

University Institute of Engineering
Department of Computer Science & Engineering

EXPERIMENT: 8

NAME : Johnson Kumar **UID** : 23BCS12654
SECTION : KRG_2A **SEMESTER:** 5TH
SUBJECT CODE: 23CSP-339 **SUBJECT** : ADBMS

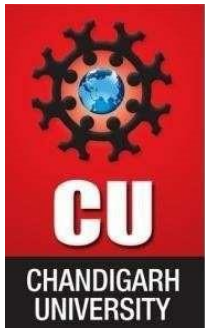
Problem Statement:

1. Design a trigger which: (Medium lvl.)
 - Whenever there is a insertion on student table then, the currently inserted or deleted row should be printed as it as on the output console window.
2. Design a postgresql triggers that: (Hard lvl.)
 - Whenever a new employee is inserted in tbl_employee, a record should be added to tbl_employee_audit like:
 - "Employee name <emp_name> has been added at <current_time>"
 - Whenever an employee is deleted from tbl_employee, a record should be added to tbl_employee_audit like:
 - "Employee name <emp_name> has been deleted at <current_time>"

The solution must use PostgreSQL triggers.

Solution:

```
-----MEDIUM LEVEL PROBLEM-----  
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
```



University Institute of Engineering

Department of Computer Science & Engineering

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

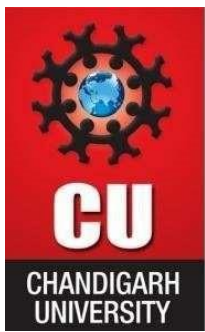
```
RETURN NULL;  
END;  
$$
```

-----HARD LEVEL PROBLEM-----

```
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;
```



University Institute of Engineering

Department of Computer Science & Engineering

\$\$

```
CREATE TRIGGER trg_employee_audit
AFTER INSERT OR DELETE
ON
tbl_employee
FOR EACH ROW
EXECUTE FUNCTION audit_employee_changes();
-- Insert an employee
INSERT INTO tbl_employee(emp_name, emp_salary) VALUES ('Aman', 50000);

-- Delete an employee
DELETE FROM tbl_employee WHERE emp_name = 'Aman';

-- Check audit log
SELECT * FROM tbl_employee_audit;
```

Output: NOTICE: Transaction Successfully Done Query returned successfully in 45 msec.

Learning outcomes:

- I learnt how to create and reset tables using DROP TABLE IF EXISTS and CREATE TABLE in PostgreSQL.
- I understood the use of DO \$\$ blocks for procedural control and structured exception handling.
- I gained hands-on experience with inserting multiple rows within a transaction to ensure atomic operations.
- I learnt how to manage errors using BEGIN...EXCEPTION...END and provide feedback with RAISE NOTICE.
- I discovered that a failure within a transaction block causes the entire operation to roll back automatically.
- I practiced using manual transactions with BEGIN, COMMIT, and ROLLBACK to control data flow precisely.