MySQL Partitions

Author : Shivshankar Chandankhede

References:

1) https://www.javatpoint.com/mysql-partitioning
2) https://www.w3resource.com/mysql/mysql-partition.php
3) https://www.vertabelo.com/blog/everything-you-need-to-know-about-mysql-partitions/

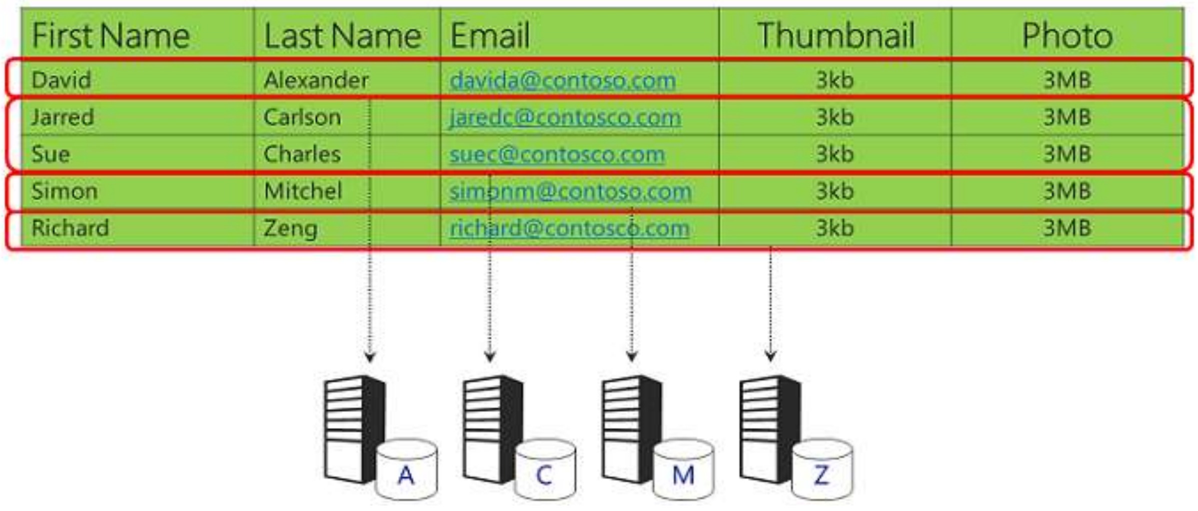
What is MySQL partitioning?

- Partitioning in MySQL is used to split or partition the rows of a table into separate tables in different locations, but still, it is treated as a single table.
- It distributes the portions of the table's data across a file system based on the rules we have set as our requirement.
- The rule that we have set to accomplish the division of table data is called as a partitioning function (modulus, a linear or internal hashing function, etc.).
- The selected function is based on the partitioning type we have specified and takes a user-supplied expression as its parameter.
- The user- expression can be a column value or a function acting on column values, depending on the type of partitioning used.

What kind of partition types are there?

1. Horizontal Partitioning

- This partitioning split the rows of a table into multiple tables based on our logic.
- In horizontal partitioning, the number of columns is the same in each table, but no need to keep the same number of rows.
- It physically divides the table but logically treated as a whole.
- Currently, MySQL supports this partitioning only.

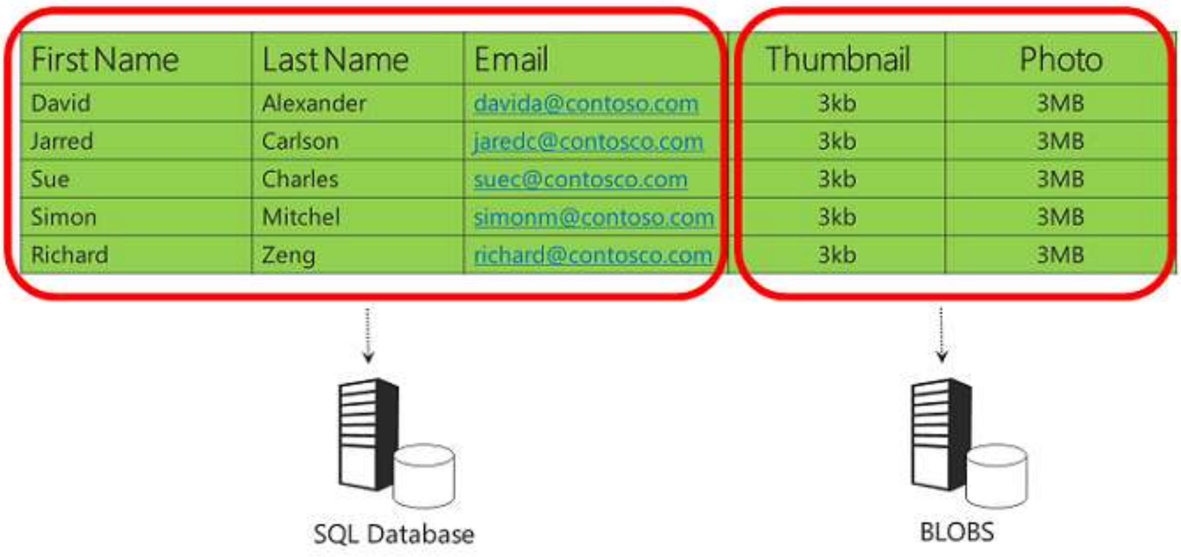


First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Fig. shows Horizontal Partitioning which is about splitting of rows

2. Vertical Partitioning

- This partitioning splits the table into multiple tables with fewer columns from the original table.
- It uses an additional table to store the remaining columns.
- Currently, MySQL does not provide supports for this partitioning.



First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Fig. shows Vertical Partitioning which is about splitting of columns

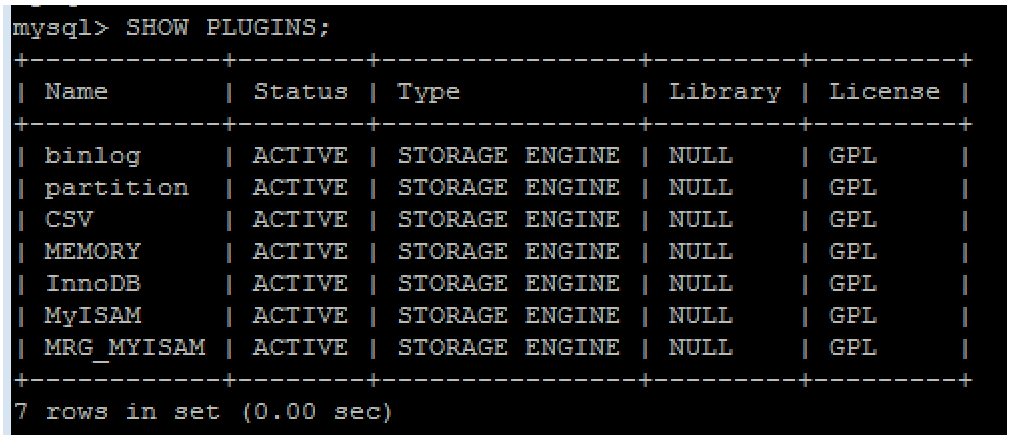
Benefits of Partitioning

The following are the benefits of partitioning in MySQL:

- It provides more control to manage the data in your database.
- Storage: It is possible to store more data in one table than can be held on a single disk or file system partition.
- Deletion: Dropping a useless partition is almost instantaneous, but a classical DELETE query run in a very large table could take minutes.
- Partition Pruning: This is the ability to exclude non-matching partitions and their data from a search; it makes querying faster. It optimizes the query performance. When we query on the table, it scans only the portion of a table that will satisfy the particular statement. Also, MySQL 5.7 supports explicit partition selection in queries, which greatly increases the search speed. (Obviously, this only works if you know in advance which partitions you want to use.) This also applies for DELETE, INSERT, REPLACE, and UPDATE statements as well as LOAD DATA and LOAD XML.

How do you know if this is something your database engine supports?

By default, community binaries include partitioning support. You can check if it's being supported in your current instance by running the SHOW PLUGINS statement. The output should display something like the following row:



```
mysql> SHOW PLUGINS;
+------------+--------+----------------+---------+---------+
| Name       | Status | Type           | Library | License |
+------------+--------+----------------+---------+---------+
| binlog     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| partition  | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| CSV        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MEMORY     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| InnoDB     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MyISAM     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
+------------+--------+----------------+---------+---------+
7 rows in set (0.00 sec)
```

Types of MySQL Partitioning

MySQL has mainly six types of partitioning, which are given below:

- RANGE Partitioning
- LIST Partitioning
- COLUMNS Partitioning
- HASH Partitioning
- KEY Partitioning
- Subpartitioning

MySQL RANGE Partitioning

- This partitioning allows us to partition the rows of a table based on column values that fall within a specified range.
- The given range is always in a contiguous form but should not overlap each other, and also use the VALUES LESS THAN operator to define the ranges.

To perform the practical, you can download the sales database(sales.sql file) from the URL mentioned below:

https://github.com/shiv2022/MySQL_tutorials/tree/main/MySQL%20Partitioning

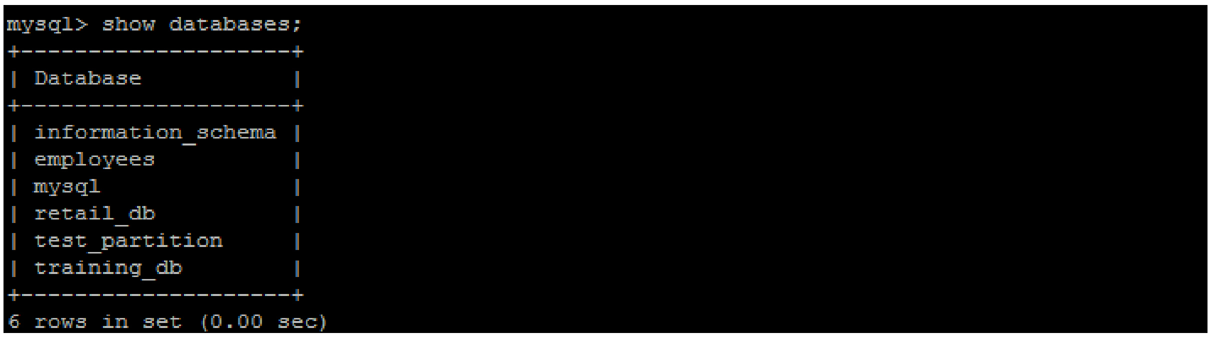
Query to create the RANGE Partition:

```
CREATE TABLE test_partition.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));
```

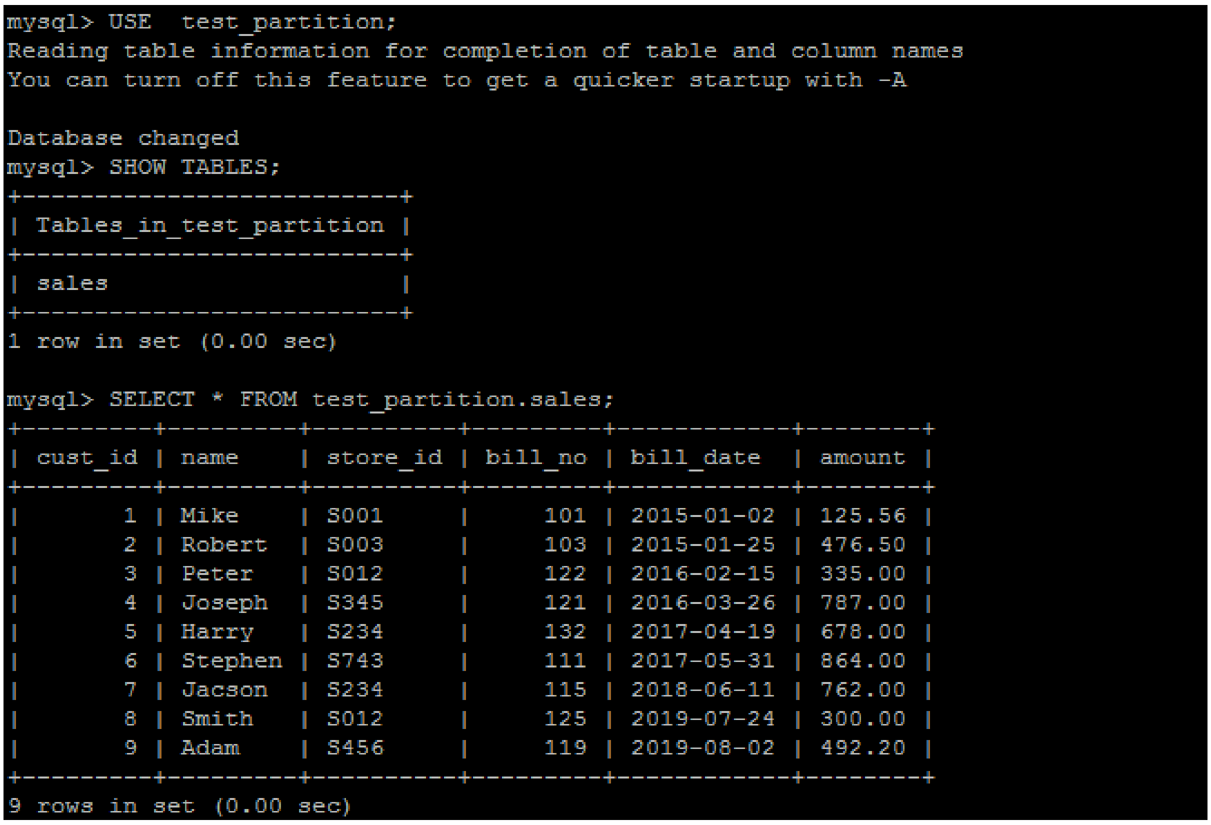
Creating the database and table schema with the partition by range clause using the below command:

```
mysql -u root -p < sales.sql
```

Once the script has completed, you should be able to see the below entries:



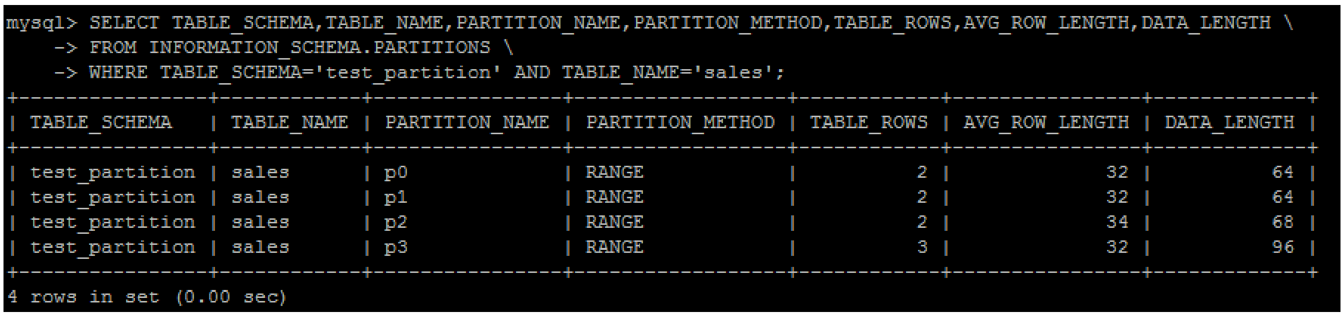
```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| employees          |
| mysql              |
| retail_db          |
| test_partition     |
| training_db        |
+--------------------+
6 rows in set (0.00 sec)
```



```
mysql> USE  test_partition;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+--------------------------+
| Tables_in_test_partition |
+--------------------------+
| sales                    |
+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM test_partition.sales;
+---------+---------+----------+---------+------------+--------+
| cust_id | name    | store_id | bill_no | bill_date  | amount |
+---------+---------+----------+---------+------------+--------+
|       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 |
+---------+---------+----------+---------+------------+--------+
9 rows in set (0.00 sec)
```

You can check the partition type and number of partition name and other important parameters associated with the table using the below query:

```
mysql> SELECT
TABLE_SCHEMA,TABLE_NAME,PARTITION_NAME,PARTITION_METHOD,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH \
    -> FROM INFORMATION_SCHEMA.PARTITIONS \
    -> WHERE TABLE_SCHEMA='test_partition' AND TABLE_NAME='sales';
```



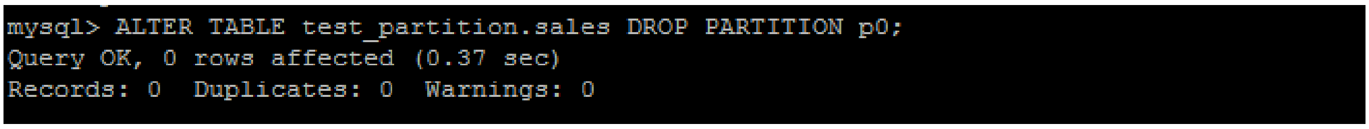
```
mysql> SELECT TABLE_SCHEMA,TABLE_NAME,PARTITION_NAME,PARTITION_METHOD,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH \
    -> FROM INFORMATION_SCHEMA.PARTITIONS \
    -> WHERE TABLE_SCHEMA='test_partition' AND TABLE_NAME='sales';
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| TABLE_SCHEMA   | TABLE_NAME | PARTITION_NAME | PARTITION_METHOD | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| test_partition | sales      | p0             | RANGE            |          2 |             32 |          64 |
| test_partition | sales      | p1             | RANGE            |          2 |             32 |          64 |
| test_partition | sales      | p2             | RANGE            |          2 |             34 |          68 |
| test_partition | sales      | p3             | RANGE            |          3 |             32 |          96 |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
4 rows in set (0.00 sec)
```

Dropping MySQL Partition

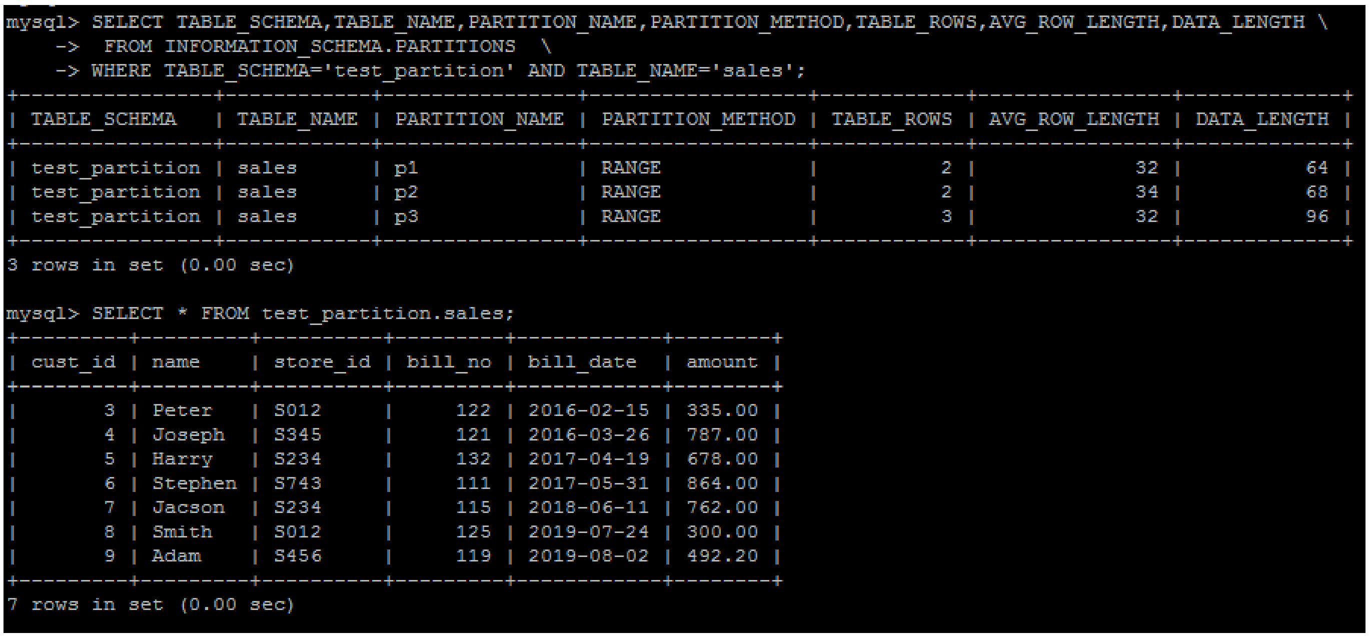
Sometimes our table contains the data that is useless in the partition table. In that case, we can drop single or multiple partitions based on the need. The following statement is used to delete all rows from the partition p0 of table Sales:

```
ALTER TABLE sales DROP PARTITION p0;
```

Verify the partition table by running the above mentioned query:



```
mysql> ALTER TABLE test_partition.sales DROP PARTITION p0;
Query OK, 0 rows affected (0.37 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
mysql> SELECT TABLE_SCHEMA,TABLE_NAME,PARTITION_NAME,PARTITION_METHOD,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH \
    ->  FROM INFORMATION_SCHEMA.PARTITIONS  \
    -> WHERE TABLE_SCHEMA='test_partition' AND TABLE_NAME='sales';
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| TABLE_SCHEMA   | TABLE_NAME | PARTITION_NAME | PARTITION_METHOD | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| test_partition | sales      | p1             | RANGE            |          2 |             32 |          64 |
| test_partition | sales      | p2             | RANGE            |          2 |             34 |          68 |
| test_partition | sales      | p3             | RANGE            |          3 |             32 |          96 |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM test_partition.sales;
+---------+---------+----------+---------+------------+--------+
| cust_id | name    | store_id | bill_no | bill_date  | amount |
+---------+---------+----------+---------+------------+--------+
|       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 |
+---------+---------+----------+---------+------------+--------+
7 rows in set (0.00 sec)
```

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

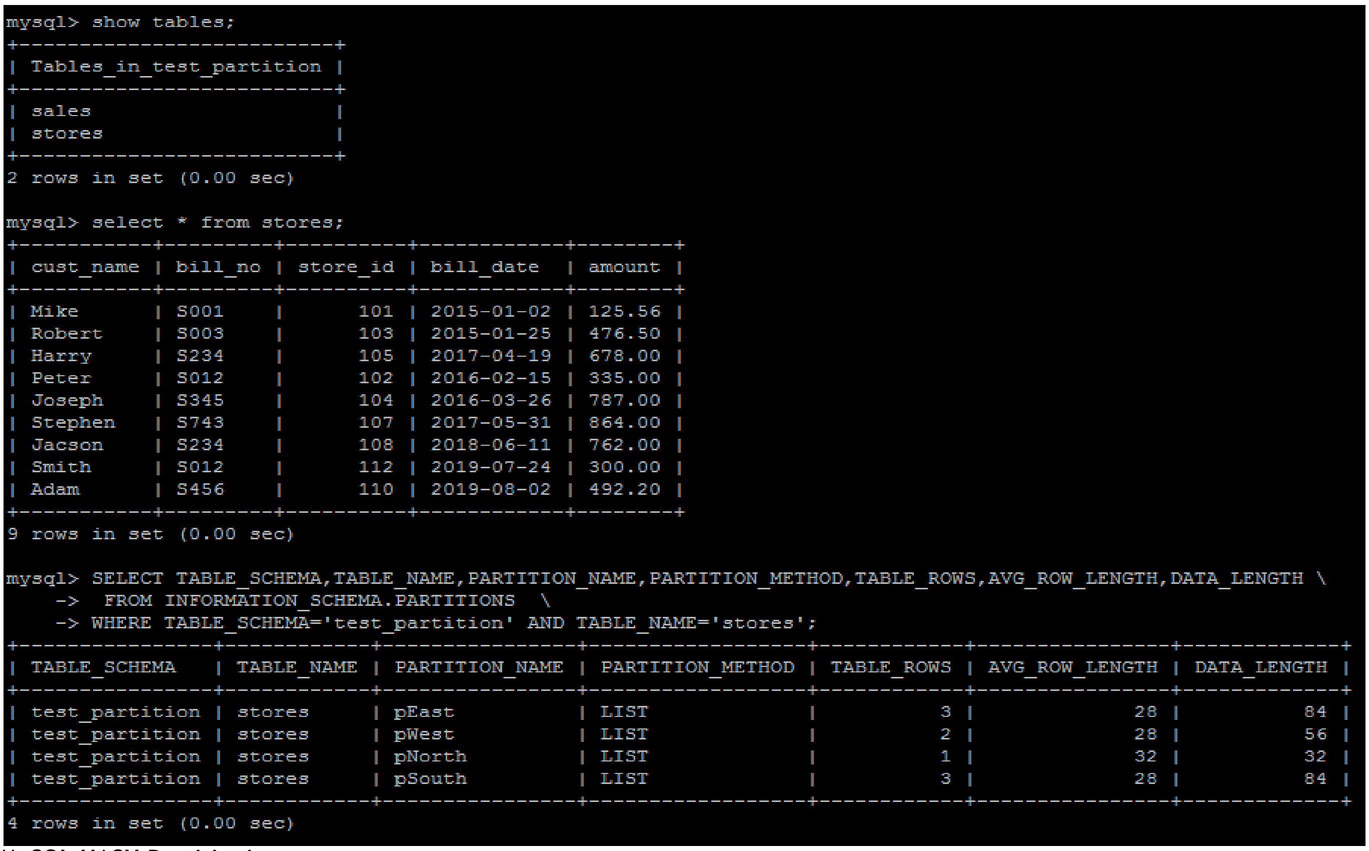
We can partition the above table where rows for stores belonging to the same region and will be stored in the same partition.

Query to create List Partition:

```
CREATE TABLE IF NOT EXISTS test_partition.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));
```

You can download the store.sql file by clicking here :
https://github.com/shiv2022/MySQL_tutorials/blob/main/MySQL%20Partitioning/stores.sql

```
mysql -u root -p < stores.sql
```

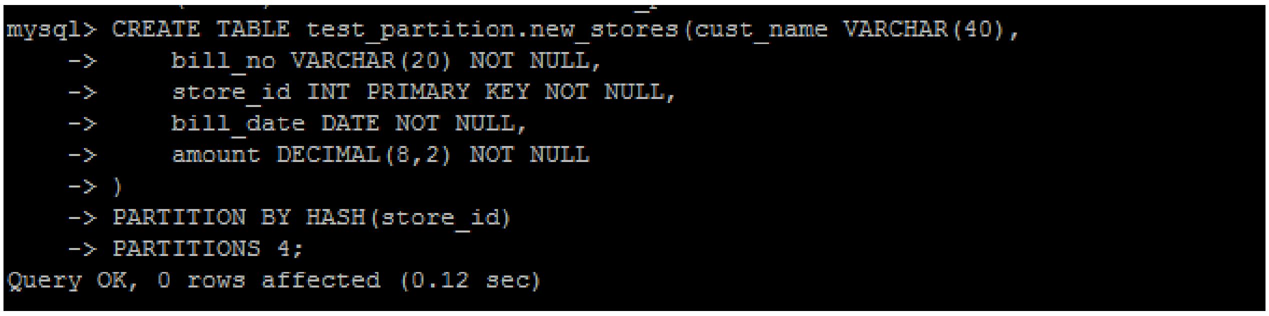

```
mysql> show tables;
+--------------------------+
| Tables_in_test_partition |
+--------------------------+
| sales                    |
| stores                   |
+--------------------------+
2 rows in set (0.00 sec)

mysql> select * from stores;
+-----------+---------+----------+------------+--------+
| cust_name | bill_no | store_id | bill_date  | amount |
+-----------+---------+----------+------------+--------+
| Mike      | S001    |      101 | 2015-01-02 | 125.56 |
| Robert    | S003    |      103 | 2015-01-25 | 476.50 |
| Harry     | S234    |      105 | 2017-04-19 | 678.00 |
| Peter     | S012    |      102 | 2016-02-15 | 335.00 |
| Joseph    | S345    |      104 | 2016-03-26 | 787.00 |
| Stephen   | S743    |      107 | 2017-05-31 | 864.00 |
| Jacson    | S234    |      108 | 2018-06-11 | 762.00 |
| Smith     | S012    |      112 | 2019-07-24 | 300.00 |
| Adam      | S456    |      110 | 2019-08-02 | 492.20 |
+-----------+---------+----------+------------+--------+
9 rows in set (0.00 sec)

mysql> SELECT TABLE_SCHEMA,TABLE_NAME,PARTITION_NAME,PARTITION_METHOD,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH \
    ->  FROM INFORMATION_SCHEMA.PARTITIONS  \
    -> WHERE TABLE_SCHEMA='test_partition' AND TABLE_NAME='stores';
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| TABLE_SCHEMA   | TABLE_NAME | PARTITION_NAME | PARTITION_METHOD | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| test_partition | stores     | pEast          | LIST             |          3 |             28 |          84 |
| test_partition | stores     | pWest          | LIST             |          2 |             28 |          56 |
| test_partition | stores     | pNorth         | LIST             |          1 |             32 |          32 |
| test_partition | stores     | pSouth         | LIST             |          3 |             28 |          84 |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
4 rows in set (0.00 sec)
```

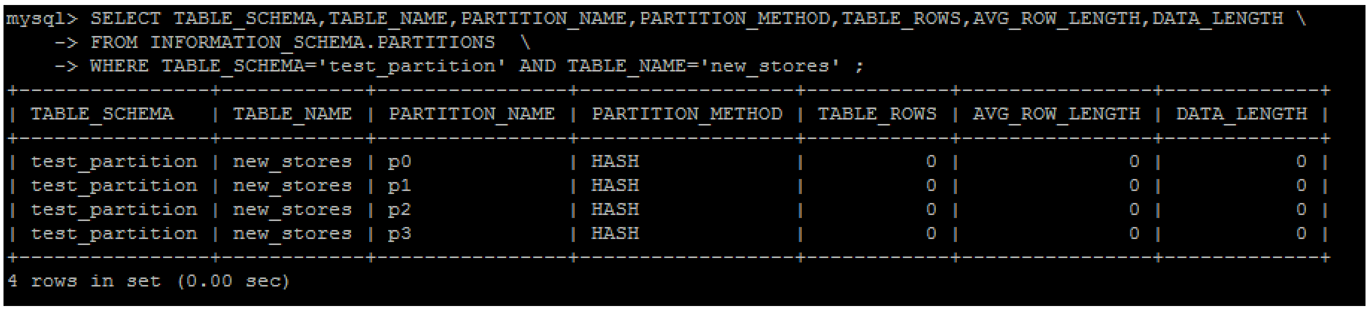
MySQL HASH Partitioning

- This partitioning is used to distribute data based on a predefined number of partitions.
- In other words, it splits the table as of the value returned by the user-defined expression.
- It is mainly used to distribute data evenly into the partition.
- It is performed with the PARTITION BY HASH(expr) clause. Here, we can specify a column value based on the column_name to be hashed and the number of partitions into which the table is divided.

This statement is used to create table Store using CREATE TABLE command and uses hashing on the store_id column that divided it into four partitions:

```
CREATE TABLE test_partition.new_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 HASH(store_id)
PARTITIONS 4;
```


```
mysql> CREATE TABLE test_partition.new_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 HASH(store_id)
    -> PARTITIONS 4;
Query OK, 0 rows affected (0.12 sec)
```


```
mysql> SELECT TABLE_SCHEMA,TABLE_NAME,PARTITION_NAME,PARTITION_METHOD,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH \
    -> FROM INFORMATION_SCHEMA.PARTITIONS  \
    -> WHERE TABLE_SCHEMA='test_partition' AND TABLE_NAME='new_stores' ;
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| TABLE_SCHEMA   | TABLE_NAME | PARTITION_NAME | PARTITION_METHOD | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
| test_partition | new_stores | p0             | HASH             |          0 |              0 |           0 |
| test_partition | new_stores | p1             | HASH             |          0 |              0 |           0 |
| test_partition | new_stores | p2             | HASH             |          0 |              0 |           0 |
| test_partition | new_stores | p3             | HASH             |          0 |              0 |           0 |
+----------------+------------+----------------+------------------+------------+----------------+-------------+
4 rows in set (0.00 sec)
```

NOTE: If you do not use the `PARTITIONS` clause, the number of partitions will be one by default. If you do not specify the number with the `PARTITIONS` keyword, it will throw an error.

MySQL COLUMN Partitioning

- This partitioning allows us to use the multiple columns in partitioning keys.
- The purpose of these columns is to place the rows in partitions and determine which partition will be validated for matching rows.

It is mainly divided into two types:
- RANGE Columns Partitioning
- LIST Columns Partitioning

They provide supports for the use of non-integer columns to define the ranges or value lists. They support the following data types:
- **All Integer Types:** TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT.
- **String Types:** CHAR, VARCHAR, BINARY, and VARBINARY.
- DATE and DATETIME data types.

Range Column Partitioning: It is similar to the range partitioning with one difference. It defines partitions using ranges based on various columns as partition keys. The defined ranges are of column types other than an integer type.

The following are the syntax for Range Columns Partitioning.

```
CREATE TABLE tab_name
PARTITIONED BY RANGE COLUMNS(colm_list) (
    PARTITION part_name VALUES LESS THAN (val_list)[,
    PARTITION parti_name VALUES LESS THAN (val_list)][,
    ...]
)
```

`colm_list`: It is a list of one or more columns.

```
    colm_name[, colm_name][, ...]
```

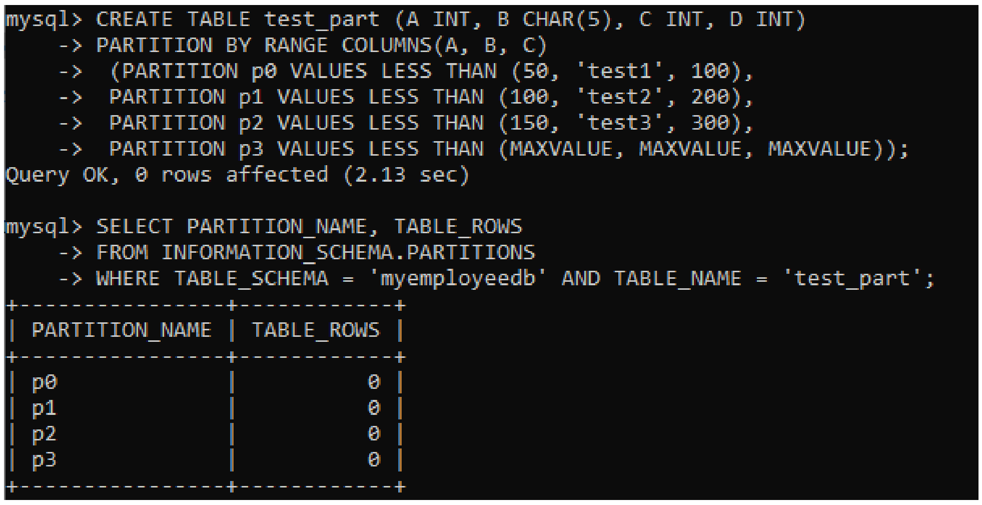
`val_list`: It is a list of values that supplied for each partition definition and have the same number of values as of columns.

```
  val[, val][, ...]
```

Consider the below example:

```
CREATE TABLE test_partition.test_part (A INT, B CHAR(5), C INT, D INT)
PARTITION BY RANGE COLUMNS(A, B, C)
 (PARTITION p0 VALUES LESS THAN (50, 'test1', 100),
 PARTITION p1 VALUES LESS THAN (100, 'test2', 200),
 PARTITION p2 VALUES LESS THAN (150, 'test3', 300),
 PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE));
```

- In this example, the table "`test_part`" contains the four columns A, B, C, and D.
- We have used the first three columns in partitioning in the order of A, B, C.
- And, each list value is used to define a partition that contains three values in the same order as **`INT`**, **`CHAR`**, and **`INT`**. After execution, we will get the output as below and verified by the **`SELECT`** statement successfully.



```
mysql> CREATE TABLE test_part (A INT, B CHAR(5), C INT, D INT)
    -> PARTITION BY RANGE COLUMNS(A, B, C)
    ->  (PARTITION p0 VALUES LESS THAN (50, 'test1', 100),
    ->  PARTITION p1 VALUES LESS THAN (100, 'test2', 200),
    ->  PARTITION p2 VALUES LESS THAN (150, 'test3', 300),
    ->  PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE));
Query OK, 0 rows affected (2.13 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    -> FROM INFORMATION_SCHEMA.PARTITIONS
    -> WHERE TABLE_SCHEMA = 'myemployeedb' AND TABLE_NAME = 'test_part';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          0 |
| p2             |          0 |
| p3             |          0 |
+----------------+------------+
```

List Columns Partitioning: It takes a list of single or multiple columns as partition keys. It enables us to use various columns of types other than integer types as partitioning columns. In this partitioning, we can use String data types, DATE, and DATETIME columns.

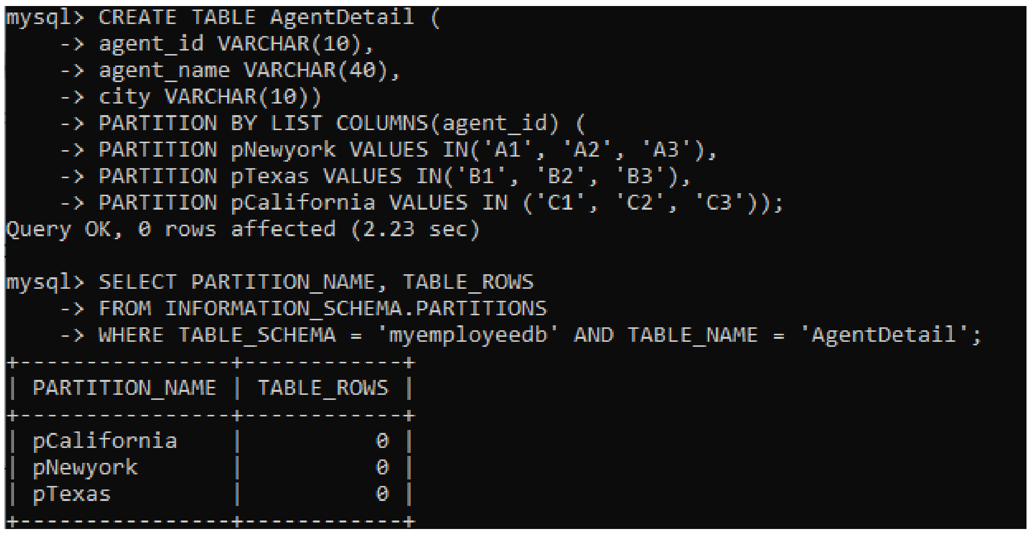
The following example explains it more clearly. Suppose a company has many agents in three cities for marketing purposes. We can organize it as below:

City	Marketing Agents
New York	A1, A2, A3
Texas	B1, B2, B3
California	C1, C2, C3

The following statement uses a List Columns Partitioning to organize the agents:

```
CREATE TABLE AgentDetail (
agent_id VARCHAR(10),
agent_name VARCHAR(40),
city VARCHAR(10))
PARTITION BY LIST COLUMNS(agent_id) (
PARTITION pNewyork VALUES IN('A1', 'A2', 'A3'),
PARTITION pTexas VALUES IN('B1', 'B2', 'B3'),
PARTITION pCalifornia VALUES IN ('C1', 'C2', 'C3'));
```

After the successful execution, we will get the output as below:



```
mysql> CREATE TABLE AgentDetail (
    -> agent_id VARCHAR(10),
    -> agent_name VARCHAR(40),
    -> city VARCHAR(10))
    -> PARTITION BY LIST COLUMNS(agent_id) (
    -> PARTITION pNewyork VALUES IN('A1', 'A2', 'A3'),
    -> PARTITION pTexas VALUES IN('B1', 'B2', 'B3'),
    -> PARTITION pCalifornia VALUES IN ('C1', 'C2', 'C3'));
Query OK, 0 rows affected (2.23 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    -> FROM INFORMATION_SCHEMA.PARTITIONS
    -> WHERE TABLE_SCHEMA = 'myemployeedb' AND TABLE_NAME = 'AgentDetail';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| pCalifornia    |          0 |
| pNewyork       |          0 |
| pTexas         |          0 |
+----------------+------------+
```

MySQL KEY Partitioning

- It is similar to the HASH partitioning where the hash partitioning uses the user-specified expression, and MySQL server supplied the hashing function for key.
- If we use other storage engines, the MySQL server employs its own internal hashing function that is performed by using the PARTITION BY KEY clause.
- Here, we will use KEY rather than HASH that can accept only a list of zero or more column names.
- If the table contains a PRIMARY KEY and we have not specified any column for partition, then the primary key is used as partitioning key.

The below example explains it more clearly:

```
CREATE TABLE AgentDetail (
    agent_id INT NOT NULL PRIMARY KEY,
    agent_name VARCHAR(40)
)
PARTITION BY KEY()
PARTITIONS 2;
```

If the table have unique key but not contains the primary key, then a UNIQUE KEY is used as a partition key.

```
CREATE TABLE AgentDetail (
    agent_id INT NOT NULL UNIQUE KEY,
    agent_name VARCHAR(40)
)
PARTITION BY KEY()
PARTITIONS 2;
```

SUBPARTITIONING

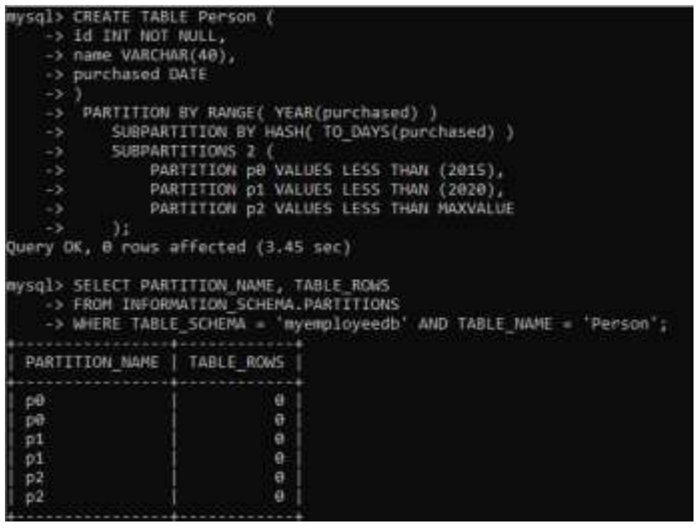
It is a composite partitioning that further splits each partition in a partition table. The below example helps us to understand it more clearly:

```
CREATE TABLE Person (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(40),
    purchased DATE
)
 PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (2015),
        PARTITION p1 VALUES LESS THAN (2020),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

Execute the below statement to verify the sub-partitioning:

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

It will give the output as below:



```
mysql> CREATE TABLE Person (
    -> id INT NOT NULL,
    -> name VARCHAR(40),
    -> purchased DATE
    -> )
    ->  PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (2015),
    ->         PARTITION p1 VALUES LESS THAN (2020),
    ->         PARTITION p2 VALUES LESS THAN MAXVALUE
    ->     );
Query OK, 0 rows affected (3.45 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    -> FROM INFORMATION_SCHEMA.PARTITIONS
    -> WHERE TABLE_SCHEMA = 'myemployeedb' AND TABLE_NAME = 'Person';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p0             |          0 |
| p1             |          0 |
| p1             |          0 |
| p2             |          0 |
| p2             |          0 |
+----------------+------------+
```