**MYSQL Partitioning Tables**

Partitioning (a database design technique) improves performance, manageability, simplifies maintenance and reduce the cost of storing large amounts of data. Partitioning can be achieved without splitting tables by physically putting tables on individual disk drives. Partitioning allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, therefore queries that access only a fraction of the data can run faster because there is fewer data to scan.

**Advantages of partitioning**

* During the scan operation, MySQL optimizer accesses those partitions that will satisfy a particular query. For example, a whole year sale records table may be broken up into 4 partitions (i.e. sale data from of Apr-Jun (partition p0), Jul-Sep (partition p1) , Oct-Dec (partition p2), Jam-Mar (partition p0)) . If a query is issued that contains sale data between Jul-Sep quarter, then it scans the partition p1 only instead of total table records and the query will complete much sooner.
* Partitioning allows you to have more control over how data is managed inside the database. For example, you can drop specific partitions in a partitioned table where data loses its usefulness. The process of adding new data, in some cases, be greatly facilitated by adding one or more new partitions for storing that data using ALTER TABLE command.
* In partitioning, it is possible to store more data in one table than can be held on a single disk or file system partition.
* MySQL 5.6 supports explicit partition selection for queries. For example, SELECT \* FROM table1 PARTITION (p0,p1) WHERE col1< 10 selects only those rows in partitions p0 and p1 that match the WHERE condition, this can greatly speed up queries
* Partition selection also supports the data modification statements DELETE, INSERT, REPLACE, UPDATE, and LOAD DATA, LOAD XML.

The partitioning algorithms are:

■RANGE—Selects a partition by determining if the partition key is inside a range of values.

An example is a partition for all rows based on customer ID; values less than 500,000 can

go into one partition, and values greater than or equal to 500,000 can go into another.

■ LIST— A partition is assigned based on a list of integer values. If the partition key has

one of these values, the partition is chosen.

For example, all rows where the field city\_id

contains integers that reference New Delhi, Noida, Gandhi Nagar, or

gurugram could form one partition for Florida cities, and another partition could contain

city\_id values for cities in Massachusetts such as Boston, Cambridge, Amherst, and

Northampton.

■ HASH —The value of a hash function determines membership in a partition. If there are

10 partitions, the hash function returns a value from 0 to 9.

■ KEY—An internal algorithm is used by mysqld to try to evenly distribute the data across

the partitions. A field, called the *key column*, is used for the determination of data

distribution.

**RANGE and LIST** partitions can be subpartitioned using HASH or KEY partitioning. This is

known as ***composite partitioning*.**

Each of these partitioning algorithms utilizes a partition key for determining in which partition

the data is stored. This partition key is defined when creating the partition and must be an integer.

Additionally, if the table has a primary key, the primary key must include all the fields in

the partition key. If the table has no primary key but has a unique key, the unique key must

include all the fields in the partition key. However, a table can use partitioning if it has no primary and unique keys specified.

No matter what the partition algorithm is, the advantage of using partitioning is *partition pruning*.

If the field used in the partition key is part of a join or a filter (i.e., in the JOIN or WHERE

clauses), the query optimizer will know which partition(s) it needs to search. Thus, the list of

partitions to search is pruned, and query execution is faster. Horizontal partitioning makes data

more manageable, and partition pruning can make queries on partitioned data much faster.

**RANGE partitioning**

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the VALUES LESS THAN operator.

Below table can be partitioned by range in a number of ways, depending on your needs. One way would be to use the store\_id column. For instance, you might decide to partition the table in number of ways by adding a PARTITION BY RANGE clause as shown here:

**CREATE DATABASE part;**

**USE part;**

CREATE TABLE employees (

id INT NOT NULL,

name VARCHAR(30),

fname VARCHAR(30),

job\_code INT NOT NULL,

**store\_id INT NOT NULL,**

dept varchar(30),

country varchar(30)

)

PARTITION BY RANGE (store\_id)

(

PARTITION StoreP0 VALUES LESS THAN (10),

PARTITION StoreP1 VALUES LESS THAN (20),

PARTITION StoreP2 VALUES LESS THAN (30),

PARTITION StoreP3 VALUES LESS THAN (40)

);

Insert into employees values (101,'Scott','turner',501,15,'DW HOUSE DEPT','Scotland');

Insert into employees values (102,'krilson','tommy',501,13,'NETWORK DEPT','Swedon');

Insert into employees values (103,'Tracy','ekia',501,20,'DW HOUSE DEPT','Singapure');

Insert into employees values (104,'Sush','ogiri',501,42,'DW HOUSE DEPT','china');

Throws error because 42 id not available

---To avoid this you can alter the table

alter table employees add partition ( partition p4 values less than maxvalue );

Again this …

Insert into employees values (104,'Sush','ogiri',501,22,'DW HOUSE DEPT','china');

This table can be partitioned by range in various of ways, depending on your requirement. Here we have used the age column and adding a PARTITION BY RANGE clause. In these partitions the range of the people table age column (age) are as of follow :

CREATE TABLE people\_age

(

id INT NOT NULL,

name VARCHAR(30),

fname VARCHAR(30),

age INT NOT NULL,

city varchar(30),

state varchar(30)

)

PARTITION BY RANGE (age) (

PARTITION partage0 VALUES LESS THAN (20),

PARTITION partage1 VALUES LESS THAN (30),

PARTITION partage2 VALUES LESS THAN (40),

PARTITION partage3 VALUES LESS THAN (50),

PARTITION partage4 VALUES LESS THAN (60)

);

Insert into people\_age values (101,'krishna','Mohan',15,'Hyderabad','Telangana');

Insert into people\_age values (102,'Srinivas','Reddy',32,'Tirupati', 'Andhra');

Insert into people\_age values (103,'Suresh','kumar',49,'Chennai','Tamilnadu');

Insert into people\_age values (104,'Sushma','Ogiri',42,'Bangalore','Karnataka');

Insert into people\_age values (105,'Janvika','sarath',8,'Bangalore','Karnataka');

Insert into people\_age values (107,'mamata','varma',30,'Hyderabad','Telangana');

Insert into people\_age values (106,'mohan','krishna murthy',70,'Bangalore','Karnataka');

Throws error because 70 age not available

This employee table can be partitioned by range in various of ways, depending on your requirement. Here we have used the age column and adding a PARTITION BY RANGE clause with max value. In these partitions the range of the employee age Column (age) with MAX Value are as of follow :

CREATE TABLE employee

(

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30),

job\_code INT NOT NULL,

age INT NOT NULL,

dept varchar(30),

country varchar(30)

)

PARTITION BY RANGE (age) (

PARTITION p0 VALUES LESS THAN (20),

PARTITION p1 VALUES LESS THAN (30),

PARTITION p2 VALUES LESS THAN (40),

PARTITION p3 VALUES LESS THAN (50),

PARTITION p4 VALUES LESS THAN (60),

**PARTITION p5 VALUES LESS THAN MAXVALUE**

);

Insert into employee values (101,'Scott','turner',501,15,'DW HOUSE DEPT','Scotland');

Insert into employee values (102,'krilson','tommy',501,13,'NETWORK DEPT','Swedon');

Insert into employee values (103,'Tracy','Ekia',501,20,'DW HOUSE DEPT','Singapure');

Insert into employee values (105,'janu','chowdary',501,19,'CIVIL DEPT','India');

Insert into employee values (104,'Sush','ogiri',501,43,'DBA DEPT','china');

Insert into employee values (106,'yashoda','Babu',532,36,'QA DEPT','India');

Insert into employee values (107,'Anusha','Naidu',511,25,'EEE DEPT','china');

Now you can check

**SELECT TABLE\_NAME,PARTITION\_NAME,SUBPARTITION\_NAME,TABLE\_ROWS FROM INFORMATION\_SCHEMA.PARTITIONS WHERE TABLE\_SCHEMA='part';**

One more example with Insurance table example on date with year , on **renewal** column created with Range Partition.

CREATE TABLE Insurance (

first\_name VARCHAR(25),

last\_name VARCHAR(25),

street VARCHAR(30),

city VARCHAR(15),

renewal DATE

)

PARTITION BY RANGE COLUMNS(renewal) (

PARTITION pWeek\_1 VALUES LESS THAN('2020-02-01'),

PARTITION pWeek\_2 VALUES LESS THAN('2020-03-01'),

PARTITION pWeek\_3 VALUES LESS THAN('2020-04-01'),

PARTITION pWeek\_4 VALUES LESS THAN('2020-05-01')

);

ALTER TABLE CUSTOMERS\_3 ENGINE=MYISAM;

**LIST Partitioning**

List partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by RANGE, each partition must be explicitly defined. The chief difference between the two types of partitioning is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values.

---Normal Table

CREATE TABLE Example (

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30),

job\_code INT,

store\_id INT

);

| **City Region** | **Store ID Numbers** |
| --- | --- |
| North | 3, 5, 6, 9, 17 |
| East | 1, 2, 10, 11, 19, 20 |
| West | 4, 12, 13, 14, 18 |
| South | 7, 8, 15, 16 |

CREATE TABLE store (

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30),

Dept varchar(20),

store\_id INT

)

PARTITION BY LIST(store\_id) (

PARTITION Part\_North VALUES IN (3,5,6,9,17),

PARTITION Part\_East VALUES IN (1,2,10,11,19,20),

PARTITION Part\_West VALUES IN (4,12,13,14,18),

PARTITION Part\_South VALUES IN (7,8,15,16)

);

Insert into store values (101,'Scott','turner', 'IT DEPT',5);

Insert into store values (102,'Rakesh','Singh','IT IS DEPT',11);

Insert into store values (103,'Tracy','ekia','WAREHOUSE DEPT',14);

Insert into store values (104,'ramya','eerthi','Electrical DEPT',13);

Insert into store values (105,'mamata','varma','EVENTS DEPT',14);

Insert into store values (106,'Harita','Royal','DOT NET DEPT',16);

One more example with cities, city column created with List Partition

CREATE TABLE customers (

name VARCHAR(25),

street VARCHAR(30),

city VARCHAR(15)

)

PARTITION BY LIST COLUMNS(city) (

PARTITION pRegion\_1 VALUES IN('Delhi', 'Agra', 'Noida'),

PARTITION pRegion\_2 VALUES IN('Mumbai', 'Pune', 'Gandhinagar'),

PARTITION pRegion\_3 VALUES IN('Hyderabad', 'Visakhapatnam', 'Tirupati'),

PARTITION pRegion\_4 VALUES IN('Chennai', 'Bangalore', 'Cochin')

);

Insert into customers values ('sarath','Sri Nagar Colony','Hyderabad');

Insert into customers values ('santhosh','Crompet','Chennai');

Insert into customers values ('anil','Thane','mumbai');

Insert into customers values ('suresh','Gandhi road ','cochin');

Insert into customers values ('Sunil',' Gandhi Colony','Bhopal');

Error….

Insert into customers values ('anil kumar',' Gandhi Colony','pune');

**HASH Partitioning**

HASH partitions evenly distribute data across a predetermined number of partition tables. HASH

partitioning uses the modulus operator (%) to distribute the data across these partitions.

Unlike RANGE and LIST partitioning, partition names and matching criteria for the partition key are not user-defined. To create a table for employees partitioned into 10 partitions.

CREATE TABLE emp\_hash (

emp\_id INT NOT NULL,

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(30) NOT NULL,

dept VARCHAR(30) NOT NULL,

store\_it INT NOT NULL

)

PARTITION BY HASH (store\_it) PARTITIONS 9;

Insert into emp\_hash values (11,'Scott','turner','DW HOUSE DEPT',5);

Insert into emp\_hash values (1,'Rakesh','Singh''IT IS DEPT','USA',11);

Insert into emp\_hash values (14303,'Tracy','ekia','Calder HOUSE',7);

Insert into emp\_hash values (14,'Tim','Chris','DOT NET DEPT',100);

**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. Here is an example :

CREATE TABLE keytable (ID INT NOT NULL PRIMARY KEY,

fname VARCHAR(25),name VARCHAR(25))

PARTITION BY KEY() PARTITIONS 7;

Insert into keytable values (101,'Kiran','kumar');

Insert into keytable values (201,'anil','kumar');

CREATE TABLE keytbl ( id INT NOT NULL, fname VARCHAR(25),

lname VARCHAR(25),

UNIQUE KEY (id))

PARTITION BY KEY() PARTITIONS 6;

Insert into keytbl values (111,'Praveen','kumar');

Insert into keytbl values (301,'arul','kumar');

Insert into keytbl values (114,'jyothi','reddy');

Note:- Complicate table [ Range Partition ]

CREATE TABLE members (

id INT,

fname VARCHAR(25),

lname VARCHAR(25),

dob DATE

)

PARTITION BY RANGE( YEAR(dob) ) (

PARTITION p0 VALUES LESS THAN (1980),

PARTITION p1 VALUES LESS THAN (1990),

PARTITION p2 VALUES LESS THAN (2000)

);

Insert into members values (101,'mahesh','kumar','1980-02-01');

Insert into members values (102,'sathish','kumar','1991-03-02');

Insert into members values (103,'pradeep','kumar','1970-08-03');

Insert into members values (104,'anil','kumar','1999-03-04');

Insert into members values (105,'sunil','kumar','2007-03-01');

Insert into members values (106,'Anitha','sharma','2015-02-01');

---6 records try to insert

ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2010));

ALTER TABLE members ADD PARTITION (PARTITION p4 VALUES LESS THAN (2020));

ALTER TABLE members ADD PARTITION (PARTITION p5 VALUES LESS THAN (1970));

ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition

ALTER TABLE members REORGANIZE PARTITION **p0** INTO (PARTITION split1 VALUES LESS THAN (1970),PARTITION split2 VALUES LESS THAN (1980) );

ALTER TABLE members REORGANIZE PARTITION **split1** INTO (PARTITION splitA VALUES LESS THAN (1960),PARTITION splitB VALUES LESS THAN (1970));

SHOW CREATE TABLE members\G

ALTER TABLE members REORGANIZE PARTITION split0 INTO ( PARTITION s01 VALUES LESS THAN (1960),PARTITION s02 VALUES LESS THAN (1970) );

MySQL Sub-Partitioning

Sub-partitioning is a method to divide each partition further in a partitioned table.

See the following CREATE TABLE statement :

CREATE TABLE sub\_part (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 3(

PARTITION part1 VALUES LESS THAN (1990),

PARTITION part2 VALUES LESS THAN (2000),

PARTITION part3 VALUES LESS THAN (2010),

PARTITION part4 VALUES LESS THAN (2020),

PARTITION partmax VALUES LESS THAN MAXVALUE

);

* The table has 5 RANGE partitions.
* Each of these partitions—part1, part2, part3,.. partmax—is further divided into 3 sub partitions.
* Therefore the entire table is divided into 5 \* 3 = 15 partitions.

**To View / Select the Partitions:-**

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

select \* from people\_age partition (partage2);

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

**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 following statement :

SHOW CREATE TABLE store\G

--Truncate the data in that partition

ALTER TABLE store TRUNCATE PARTITION Part\_East;

--Drop the partition permanently

ALTER TABLE store drop PARTITION Part\_West;

SELECT \* FROM store;

**Split partitions: ( Discussed on Year Basics )**

ALTER TABLE employee REORGANIZE PARTITION mainpart01 INTO (PARTITION splitp0 VALUES LESS THAN (6), PARTITION splitp1 VALUES LESS THAN (11));

OPTIMISE, ANALYSE, REPAIR AND CHECK PARTITIONS

**Rebuild partitions:**

mysql> ALTER TABLE employee REBUILD PARTITION p0, p1;

Rebuild the partitions, can be a method to remove fragmentation as an example.

reclaim any unused space and to defragment the partition data file

mysql> ALTER TABLE employee OPTIMIZE PARTITION p0,p1;

You can check partitions for errors in much the same way that you can use CHECK TABLE with nonpartitioned tables.

mysql> ALTER TABLE employee CHECK PARTITION p0;

This reads and stores the key distributions for partitions.

mysql> ALTER TABLE employee ANALYZE PARTITION p1;

This repairs corrupted partitions.

mysql> ALTER TABLE employee REPAIR PARTITION p0;

**Restrictions and Limitations on Partitioning: -**

1. User-defined partitioning and the MERGE storage engine are not compatible. Tables using the MERGE storage engine cannot be partitioned.
2. FEDERATED storage engine. Partitioning of FEDERATED tables is not supported.
3. CSV storage engine. Partitioned tables using the CSV storage engine are not supported.
4. BLACKHOLE storage engine will support, but data will loss, it will store in binary logs.

**Altering the Non Partition table to Partition table**

Suppose that we have a MySQL table with the following schema

CREATE TABLE emp (

id INT NOT NULL,

name VARCHAR(30),

fname VARCHAR(30),

job\_code INT NOT NULL,

**store\_id** INT NOT NULL,

dept varchar(30),

country varchar(30)

);

Insert into emp values (101,'Scott','turner',501,15,'DW HOUSE DEPT','Scotland');

Insert into emp values (102,'krilson','tommy',501,13,'NETWORK DEPT','Swedon');

Insert into emp values (103,'Tracy','ekia',501,20,'DW HOUSE DEPT','Singapure');

Insert into emp values (104,'Sush','ogiri',501,22,'DW HOUSE DEPT','China');

Insert into emp values (105,'janu','chowdary',501,19,'CIVIL DEPT','India');

Insert into emp values (106,'yashoda','chowdary',532,36,'QA DEPT','India');

Insert into emp values (107,'Anusha','Chowdary',511,25,'EEE DEPT','china');

Insert into emp values (108,'Janvika','Sarath',03,08,'KIDS DEPT','India');

**Range partitioning**

alter table emp

PARTITION BY RANGE (store\_id) (

PARTITION p0 VALUES LESS THAN (100),

PARTITION p1 VALUES LESS THAN (200),

PARTITION p2 VALUES LESS THAN (300),

PARTITION p3 VALUES LESS THAN (400)

);

select \* from emp partition (p2);

select \* from emp partition (p0);

CREATE TABLE `employee` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(50) DEFAULT NULL,

`email` varchar(100) DEFAULT NULL,

`country` varchar(50) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB;

We will be creating some partitions on the MySQL table to understand how to create the partitions.

insert into employee values (1,'sarath','sarath.dba@gmail.com','india');

insert into employee values (220,'kiran','kiran.dba@gmail.com','india');

insert into employee values (333,'kumar','kumar.dba@gmail.com','USA');

insert into employee values (311,'Praveen kumar','Pkumar.dba@gmail.com','uk');

insert into employee values (431,'sairam','sai.dba@gmail.com','USA');

insert into employee values (500,'priya','priya.dba@gmail.com','uk');

insert into employee values (301,'karan kumar','kumar.dba@gmail.com','uk');

insert into employee values (303,'Praveen Reddy','PRdba@gmail.com','uk');

insert into employee values (353,'Sarath kumar','Skumar.dba@gmail.com','uk');

insert into employee values (403,'Sridevi','Sridevi.dba12@gmail.com','India');

insert into employee values (404,'Sridev','Sridev.db@gmail.com','AUSTRALIA');

In below query the table employee will be partitioned into 10 even size hash partitions

**ALTER table** employee **PARTITION BY HASH(id) PARTITIONS 10;**

The above query will create 7 even size partition of the table employee on the basis of id.

**ALTER table** employee **PARTITION BY KEY(id) PARTITIONS 5;**

**SYSTEM VIEWS**

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

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