EXPERIMENT - 7

Name: Neha Sharma	**UID**: 23BCS10766
Branch: CSE	**Section**: KRG1-B
Semester: 5	**Date of Performance**: 23/10/2025
Subject: ADBMS	**Subject Code**: 23CSP-333

Medium-Level Problem

1. **Aim:** Design a trigger in PostgreSQL which, whenever there is an insertion or deletion on the student table, prints the inserted or deleted row exactly as it is on the output console window.

2. **Objective:**
   - To understand how PostgreSQL triggers work.
   - To use `NEW` and `OLD` pseudo-records for accessing row data.
   - To display the affected record dynamically upon data changes.

3. **DBMS script and output:**

```
-- Creating Table
CREATE TABLE student (
   id SERIAL PRIMARY
   KEY, name
   VARCHAR(100),
   age INT,
   class VARCHAR(20)
);
```

```
-- Creating Trigger Function
CREATE OR REPLACE FUNCTION
fn_student_audit()
RETURNS TRIGGER
LANGUAGE
plpgsql AS
$$
BEGIN
    IF TG_OP = 'INSERT' THEN
      RAISE NOTICE 'Inserted Row -> ID: %, Name: %, Age: %,
            Class: %', NEW.id, NEW.name, NEW.age,
            NEW.class;
      RETURN NEW;

    ELSIF TG_OP = 'DELETE' THEN
      RAISE NOTICE 'Deleted Row -> ID: %, Name: %, Age: %,
            Class: %', OLD.id, OLD.name, OLD.age, OLD.class;
      RETURN
    OLD;
    END IF;

    RETURN
  NULL; END;
  $$;

  -- Creating Trigger
  CREATE TRIGGER
  trg_student_audit
  AFTER INSERT OR DELETE
  ON student
  FOR EACH ROW
  EXECUTE FUNCTION fn_student_audit();

  -- Testing the Trigger
  INSERT INTO student(name, age, class) VALUES ('Aman', 19,
  '11th'); DELETE FROM student WHERE name = 'Aman';
```

4. Output:

Data Output | Messages | Notifications

```
NOTICE:  Inserted Row -> ID: 1, Name: Aman, Age: 19, Class: 11th
NOTICE:  Deleted Row -> ID: 1, Name: Aman, Age: 19, Class: 11th
DELETE 1

Query returned successfully in 168 msec.
```

Hard-Level Problem

1. Aim: Design a PostgreSQL trigger that automatically logs employee addition and deletion activity into an audit table with a timestamped message.

2. Objective:

- To implement trigger-based audit tracking for INSERT and DELETE operations.
- To maintain an audit history of employee changes.
- To use the `NOW()` function to record timestamps automatically.

3. DBMS script and output:

```
-- Creating Main and Audit Tables CREATE
TABLE tbl_employee (
   emp_id SERIAL PRIMARY KEY, emp_name
   VARCHAR(100) NOT NULL,
   emp_salary NUMERIC
);

CREATE TABLE tbl_employee_audit ( sno
   SERIAL PRIMARY KEY,
   message TEXT
);

-- Creating Trigger Function
CREATE OR REPLACE FUNCTION audit_employee_changes()
RETURNS TRIGGER
LANGUAGE plpgsql AS
$$ BEGIN
   IF TG_OP = 'INSERT' THEN
      INSERT INTO tbl_employee_audit(message)
      VALUES ('Employee name ' || NEW.emp_name || ' has been added at ' || NOW());
      RETURN NEW;
```

```
ELSIF TG_OP = 'DELETE' THEN
  INSERT INTO tbl_employee_audit(message)
  VALUES ('Employee name ' || OLD.emp_name || ' has been deleted at ' || NOW());
```

4. Output:

Data Output | Messages | Notifications

Showing rows: 1 to 2 | Page No:

	sno [PK] integer	message text
1	1	Employee name Aman has been added at 2025-11-10 22:21:08.893046+05:30
2	2	Employee name Aman has been deleted at 2025-11-10 22:21:08.893046+05:30