Batch: B-2	**Roll No:-** 16010122151
Experiment / assignment / tutorial No. 5	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

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 COMPAN-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):

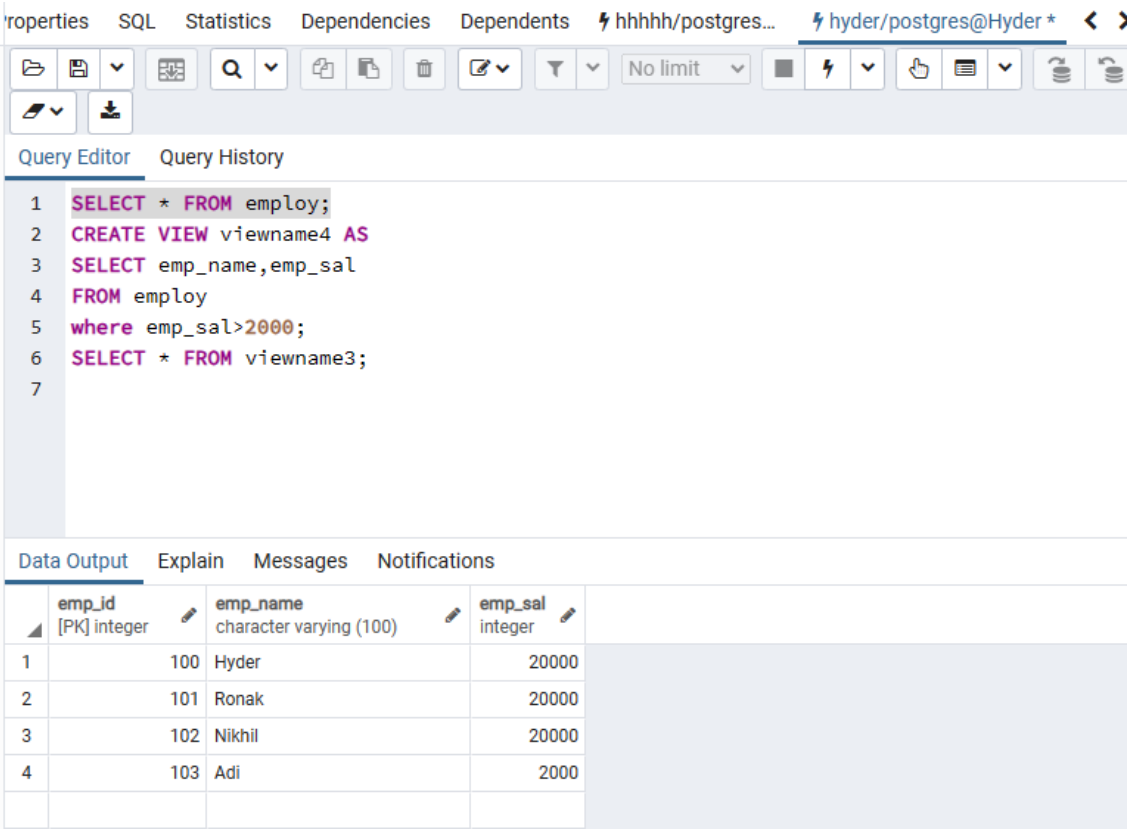
View Example:-

Code:-

```
SELECT * FROM employ;
CREATE VIEW viewname3 AS
SELECT emp_name,emp_sal
FROM employ
where emp_sal>2000;
SELECT * FROM viewname3;
```

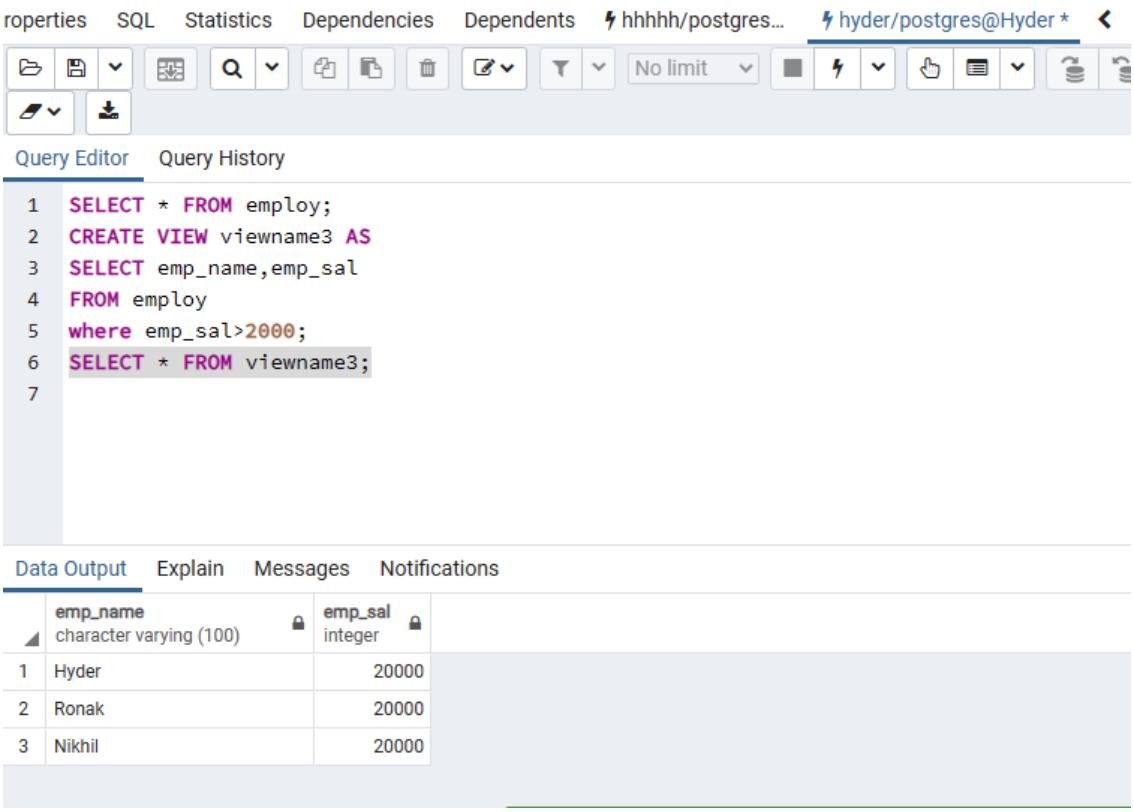
Output:-

Database :-



Output using view

Now it is only showing employee_name and employee_salary
Note:- We are also hiding employee_id



Trigger:-

Code:-

```
CREATE TABLE COMPANY (
ID INT PRIMARY KEY NOT NULL,
NAME TEXT NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (50),
SALARY REAL );
CREATE TABLE AUDIT (
        EMP_ID INT NOT NULL,
        ENTRY_DATE TEXT NOT NULL
```

```
        );
-- Create the auditlogfunc function
CREATE OR REPLACE FUNCTION auditlogfunc() RETURNS
TRIGGER AS $example_table$
BEGIN
   INSERT INTO AUDIT (EMP_ID, ENTRY_DATE) VALUES
(NEW.ID, CURRENT_TIMESTAMP);
   RETURN NEW;
END;
$example_table$ LANGUAGE plpgsql;

-- Create the example_trigger trigger
CREATE TRIGGER example_trigger AFTER INSERT ON
COMPANY
   FOR EACH ROW EXECUTE FUNCTION auditlogfunc();

        INSERT INTO COMPANY (ID, NAME, AGE, ADDRESS,
SALARY)
        VALUES (1, 'Paul', 32, 'California', 20000.00);
        SELECT * FROM COMPANY
        SELECT * FROM AUDIT
        SELECT * FROM pg_trigger
```

Created a table COMPANY

```sql
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL
);
```

Data Output | Explain | Messages | Notifications

id [PK] integer	name text	age integer	address character (50)	salary real

Created an audit table

```sql
CREATE TABLE AUDIT(
   EMP_ID INT NOT NULL,
   ENTRY_DATE TEXT NOT NULL
);
```

Data Output | Explain | Messages | Notifications

emp_id integer	entry_date text

Created a trigger on COMPANY table

```
CREATE TRIGGER example_trigger AFTER INSERT ON COMPANY
FOR EACH ROW EXECUTE PROCEDURE auditlogfunc();

CREATE OR REPLACE FUNCTION auditlogfunc() RETURNS TRIGGER AS $example_table$
   BEGIN
      INSERT INTO AUDIT(EMP_ID, ENTRY_DATE) VALUES (new.ID, current_timestamp);
      RETURN NEW;
   END;
$example_table$ LANGUAGE plpgsql;
```

Added values to COMPANY

Query Editor Query History

```
INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Paul', 32, 'California', 20000.00 );
```

Data Output Explain Messages Notifications

	id [PK] integer	name text	age integer	address character (50)	salary real
1	1	Paul	32	California	20000

Audit table gets updated

Data Output | Explain | Messages | Notifications

	emp_id integer	entry_date text
1	1	2023-04-03 11:...

Listing all TRIGGERS

Query Editor | Query History

```
SELECT * FROM pg_trigger;
```

Data Output | Explain | Messages | Notifications

	oid oid	tgrelid oid	tgname name	tgfoid oid	tgtype smallint	tgenabled "char" (1)	tgisinternal boolean	tgconstrrelid oid	tgconstrindid oid	tgco oid
1	16573	16558	example_tri...	16572	5	O	false	0	0	

Output:-

hyderdatabase/postgres@Hyderserver

Query | Query History | Scratch Pad

```
END;
$example_table$ LANGUAGE plpgsql;

-- Create the example_trigger trigger
CREATE TRIGGER example_trigger AFTER INSERT ON COMPANY
    FOR EACH ROW EXECUTE FUNCTION auditlogfunc();

    INSERT INTO COMPANY (ID, NAME, AGE, ADDRESS, SALARY)
    VALUES (1, 'Paul', 32, 'California', 20000.00);
    SELECT * FROM COMPANY
    SELECT * FROM AUDIT
```

Data Output | Messages | Notifications

	id [PK] integer	name text	age integer	address character	salary real
1	1	Paul	32	California	20000

hyderdatabase/postgres@Hyderserver

Query | Query History | Scratch Pad

```
END;
$example_table$ LANGUAGE plpgsql;

-- Create the example_trigger trigger
CREATE TRIGGER example_trigger AFTER INSERT ON COMPANY
    FOR EACH ROW EXECUTE FUNCTION auditlogfunc();

    INSERT INTO COMPANY (ID, NAME, AGE, ADDRESS, SALARY)
    VALUES (1, 'Paul', 32, 'California', 20000.00);
    SELECT * FROM COMPANY
    SELECT * FROM AUDIT
```

Data Output | Messages | Notifications

	emp_id integer	entry_date text
1	1	2024-03-16 12:42:57.617221+05:30

Query Query History Scratch Pad

```
$example_table$ LANGUAGE plpgsql;

-- Create the example_trigger trigger
CREATE TRIGGER example_trigger AFTER INSERT ON COMPANY
    FOR EACH ROW EXECUTE FUNCTION auditlogfunc();

    INSERT INTO COMPANY (ID, NAME, AGE, ADDRESS, SALARY)
    VALUES (1, 'Paul', 32, 'California', 20000.00);
    SELECT * FROM COMPANY
    SELECT * FROM AUDIT
    SELECT * FROM pg_trigger
```

Data Output Messages Notifications

	oid [PK] oid	tgrelid oid	tgparentid oid	tgname name	tgfoid oid	tgtype smallint	tgenabled "char"	tgisinternal boolean	tgconstrrelid oid	tgco oid
1	16543	16530	0	example_trigger	16542	5	O	false	0	

Conclusion:- Views and Triggers are essential tools for database management, offering increased efficiency, security, and maintainability to SQL databases.

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 of Triggers:-

1. **Automation:-** Triggers automate tasks based on database events, reducing manual effort.
2. **Data Integrity:-**They ensure data correctness by enforcing rules before changes are made.
3. **Audit Trail:-** Triggers help track changes, providing a history for compliance or analysis.

Disadvantages of Triggers:-

1. **Performance Impact:-**They can slow down database operations, especially with complex logic.
2. **Hidden Logic:-** Triggers hide logic within the database, making debugging harder.
3. **Concurrency Problems:-** They may lead to issues when multiple users access the database simultaneously.