Batch: D2 Roll No.: 16010122323 **Experiment / assignment / tutorial No. 5** **Grade: AA / AB / BB / BC / CC / CD /DD**

Title: Queries based Views and Triggers

Objective: To be able to use SQL view and triggers.

Expected Outcome of Experiment:

CO 2	Develop relational database design using the designed Entity-Relationship model.
CO 3	Use SQL for Relational database creation, maintenance and query processing

Books/ Journals/ Websites referred:

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Slberchatz, Sudarshan : "Database Systems Concept", 5th Edition , McGraw Hill
4. Elmasri and Navathe,"Fundamentals of database Systems", 4th Edition,PEARSON Education.

Resources used: Postgresql

Theory

Views are pseudo-tables. That is, they are not real tables; nevertheless appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, you can use them to restrict table access so that the users see only specific rows or columns of a table.

A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL 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 only see limited data instead of complete table.
- Summarize data from various tables, which can be used to generate reports.

Since views are not ordinary tables, you may not be able to execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a RULE to correct this problem of using DELETE, INSERT or UPDATE on a view.

Syntax

CREATE [TEMP | TEMPORARY] VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];

Ex:

CREATE VIEW COMPANY-VIEW AS

SELECT ID, NAME, AGE

FROM COMPANY;

select * from Company-View

Insert into Company-View values (123,'alpha', 10)

select * from Company

Dropping Views

Syntax: DROP VIEW view_name;

Triggers

The basic syntax of creating a trigger is as follows –

CREATE TRIGGER trigger_name [BEFORE|AFTER|INSTEAD OF] event_name

ON table_name

[

-- Trigger logic goes here....

];

event_name could be INSERT, DELETE, UPDATE, and TRUNCATE database operation on the mentioned table table_name. You can optionally specify FOR EACH ROW after table name.

The following is the syntax of creating a trigger on an UPDATE operation on one or more specified columns of a table as follows –

CREATE TRIGGER trigger_name [BEFORE|AFTER] UPDATE OF column_name

ON table_name

[

-- Trigger logic goes here....

];

Example :

creates a log table and a trigger that inserts a row in the log table after any **UPDATE** statement affects the **SALARY** column of the **EMPLOYEES** table, and then pdates **EMPLOYEES**.**SALARY** and shows the log table.

CREATE TABLE Emp_log (Emp_id NUMBER, Log_date DATE, New_salary NUMBER, Action VARCHAR2(20));

- Create trigger that inserts row in log table after EMPLOYEES.SALARY is update

CREATE OR REPLACE TRIGGER log_salary_increase

AFTER UPDATE OF salary ON employees

FOR EACH ROW BEGIN INSERT INTO Emp_log (Emp_id, Log_date, New_salary, Action) VALUES (:NEW.employee_id, SYSDATE, :NEW.salary, 'New Salary');

END;

Update EMPLOYEES.SALARY:

UPDATE employees SET salary = salary + 1000.0 WHERE Department_id = 20;

Result:

2 rows updated. Show log table:

SELECT * FROM Emp_log;

Result:

EMP_ID LOG_DATE NEW_SALARY ACTION

201 28-APR-10 15049.13 New Salary

202 28-APR-10 6945.75 New Salary

2 rows selected.

Implementation Screenshots (Problem Statement, Query and Screenshots of Results):

1) Created a table DONOR

```
1  CREATE TABLE DONOR (
2      DONOR_ID INT NOT NULL,
3      DONOR_NAME VARCHAR(50) NOT NULL,
4      PASSKEY VARCHAR(16) NOT NULL,
5      DONATION_HISTORY INTEGER,
6      PHONE_NUMBER VARCHAR(10),
7      EMAIL VARCHAR(30),
8      PRIMARY KEY (DONOR_ID)
9  );
```

	donor_id [PK] integer	donor_name character varying (50)	passkey character varying (16)	donation_history integer	phone_number character varying (10)	email character varying (30)
1	1	JIYA	jiya123	50000	8282828282	jiya@gmail.com
2	2	Rahul	rahul456	200000	7878787878	rahul@gmail.com
3	3	Aisha	aisha789	100000	9090909091	aisha@gmail.com

2) Created Table NGO

```
15  CREATE TABLE NGO (
16      NGO_ID INT NOT NULL,
17      NGO_NAME VARCHAR(50) NOT NULL,
18      ADDRESS VARCHAR(100) NOT NULL,
19      DONATION_RECIEVED INTEGER,
20      PHONE_NUMBER VARCHAR(10),
21      EMAIL VARCHAR(30),
22      DONOR_ID INT NOT NULL,
23      PRIMARY KEY (NGO_ID),
24      FOREIGN KEY (DONOR_ID) REFERENCES DONOR
25  );
```

	ngo_id [PK] integer	ngo_name character varying (50)	address character varying (100)	donation_recieved integer	phone_number character varying (10)	email character varying (30)	donor_id integer
1	1	ROBIN HOOD ARMY	MUMBAI	50000	9090909090	robinhood@gmail.com	1
2	2	Greenpeace	New Delhi	200000	8080808080	greenpeace@gmail.com	2
3	3	CARE India	Kolkata	100000	7070707070	careindia@gmail.com	3

3) Created a view donor_NGO_view (combining NGO and DONOR table)

```
147 CREATE VIEW Donor_NGO_View AS
148 SELECT D.DONOR_ID, D.DONOR_NAME, D.DONATION_HISTORY, N.NGO_NAME
149 FROM DONOR D
150 JOIN NGO N ON D.DONOR_ID = N.DONOR_ID;
```

	donor_id integer	donor_name character varying (50)	donation_history integer	ngo_name character varying (50)
1	1	JIYA	50000	ROBIN HOOD ARMY
2	2	Rahul	200000	Greenpeace
3	3	Aisha	100000	CARE India

4) Created a trigger on donor_NGO_View

```
152 CREATE OR REPLACE FUNCTION update_donor_ngo_view()
153 RETURNS TRIGGER AS
154 $$
155 BEGIN
156   IF TG_OP = 'UPDATE' THEN
157
158     UPDATE DONOR
159     SET DONATION_HISTORY = NEW.DONATION_HISTORY
160     WHERE DONOR_ID = NEW.DONOR_ID;
161
162     RETURN NEW;
163   END IF;
164 END;
165 $$
166 LANGUAGE plpgsql;
167
168 CREATE TRIGGER instead_of_update_donor_ngo_view
169 INSTEAD OF UPDATE ON Donor_NGO_View
170 FOR EACH ROW
171 EXECUTE FUNCTION update_donor_ngo_view();
```

5) Updated trigger

```
173 UPDATE Donor_NGO_View
174 SET DONATION_HISTORY = DONATION_HISTORY - 50000
175 WHERE DONOR_ID = 3;
```

	donor_id integer	donor_name character varying (50)	donation_history integer	ngo_name character varying (50)
1	1	JIYA	50000	ROBIN HOOD ARMY
2	2	Rahul	200000	Greenpeace
3	3	Aisha	50000	CARE India

Conclusion: The experiment on Views and triggers was learnt and implemented successfully

Post Lab Questions:

1. **What is a view?**

   a) A view is a special stored procedure executed when certain event occurs
   b) A view is a virtual table which results of executing a pre-compiled query
   c) A view is a database diagram
   d) None of the Mentioned

2. **List Advantages and disadvantages of triggers**

   Advantages:

   - Triggers help enforce business rules and maintain data integrity by automatically executing predefined actions when specific events occur, ensuring that the data conforms to the desired rules.
   - Triggers allow the execution of complex actions that involve multiple steps or multiple tables in response to a single event.
   - They automate repetitive tasks, reducing the need for manual intervention and minimizing the chances of errors.

   Disadvantages:

   - Triggers can have a performance impact because they are executed automatically whenever the specified event occurs. Poorly designed triggers or triggers on frequently modified tables can lead to performance degradation.
   - Triggers can introduce complexity to the database schema and logic. Managing and maintaining complex trigger logic can be challenging, especially in large databases.
   - Triggers are implicitly executed, and developers may not be immediately aware of their existence, potentially leading to unexpected behavior if not properly documented.