): QuadraWhat
1 row in set (0.01 sec)
```

转换函数

CAST(*expr* AS *type*)

将某种数据类型的表达式显式转换为另一种数据类型。

将*expr*字段值转换为*type*数据类型。数据类型参见数据类型章节。

```
Oceanbase>SELECT CAST(123 AS BOOL);
+-------------------+
| CAST(123 AS bool) |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)
```

参数说明：

expr:表示任何有效的SQL表达式。

AS: 用于分隔两个参数，在AS之前的是要处理的数据，在AS之后是要转换的数据类型。 *type*: 表示目标系统所提供的数据类型。可以是以下值其中的一个：

CHAR[(N)]（CHAR[N]会使 cast 使用该参数的不多于N 个字符）

DATE

DATETIME

DECIMAL

SIGNED [INTEGER]

TIME

UNSIGNED [INTEGER]

在使用CAST函数进行数据类型转换时，在下列情况下能够被接受：

两个表达式的数据类型完全相同；

两个表达式可隐式转换；

必须显式转换数据类型。

如果试图进行不可能的转换，OceanBase将显示一条错误信息。

如果转换时没有指定数据类型的长度，则使用 OceanBase系统内部最大长度。如varchar是262,143字节，number是65个bit位的浮动精度。

支持带符号和无符号的64比特值的运算。若你正在使用数字操作符 (如 +) 而其中一个操作数为无符号整数，则结果为无符号。可使用SIGNED 和UNSIGNED cast操作符来覆盖它。将运算分别派给带符号或无符号64比特整数。

假如任意一个操作数为一个浮点值，则结果为一个浮点值。

```
Oceanbase>select cast(1-2 as unsigned), cast(cast(1-2 as unsigned) as signed);
+----------------------+--------------------------------------+
| cast(1-2 as unsigned) | cast(cast(1-2 as unsigned) as signed) |
+----------------------+--------------------------------------+
|  18446744073709551615 |                                   -1 |
+----------------------+--------------------------------------+
1 row in set (0.00 sec)

Oceanbase>SELECT CAST(1 AS UNSIGNED) - 2.0;
+---------------------------+
| CAST(1 AS UNSIGNED) - 2.0 |
+---------------------------+
|                      -1.0 |
+---------------------------+
1 row in set (0.00 sec)

Oceanbase>select cast(0 as date);
+-----------------+
| cast(0 as date) |
+-----------------+
| 0000-00-00      |
+-----------------+
1 row in set (0.00 sec)
```

聚合函数

聚合函数对一组值执行计算并返回单一的值。聚合函数忽略空值。聚合函数经常与SELECT语句的GROUP BY子句一同使用。

所有聚合函数都具有确定性。任何时候用一组给定的输入值调用它们时，都返回相同的值。

在OceanBase的聚合函数中，Value表达式只能出现一个。例如：不支持COUNT(c1, c2)，仅支持COUNT(c1）。

AVG(([DISTINCT] *expr*)

返回指定组中的平均值，空值被忽略。DISTINCT选项可用于返回expr的不同值的平均值。若找不到匹配的行，则AVG()返回NULL。

```
Oceanbase>select * from oceanbasetest;
+----+------+------+
| id | ip   | ip2  |
+----+------+------+
|  1 |    4 | NULL |
|  3 |    3 | NULL |
|  4 |    3 | NULL |
+----+------+------+
3 rows in set (0.01 sec)

Oceanbase>select avg(ip2), avg(ip), avg(distinct(ip)) from oceanbasetest;
+----------+---------+-------------------+
| avg(ip2) | avg(ip) | avg(distinct(ip)) |
+----------+---------+-------------------+
|     NULL |  3.3333 |            3.5000 |
+----------+---------+-------------------+
1 row in set (0.00 sec)

Oceanbase>select avg(distinct(ip)),avg(ip),avg(ip2) from oceanbasetest;
+-------------------+---------+----------+
| avg(distinct(ip)) | avg(ip) | avg(ip2) |
+-------------------+---------+----------+
|            3.5000 |  3.3333 |     NULL |
+-------------------+---------+----------+
1 row in set (0.00 sec)
```

COUNT([DISTINCT] *expr*)

COUNT([DISTINCT] *expr*)返回SELECT语句检索到的行中非NULL值的数目。若找不到匹配的行，则COUNT()返回0。DISTINCT选项可用于返回expr的不同值的数目。

COUNT(*)的稍微不同之处在于，它返回检索行的数目，不论其是否包含NULL值。

```
Oceanbase>select * from oceanbasetest;
+----+------+------+
| id | ip   | ip2  |
+----+------+------+
|  1 |    4 | NULL |
|  3 |    3 | NULL |
|  4 |    3 | NULL |
+----+------+------+
```

```
3 rows in set (0.00 sec)

Oceanbase>select count(ip2), count(ip), count(distinct(ip)), count(*) from oceanbasetest;
+------------+-----------+--------------------+----------+
| count(ip2) | count(ip) | count(distinct(ip)) | count(*) |
+------------+-----------+--------------------+----------+
|          0 |         3 |                  2 |        3 |
+------------+-----------+--------------------+----------+
1 row in set (0.00 sec)
```

MAX([DISTINCT] *expr*)

返回指定数据中的最大值。

MAX()的取值可以是一个字符串参数；在这些情况下，它们返回最大字符串值。DISTINCT关键字可以被用来查找*expr*的不同值的最大值，这产生的结果与省略DISTINCT 的结果相同。

假设表a有三行数据：id=1，num=10；id=2，num=20；id=3，num=30。

```
Oceanbase>SELECT MAX(num) FROM a;
+-----------------+
| MAX(num)        |
+-----------------+
|             30 |
+-----------------+
1 row in set (0.00 sec)
```

MIN([DISTINCT] *expr*)

返回指定数据中的最小值。

MIN()的取值可以是一个字符串参数；在这些情况下，它们返回最小字符串值。DISTINCT关键字可以被用来查找expr 的不同值的最小值，然而，这产生的结果与省略DISTINCT 的结果相同。

假设表a有三行数据：id=1，num=10；id=2，num=20；id=3，num=30。

```
Oceanbase>SELECT MIN(num) FROM a;
+----------------+
| MIN(num)       |
+----------------+
|             10 |
+----------------+
1 row in set (0.00 sec)
```

SUM([DISTINCT] *expr*)

返回*expr* 的总数。若返回集合中无任何行，则SUM()返回NULL。DISTINCT关键字可用于求得*expr*不同值的总和。

若找不到匹配的行，则SUM()返回NULL。

```
Oceanbase>select * from oceanbasetest;
+------+------+------+
| id   | ip   | ip2  |
+------+------+------+
|    1 |    4 | NULL |
|    3 |    3 | NULL |
|    4 |    3 | NULL |
+------+------+------+
3 rows in set (0.00 sec)

Oceanbase>select sum(ip2),sum(ip),sum(distinct(ip)) from oceanbasetest;
+----------+---------+-------------------+
| sum(ip2) | sum(ip) | sum(distinct(ip)) |
+----------+---------+-------------------+
|     NULL |      10 |                 7 |
+----------+---------+-------------------+
1 row in set (0.00 sec)
```

GROUP_CONCAT([DISTINCT] *expr*)

该函数返回带有来自一个组的连接的非NULL值的字符串结果。

语法

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
[ORDER BY {unsigned_integer | col_name | expr}
[ASC | DESC] [,col_name ...]]
[SEPARATOR str_val])
```

举例

```
Oceanbase>select * from book;     //表book（书编号，书名，出版社）+--------+--------------------------------+-------
----------------------+
| bookid | bookname                       | publishname                |
+--------+--------------------------------+----------------------------+
|      1 | git help                       | alibaba group publisher    |
|      2 | MySQL性能优化                  | 浙江大学图文出版社         |
|      3 | JAVA编程指南                   | 机械工业出版社             |
|      3 | JAVA编程指南                   | 机械工业出版社             |
|      4 | 大规模分布式存储系统           | 机械工业出版社             |
+--------+--------------------------------+----------------------------+
5 rows in set (0.00 sec)

//查找书名信息
Oceanbase>select group_concat(bookname) from book group by bookname;
+----------------------------------+
| group_concat(bookname)           |
+----------------------------------+
| git help                         |
```

```
| JAVA编程指南,JAVA编程指南        |
| MySQL性能优化                   |
| 大规模分布式存储系统             |
+----------------------------------+
4 rows in set (0.00 sec)

//查找书名信息，书名唯一
Oceanbase>select group_concat(distinct(bookname)) from book group by bookname;
+---------------------------------+
| group_concat(distinct(bookname)) |
+---------------------------------+
| git help                        |
| JAVA编程指南                    |
| MySQL性能优化                   |
| 大规模分布式存储系统             |
+---------------------------------+
4 rows in set (0.01 sec)

//查找书名和出版社信息，以书名分组，出版社信息降序排序显示
Oceanbase>select bookname, group_concat(publishname order by publishname desc separator  ';' ) from book
group by bookname;
+--------------------------------+-------------------------------------------------------------------------------+
| bookname                       | group_concat(publishname order by publishname desc separator  ';' ) |
+--------------------------------+-------------------------------------------------------------------------------+
| git help                       | alibaba group publisher                                                       |
| JAVA编程指南                   | 机械工业出版社;机械工业出版社                                                 |
| MySQL性能优化                  | 浙江大学图文出版社                                                            |
| 大规模分布式存储系统           | 机械工业出版社                                                                |
+--------------------------------+-------------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

数学函数

数学函数能够对数字表达式进行数学计算。

ROUND(X), ROUND(X,D)

返回一个数值，四舍五入到指定的长度或精度。

返回参数X, 其值接近于最近似的整数。在有两个参数的情况下，返回X，其值保留到小数点后D位，而第D位的保留方式为四舍五入。若要接保留X值小数点左边的D位，可将D设为负值。

返回值的类型同第一个自变量相同(假设它是一个整数、双精度数或小数)。这意味着对于一个整数参数，结果也是一个整数(无小数部分)。

- 对于准确值数字,ROUND()使用"四舍五入" 或"舍入成最接近的数" 的规则：对于一个分数部分为 .5或大于 .5的值，正数则上舍入到邻近的整数值，负数则下舍入临近的整数值。(换言之, 其舍入的方向是数轴上远离零的方向）。对于一个分数部分小于.5 的值，正数则下舍入下一个整数值，负数则下舍入邻近的整数值，而正数则上舍入邻近的整数值。

- 对于近似值数字， ROUND()遵循银行家规则"四舍--大于五入--五取最接近的偶数"的规则： 一个带有任何小数部分的值会被舍入成最接近的偶数整数。

```
Oceanbase>select round(2.15,2);
+---------------+
| round(2.15,2) |
+---------------+
|          2.15 |
+---------------+
1 row in set (0.00 sec)

Oceanbase>select round(2555e-2,1);
+------------------+
| round(2555e-2,1) |
+------------------+
|             25.6 |
+------------------+
1 row in set (0.01 sec)

Oceanbase>select round(25e-1), round(25.3e-1),round(35e-1);
+--------------+----------------+--------------+
| round(25e-1) | round(25.3e-1) | round(35e-1) |
+--------------+----------------+--------------+
|            3 |              3 |            4 |
+--------------+----------------+--------------+
1 row in set (0.00 sec)
```

CEIL(*expr*)

返回大于或者等于指定表达式的最小整数。

还支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)；

如果输入NULL，返回值为NULL。

如果输入纯数字的字符串，支持自动转换成数字类型。

返回值会被转化为一个BIGINT。

```
Oceanbase>select ceil(1.2), ceil(-1.2), ceil(1+1.5), ceil(1=1),ceil(1<1),ceil(null);
+-----------+-----------+-------------+-----------+-----------+------------+
| ceil(1.2) | ceil(-1.2) | ceil(1+1.5) | ceil(1=1) | ceil(1<1) | ceil(null) |
+-----------+-----------+-------------+-----------+-----------+------------+
|         2 |        -1 |           3 |         1 |         0 |       NULL |
+-----------+-----------+-------------+-----------+-----------+------------+
1 row in set (0.00 sec)

Oceanbase>select ceil(name);
ERROR 1166 (42703): Unkown column name 'name'

Oceanbase>select ceil('2');
+------------+
| ceil('2') |
```

```
+-----------+
|         2 |
+-----------+
1 row in set (0.00 sec)
```

FLOOR(*expr*)

和CEIL(*expr*)函数功能类似，返回小于或者等于指定表达式的最大整数。

还支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)；

如果输入NULL，返回值为NULL。

如果输入纯数字的字符串，支持自动转换成数字类型。

返回值会被转化为一个BIGINT。

```
Oceanbase>select floor(1.2), floor(-1.2), floor(1+1.5), floor(1=1),floor(1<1),floor(null);
+------------+-------------+-------------+------------+------------+-------------+
| floor(1.2) | floor(-1.2) | floor(1+1.5) | floor(1=1) | floor(1<1) | floor(null) |
+------------+-------------+-------------+------------+------------+-------------+
|          1 |          -2 |           2 |          1 |          0 |        NULL |
+------------+-------------+-------------+------------+------------+-------------+
1 row in set (0.00 sec)

Oceanbase>select floor(name);
ERROR 1166 (42703): Unkown column name 'name'

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

ABS(*expr*)

绝对值函数，求表达式绝对值，函数返回值类型与数值表达式的数据类型相同。

还支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)；

如果输入NULL，返回值为NULL。

如果输入纯数字的字符串，支持自动转换成数字类型。

返回值会被转化为一个BIGINT。

```
Oceanbase>select abs(5),abs(-5.777),abs(0),abs(1/2),abs(1-5);
+--------+-------------+--------+----------+----------+
| abs(5) | abs(-5.777) | abs(0) | abs(1/2) | abs(1-5) |
+--------+-------------+--------+----------+----------+
```

```
|     5 |      5.777 |      0 |   0.5000 |       4 |
+--------+------------+--------+----------+----------+
1 row in set (0.00 sec)
```

NEG(*expr*)

求补函数，对操作数执行求补运算：用零减去操作数，然后结果返回操作数。

支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)，再对结果求补。

```
Oceanbase>select neg(1),neg(1+1),neg(2*3),neg(1=1),neg(5<1);
+--------+----------+----------+----------+----------+
| neg(1) | neg(1+1) | neg(2*3) | neg(1=1) | neg(5<1) |
+--------+----------+----------+----------+----------+
|    -1 |       -2 |       -6 |       -1 |        0 |
+--------+----------+----------+----------+----------+
1 row in set (0.01 sec)
```

SIGN(X)

SIGN(X)返回参数作为-1、 0或1的符号，该符号取决于X的值为负、零或正。

支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)；

如果输入NULL，返回值为NULL。

支持浮点数、十六进制数。

```
Oceanbase>SELECT SIGN(-32), SIGN(0), SIGN(234);
+-----------+---------+-----------+
| SIGN(-32) | SIGN(0) | SIGN(234) |
+-----------+---------+-----------+
|        -1 |       0 |         1 |
+-----------+---------+-----------+
1 row in set (0.01 sec)

Oceanbase>select sign(null),sign(false),sign(0x01);
+------------+-------------+------------+
| sign(null) | sign(false) | sign(0x01) |
+------------+-------------+------------+
|       NULL |           0 |          1 |
+------------+-------------+------------+
1 row in set (0.00 sec)
```

CONV(N, *from_base, to_base*)

不同数基间转换数字。返回值为一个字符串，由*from_base*基转化为*to_base*基。输入参数N可以是一个整数或字符串。最小基数为2，而最大基数则为36。如果*to_base*是一个负数，则N被看作一个带符号数。否则，N被

看作无符号数。*from_base*如果是负数，则被当作整数处理，符号被忽略。N参数仅支持int类型和字符串类型输入；*from_base*和*to_base*参数仅支持十进制int类型输入，且取值范围为[-36,-2]U[2,36]。

非法输入将导致报错，其中非法输入包括以下情况：

- *from_base*或者*to_base*不是一个合法的十进制int类型输入；
- *from_base*或者*to_base*超出[-36,-2]U[2,36]的取值范围；
- N不是一个合法的数字表示，例如取值超出0～9，a～z，A～Z的字符范围；
- N超出了*from_base*基的取值范围，例如*from_base*为2，而N取值为3；
- N的取值超出了int64的最大表示范围，即[-9223372036854775807, 9223372036854775807]。

```
Oceanbase>select conv(9223372036854775807,10,2);
+----------------------------------------------------------------+
| conv(9223372036854775807,10,2)                                 |
+----------------------------------------------------------------+
| 1111111111111111111111111111111111111111111111111111111111111111 |
+----------------------------------------------------------------+
1 row in set (0.00 sec)

Oceanbase>select conv('-acc',21,-7);
+--------------------+
| conv('-acc',21,-7) |
+--------------------+
| -16425             |
+--------------------+
1 row in set (0.00 sec)
```

MOD(*N*,*M*)

取余函数。MOD(*N*，*M*)， *N* % *M*， *N* MOD *M* 三种形式是等效的。

MOD()对于带有小数部分的数值也起作用，它返回除法运算后的精确余数。

N,M 中任何一个参数为 NULL，返回值都为 NULL。M 为 0 是，也返回 NULL。

```
Oceanbase>select mod(29,19), 29 mod 19, 29 % 19;
+------------+-----------+---------+
| mod(29,19) | 29 mod 19 | 29 % 19 |
+------------+-----------+---------+
|         10 |        10 |      10 |
+------------+-----------+---------+
1 row in set (0.00 sec)
```

POW(*X*,*Y*)

返回X的Y次方。

X与Y中任何一个参数为NULL，返回值都为NULL。

```
Oceanbase>select pow(4,2), pow(4,-2), pow(1,null);
+----------+-----------+-------------+
| pow(4,2) | pow(4,-2) | pow(1,null) |
+----------+-----------+-------------+
|       16 |    0.0625 |        NULL |
+----------+-----------+-------------+
1 row in set (0.00 sec)
```

数据定义语言

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。

OceanBase支持的DDL主要有CREATE DATABASE, ALTER DATABASE, DROP DATABASE, CREATE TABLE，DROP TABLE，ALTER TABLE，CREATE INDEX，DROP INDEX，CREATE VIEW, DROP VIEW，ALTER VIEW，TRUNCATE TABLE等。

CREATE DATABASE语句

格式

```
CREATE DATABASE [IF NOT EXISTS] dbname
    [create_specification_list];

create_specification_list:
    create_specification [, create_specification...]

create_specification:
    [DEFAULT] {CHARACTER SET | CHARSET} [=] charsetname
    | [DEFAULT] COLLATE [=] collationname
    | REPLICA_NUM [=] num
    | PRIMARY_ZONE [=] zone
```

CREATE DATABASE用于创建数据库，并可以指定数据库的默认属性（如数据库默认字符集,校验规则等）。

- REPLICA_NUM指定副本数,副本数必须与zone的数量相等，默认情况下无需设置；
- PRIMARY_ZONE指定主zone的名称。

举例

```
Oceanbase>create database test1 default charset utf8mb4 collate utf8mb4_general_ci replica_num 3 primary_zone
ET15SQA_1;
Query OK, 0 rows affected (0.04 sec)

Oceanbase>show create database test1;
+----------+-----------------------------------------------------------------------------------------------------+
| Database | Create Database                                                                                     |
+----------+-----------------------------------------------------------------------------------------------------+
| test1    | CREATE DATABASE `test1` DEFAULT CHARACTER SET = utf8mb4 REPLICA_NUM = 3 PRIMARY_ZONE =
ET15SQA_1 |
+----------+-----------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

ALTER DATABASE语句

格式

```
ALTER DATABASE [dbname]
    alter_specification_list;

alter_specification_list:
     alter_specification [, alter_specification...]

alter_specification:
     [DEFAULT] {CHARACTER SET| CHARSET} [=] charsetname
     | [DEFAULT] COLLATE [=] collationname
     | REPLICA_NUM [=] num
     | PRIMARY_ZONE [=] zone
```

修改database的属性，如字符集、校对规则、REPLICA_NUM指定副本数，PRIMARY_ZONE指定主集群等。

数据库名称可以忽略，此时，语句对应于当前默认数据库。

举例

```
Oceanbase>alter database test1 default collate utf8mb4_bin;
Query OK, 0 rows affected (0.03 sec)
```

DROP DATABASE语句

格式

```
DROP DATABASE [IF EXISTS] dbname;
```

DROP DATABASE用于取消数据库中的所用表格和取消数据库。

IF EXISTS用于防止当数据库不存在时发生错误。

举例

```
Oceanbase>drop database test3;
ERROR 1008 (HY000): Can't drop database 'test3';database doesn't exist

Oceanbase>drop database if exists test3;
Query OK, 0 rows affected (0.01 sec)
```

CREATE TABLE语句

该语句用于在OceanBase数据库中创建新表。

格式

```
CREATE TABLE [IF NOT EXIST] tblname
(create_definition,...)
[table_options]
[partition_options];

CREATE TABLE [IF NOT EXISTS] tblname
  LIKE oldtblname;

create_definition:
   colname column_definition
  | PRIMARY KEY (index_col_name[, index_col_name...]) [index_option]...
  | {INDEX|KEY} [indexname] (index_col_name,...) [index_option]...
  | UNIQUE {INDEX|KEY} [indexname] (index_col_name,...) [index_option]...

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT defaultvalue]
     [AUTO_INCREMENT] [UNIQUE [KEY]] | [[PRIMARY] KEY]
     [COMMENT 'string']

data_type:
  TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME[(fsp)]
```

```
  | TIMESTAMP[(fsp)]
  | DATETIME[(fsp)]
  | YEAR
  | CHAR[(length)]
      [CHARACTER SET charsetname] [COLLATE collationname]
  | VARCHAR(length)
      [CHARACTER SET charsetname] [COLLATE collationname]
  | BINARY[(length)]
  | VARBINARY(length)
index_col_name:
    colname [(length)] [ASC | DESC] ( OceanBase1.0暂不支持前缀索引 )

index_option:
    GLOBAL [LOCAL]
    |COMMENT 'string'
    |COMPRESSION [=] '{NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}'
    |BLOCK_SIZE [=] size
    |STORING(columname_list)

table_options:
    table_option [[,] table_option]...

table_option:
    [DEFAULT] { CHARACTER SET | CHARSET } [=] value
    | [DEFAULT] COLLATE [=] collationname
    | COMMENT [=] 'comment string'
    | COMPRESSION [=] '{NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}'
    | EXPIRE_INFO [=] (expr)
    | REPLICA_NUM [=] num
    | TABLE_ID [=] id
    | BLOCK_SIZE [=] size
    | USE_BLOOM_FILTER [=] {True| False}
    | PROGRESSIVE_MERGE_NUM [=] num
    | TABLEGROUP [=] 'tablegroupname'
    | PRIMARY_ZONE [=] 'zone'
    | AUTO_INCREMENT [=] num

partition_options:
    PARTITION BY
        HASH(expr)
        |KEY(column_list)
        [PARTITIONS num]
        [partition_definition ...]

partition_definition:
    COMMENT [=] 'string' (暂不支持)
```

OceanBase内部数据以b树为索引，按照Primary Key排序。OceanBase 0.4.2及以前版本不允许用户创建只有主键列的表；OceanBase 0.5.0及以上版本支持创建只有主键列的表。OceanBase 1.0可以不指定主键，系统会自动生成。

CREATE TABLE支持UNIQUE约束；暂不支持创建临时表，暂不支持CHECK约束；不支持创建表的同时从其他表导入功能。

- 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"请参见数据类型章节。
- NOT NULL，DEFAULT，AUTO_INCREMENT用于列的完整性约束。
- "table_option"内容参考下表，各子句间用","隔开。
- "index_option"中，可以指定GLOBAL，LOCAL关键字，表述全局或局部索引。默认是GLOBAL index。创建带有Parition的表时index一定要加LOCAL关键字。如果没有加，系统将报错。

参数	含义	举例
CHARACTER SET	指定该表所有字符串的编码，用于对外提供元数据信息。目前仅支持UTF8MB4。	CHARACTER SET = 'utf8mb4'
COMMENT	添加注释信息。	COMMENT='create by Bruce'
COMPRESSION	存储数据时使用的压缩方法名，目前提供的方法有以下几种： · none（默认值，表示不作压缩） · LZ4_1.0 · LZO_1.0 · SNAPPY_1.0 · ZLIB_1.0	COMPRESSION = 'NONE'
EXPIRE_INFO	在MemTable中的动态数据和SSTable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 c1 < date_add(merging_frozen_time(),interval -1 HOUR); merging_frozen_time()表示当前最新的合并时间，此函数只适用于过期条件。
TABLE_ID	指定表的ID。如果指定的table_id小于50000，需要打开RootServer的配置项开关"enable_sys_table_ddl"，普通用户不建议指定。	TABLE_ID =4000
BLOCK_SIZE	设置Partition的微块大小。	默认为16K。
USE_BLOOM_FILTER	对本表读取数据时，是否使用Bloom Filter。 · False：默认值，不使用。 · True：使用。	USE_BLOOM_FILTER = False
PROGRESSIVE_MERGE_NUM	设置渐近合并步数。PROGRESSIVE_MERGE_NUM现在在限制是1~64。	默认值为1。PROGRESSIVE_MERGE_NUM = 5
TABLEGROUP	表所属表格组。	
REPLICA_NUM	这个表的partition总副本数，默认值为3。	REPLICA_NUM = 3
ZONE_LIST	集群列表	
PRIMARY_ZONE	主集群。	

AUTO_INCREMENT	自增字段初始值	AUTO_INCREMENT = 5

AUTO_INCREMENT基本说明

在OceanBase中可以把表中的数据类型为整数型的某个字段定义为自增属性AUTO_INCREMENT，数据库会自动递增方式生产唯一值。

允许给TINYINT、SMALLINT、MEDIUMINT、INT、INTEGER、BIGINT各种整数类型的字段指定AUTO_INCREMENT，其它数据类型的字段不能指定AUTO_INCREMENT。

一个表只允许有一个属性为AUTO_INCREMENT的字段。

```
Oceanbase>create table t1 (id int auto_increment,name varchar(20) primary key);
Query OK, 0 rows affected (0.01 sec)
```

举例

例1:

执行以下命令，创建数据库表。

```
ceanbase>CREATE TABLE test (c1 int primary key, c2 varchar(10)) REPLICA_NUM = 3, COMPRESSION = 'NONE';
Query OK, 0 rows affected (0.40 sec)
```

执行命令查看表信息，如：

```
Oceanbase>describe test;
+-------+-------------+------+------+---------+-------+
| Field | Type        | Null | Key  | Default | Extra |
+-------+-------------+------+------+---------+-------+
| c1    | int(11)     | NO   | PRI  | NULL    |       |
| c2    | varchar(10) | YES  |      | NULL    |       |
+-------+-------------+------+------+---------+-------+
2 rows in set (0.01 sec)
```

例2:

```
Oceanbase>CREATE TABLE example_2(custid INT,
thedate TIMESTAMP, cost INT, PRIMARY KEY(custid, thedate))
EXPIRE_INFO = (thedate < date_sub(merging_frozen_time(), INTERVAL 2 DAY)),  USE_BLOOM_FILTER = FALSE;
Query OK, 0 rows affected (0.11 sec)
```

(thedate < date_sub(merging_frozen_time(), INTERVAL 2 DAY))，表示删除（过期）thedate字段值为2天前的数据，删除数据动作在数据合并时候真正执行。其中 merging_frozen_time()表示当前最新的合并时间。（merging_frozen_time()只能放在过期条件中使用）

ALTER TABLE语句

该语句用于修改已存在的表的结构，比如：修改表及表属性、新增列、修改列及属性、删除列等。

格式

```
ALTER TABLE tblname
    alter_specification [, alter_specification]... ;

alter_specification:
   ADD [COLUMN] colname column_definition
  | ADD [COLUMN] (colname column_definition,...)
  | ADD [UNIQUE]{INDEX|KEY} [indexname] (index_col_name,...) [index_options]
  | ADD PRIMARY KEY (index_col_name,...) [index_options](暂不支持)
  | ALTER [COLUMN] colname {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] oldcolname newcolname column_definition
  | MODIFY [COLUMN] colname column_definition
  | DROP [COLUMN] colname
  | DROP PRIMARY KEY (暂不支持)
  | DROP {INDEX | KEY} indexname
  | RENAME [TO] newtblname
  | ORDER BY colname (暂不支持)
  | CONVERT TO CHARACTER SET charsetname [COLLATE collationname] (暂不支持)
  | [DEFAULT] CHARACTER SET charsetname [COLLATE collationname] (暂不支持)
  | table_options
  | partition_options
  | DROP TABLEGROUP
  | AUTO_INCREMENT [=] num

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT defaultvalue]
     [AUTO_INCREMENT] [UNIQUE [KEY]](暂不支持) [[PRIMARY] KEY] (暂不支持)
     [COMMENT 'string']

table_options:
    [SET] table_option [[,] table_option]...

table_option:
[DEFAULT] {CHARACTER SET | CHARSET} [=] charsetname
| [DEFAULT] COLLATE [=] collationname
| COMMENT [=] 'string'
| COMPRESSION [=] '{NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 |ZLIB_1.0}'
| EXPIRE_INFO [=] (expr)
| REPLICA_NUM [=] num
| TABLE_ID [=] id
| BLOCK_SIZE [=] size
| USE_BLOOM_FILTER [=] {True | False}
| PROGRESSIVE_MERGE_NUM [=] num
| TABLEGROUP [=] tablegroupname
| PRIMARY_ZONE [=] zone
| AUTO_INCREMENT [=] num
partition_options:
    PARTITION BY
        HASH(expr)
```

```
        | KEY(column_list)
        [PARTITIONS num]
        [partition_definition ...]

partition_definition:
    COMMENT [=] 'commenttext' (暂不支持)
```

增加列

```
ALTER TABLE tblname
   ADD [COLUMN] colname data_type
   [NOT NULL | NULL]
   [DEFAULT defaultvalue];
```

- data_type请参见数据类型章节

修改列属性

```
ALTER TABLE tblname
  ALTER [COLUMN] colname
  [SET DEFAULT literal| DROP DEFAULT];
```

修改列类型

```
ALTER TABLE tblname
   MODIFY colname column_definition;
```

删除列

```
ALTER TABLE tblname
   DROP [COLUMN] colname;
```

- 不允许删除主键列或者包含索引的列。

表重命名

```
ALTER TABLE tblname
  RENAME TO newtblname;
```

列重命名

```
ALTER TABLE tblname
```

```
CHANGE [COLUMN] oldcolname newcolname column_definition;
```

注意：

1. 对于varchar类型的列，只允许将varchar的长度变大，不允许减小。
2. 在 OceanBase 0.5 里列重命名是用 ALTER TABLE tblname RENAME [COLUMN] oldcolname TO newcolname，OceanBase 1.0的实现与MySQL兼容。

例1：

```
#把表t2的字段d改名为c，并同时修改了字段类型
ALTER TABLE t2 CHANGE COLUMN d c CHAR(10);
```

设置过期数据删除

```
ALTER TABLE tblname
   SET EXPIRE_INFO [ = ] expr;
```

例2：

```
CREATE TABLE example_1(custid INT
   , thedate TIMESTAMP
   , cost INT
   , PRIMARY KEY(custid, thedate)
   ) EXPIRE_INFO = (thedate < date_sub(merging_frozen_time(), INTERVAL 2 DAY)),
   USE_BLOOM_FILTER = FALSE;
```

(thedate < date_sub(merging_frozen_time(), INTERVAL 2 DAY))，表示删除（过期）thedate字段值为冻结时间2天前的数据，删除数据动作在数据合并时候真正执行。

修改过期条件

```
# 删除（过期）thedate字段值为1天前的数据。
alter table example_1 set EXPIRE_INFO = (thedate < date_sub(merging_frozen_time(), INTERVAL 1 DAY))
```

设置Partition表BLOCK大小

```
ALTER TABLE tblname
   SET BLOCK_SIZE [=] blocksize;
```

设置该表的副本数

```
ALTER TABLE tblname
   SET REPLICA_NUM [=] num;
```

这里是指表的副本总数多少。

设置该表的压缩方式

```
ALTER TABLE tblname
   SET COMPRESSION [=] '{NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}';
```

设置是否使用BloomFilter

```
ALTER TABLE tblname
   SET USE_BLOOM_FILTER [=] {True | Flase};
```

设置注释信息

```
ALTER TABLE tblname
   SET COMMENT [=] 'commentstring';
```

设置渐进合并步数

```
ALTER TABLE tblname
  SET PROGRESSIVE_MERGE_NUM [=] num;
```

此功能是设置渐近合并步数，PROGRESSIVE_MERGE_NUM现在在限制是1~64。

设置表的ZONE属性

```
ALTER TABLE tblname
   zone_specification...;

zone_specification:
   PRIMARY_ZONE [=] zone
```

修改AUTO_INCREMENT字段的起始值

可在create talbe时指定AUTO_INCREMENT的起始值，也可用alter table tbl_name AUTO_INCREMENT=n命令来重设自增的起始值，但是如果设置的n比AUTO_INCREMENT字段的当前值小的话，执行的sql不会报错，但是不会生效！

```
Oceanbase>create table t1(id int auto_increment primary key,name varchar(10)) auto_increment=100;
Query OK, 0 rows affected (0.10 sec)

Oceanbase>alter table t1 auto_increment=50;
Query OK, 0 rows affected (0.04 sec)
#用alter table语句把t1表的auto_increment起始值调整为50,由于50小于100，修改无效

Oceanbase>show create table t1;
+-------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| t1    | CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) AUTO_INCREMENT = 100 DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384
USE_BLOOM_FILTER = FALSE |
+-------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+

Oceanbase>alter table t1 auto_increment=110;
Query OK, 0 rows affected (0.03 sec)

Oceanbase>show create table t1;
+-------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| t1    | CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) AUTO_INCREMENT = 110 DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384
USE_BLOOM_FILTER = FALSE |
+-------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

DROP TABLE语句

该语句用于删除OceanBase数据库中的表。

格式

```
DROP TABLE [IF EXISTS] tbl_list;

tbl_list:
```

```
    tblname [, tblname ...]
```

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;
```

TRUNCATE TABLE语句

格式

```
TRUNCATE [TABLE] tblname;
```

该语句用于完全清空指定表，但是保留表结构，包括表中定义的Partition信息。从逻辑上说，该语句与用于删除所有行的DELETE FROM语句相同。执行TRUNCATE语句，必须具有表的删除和创建权限。它属于DDL语句。

TRUNCATE TABLE语句与DELETE FROM语句有以下不同：

- 删减操作会取消并重新创建表，这比一行一行的删除行要快很多。
- TRUNCATE TABLE语句执行结果显示影响行数始终显示为0行。
- 使用TRUNCATE TABLE语句，表管理程序不记得最后被使用的AUTO_INCREMENT值，但是会从头开始计数。
- TRUNCATE语句不能在进行事务处理和表锁定的过程中进行，如果使用，将会报错。
- 只要表定义文件是合法的，则可以使用TRUNCATE TABLE把表重新创建为一个空表，即使数据或索引文件已经被破坏。

示例

创建分区表：

```
Oceanbase>create table tp(a int, b int) partition by hash(a) partitions 7;
Query OK, 0 rows affected (0.37 sec)

Oceanbase>show create table tp;
+-------+-----------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| tp    | CREATE TABLE `tp` (
  `a` int(11) DEFAULT NULL,
```

```
  `b` int(11) DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE partition by
hash(a) partitions 7 |
+-------+-----------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

插入若干条数据：

```
Oceanbase>insert into tp values(1,2);
Query OK, 1 row affected (0.02 sec)

Oceanbase>insert into tp values(2,2);
Query OK, 1 row affected (0.04 sec)

Oceanbase>insert into tp values(3,2);
Query OK, 1 row affected (0.02 sec)

Oceanbase>insert into tp values(5,2);
Query OK, 1 row affected (0.01 sec)

Oceanbase>insert into tp values(6,2);
Query OK, 1 row affected (0.01 sec)


Oceanbase>select * from tp;
+------+------+
| a    | b    |
+------+------+
|    1 |    2 |
|    2 |    2 |
|    3 |    2 |
|    5 |    2 |
|    6 |    2 |
+------+------+
5 rows in set (0.02 sec)
```

truncate 分区表：

```
Oceanbase>truncate tp;
Query OK, 0 rows affected (0.46 sec)

Oceanbase>select * from tp;
Empty set (0.04 sec)

Oceanbase>show create table tp;
+-------+-----------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| tp    | CREATE TABLE `tp` (
  `a` int(11) DEFAULT NULL,
```

```
  `b` int(11) DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE partition by
hash(a) partitions 7 |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

RENAME TABLE语句

格式

```
RENAME TABLE tblname TO newtblname
    [, tb1name2 TO newtblname ...];
```

RENAME TABLE语句用于对一个或多个表进行重命名 。

重命名操作自动进行，重命名正在进行时，其他线程不能读取任何表。例如，如果您有一个原有的表oldtable,您可以创建另一个具有相同结果的空表newtable,然后用此空表替换原有的表：

```
CREATE TABLE newtable(...);
RENAME TABLE oldtable TO backuptable, newtable TO oldtable;
```

如果此语句用于对多个表进行重命名，则重命名操作从左到右进行。

如果你想要交换两个表的名称，可以这样做(假设不存在表tmptable)

```
RENAME TABLE oldtable TO tmptable,
newtable TO oldtable,
temptable TO newtable
```

同一个租户下，可以对数据库中的表进行重命名，把表从一个数据库中移动到另一个数据库中：

```
RENANME TABLE currentdb.tblname TO otherdb.tblname
```

当您执行RENAME时，您不能有被锁定的表，也不能有处于活性状态的事务。还必须拥有原表的ALTER和DROP权限，以及新表的 CREATE和INSERT权限。

RENAME TABLE也可以用于视图，只要确保是在同一个数据库中。

CREATE INDEX语句

索引是创建在表上的，对数据库表中一列或多列的值进行排序的一种结构。其作用主要在于提高查询的速度，降低数据库系统的性能开销。

OceanBase中，新创建的索引需要等待一次每日合并后，才生效。

格式

```
CREATE [UNIQUE] INDEX indexname
   ON tblname (index_col_name,...)
   [index_type][index_options];

index_type:
   USING BTREE

index_options:
   index_option [index_option...]

index_option:
   GLOBAL [LOCAL]
   | COMMENT 'string'
   | COMPRESSION [=] '{NONE | LZ4_1.0 | LZO_1.0 | SNAPPY1.0 | ZLIB_1.0}'
   | BLOCK_SIZE [=] size
   | STORING(columname_list)
   | index_using_algorithm

columname_list:
   colname [, colname...]

index_col_name:
    colname [ASC | DESC]
```

index_col_name中，每个列名后都支持ASC（升序）和DESC（降序）。默认就为升序。

本语句建立索引的排列方式为：首先以index_col_name中第一个列的值排序；该列值相同的记录，按下一列名的值排序；以此类推。

执行"SHOW INDEX FROM tblname"可以查看创建的索引。

"index_option"中，可以指定GLOBAL，LOCAL关键字，表述全局或局部索引。默认是GLOBAL index. 创建带有Parition的表时index一定要加LOCAL关键字。如果没有加，系统将报错。多个index option以空格分隔。

使用可选字段STORING，表示索引表中冗余存储某些列，以提高系统查询系统。（*为OceanBase数据库特有*）

说明: OceanBase内部数据以b树为索引。

举例

```
Oceanbase>CREATE TABLE test (c1 int primary key, c2 varchar(10));
Query OK, 0 rows affected (0.10 sec)

Oceanbase>create index test_index on test(c1, c2 desc) using hash;
Query OK, 0 rows affected (0.04 sec)

Oceanbase>show index from test;
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+---
---------+-------------+---------------+
| Table | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null |
Index_type | Comment     | Index_comment |
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+---
---------+-------------+---------------+
| test  |          0 | PRIMARY    |            1 | c1          | A         |        NULL | NULL     | NULL   |      | BTREE      | available   |
|
| test  |          1 | test_index |            1 | c1          | A         |        NULL | NULL     | NULL   |      | BTREE      | unavailable |
|
| test  |          1 | test_index |            2 | c2          | A         |        NULL | NULL     | NULL   | YES  | BTREE      | unavailable |
|
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+---
---------+-------------+---------------+
3 rows in set (0.00 sec)
```

DROP INDEX语句

当索引过多时，维护开销增大，因此，需要删除不必要的索引。

删除索引的时候需要等待一段时间才能完全删除。

格式

```
DROP INDEX indexname
   ON tblname;
```

举例

```
DROP INDEX test_index ON test;
```

CREATE VIEW语句

格式

```
CREATE [OR REPLACE] VIEW viewname
 [(column_list)] AS select_stmt;
```

创建视图语句，如果指定了OR REPLACE子句，该语句能够替换已有的视图。

select_stmt是一种SELECT语句。它给出了视图的定义。该语句可以从基表或其他视图进行选择。

视图必须具有唯一的列名，不得有重复，就像基表那样。默认情况下，由SELECT语句检索的列名将用作视图列名。要想为视图列定义明确的名称，可使用可选的column_list子句，列出由逗号隔开的ID。column_list中的名称数目必须等于SELECT语句检索的列数。

SELECT语句检索的列可以是对表列的简单引用。也可以是使用函数、常量值、操作符等的表达式。

视图在数据库中实际上并不是以表的形式存在。每次使用时它们就会派生。视图是作为在CREATE VIEW语句中指定的SELECT语句的结果而派生出来的。

OceanBase 1.0只支持不可更新视图。

示例

创建基本表和视图：

```
Oceanbase>create table test(a int, b int);
createQuery OK, 0 rows affected (1.23 sec)

Oceanbase>create view test_view(va, vb) as select a, a+b from test;
Query OK, 0 rows affected (0.04 sec)

Oceanbase>show create view test_view;
+-----------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| View      | Create View                                                                                                                                                                 |
character_set_client | collation_connection |
+-----------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| test_view | CREATE VIEW `test_view` AS select `ocp_test`.`test`.`a` AS `va`,(`ocp_test`.`test`.`a` + `ocp_test`.`test`.`b`)
AS `vb` from `ocp_test`.`test` | utf8mb4              | utf8mb4_general_ci   |
+-----------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
1 row in set (0.01 sec)
```

向基本表中插入数据，并查看视图数据：

```
Oceanbase>insert into test values(1,2),(2,3),(3,4);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

Oceanbase>select * from test_view;
+------+------+
| va   | vb   |
+------+------+
|    1 |    3 |
|    2 |    5 |
|    3 |    7 |
+------+------+
3 rows in set (0.01 sec)
```

不支持更新视图：

```
Oceanbase>update test_view set va = 5;
ERROR 4016 (HY000): view table just supports select
```

ALTER VIEW语句

格式

```
ALTER VIEW viewname [(column_list)]
    [(column_list)] AS select_statement
```

该语句用于更改已有视图的定义。其语法与CREATE VIEW类似。

> **注意** OceanBase目前暂不支持alter view语句，即目前无法修改视图。由于视图本质上是逻辑表，如果有修改视图的需求，可以先将原先的视图删除，再重新创建即可。

DROP VIEW语句

格式

```
DROP VIEW [IF EXISTS]
   viewname [, viewname ...];
```

DROP VIEW能够删除1个或多个视图。必须在每个视图上有DROP权限。

使用IF EXISTS关键字来防止因不存在的视图而出错。

数据操作语言

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。

OceanBase支持的DML主要有INSERT、REPLACE、SELECT、UPDATE和DELETE。

PARTITION表的DML限制:

因OceanBase 1.0是按partition进行数据打散的，从性能上考虑，SELECT、UPDATE和DELETE都的WHERE条件里都需要带上"partition(p0, p1,...)"，不带"partition(partition_list)"的SQL需要所有分区。

不带"partition(partition_list)"的SELECT、UPDATE和DELETE语法并不会报错，但会引起全表扫描。

DML中REPLACE,INSERT,UPDATE,DELETE语句只支持同一个Partition的操作，跨Partition的操作直接报错；

SELECT语句跨Partition的情况:

跨Partition Group的读请求，只支持跨分区弱一致读取；

同一个Partition Group的读请求，支持跨分区强一致读取。需要注意的是，OB后台数据迁移的过程中，同一个Partition Group的多个Partition有很低的概率在较短的时间内（毫秒级别）不在一台机器，这种情况下，强一致读取会失败。

INSERT语句

该语句用于添加一个或多个记录到表。

格式

```
INSERT [INTO] tblname
    [(colname,...)]
    {VALUES|VALUE} ({expr | DEFAULT},...)
    [ ON DUPLICATE KEY UPDATE
     colname=expr
    [, colname=expr] ... ] ;
```

或者

```
INSERT [INTO] tblname
       [(colname,...)]
      {VALUES|VALUE} (colvalues,...)
      [ON DUPLICATE KEY UPDATE
        colname=expr
       [, colname=expr] ... ] ;
```

[(colname,...)]用于指定插入数据的列。

同时插入多列时，用","隔开。

支持ON DUPLICATE KEY UPDATE

Insert语句后面不支持set操作

INSERT...ON DUPLICATE KEY UPDATE... 语句执行，affect row的计算：

在没有设置client_capabilities 中CLIENT_FOUND_ROWS的情况下：

作为新行插入的话，affected_row= 1；

存在冲突行的情况下，如果更新前后数据相同的话， affected_row = 0； 否则affectd_row = 2;

如果设置了CLIENT_FOUND_ROWS：

作为新行插入的话，affected_row=1；

如果更新前后数据相同的话，affectd_row=1；

如果更新前后数据不相同的话，affected_row=2;

CLIENT_FOUND_ROWS的影响在于：不设置CLIENT_FOUND_ROWS的情况下，计算affected_row的值，只计算实际更新了行数，而设置了CLIENT_FOUND_ROWS的话，会把所有touched行数（满足冲突条件的行）都记上，而不管其是否发生了真正的数据修改。

示例

```
Oceanbase>show create table test;
+-------+--------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+--------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| test  | CREATE TABLE `test` (
 `a` int(11) DEFAULT NULL,
 `b` int(11) DEFAULT NULL,
 UNIQUE KEY `a_uniq` (`a`) BLOCK_SIZE 16384
```

```
) DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE |
+-------+-------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------+
1 row in set (0.01 sec)

Oceanbase>insert into test values(1,2);
Query OK, 1 row affected (0.01 sec)

Oceanbase>insert into test values(1,3);
ERROR 1062 (23000): Duplicate entry '1' for key 'a_uniq'

Oceanbase>insert into test values(1,3) on duplicate key update a = a+1;
Query OK, 2 rows affected (0.01 sec)

Oceanbase>select * from test;
+------+------+
| a    | b    |
+------+------+
|    2 |    2 |
+------+------+
1 row in set (0.01 sec)
```

注意: 同时插入多个值时，如果这些值在不同的分区，会报错。

```
Oceanbase>show create table tp;
+-------+-------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| Table | Create Table
|
+-------+-------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| tp    | CREATE TABLE `tp` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 REPLICA_NUM = 3 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE partition by
hash(a) partitions 7 |
+-------+-------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
1 row in set (0.01 sec)

Oceanbase>insert into tp values(1,2),(2,3),(3,4);
ERROR 1235 (0A000): Insert data cross partition not supported
```

UPDATE语句

该语句用于修改表中的字段值。

格式

```
UPDATE tblname
```

```
    SET colname1={expr1|DEFAULT}
    [, colname2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY order_list]
    [LIMIT row_count];

order_list:
    colname [ASC|DESC] [, colname [ASC|DESC]...]
```

REPLACE语句

REPLACE的运行与INSERT相似，只有一点除外，如果表中的一个旧记录与一个用于PRIMARY KEY或一个UNIQUE索引的新记录具有相同的值，则在新记录被插入之前，旧记录将被删除。

为了能够使用REPLACE，必须同时拥有表的INSERT和DELETE权限。

格式

```
REPLACE [INTO] tblname
   [(colname,...)]
   {VALUES|VALUE} ({expr | DEFAULT},...);
```

[(colname,...)]用于指定插入数据的列。

同时替换多列时，用","隔开。

关于REPLACE语句执行，affect row的值计算：

如果是当新行插入的话，affected_row=1;

如果replace前后新值和旧值不一致的话，affectd_row=2，（这里假定产生冲突的行为1行，如果为多行的话，那么affectd_row = 冲突行数+1）

如果replace前后新值和旧值一致的话，分两种情况：

- 如果产生冲突的是表上最后一个唯一索引&&没有外键约束&&没有ON DELETE TRIGGER，affected_row=1；
- 其他情况，affectd_row = 2。

DELETE语句

该语句用于删除表中符合条件的行。

格式

```
DELETE FROM tblname
      [WHERE where_condition]
      [ORDER BY order_list]
      [LIMIT row_count];
```

SELECT语句

该语句用于查询表中的内容。

基本查询

格式

```
SELECT
  [ALL | DISTINCT]
   selectexpr [AS othername] [, selectexpr ...]
   [FROM table_references]
   [WHERE where_conditions]
   [GROUP BY group_by_list]
  [HAVING search_confitions]
  [ORDER BY order_list]
  [LIMIT {[offset,] rowcount | rowcount OFFSET offset}]
   [FOR UPDATE];
```

SELECT子句说明如下所示。

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_expr	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_references	指名了从哪个表或哪些表中读取数据。（支持多表查询）
WHERE where_conditions	可选项，WHERE子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_confitions	HAVING子句与WHERE子句类似，但是HAVING子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。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)。 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。
FOR UPDATE	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。

JOIN句法

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。两个表连接后，可以使用ON指定条件进行筛选。

OceanBase1.0版本里JOIN可以支持USING子句，JOIN的连接条件中必须至少有一个等值连接条件。

内连接：结果中只包含两个表中同时满足条件的行。

左连接：结果中包含位于关键字LEFT [OUTER] JOIN左侧的表中的所有行，以及该关键字右侧的表中满足条件的行。

右连接：结果中包含位与关键字[RIGHT] [OUTER] JOIN右侧的表中的所有行，以及该关键字左侧的表中满足条件的行。

全连接：结果中包含两个表中的所有行。

集合操作

OceanBase中的集合操作主要包括UNION、EXCEPT和INTERSECT。

UNION句法

UNION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撷取资料。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。

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。

INTERSECT句法

INTERSECT用于查询在两个集合中都存在的数据。

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,] rowcount | rowcount OFFSET offset}];
```

SELECT...FOR UPDATE句法

格式

```
SELECT ... FOR UPDATE [WAIT n| NOWAIT];
```

其中:

WAIT子句指定等待其他用户释放锁的秒数，防止无限期的等待。

NOWAIT不等待行锁释放。

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。

注意，目前OceanBase实现有如下限制：必须是单表查询。

例如：

```
SELECT * FROM a FOR UPDATE;
```

IN和OR

OceanBase支持逻辑运算"IN"和"OR"。

数据库管理语言

用户及权限管理

数据库用户权限管理包括新建用户、删除用户、修改密码、修改用户名、锁定用户、用户授权和撤销授权等。

OceanBase 1.0中用户分为两类：系统租户下的用户，一般租户下的用户。创建用户时，如果Session当前租户为系统租户，则新建的用户为系统租户下的用户，反之为一般租户下的用户。

用户名称在租户内是唯一的，不同租户下的用户可以同名。用户名@租户名在系统全局唯一。为区别系统租户和一般租户下的用户，系统租户下的用户名称使用特定前缀。系统租户和普通租户都有一个内置用户root，系统租户的root为系统管理员和普通租户的root为租户管理员，购买了某个普通租户的客户得到普通租户root和密码，进行本租户范围的管理工作。

一般租户下的用户只能拥有该租户下对象的访问权限，权限和MySQL兼容；系统租户下的用户可以被授予跨租户的对象访问权限。当前系统租户下的用户不允许访问一般租户下的用户表数据。 用户在登录OceanBase系统时需指定唯一的租户名。对于系统租户下的用户，在登录后，可以使用CHANGE EFFECTIVE TENANT tenantname语句来切换当前访问的租户；对于一般租户下的用户，不能切换到其他租户。

新建用户

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

格式

```
CREATE USER user_specification_list;
```

```
user_specification_list:
    user_specification [, user_specification]...;

user_specification:
    user IDENTIFIED BY 'authstring'
    user IDENTIFIED BY PASSWORD 'hashstring'
```

- 必须拥有全局的CREATE USER权限，才可以使用CREATE USER命令。
- 新建用户后，"mysql.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。
- user IDENTIFIED BY 'authstring'此处密码为明文，存入"mysql.user"表后，服务器端会变为密文存储。
- user IDENTIFIED BY PASSWORD 'hashstring'此处密码为密文。
- 同时创建多个用户时，用","隔开。

举例

```
Oceanbase>CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.07 sec)

Oceanbase>select user from user;
+-----------+
| user      |
+-----------+
| root      |
| test      |
| sqluser01 |
| sqluser02 |
+-----------+
4 rows in set (0.01 sec)
```

删除用户

DROP USER语句用于删除一个或多个OceanBase用户。

格式

```
DROP USER username [, username...];
```

必须拥有全局的CREATE USER权限，才可以使用DROP USER命令。

不能对"mysql.user"表进行DELETE方式进行权限管理。

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"sqluser02"用户。

```
DROP USER 'sqluser02';
```

修改密码

修改OceanBase登录用户的密码。

格式

```
SET PASSWORD [FOR user] = password_option;

password_option: {
    PASSWORD('authstring')
    |'hashstring'}
```

或者

```
ALTER USER username 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。必须拥有全局CREATE USER权限，才可以修改指定用户的密码。

举例

执行以下命令将"sqluser01"的密码修改为"abc123"。

```
ALTER USER sqluser01 IDENTIFIED BY 'abc123';
```

使用SET PASSWORD修改密码如下：

```
Oceanbase>set password for test = password('abc123');
Query OK, 0 rows affected (0.03 sec)

# 不指定password函数，会报错如下：
Oceanbase>set password for test = 'abc123';
ERROR 1827 (42000): The password hash doesn't have the expected format. Check if the correct password
algorithm is being used with the PASSWORD() function.
```

修改用户名

用于修改OceanBase登录用户的用户名。

格式

```
RENAME USER
   'oldusername' TO 'newusername'
   [,'oldusername' TO 'newusername'...];
```

必须拥有全局CREATE USER权限，才可以使用本命令。

同时修改多个用户名时，用","隔开。

修改前后，新旧用户权限保持一致。

用户名长度限制：用户名占用字节小于等于16。

举例

```
Oceanbase>select user from user;
+---------+
| user    |
+---------+
| root    |
| testall |
+---------+
2 rows in set (0.00 sec)
```

修改用户名

```
Oceanbase>rename user testall to test;
Query OK, 0 rows affected (0.03 sec)

Oceanbase>select user from user;
+------+
| user |
+------+
| root |
| test |
+------+
2 rows in set (0.00 sec)
```

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

```
ALTER USER user [lock_option]

lock_option:{
    ACCOUNT LOCK
    | ACCOUNT UNLOCK}
```

必须拥有全局UPDATE USER权限，才可以执行本命令。

举例

锁定用户

```
Oceanbase>alter user test account lock;
Query OK, 0 rows affected (0.04 sec)
```

解锁用户

```
Oceanbase>alter user test account unlock;
Query OK, 0 rows affected (0.02 sec)
```

用户授权

GRANT语句用于系统管理员授予User某些权限。

格式

```
GRANT priv_type
ON priv_level
   TO user_specification [, user_specification]...
    [WITH with_option ...]


priv_level:
     *
    | *.*
    | db_name.*
    | db_name.tbl_name
    | tbl_name

user_specification:
    user [IDENTIFIED BY [PASSWORD] 'password']

with_option:
     GRANT OPTION
```

权限可以分为以下几个层级：

全局层级：适用于所有的数据库。使用GRANT ALL ON *.*授予全局权限。

数据库层级：适用于一个给定数据库中的所有目标。使用GRANT ALL ON db_name.*授予数据库权限。

表层级：表权限适用于一个给定表中的所有列。使用GRANT ALL ON db_name.tbl_name授予表权限。

给特定用户授予权限。如果用户不存在，可以直接创建用户。
(sql_mode='no_auto_create_user'，同时没有identified by指定密码时候，不可以直接创建用户。)

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OceanBase，权限才能生效。

用"*"代替table_name，表示赋予全局权限，即对数据库中的所有表赋权。

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的值如下表所示。

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 PRIVILEGES的权限。
CREATE TABLEGROUP	全局CREATE TABLEGROUP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。
SUPER	SET GLOBAL修改全局系统参数的权限。
SHOW DATABASES	全局 SHOW DATABASES的权限。
INDEX	CREATE INDEX, DROP INDEX的权限
CREATE VIEW	创建、删除视图的权限

SHOW VIEW	SHOW CREATE VIEW权限

说明：目前没有change effective tenant的权限控制，sys租户下的用户都可以。

撤销权限

REVOKE语句用于系统管理员撤销User某些权限。

格式

```
REVOKE priv_type
ON database.tblname
 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，当前用户必须拥有全局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。

用"*"代替table_name，表示撤销全局权限，即撤销对数据库中所有表的操作权限。

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的值如上表所示。

举例

执行以下命令撤销"obsqluser01"的所有权限。

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'obsqluser01';
```

查看权限

SHOW GRANTS语句用于系统管理员查看User的操作权限。

格式

```
SHOW GRANTS [FOR username];
```

- 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"mysql.user"的SELECT权限。

举例

```
Oceanbase>show grants for test;
+------------------------------+
| Grants for test              |
+------------------------------+
| GRANT USAGE ON *.* TO 'test' |
+------------------------------+
1 row in set (0.01 sec)
```

SET语句

设置字符集

字符集是一套符号和编码。校对规则是在字符集内用于比较字符的一套规则。

OceanBase 1.0支持utf8mb4字符集，utf8mb4是utf8的超集，最大编码长度4字节，最大有效位数21比特。与utf8相比，utf8mb4支持ios表情符号等新字符。通常将utf8支持的字符称为BMP（Basic Multilingual Plane），将utf8mb4新增支持的字符称为扩展集。

用户可以在租户级、Database级、表级、字段级、session级设置字符集，因OceanBase 1.0只支持utf8mb4这一种字符集，通常不需要用户去单独设置字符集。

在OceanBase 1.0中，utf8mb4字符集对应的collation支持utf8mb4_bin、utf8mb4_general_ci这二种，默认为utf8mb4_general_ci。主要不同点是对排序和字符串比较有影响，其中utf8mb4_bin大小写敏感、utf8mb4_general_ci大小写不敏感。

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

设置用户变量

```
Oceanbase>SET @a=2, @b=3;
Query OK, 0 rows affected (0.01 sec)

Oceanbase>SET @c = @a + @b;
Query OK, 0 rows affected (0.00 sec)
```

查看用户变量：

```
Oceanbase>SELECT @a, @b, @c;
+------+------+------+
| @a   | @b   | @c   |
+------+------+------+
|   2 |   3 |   5 |
+------+------+------+
1 row in set (0.01 sec)
```

或者通过select语句设置用户变量：

```
Oceanbase> SELECT @a:=2, @b:=3, @c:=@a+@b;
+-------+-------+-----------+
| @a:=2 | @b:=3 | @c:=@a+@b |
+-------+-------+-----------+
|    2 |    3 |       5 |
+-------+-------+-----------+
1 row in set (0.01 sec)
```

修改系统变量

系统变量和SQL功能相关，存放在"__all_sys_variable"表中，如autocommit，tx_isolation等。

OceanBase维护两种变量：

全局变量

影响OceanBase整体操作。当OceanBase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OceanBase的客户端。在客户端连接OceanBase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。

全局变量的更改不影响目前已经连接的客户端的会话变量，即使客户端执行SET GLOBAL语句也不影响。

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;
```

查看系统变量值的格式，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

- 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;
```

- 执行以下命令，查看"ob_trx_timeout"的值。

```
Oceanbase>show variables like 'ob_trx_timeout';
+----------------+-----------+
| Variable_name  | Value     |
+----------------+-----------+
```

```
| ob_trx_timeout | 100000000 |
+----------------+-----------+
1 row in set (0.02 sec)
```

同样可以使用select语句查询变量的值。

```
Oceanbase>select @@ob_trx_timeout;
+------------------+
| @@ob_trx_timeout |
+------------------+
|        100000000 |
+------------------+
1 row in set (0.00 sec)
```

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。

显示事务是用户自定义或用户指定的事务。通过BEGIN TRANSACTION，或BEGIN和BEGIN WORK（被作为START TRANSACTION的别名受到支持）语句显示开始，以COMMIT或ROLLBACK语句显示结束。

格式

开启事务语句格式如下：

```
START TRANSACTION
   [WITH CONSISTENT SNAPSHOT];
BEGIN [WORK] ;
COMMIT [WORK] ;
ROLLBACK [WORK];
```

OceanBase 1.0 只支持 READ COMMITTED 隔离级别。

> WITH CONSISTENT SNAPSHOT子句用于启动一个一致的读取。该子句的效果与发布一个START TRANSACTION，后面跟一个来自任何OceanBase表的SELECT的效果一样。OceanBase 1.0语法上支持 WITH CONSISTENT SNAPSHOT子句，其WITH CONSISTENT SNAPSHOT功能暂时还未实现。
>
> 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(点对点)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，不包括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];
```

回滚当前事务语句格式如下：

```
ROLLBACK [WORK];
```

SQL模式

OceanBase 服务器可以以不同的 SQL 模式操作，并且可以为不同客户端应用不同模式。这样每个应用程序可以根据自己的需求来定制服务器的操作模式。

模式定义 OceanBase 应支持哪些 SQL 语法，以及应执行哪种数据验证检查。这样可以更容易地在不同的环境中使用 OceanBase。

可以用--sql-mode="modes"选项来设置SQL模式。"严格模式"，表示至少STRICT_TRANS_TABLES 或 STRICT_ALL_TABLES 被启用的模式。

ALLOW_INVALID_DATES 在严格模式下不要检查全部日期。只检查1到12之间的月份和1到31之间的日。这在 Web 应用程序中，当你从三个不同的字段获取年、月、日，并且想要确切保存用户插入的内容(不进行日期验证)时很重要。该模式适用于 DATE 和DATETIME 列。不适合 TIMESTAMP 列，TIMESTAMP 列需要验证日期。启用严格模式后，服务器需要合法的月和日，不仅仅是分别在 1 到 12 和 1到 31 范围内。例如，禁用严格模式时'2004-04-31'是合法的，但启用严格模式后是非法的。要想在严格模式允许遮掩固定日期，还应启用 ALLOW_INVALID_DATES。

ANSI_QUOTES 将""视为识别符引号(``引号字符)，不要视为字符串的引号字符。在 ANSI 模式，你可以仍然使用``来引用识别符。启用 ANSI_QUOTES 后，你不能 用双引号来引用字符串，因为它被解释为识别符。

HIGH_NOT_PRECEDENCE NOT 操作符的优先顺序是表达式, 例如 NOT a BETWEEN b AND c 被解释为 NOT (a BETWEEN b AND c)。如果需要表达式被解释为(NOT a) BETWEEN b AND c，则启 用 HIGH_NOT_PRECEDENCESQL 模式，可以获得 NOT 的更高优先级的结果。

NO_UNSIGNED_SUBTRACTION 在减运算中，如果某个操作数没有符号，不要将结果标记为 UNSIGNED。

ONLY_FULL_GROUP_BY 不要让 GROUP BY 部分中的查询指向未选择的列。

PAD_CHAR_\TO_FULL_LENGTH 默认情况下，CHAR列值检索时后面的空格会被截断，在

PAD_CHAR_TO_FULL_LENGTH 模式下，截断不会发生。这种模式不适用于VARCHAR列值。

PIPES_AS_CONCAT 将||视为字符串连接操作符（ + ）(同 CONCAT())，而不视为 OR。

REAL_AS_FLOAT 将 REAL 视为 FLOAT 的同义词，而不是 DOUBLE 的同义词。

STRICT_ALL_TABLES 为所有存储引擎启用严格模式。非法数据值被拒绝。

STRICT_TRANS_TABLES 为事务存储引擎启用严格模式，也可能为非事务存储引擎启用严格模式。

严格模式控制 OceanBase 如何处理非法或丢失的输入值。有几种原因可以使一个值为非法。例如，数据类型错误，不适合列，或超出范围。当新插入的行不包含某列的没有显示定义 DEFAULT 子句的值，则该值被丢失。

对于事务表，当启用 STRICT_ALL_TABLES 或 STRICT_TRANS_TABLES 模式时，如果语句中有非法或丢失值，则会出现错误。语句被放弃并滚动。

对于非事务表，如果插入或更新的第 1 行出现坏值，两种模式的行为相同。语句被放弃，表保持不变。如果语句插入或修改多行，并且坏值出现在第 2 或后面的行，结果取决于启用了哪个严格选项：

- 对于 STRICT_ALL_TABLES，OceanBase 返回错误并忽视剩余的行。但是，在这 种情况下，前面的行已经被插入或更新。这说明你可以部分更新，这可能不是 你想要的。要避免这点，最好使用单行语句，因为这样可以不更改表即可以放 弃。

  对于 STRICT_TRANS_TABLES，OceanBase 将非法值转换为最接近该列的合法值 并插入调整后的值。如果值丢失，MySQL 在列中插入隐式 默认值。在任何情 况下，MySQL 都会生成警告而不是给出错误并继续执行语句。

  严格模式不允许非法日期，例如'2004-04-31'。它不允许禁止日期使用"零" 部分，例如'2004-04-00'或"零"日期。

  如果你不使用严格模式(即不启用 STRICT_TRANS_TABLES 或 STRICT_ALL_TABLES 模式)，对于非法或丢失的值，OceanBase 将插入调整后的 值并给出警告。