Assignment No. 3

Title: Partitioning and its various types

Aim:

Software required:

Theory:

Program/Query:

First create a Database ……

Say

```
Create database example;
Use example;
CREATE TABLE Sales ( cust_id INT NOT NULL, name VARCHAR(40),
store_id VARCHAR(20) NOT NULL, bill_no INT NOT NULL,
bill_date DATE PRIMARY KEY NOT NULL, amount DECIMAL(8,2) NOT NULL)
PARTITION BY RANGE (year(bill_date))
(PARTITION p0 VALUES LESS THAN (2016),
PARTITION p1 VALUES LESS THAN (2017),
PARTITION p2 VALUES LESS THAN (2018),
PARTITION p3 VALUES LESS THAN (2020));
```

Insert values

```
INSERT INTO Sales VALUES
(1, 'Mike', 'S001', 101, '2015-01-02', 125.56),
(2, 'Robert', 'S003', 103, '2015-01-25', 476.50),
(3, 'Peter', 'S012', 122, '2016-02-15', 335.00),
(4, 'Joseph', 'S345', 121, '2016-03-26', 787.00),
(5, 'Harry', 'S234', 132, '2017-04-19', 678.00),
(6, 'Stephen', 'S743', 111, '2017-05-31', 864.00),
```

(7, 'Jacson', 'S234', 115, '2018-06-11', 762.00),

(8, 'Smith', 'S012', 125, '2019-07-24', 300.00),

(9, 'Adam', 'S456', 119, '2019-08-02', 492.20);

We can see the partition created by CREATE TABLE statement using the below query:

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'mydata' AND TABLE_NAME = 'Sales';
```

we can see the single partition records as

```
SELECT * FROM sales PARTITION (p1);

SELECT * FROM sales PARTITION (p3);

SELECT * FROM sales PARTITION (p2);

SELECT * FROM sales PARTITION (p0);
```

List Partitioning

Implementation of Data partitioning through List partition.

MySQL LIST Partitioning

It is the same as Range Partitioning. Here, the partition is defined
and selected based on columns matching one of a set of discrete value lists
rather than a set of a contiguous range of values. It is performed by the
PARTITION BY LIST(exp) clause.
The exp is an expression or column value that returns an integer value.
The VALUES IN(value_lists) statement will be used to define each partition.

In the below example, suppose we have 12 stores distributed
among four franchises based on their region. The table explains it more
clearly:

Region Store ID Number
East 101, 103, 105
West 102, 104, 106
North 107, 109, 111
South 108, 110, 112

We can partition the above table where rows for stores belonging
to the same region and will be stored in the same partition.
The following statement arranges the stores in the same region using LIST partitioning,
as shown below:

```
CREATE TABLE Stores (
cust_name VARCHAR(40),
```

```
bill_no VARCHAR(20) NOT NULL,
store_id INT PRIMARY KEY NOT NULL,
bill_date DATE NOT NULL,
amount DECIMAL(8,2) NOT NULL)
PARTITION BY LIST(store_id)
(PARTITION pEast VALUES IN (101, 103, 105),
PARTITION pWest VALUES IN (102, 104, 106),
PARTITION pNorth VALUES IN (107, 109, 111),
PARTITION pSouth VALUES IN (108, 110, 112));
```

We can see the partition created by CREATE TABLE statement using the below query:

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'example' AND TABLE_NAME = 'Stores';
```

we can see the single partition records as

```
SELECT * FROM stores PARTITION (pEast);

SELECT * FROM stores PARTITION (pWest);

SELECT * FROM stores PARTITION (pNorth);

SELECT * FROM stores PARTITION (pSouth);
```

EXAMPLE 2

MySQL RANGE Partitioning

In MySQL, RANGE partitioning mode allows us to specify various ranges for which data is assigned. Ranges should be contiguous but not overlapping, and are defined using the VALUES LESS THAN operator. In the following example, sale_mast table contains four columns: bill_no, bill_date, cust_code and amount. This table can be partitioned by range in various ways, depending on your requirement. Here we have used the bill_date column and decide to partition the table 4 ways by adding a PARTITION BY RANGE clause. In these partitions the range of the sale date (sale_date) are as of follow :

- partition p0 (sale between 01-01-2013 to 31-03-2013)
- partition p1 (sale between 01-04-2013 to 30-06-2013)
- partition p2 (sale between 01-07-2013 to 30-09-2013)
- partition p3 (sale between 01-10-2013 to 30-12-2013)

Let create the table :

```
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')));
```

Now 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);
```

Here is the partition status of sale_mast table:

```
SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE
TABLE_NAME='sale_mast';
```

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

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

- 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 is 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:

```
MySQL> ALTER TABLE sale_mast TRUNCATE PARTITION p0;
```

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 :

City	Agent ID
A	1, 2, 3
B	4, 5, 6
C	7, 8, 9, 10, 11

```
CREATE TABLE sale_mast2 (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));
```

FAQs

1. What is partitioning? List Out its types?
2. Why is partitioning important in databases?
3. Write an application of partitioning?
4. What is HASH partitioning?
5. What is round robin partitioning?