Deccan Education Society's (DES)
Pune University, Pune
School of Engineering and Technology
Department of Computer Engineering and Technology
Program: B. Tech in Computer Science and Engineering

Academic Year: 2024-25	**Year:** Second Year	**Semester:** II
PRN No.: 1012412079	**Name:** Ratnajeet Patil	
Subject: Database Management System		
Assignment No.: 8		
Date:		

Lab Assignment: 08

Title: Basic Trigger for Audit Logs:

Write and Execute Triggers to perform following kind of operations:

Theory:

What is Trigger?

A trigger in a database is a stored procedure that automatically executes when a specified event occurs on a table or view.

Row Level and Statement Level Trigger

1. Row-Level Trigger

- Executes once for each row affected by the triggering event.
- Useful when you want to inspect or manipulate data row-by-row.
- Uses FOR EACH ROW clause.
- Accesses NEW and OLD row data.
- If you insert 5 users, the trigger runs 5 times, once per user.

2. Statement-Level Trigger

- Executes once per SQL statement, regardless of how many rows are affected.
- Default if FOR EACH ROW is not specified.
- Does not have access to individual NEW or OLD row values.

- If you insert 5 users, the trigger runs once **total**

Execution:

1. **Create a trigger that logs new user registrations into a separate audit table.**
   Table: users(id, name, email)
   Audit Table: user_log(id, name, email, action, log_time)

2. **Write a trigger to ensure that any newly inserted employee has a default department set if not specified.**
   Table: employees(id, name, department_id)

3. **Create a trigger that automatically calculates and inserts a loyalty point entry when a new order is placed.**
   Table: orders(order_id, user_id, amount)
   Loyalty Table: points(user_id, earned_points)

4. **Write a trigger that stores the old salary of an employee into a salary history table before it gets updated.**
   Table: employees(id, name, salary)
   History Table: salary_history(emp_id, old_salary, change_date)
5. **Create a BEFORE UPDATE trigger if anyone tries to update salary of employees.**
   Table: employees(id, name, salary)

6. **Design a trigger that automatically updates the last_modified field whenever a row is updated.**
   Table: documents(id, title, content, last_modified)

7. **Create a trigger that saves deleted customer records into a backup table, along with the date when deletion occurs.**
   Table: customers(id, name, email)
   Backup Table: deleted_customers(id, name, email, deleted_at)

8. **Write an AFTER DELETE trigger that removes all orders linked to a deleted user.**
   Tables: users(id), orders(order_id, user_id)

9. **Create a trigger that logs the deletion of product entries including name and reason.**
   Table: products(id, name)
   Log Table: product_deletion_log(product_id, name, reason, log_time)

```
dbms=# -- 1. Trigger: Log new user registrations
dbms=# CREATE OR REPLACE FUNCTION log_new_user() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     INSERT INTO user_log(id, name, email, action, log_time)
dbms$#     VALUES (NEW.id, NEW.name, NEW.email, 'REGISTER', NOW());
dbms$#     RETURN NEW;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_log_new_user
dbms-# AFTER INSERT ON users
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION log_new_user();
CREATE TRIGGER
dbms=#
dbms=# -- 2. Trigger: Set default department if NULL
dbms=# CREATE OR REPLACE FUNCTION set_default_department() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     IF NEW.department_id IS NULL THEN
dbms$#         NEW.department_id := 1; -- Default department ID
dbms$#     END IF;
dbms$#     RETURN NEW;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_set_default_department
dbms-# BEFORE INSERT ON employees
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION set_default_department();
CREATE TRIGGER
dbms=#
dbms=# -- 3. Trigger: Insert loyalty points on new order
dbms=# CREATE OR REPLACE FUNCTION calc_loyalty_points() RETURNS TRIGGER AS $$
dbms$# DECLARE
dbms$#     points_earned INT;
dbms$# BEGIN
dbms$#     points_earned := FLOOR(NEW.amount / 10); -- Example: 1 point per 10 units
dbms$#     INSERT INTO points(user_id, earned_points)
dbms$#     VALUES (NEW.user_id, points_earned);
dbms$#     RETURN NEW;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_loyalty_points
dbms-# AFTER INSERT ON orders
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION calc_loyalty_points();
CREATE TRIGGER
```

```
dbms=#
dbms=# -- 4. Trigger: Store old salary before update
dbms=# CREATE OR REPLACE FUNCTION store_old_salary() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     INSERT INTO salary_history(emp_id, old_salary, change_date)
dbms$#     VALUES (OLD.id, OLD.salary, NOW());
dbms$#     RETURN NEW;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_store_old_salary
dbms-# BEFORE UPDATE OF salary ON employees
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION store_old_salary();
CREATE TRIGGER
dbms=#
dbms=# -- 5. Trigger: Prevent salary updates
dbms=# CREATE OR REPLACE FUNCTION prevent_salary_update() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     RAISE EXCEPTION 'Salary updates are not allowed.';
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_prevent_salary_update
dbms-# BEFORE UPDATE OF salary ON employees
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION prevent_salary_update();
CREATE TRIGGER
dbms=#
dbms=# -- 6. Trigger: Auto-update last_modified on document update
dbms=# CREATE OR REPLACE FUNCTION update_last_modified() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     NEW.last_modified := NOW();
dbms$#     RETURN NEW;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_update_last_modified
dbms-# BEFORE UPDATE ON documents
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION update_last_modified();
CREATE TRIGGER
dbms=#
```

```
dbms=# -- 7. Trigger: Backup deleted customer
dbms=# CREATE OR REPLACE FUNCTION backup_deleted_customer() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     INSERT INTO deleted_customers(id, name, email, deleted_at)
dbms$#     VALUES (OLD.id, OLD.name, OLD.email, NOW());
dbms$#     RETURN OLD;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_backup_deleted_customer
dbms-# BEFORE DELETE ON customers
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION backup_deleted_customer();
CREATE TRIGGER
dbms=#
dbms=# -- 8. Trigger: Remove all orders linked to deleted user
dbms=# CREATE OR REPLACE FUNCTION delete_user_orders() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     DELETE FROM orders WHERE user_id = OLD.id;
dbms$#     RETURN OLD;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_delete_user_orders
dbms-# AFTER DELETE ON users
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION delete_user_orders();
CREATE TRIGGER
dbms=#
dbms=# -- 9. Trigger: Log product deletion
dbms=# CREATE OR REPLACE FUNCTION log_product_deletion() RETURNS TRIGGER AS $$
dbms$# BEGIN
dbms$#     INSERT INTO product_deletion_log(product_id, name, reason, log_time)
dbms$#     VALUES (OLD.id, OLD.name, 'Deleted by user action', NOW());
dbms$#     RETURN OLD;
dbms$# END;
dbms$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
dbms=#
dbms=# CREATE TRIGGER trg_log_product_deletion
dbms-# BEFORE DELETE ON products
dbms-# FOR EACH ROW
dbms-# EXECUTE FUNCTION log_product_deletion();
CREATE TRIGGER
dbms=#
```

FAQs:

1. What are the different types of triggers?

   Triggers are classified based on:

   A. Event Type:

- BEFORE INSERT
- AFTER INSERT
- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELETE

B. Execution Level:

- Row-Level Trigger: Executes once per row affected.
- Statement-Level Trigger: Executes once per SQL statement.

2. How do you create a trigger in MySQL?

```
1. CREATE TRIGGER trigger_name
2. [BEFORE | AFTER] [INSERT | UPDATE | DELETE]
3. ON table_name
4. FOR EACH ROW
5. BEGIN
6.     -- trigger logic
7. END;
8.
```

3. What is the purpose of OLD and NEW in a trigger?

   `OLD`: Refers to the existing row before the operation (used in `UPDATE` or `DELETE`).
   `NEW`: Refers to the new row being inserted or updated (used in `INSERT` or `UPDATE`).

4. Can triggers be used for audit purposes?

   Triggers can log changes to a separate audit table, helping track:

   - Who modified data
   - When it was modified

- What was changed

5. What are the restrictions on triggers?
    - Cannot call `COMMIT` or `ROLLBACK` inside a trigger.
    - Cannot create or drop tables dynamically inside triggers.
    - Recursive trigger calls may be limited or disallowed by default.
    - Triggers cannot accept parameters.
    - In MySQL, each table can have only one `BEFORE` and one `AFTER` trigger per action.