**Work with MYSQL Partitions**

**Create Table with partition**

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

(create\_definition,...)

[table\_options]

[partition\_options]

**partition\_options:**

PARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY(column\_list)

| RANGE(expr)

| LIST(expr) }

[PARTITIONS num]

[SUBPARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY(column\_list) }

[SUBPARTITIONS num]

]

[(partition\_definition[, partition\_definition] ...)

**partition\_definition:**

PARTITION partition\_name

[VALUES

{LESS THAN {(expr) | MAXVALUE}

|

IN (value\_list)}]

[[STORAGE] ENGINE [=] engine\_name]

[COMMENT [=] 'comment\_text' ]

[DATA DIRECTORY [=] 'data\_dir']

[INDEX DIRECTORY [=] 'index\_dir']

[MAX\_ROWS [=] max\_number\_of\_rows]

[MIN\_ROWS [=] min\_number\_of\_rows]

[TABLESPACE [=] tablespace\_name]

[NODEGROUP [=] node\_group\_id]

[(subpartition\_definition [, subpartition\_definition] ...)]

**Sub-partition\_definition:**

SUBPARTITION logical\_name

[[STORAGE] ENGINE [=] engine\_name]

[COMMENT [=] 'comment\_text' ]

[DATA DIRECTORY [=] 'data\_dir']

[INDEX DIRECTORY [=] 'index\_dir']

[MAX\_ROWS [=] max\_number\_of\_rows]

[MIN\_ROWS [=] min\_number\_of\_rows]

[TABLESPACE [=] tablespace\_name]

[NODEGROUP [=] node\_group\_id]

**ALTER TABLE: Partition operations**

ALTER TABLE statement can be used for adding, dropping, merging, and splitting partitions,

and for performing partitioning maintenance.

Here we define a non-partitioned table:

CREATE TABLE sale\_mast (

bill\_no INT,

bill\_date DATETIME

);

This table can be partitioned by HASH (or in another type), using the bill\_no column as the partitioning key,

into 6 (or other) partitions using ALTER TABLE statement :

ALTER TABLE t1

PARTITION BY HASH(id)

PARTITIONS 6;

**Partition naming :**

Names of partitions follow the rules of other MySQL identifiers such as databases, tables, constraint, stored procedure etc.

**Partition names are not case-sensitive.**

CREATE TABLE sale\_mast (bill\_no INT NOT NULL, bill\_date TIMESTAMP NOT NULL,

cust\_code VARCHAR(15) NOT NULL, amount DECIMAL(8,2) NOT NULL)

PARTITION BY RANGE (UNIX\_TIMESTAMP(bill\_date))(

PARTITION p0 VALUES LESS THAN (UNIX\_TIMESTAMP('2013-04-01')),

PARTITION p1 VALUES LESS THAN (UNIX\_TIMESTAMP('2013-07-01')),

PARTITION p2 VALUES LESS THAN (UNIX\_TIMESTAMP('2013-10-01')),

PARTITION p3 VALUES LESS THAN (UNIX\_TIMESTAMP('2014-01-01')));

insert some records in sale\_mast table :

mysql> INSERT INTO sale\_mast VALUES (1, '2013-01-02', 'C001', 125.56),

(2, '2013-01-25', 'C003', 456.50),

(3, '2013-02-15', 'C012', 365.00),

(4, '2013-03-26', 'C345', 785.00),

(5, '2013-04-19', 'C234', 656.00),

(6, '2013-05-31', 'C743', 854.00),

(7, '2013-06-11', 'C234', 542.00),

(8, '2013-07-24', 'C003', 300.00),

(8, '2013-08-02', 'C456', 475.20);

SELECT \* FROM sale\_mast;

The partition status of sale\_mast table:

SELECT PARTITION\_NAME, TABLE\_ROWS FROM INFORMATION\_SCHEMA.PARTITIONS WHERE TABLE\_NAME='sale\_mast';

In the above you can partition the table based on sale amount (amount.)

In these partitions the range of the sale amount (amount) are as

* partition p0 ( sale amount < 100 )
* partition p1 ( sale amount < 500 )
* partition p2 ( sale amount <1000 )
* partition p3 ( sale amount<1500 )

CREATE TABLE sale\_mast1 (bill\_no INT NOT NULL, bill\_date TIMESTAMP NOT NULL,

cust\_codE VARCHAR(15) NOT NULL, amount INT NOT NULL)

PARTITION BY RANGE (amount) (

PARTITION p0 VALUES LESS THAN (100),

PARTITION p1 VALUES LESS THAN (500),

PARTITION p2 VALUES LESS THAN (1000),

PARTITION p3 VALUES LESS THAN (1500));

**Drop a MySQL partition**

If you feel some data are useless in a partitioned table you can drop one or more partition(s).

To delete all rows from partition p0 of sale\_mast, you can use the statement :

ALTER TABLE sale\_mast DROP PARTITION p0;

DROP PARTITION can be used to drop one or more RANGE or LIST partitions. This statement cannot be used with HASH or KEY partitions; instead, use COALESCE PARTITION (see below). Any data that was stored in the dropped partitions named in the partition\_names list is discarded.

ALTER TABLE t1 DROP PARTITION p0, p1;

SELECT \* FROM sale\_mast3;

 The partition status of sale\_mast after dropping the partition p0 :

SELECT PARTITION\_NAME, TABLE\_ROWS FROM INFORMATION\_SCHEMA.PARTITIONS

WHERE TABLE\_NAME='sale\_mast';

SELECT PARTITION\_NAME, TABLE\_ROWS FROM INFORMATION\_SCHEMA.PARTITIONS

WHERE TABLE\_NAME='sale\_mast1’;

SELECT PARTITION\_NAME, TABLE\_ROWS FROM INFORMATION\_SCHEMA.PARTITIONS

WHERE TABLE\_NAME='sale\_mast2';

**MySQL LIST Partitioning**

List partition allows us to segment data based on a pre-defined set of values (e.g. 1, 2, 3). This is done by using PARTITION BY LIST(expr) where expr is a column value and then defining each partition by means of a VALUES IN (value\_list), where value\_list is a comma-separated list of integers. In MySQL 5.6, it is possible to match against only a list of integers (and possibly NULL) when partitioning by LIST. In the following example, sale\_mast2 table contains four columns bill\_no, bill\_date, agent\_code, and amount. Suppose there are 11 agents represent three cities A, B, C these can be arranged in three partitions with LIST Partitioning as follows :

CREATE TABLE sale\_mast3 (bill\_no INT NOT NULL, bill\_date TIMESTAMP NOT NULL,

agent\_codE INT NOT NULL, amount INT NOT NULL)

PARTITION BY LIST(agent\_code) (

PARTITION pA VALUES IN (1,2,3),

PARTITION pB VALUES IN (4,5,6),

PARTITION pC VALUES IN (7,8,9,10,11));

INSERT INTO sale\_mast3 VALUES (4, '2013-01-02', 2, 125), (4, '2013-03-26', 5, 785), ( (6, '2013-05-31', 2, 854),(7, '2013-06-11', 7, 542);

INSERT INTO sale\_mast3 VALUES (1, '2013-01-02', 2, 125);

INSERT INTO sale\_mast3 VALUES (2, '2013-01-25', 1, 456);

INSERT INTO sale\_mast3 VALUES (3, '2013-02-15', 3, 365);

INSERT INTO sale\_mast3 VALUES (5, '2013-04-19', 7, 656);

INSERT INTO sale\_mast3 VALUES (7, '2013-06-11', 9, 542);

SELECT PARTITION\_NAME, TABLE\_ROWS FROM INFORMATION\_SCHEMA.PARTITIONS

WHERE TABLE\_NAME='sale\_mast3';

**MySQL COLUMNS Partitioning**

In COLUMNS partitioning it is possible to use multiple columns in partitioning keys. There are two types of COLUMNS partitioning RANGE COLUMNS partitioning

LIST COLUMNS partitioning

In addition, both RANGE COLUMNS partitioning and LIST COLUMNS partitioning support the use of non-integer columns for defining value ranges or list members.

Both RANGE COLUMNS partitioning and LIST COLUMNS partitioning support following data types for defining value ranges or list members.

* All integer types: TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT.
* DATE and DATETIME.

**RANGE COLUMNS partitioning**

* RANGE COLUMNS partitioning is similar to range partitioning with some significant difference. RANGE COLUMNS accepts a list of one or more columns as partition keys. You can define the ranges using various columns of types (mentioned above) other than integer types.

CREATE TABLE table\_name

PARTITIONED BY RANGE COLUMNS(column\_list) (

PARTITION partition\_name VALUES LESS THAN (value\_list)[,

PARTITION partition\_name VALUES LESS THAN (value\_list)][,

...]

)

column\_list:

column\_name[, column\_name][, ...]

value\_list:

value[, value][, ...]

* column\_list is a list of one or more columns.
* value\_list is a list of values and must be supplied for each partition definition.
* column list and in the value list defining each partition must occur in the same order
* The order of the column names in the partitioning column list and the value lists do not have to be the same as the order of the table column definitions in CREATE TABLE statement.

CREATE TABLE table3 (col1 INT, col2 INT, col3 CHAR(5), col4 INT)

PARTITION BY RANGE COLUMNS(col1, col2, col3)

(PARTITION p0 VALUES LESS THAN (50, 100, 'aaaaa'),

PARTITION p1 VALUES LESS THAN (100,200,'bbbbb'),

PARTITION p2 VALUES LESS THAN (150,300,'ccccc'),

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

In the above table -

* Table table3 contains the columns col1, col2, col3, col4
* The first three columns have participated in partitioning COLUMNS clause, in the order col1, col2, col3.
* Each value list used to define a partition contains 3 values in the same order and (INT, INT, CHAR(5)) form.

**LIST COLUMNS partitioning**

LIST COLUMNS accepts a list of one or more columns as partition keys.You can use various columns of data of types other than integer types as partitioning columns. You can use string types, DATE, and DATETIME columns

In a company there are agents in 3 cities, for sales and marketing purposes.

create a table with LIST COLUMNS partitioning based on the above information :

mysql> CREATE TABLE salemast ( agent\_id VARCHAR(15), agent\_name VARCHAR(50),

agent\_address VARCHAR(100), city\_code VARCHAR(10))

PARTITION BY LIST COLUMNS(agent\_id) (

PARTITION pcity\_a VALUES IN('A1', 'A2', 'A3'),

PARTITION pcity\_b VALUES IN('B1', 'B2', 'B3'),

PARTITION pcity\_c VALUES IN ('C1', 'C2', 'C3', 'C4', 'C5'));

You can use DATE and DATETIME columns in LIST COLUMNS partitioning,

CREATE TABLE sale\_master (bill\_no INT NOT NULL, bill\_date DATE,

cust\_code VARCHAR(15) NOT NULL, amount DECIMAL(8,2) NOT NULL)

PARTITION BY RANGE COLUMNS (bill\_date)(

PARTITION p\_qtr1 VALUES LESS THAN ('2013-04-01'),

PARTITION p\_qtr2 VALUES LESS THAN ('2013-07-01'),

PARTITION p\_qtr3 VALUES LESS THAN ('2013-10-01'),

PARTITION p\_qtr4 VALUES LESS THAN ('2014-01-01'));

**MySQL HASH Partitioning**

MySQL HASH partition is used to distribute data among a predefined number of partitions on a column value or expression based on a column value. This is done by using PARTITION BY HASH(expr) clause, adding in CREATE TABLE STATEMENT. In PARTITIONS num clause, num is a positive integer represents the number of partitions of the table. The following statement creates a table that uses hashing on the studetn\_id column and is divided into 4 partitions :

MySQL>CREATE TABLE student (student\_id INT NOT NULL,

class VARCHAR(8), name VARCHAR(40),

date\_of\_admission DATE NOT NULL DEFAULT '2000-01-01')

PARTITION BY HASH(student\_id)

PARTITIONS 4;

Query OK, 0 rows affected (1.43 sec)

It is also possible to make a partition based on the year in which a student was admitted. See the following statement :

MySQL> CREATE TABLE student (student\_id INT NOT NULL,

class VARCHAR(8), class VARCHAR(8), name VARCHAR(40),

date\_of\_admission DATE NOT NULL DEFAULT '2000-01-01')

PARTITION BY HASH(YEAR(date\_of\_admission))

PARTITIONS 4;

Query OK, 0 rows affected (1.27 sec)

**MySQL KEY Partitioning**

MySQL KEY partition is a special form of HASH partition, where the hashing function for key partitioning is supplied by the MySQL server. The server employs its own internal hashing function which is based on the same algorithm as PASSWORD(). This is done by using PARTITION BY KEY, adding in CREATE TABLE STATEMENT. In KEY partitioning KEY takes only a list of zero or more column names. Any columns used as the partitioning key must comprise part or all of the table's primary key if the table has one. If there is a primary key in a table, it is used as partitioning key when no column is specified as the partitioning key.

MySQL> CREATE TABLE table1 ( id INT NOT NULL PRIMARY KEY,

fname VARCHAR(25), lname VARCHAR(25))

PARTITION BY KEY()

PARTITIONS 2;

Query OK, 0 rows affected (0.84 sec)

If there is no primary key but there is a unique key in a table, then the unique key is used for the partitioning key :

MySQL> CREATE TABLE table2 ( id INT NOT NULL, fname VARCHAR(25),

lname VARCHAR(25),

UNIQUE KEY (id))

PARTITION BY KEY()

PARTITIONS 2;

**MySQL Sub-partitioning**

Subpartitioning is a method to divide each partition further in a partitioned table. See the following CREATE TABLE statement :

CREATE TABLE table10 (BILL\_NO INT, sale\_date DATE, cust\_code VARCHAR(15),

AMOUNT DECIMAL(8,2))

PARTITION BY RANGE(YEAR(sale\_date) )

SUBPARTITION BY HASH(TO\_DAYS(sale\_date))

SUBPARTITIONS 4 (

PARTITION p0 VALUES LESS THAN (1990),

PARTITION p1 VALUES LESS THAN (2000),

PARTITION p2 VALUES LESS THAN (2010),

PARTITION p3 VALUES LESS THAN MAXVALUE

);

In the above statement -

* The table has 4 RANGE partitions.
* Each of these partitions—p0, p1, p2 and p3—is further divided into 4 subpartitions.
* Therefore the entire table is divided into 4 \* 4 = 16 partitions.

Here is the partition status of table10 :

mysql> SELECT PARTITION\_NAME, TABLE\_ROWS FROM

INFORMATION\_SCHEMA.PARTITIONS WHERE TABLE\_NAME='stable';

\*\*\*\*\*\*\*\*\*\*\*\*\*