ADBMS

Name: Meet Patel SAP-ID: 60004200104

Division/Batch: B/B1 Branch: Computer Engineering

Experiment 5

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

ADBMS

Meet Patel 60004200104
B 1

Experiment - 5

Aim: Perform fragmentation (Range, List, Hash and Key) in DDBS design.

Theory: MySQL partitioning is about altering - ideally optimizing - the way the database engine physically store data. It allows you to distribute portion of table data partitions across file system based on a set of user - defined rules the partitioning function.

Types of partitioning:
Horizontal Partitioning: It means that all rows matching the partitioning function will

rows matching the partitioning function will be assigned to different physical functions

key	room no
1	66
2	89
3	77
4	54

→

key	room no
1	66
2	89

<u>Vertical Partitioning</u> : It allows different table columns to be split into different physical partitions

id	fname	lname

→

id	fname

Currently MySQL supports horizontal partitioning but not vertical partitioning

Partition Types:

<u>Range</u>: This type of partition assign ~~p~~row to partitions based on column values that fall within a stated range

<u>List</u>: It is similar to Range except that the partitioning is selected based on columns matching one of set of direct values.

<u>Hash</u>: In hash partitioning, a partition is selected based on values returned by a user defined expression.

<u>Key</u>: This is very similar to Hash partitioning but the hashing function is supplied by MySQL.

<u>Conclusion</u>: From personal experience, partitioning is the last part of optimization process. In general partitioning make the most sense when your dealing with million records.

OUTPUT:

RANGE:

```
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

```
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 | Filter Rows: | Export: | Wrap Cell Content:

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

PARTITION_NAME	TABLE_ROWS
p0	2
p1	1

CONCLUSION:

Partitioning is powerful functionality that allows tables, indexes, and index-organized tables tobe subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity.