Database Management Systems Lab Experiment No: - 09

Aim: - Implementation of Views & Triggers

Views:-

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

Just like table query, we can query the view to view the data:

SELECT * FROM view_name;

Deleting View:-

A view can be deleted using the Drop View statement.

Syntax

DROP VIEW view_name;

1) **CODE**

```
mysql> use S21_89;
Database changed
mysql> create view V1 as
    -> select Client_no, Name, City
    -> from Client_master
    -> where City in ('Bombay');
Query OK, 0 rows affected (0.02 sec)
```

Aim: - Implementation of Views & Triggers

OUTPUT

```
mysql> select * from V1;
+-----------+-----------------+--------+
| Client_no | Name            | City   |
+-----------+-----------------+--------+
| C00001    | Ivan Baryons    | Bombay |
| C00003    | Pramada Jaguste | Bombay |
| C00004    | Basu Navindgi   | Bombay |
| C00006    | Rukmini         | Bombay |
+-----------+-----------------+--------+
4 rows in set (0.00 sec)
```

2) CODE

```
mysql> create view V2 as
    -> select c.Name, c.City, s.Order_status
    -> from Client_master c, sales_order s
    -> where c.Client_no = s.Client_no;
Query OK, 0 rows affected (0.01 sec)
```

OUTPUT

```
mysql> select * from V2;
+-----------------+--------+--------------+
| Name            | City   | Order_status |
+-----------------+--------+--------------+
| Ravi Sreedharan | Delhi  | IN PROCESS   |
| Vandana Saitwal | Madras | CANCELLED    |
| Ivan Baryons    | Bombay | FULFILLED    |
| Pramada Jaguste | Bombay | Fulfilled    |
| Basu Navindgi   | Bombay | CANCELLED    |
+-----------------+--------+--------------+
5 rows in set (0.00 sec)
```

Database Management Systems Lab Experiment No: - 09

Aim: - Implementation of Views & Triggers

Triggers:-

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

Types of Triggers :

Depending upon, when a trigger is fired, it may be classified as :

- Statement-level trigger
- Row-level trigger
- Before triggers
- After triggers

Statement-level Triggers :

Database Management Systems Lab Experiment No: - 09

Aim: - Implementation of Views & Triggers

A statement trigger is fired only for once for a DML statement irrespective of the number of rows affected by the statement.

For example, if you execute the following UPDATE command STUDENTS table, statement trigger for UPDATE is executed only for once.

```
update students set bcode='b3'
where bcode = 'b2';
```

However, statements triggers cannot be used to access the data that is being inserted, updated or deleted. In other words, they do not have access to keywords NEW and OLD, which are used to access data. Statement-level triggers are typically used to enforce rules that are not related to data.

For example, it is possible to implement a rule that says "no body can modify BATCHES table after 9 P.M".

Statement-level trigger is the default type of trigger.

Row-level Trigger :

A row trigger is fired once for each row that is affected by DML command.
For example, if an UPDATE command updates 100 rows then row-level trigger is fired 100 times whereas a statement-level trigger is fired only for once.

Row-level trigger are used to check for the validity of the data. They are typically used to implement rules that cannot be implemented by integrity constraints. Row-level triggers are implemented by using the option FOR EACH ROW in CREATE TRIGGER statement

Before Triggers :

While defining a trigger, you can specify whether the trigger is to be fired before the command (INSERT, DELETE, and UPDATE) is executed or after the command is executed. Before triggers are commonly used to check the validity of the data before the action is performed. For instance, you can use before trigger to prevent deletion of row if deletion should not be allowed in the given case.

AFTER Triggers :

After triggers are fired after the triggering action is completed. For example, If after trigger is associated with INSERT command then it is fired after the row is inserted into the table

1)

```
create table Student
(roll_no int(2),
```

```
name varchar(20),
city varchar(15),
state varchar(20)
);

delimiter $$
create trigger tg1
before insert on Student
for each row
begin
set new.name= upper(new.name);
set new.city= upper(new.city);
end;
$$

insert into Student values(77,'ovesh','mumbai','Maharashtra');
insert into Student values(78,'ujjwal','pune','Maharashtra');
insert into Student values(56,'Sadiya','surat','Gujurat');
$$

select *from Student;
$$
```

OUTPUT

```
mysql> select *from Student;
    -> $$
+---------+--------+--------+-------------+
| roll_no | name   | city   | state       |
+---------+--------+--------+-------------+
|      77 | OVESH  | MUMBAI | Maharashtra |
|      78 | UJJWAL | PUNE   | Maharashtra |
|      56 | SADIYA | SURAT  | Gujurat     |
+---------+--------+--------+-------------+
3 rows in set (0.00 sec)
```

2)

```
create table person
(fname char(10),
lname char(10),
id decimal(8) primary key
);
$$

create table audit_log
(ofname char(10),
olname char(10),
nfname char(10),
nname char(10),
cwhen date
);
$$

delimiter $$
create trigger t2
```

```
after update on person
for each row

begin
insert into audit_log
values(old.fname,old.lname,new.fname,new.lname,curdate()
);
end;
$$

insert into person values('ovesh','patel',1);
insert into person values('ayushi','shah',2);
$$

update person set fname='ovi' where id like 1;
$$

select * from audit_log;
$$
```

OUTPUT

```
mysql> select*from audit_log;
    -> $$
+--------+--------+--------+-------+------------+
| ofname | olname | nfname | nname | cwhen      |
+--------+--------+--------+-------+------------+
| ovesh  | patel  | ovi    | patel | 2024-04-02 |
+--------+--------+--------+-------+------------+
1 row in set (0.00 sec)
```

Conclusion:- LO mapped LO3,LO4.