Name: Dhruv Bheda **SAPID:** 60004200102

DIV: B/B1

ADBMS

Exp5

Aim: To implement Fragmentation using Range, Key, Hash and List.

Theory:

Dhruv.A.Bheda
60004200102
B1

ADBMS - Exp 5

~~Theory:~~
Aim : Implement Fragmentation (Range, Key, Hash, List)

Theory :
Partitioning in MySQL is used to split a Table by row or by column. Accordingly we have 2 major types of Fragmentations
Horizontal (by tuple)
Vertical (by attribute)
Both of the fragmentations can be implemented according to need of the users. When we need some of the users & all of the attributes of those entries, there, we use Horizontal Fragmentation
When all the attributes are not necessary but performance of one of the aspect is important we use Vertical Fragmentation.

Types:-

Range Partitioning:-
It allows to specify various ranges for which data is assigned. Ranges should be continous but not overlapping and are defined using VALUE LESS THAN operator.

List Partitioning:-
It allows us to segment data based on a pre-defined set of values (eg:- 1, 2, 3). This is done by PARTITION BY LIST (expr). where expr

FOR EDUCATIONAL USE

is a column value & then defining each partitioning by means of a VALUE IN-(val_list) where val_list is a comma-seperated list of Integers.

Hash Partition :-

It is used to distribute data among a predifined number of partitions of a column value. This is done by using PARTITION BY HASH (expr) clause, adding in CREATE TABLE STATEMENT. In PARTITION'S num clause, num is a +ve integer representing the number of partitions of the table.

Key Partition :-

It is a special form of HASH partition where the hashing function for any partitioning is supplied by 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.

Conclusion :-

Thus, we implemented different types of fragmentation.

Output:

Range:

```sql
alter table film_year
PARTITION BY RANGE (year(film_years))(
PARTITION p0 VALUES LESS THAN (2016),
PARTITION p1 VALUES LESS THAN (2017),
PARTITION p2 VALUES LESS THAN (2018),
PARTITION p3 VALUES LESS THAN (2020));



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

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PARTITION_NAME	TABLE_ROWS
p0	2
p1	2
p2	2
p3	0

List:

Limit to 1000 rows

```sql
CREATE TABLE film_genre (

    category_id INT PRIMARY KEY NOT NULL

)

PARTITION BY LIST(category_id) (
PARTITION horror VALUES IN (101, 103, 105),
PARTITION comedy VALUES IN (102, 104, 106),
PARTITION actions VALUES IN (107, 109, 111),
PARTITION romance VALUES IN (108, 110, 112));

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

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PARTITION_NAME	TABLE_ROWS
actions	0
comedy	0
horror	0
romance	0

Result Grid

Form Editor

Hash:

```
CREATE TABLE ActorDetail (
    actor_id INT NOT NULL UNIQUE KEY,
    actor_name VARCHAR(40)
)
PARTITION BY KEY()
PARTITIONS 2;



Insert into ActorDetail values
 (1,'Salman'),
 (2,'SRK'),
 (3,'HERO');

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

Result Grid

PARTITION_NAME	TABLE_ROWS
p0	2
p1	1

Key:

```
CREATE TABLE ActorDetail (
    actor_id INT NOT NULL UNIQUE KEY,
    actor_name VARCHAR(40)
)
PARTITION BY KEY()
PARTITIONS 2;



Insert into ActorDetail values
 (1,'Salman'),
 (2,'SRK'),
 (3,'HERO');

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

Result Grid

PARTITION_NAME	TABLE_ROWS
p0	2
p1	1

Conclusion:

Thus, we implemented fragmentation using different techniques.