DBMS

EXPERIMENT – 8

Name: Ayush Jain **SAP ID**: 60004200132

Batch: B1 **Branch**: Computer Engineering

Aim: Create Views and Triggers.

Theory: A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view. Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name AS SELECT
column1, column2.....
FROM table_name
WHERE [condition];
```

1) View with where clause:

```
mysql> create view details as
    -> select song_title,length,genre,artist_name,rating
    -> from song,artist
    -> where song.artist_id = artist.artist_id;
Query OK, 0 rows affected (0.77 sec)

mysql> select * from details;
+---------------+----------+---------+---------------+--------+
| song_title    | length   | genre   | artist_name   | rating |
+---------------+----------+---------+---------------+--------+
| Peaches       | 00:03:18 | Pop     | Justin Bieber |      5 |
| Hello         | 00:04:55 | soul    | Adele         |     10 |
| Smile         | 00:03:16 | Hip hop | Juice WRLD    |      8 |
| Galway Girl   | 00:02:50 | Pop     | Ed Sheeran    |      9 |
| Perfect       | 00:04:23 | Pop     | Ed Sheeran    |      9 |
| Industry Baby | 00:03:32 | Pop rap | Lil Nas X     |      6 |
+---------------+----------+---------+---------------+--------+
6 rows in set (0.02 sec)
```

2) View with group by function:

```
mysql> create view userInfo as
    -> select username, mobileNo ,email,startDate,expiryDate
    -> from user
    -> Inner Join subscription
    -> On user.sub_id = subscription.sub_id;
Query OK, 0 rows affected (0.48 sec)

mysql> select * from userInfo;
+-------------+-----------+--------------------------+-----------+------------+
| username    | mobileNo  | email                    | startDate | expiryDate |
+-------------+-----------+--------------------------+-----------+------------+
| meetp       | 9557849834 | meetp2002@gmail.office.com | 2021-12-30 | 2022-12-30 |
| codingmickey | 9619247188 | kjmickey003@gmail.com    | 2021-12-30 | 2022-12-30 |
| Deevya      | 9763657385 | deevyam@gmail.com        | 2021-12-30 | 2022-01-30 |
+-------------+-----------+--------------------------+-----------+------------+
3 rows in set (0.02 sec)
```

3) View with group by and order by function:

```
mysql> create view varity as
    -> select genre,COUNT(*) as 'number of songs'
    -> from song
    -> group by genre;
Query OK, 0 rows affected (0.11 sec)

mysql> select * from varity;
+---------+-----------------+
| genre   | number of songs |
+---------+-----------------+
| Hip hop |               1 |
| Pop     |               4 |
| Pop rap |               1 |
| soul    |               1 |
+---------+-----------------+
4 rows in set (0.10 sec)
```

Trigger:

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the table. The statement CREATE TRIGGER creates a new trigger in MySQL.

Syntax

CREATE
[DEFINER = { user | CURRENT_USER }] TRIGGER
trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body trigger_time: {
BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }

1) Trigger with default value of insert:

```
mysql> create trigger salcheck before insert on employee for each row
    -> if new.salary is null then
    -> set new.salary = 12000;
    -> end if;
    -> //
Query OK, 0 rows affected (0.12 sec)

mysql> insert into employee(employee_id, Fname, age , sex, working_hours, department)
    -> values (106, "Scott", 20 , "M", 4, "electronics games");
    -> //
Query OK, 1 row affected (0.05 sec)

mysql> select * from employee;
    -> //
+-------------+---------+------+--------------+------+--------+--------------+------------------+
| employee_id | Fname   | age  | phone_number | sex  | salary | working_hours | department       |
+-------------+---------+------+--------------+------+--------+--------------+------------------+
|         101 | John    |   25 |         NULL | M    |  25000 |             6 | cashier          |
|         102 | Yogesh  |   19 |         NULL | M    |  23000 |             5 | casual games     |
|         103 | Lydia   |   24 |         NULL | F    |  30000 |             7 | kid's games      |
|         104 | Alisson |   23 |         NULL | F    |  27000 |             5 | electronic games |
|         105 | Hetvi   |   18 |         NULL | F    |  33000 |             8 | manager          |
|         106 | Scott   |   20 |         NULL | M    |  12000 |             4 | electronics games |
+-------------+---------+------+--------------+------+--------+--------------+------------------+
6 rows in set (0.00 sec)
```

2) Trigger by creating backup table and then insert:

```
mysql> create table product_bkp(
    -> product_name varchar(50),
    -> product_id int not null primary key,
    -> stock varchar(3),
    -> employee_id int,
    -> foreign key (employee_id) references employee(employee_id),
    -> price int not null,
    -> department varchar(20));
Query OK, 0 rows affected (0.18 sec)

mysql> desc product_bkp;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| product_name  | varchar(50) | YES  |     | NULL    |       |
| product_id    | int         | NO   | PRI | NULL    |       |
| stock         | varchar(3)  | YES  |     | NULL    |       |
| employee_id   | int         | YES  | MUL | NULL    |       |
| price         | int         | NO   |     | NULL    |       |
| department    | varchar(20) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
mysql> delimiter //
mysql> create trigger backup after insert on product
    -> for each row insert into product_bkp (product_name, product_id, stock, employee_id, price, department)
    -> values (new.product_name, new.product_id, new.stock, new.employee_id, new.price, new.department);
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> insert into product
    -> values ("Monopoly", 34516, "yes", 102, 3000, "casual games");
    -> //
Query OK, 1 row affected (0.04 sec)

mysql> select * from product_bkp;
    -> //
+--------------+------------+-------+-------------+-------+--------------+
| product_name | product_id | stock | employee_id | price | department   |
+--------------+------------+-------+-------------+-------+--------------+
| Monopoly     |      34516 | yes   |         102 |  3000 | casual games |
+--------------+------------+-------+-------------+-------+--------------+
1 row in set (0.00 sec)

mysql>
```

3) Trigger with update:

```
mysql> create trigger supptrig before update on supplier for each row
    -> if new.manufacturer_id is null then set new.manufacturer_id = 001;
    -> end if;
    -> //
Query OK, 0 rows affected (0.06 sec)
```

4) Drop a trigger:

```
mysql> drop trigger backup;
Query OK, 0 rows affected (0.02 sec)
```

Conclusion: Views, trigger performed on database.