Name: Kartik Jolapara **SAPID:** 60004200107

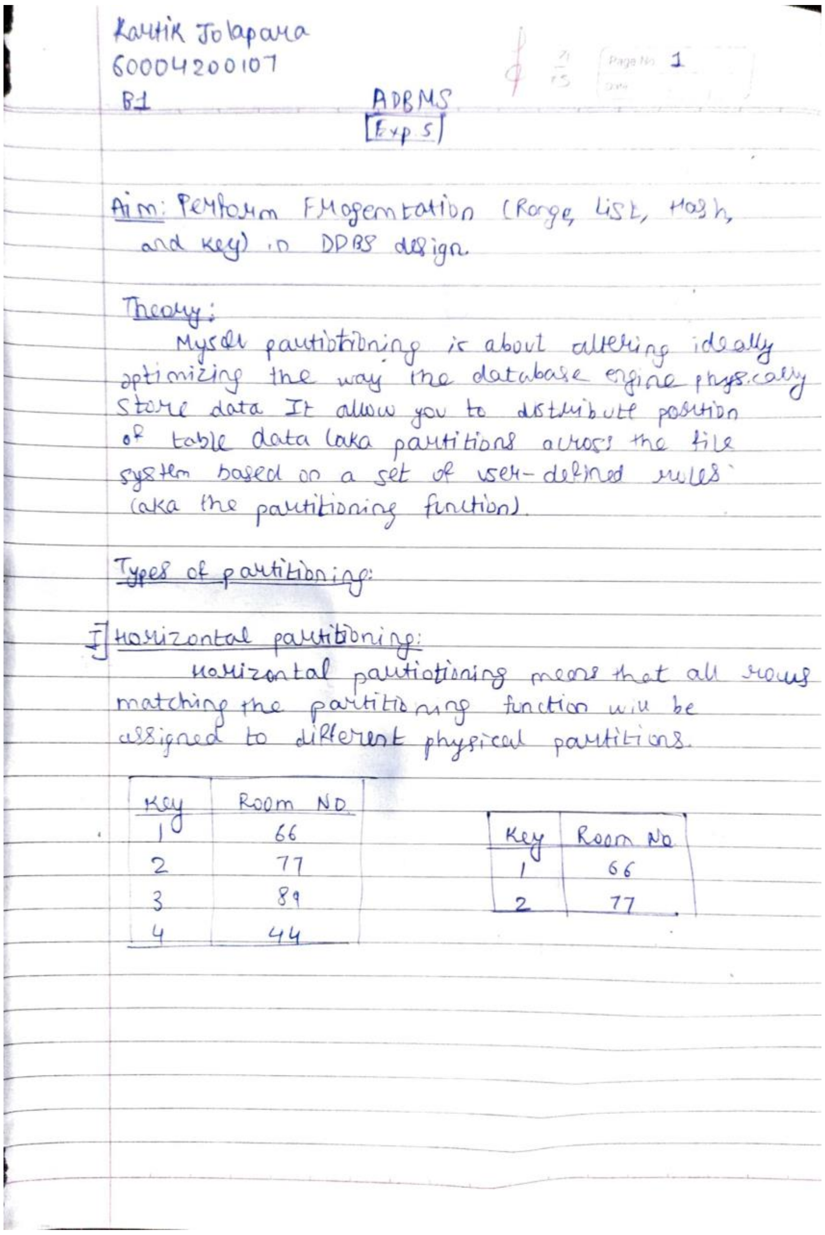
DIV: B/B1

ADBMS

Exp5

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

Theory:


Kartik Jolapara
60004200107
B1

Page No. 1

ADBMS
[Exp 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 allow you to distribute position of table data (aka partitions across the file system based on a set of user-defined rules (aka the partitioning function).

Types of partitioning:

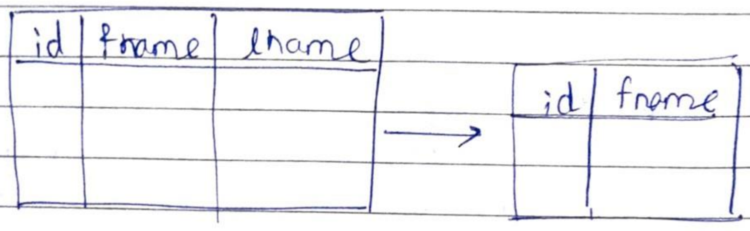
I] Horizontal partitioning:
Horizontal partitioning means that all rows matching the partitioning function will be assigned to different physical partitions.

Key	Room No
1	66
2	77
3	89
4	44

Key	Room No
1	66
2	77

II) <u>vertical partitioning.</u>

It allows different table columns to be split into different physical partitions.



currently MySQL supports horizontal partitioning but not vertical

<u>partition Types:-</u>

① <u>Range</u>

This type of partition assign row to partitions based on column values that fall within a stated range

② <u>List</u>

Partitioning is similar to RANGE except that the partion is selected based on columns matching one of of set of discrete values.

③ <u>Hash partitioning:</u>

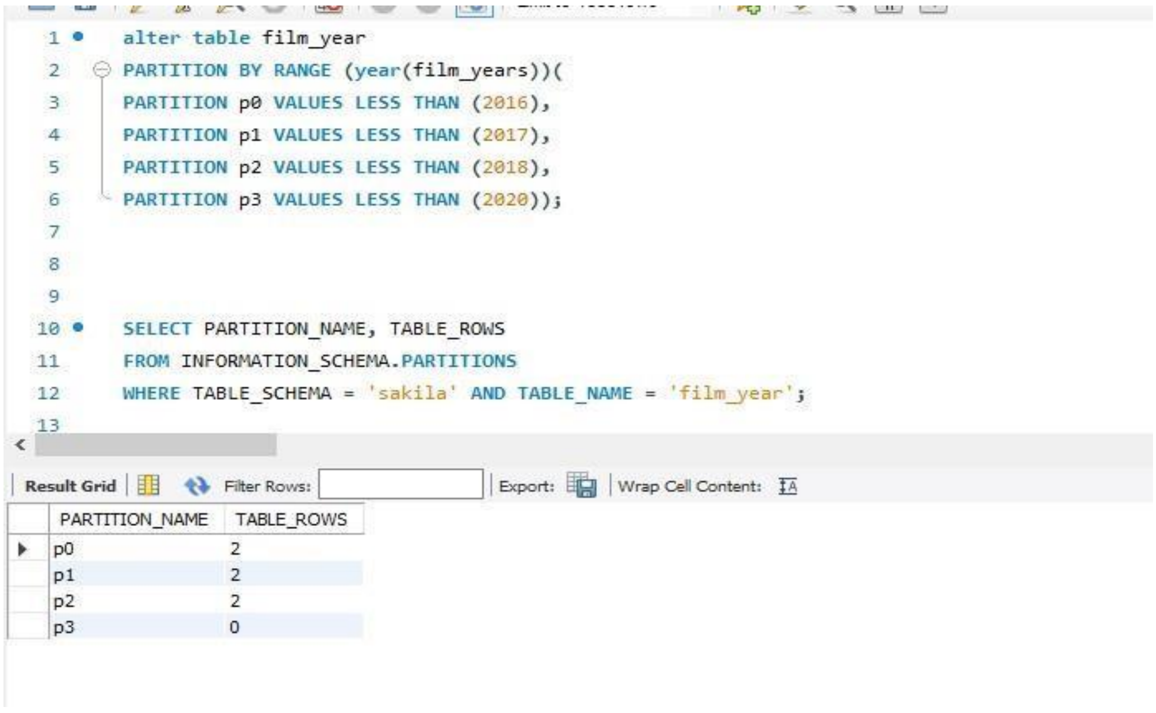
In hash partitioning a partition is selected based on the value returned by a user defined expression

④ Key
This very similar to HASH partitioning but the hashing function is supplied by MySQL.

Conclusion: From personal experience partitioning is the last part of an optimisation process I'd perform In general partitioning make the most sense when you are dealing with million record

<u>Output:</u>

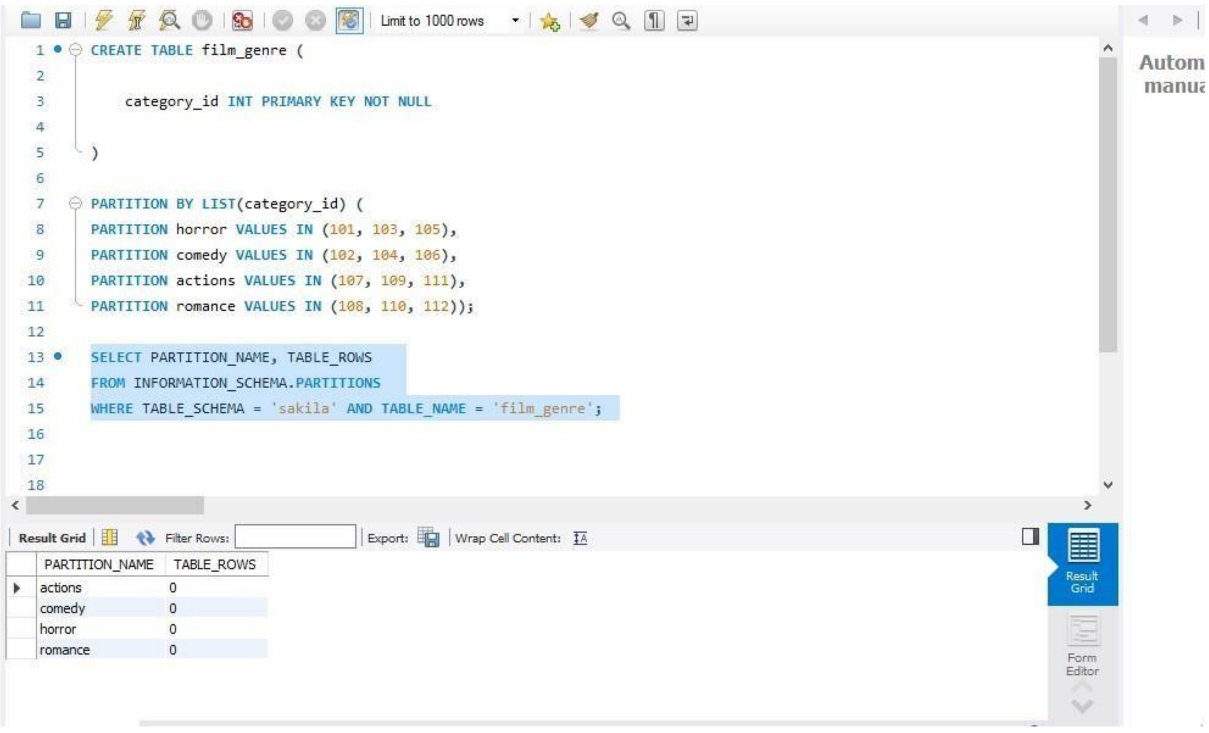
Range:



```
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';
```

PARTITION_NAME	TABLE_ROWS
p0	2
p1	2
p2	2
p3	0

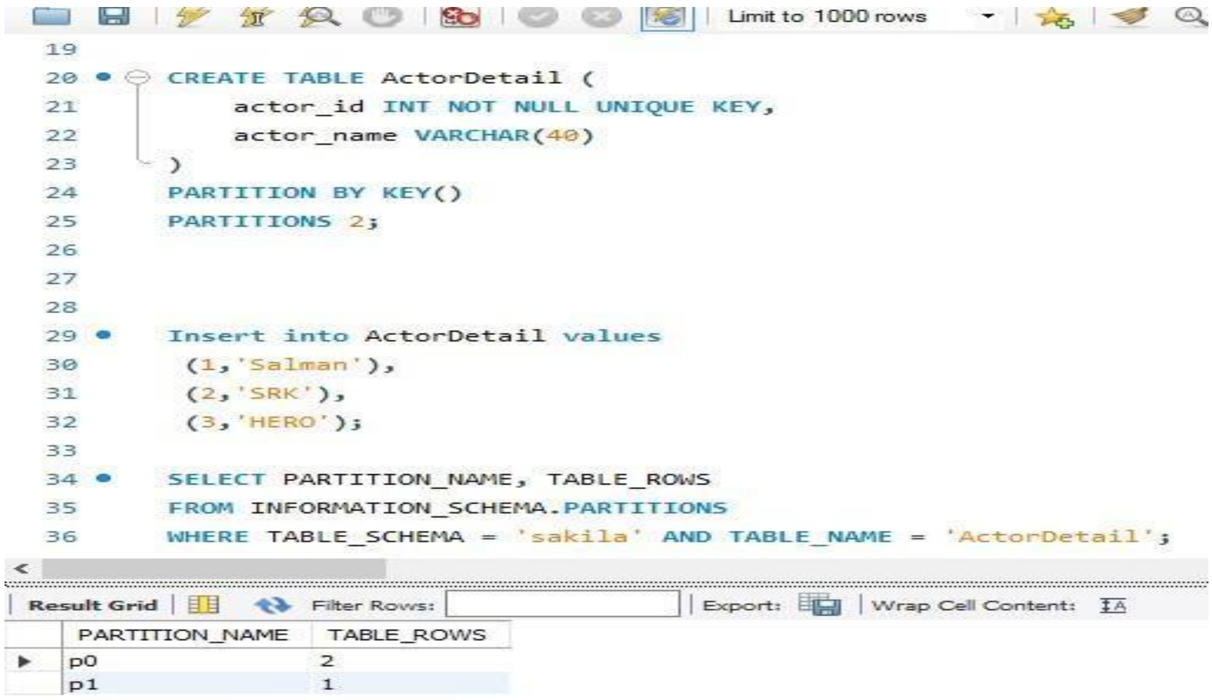
List:



```
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';
```

PARTITION_NAME	TABLE_ROWS
actions	0
comedy	0
horror	0
romance	0

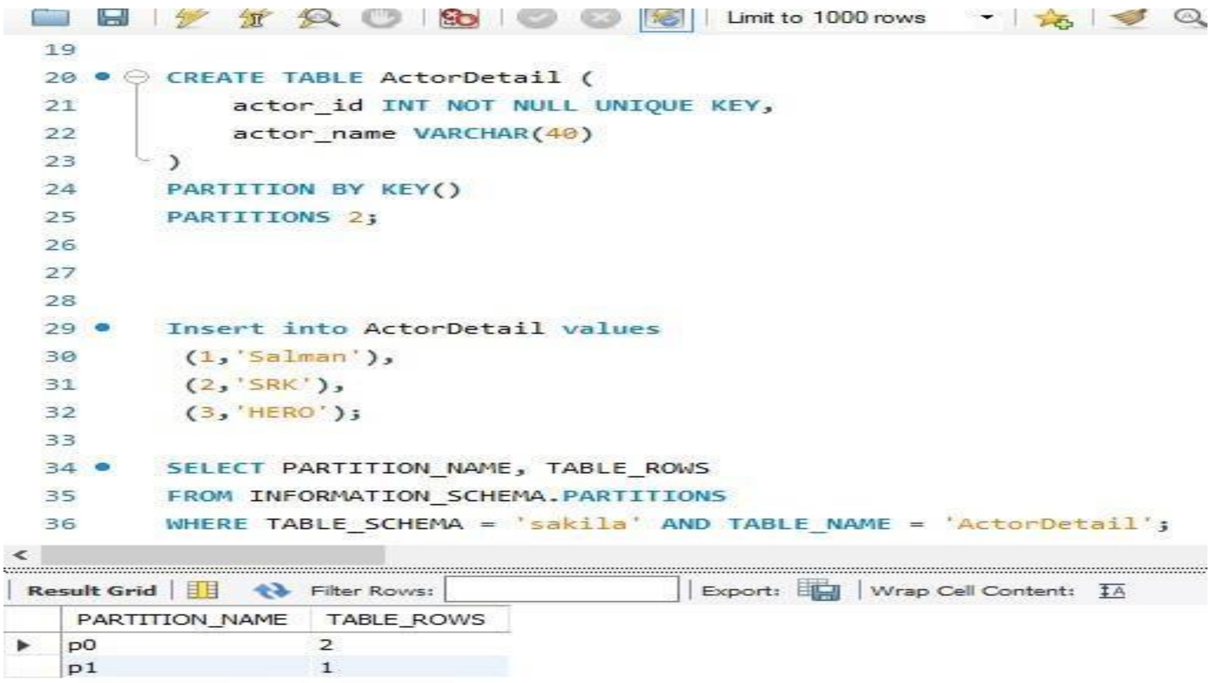
Hash:



```sql
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';
```

```
PARTITION_NAME   TABLE_ROWS
p0               2
p1               1
```

Key:



```sql
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';
```

```
PARTITION_NAME   TABLE_ROWS
p0               2
p1               1
```

Conclusion:

Thus, we implemented fragmentation using different techniques.