

CHANDIGARH
UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

Advanced Database Management System

Experiment 3.1

23CSP-333

Submitted To:

Faculty Name: Er. Alok Kumar

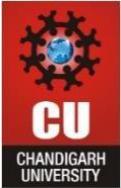
Submitted By:

Name: Garvi Dabas

UID: 23BCS11346

Section: KRG - 2B

Semester: 5th



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Aim:

Medium

To design a trigger on the student table that prints the inserted or deleted row to the console whenever an insert or delete operation occurs.

Hard

To design a trigger on the tbl_employee table that logs every insertion or deletion into an audit table tbl_employee_audit with a message containing employee name and timestamp.

Level:

Level:

Procedure:

Medium Level (Student Table Trigger)

1. Create a trigger function fn_student_audit() that handles INSERT and DELETE operations.
2. Use RAISE NOTICE inside the function to display the newly inserted (NEW) or deleted (OLD) row.
3. Attach the trigger function to the student table for INSERT and DELETE operations.
4. Test the trigger by inserting and deleting student records.

Hard Level (Employee Audit Trigger)

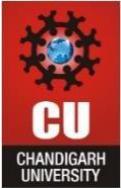
1. Create two tables: tbl_employee and tbl_employee_audit.
2. Write a trigger function audit_employee_changes() that inserts a message into tbl_employee_audit whenever an employee is added or deleted.
3. The message should include the employee name and the current timestamp.
4. Attach the trigger function to tbl_employee for AFTER INSERT OR DELETE operations.
5. Test the trigger by inserting and deleting employee records, then check the audit log.

Code:

Medium Level: Student Trigger

```
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;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
END IF;  
RETURN NULL;  
END;  
$$;
```

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

```
INSERT INTO student(name, age, class) VALUES ('Riya', 18, 10);  
DELETE FROM student WHERE name = 'Riya';
```

Hard Level: Employee Audit Trigger

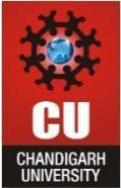
```
CREATE TABLE tbl_employee (  
emp_id SERIAL PRIMARY KEY,  
emp_name VARCHAR(100) NOT NULL,  
emp_salary NUMERIC  
);
```

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

```
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());  
RETURN OLD;  
END IF;
```

```
RETURN NULL;  
END;  
$$;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
CREATE TRIGGER trg_employee_audit
AFTER INSERT OR DELETE
ON tbl_employee
FOR EACH ROW
EXECUTE FUNCTION audit_employee_changes();
```

```
INSERT INTO tbl_employee(emp_name, emp_salary) VALUES ('Aman', 50000);
DELETE FROM tbl_employee WHERE emp_name = 'Aman';
SELECT * FROM tbl_employee_audit;
```

Output:

Output:

```
DROP TABLE
CREATE TABLE
CREATE FUNCTION
CREATE TRIGGER
INSERT 0 1
DELETE 1
DROP TABLE
CREATE TABLE
DROP TABLE
CREATE TABLE
CREATE FUNCTION
CREATE TRIGGER
INSERT 0 1
DELETE 1
sno | message
---+-----
 1 | Employee name Aman has been added at 2025-10-06 18:08:01.836971+00
 2 | Employee name Aman has been deleted at 2025-10-06 18:08:01.838896+00
(2 rows)
```

```
psql:commands.sql:9: NOTICE:  table "student" does not exist, skipping
psql:commands.sql:47: NOTICE:  Inserted Row -> ID: 1, Name: Riya, Age: 18, Class: 10
psql:commands.sql:48: NOTICE:  Deleted Row -> ID: 1, Name: Riya, Age: 18, Class: 10
psql:commands.sql:57: NOTICE:  table "tbl_employee" does not exist, skipping
psql:commands.sql:65: NOTICE:  table "tbl_employee_audit" does not exist, skipping
```

Conclusion:

This experiment demonstrated the use of **PostgreSQL triggers** to automate database actions and maintain audit logs.

- In the **medium-level problem**, the trigger displayed inserted or deleted student rows immediately in the console, providing real-time feedback.
- In the **hard-level problem**, the trigger recorded every insertion and deletion of employees into an audit table with timestamps, ensuring traceability and data integrity.

Overall, triggers help in **automating tasks, auditing changes, and maintaining consistent and reliable database operations** without manual intervention.