|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Database Management System Laboratory** | **Semester:** | **IV** |
| **Date of Performance:** | **28 / 03 / 2023** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Prof. Shila Dhande** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 7**

**Title: Queries based on Triggers**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **Objective:** To be able to use trigger on table. |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** Use SQL for Relational database creation, maintenance and query processing. |

|  |
| --- |
| **Books / Journals / Websites Referred:** |
| 1. G. K. Gupta:” Database Management Systems”, McGraw – Hill 2. Korth, Slberchatz, Sudarshan: “Database Systems Concept”, 6th Edition, McGraw Hill 3. Elmasri and Navathe, “Fundamentals of Database Systems”, 5th Edition, PEARSON Education. |

|  |
| --- |
| **Tools Required:** |
| * Postgresql Software |

|  |
| --- |
| **Theory:** |
| **Triggers** are database call-back functions, which are automatically performed/invoked when a specified database event occurs.  Before the operation is attempted on a row (before constraints are checked and the INSERT, UPDATE or DELETE is attempted)  After the operation has completed (after constraints are checked and the INSERT, UPDATE, or DELETE has completed)  Instead of the operation (in the case of inserts, updates or deletes on a view)  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....  ]; |

|  |
| --- |
| **Implementation Details (Problem Statement, Query and Screenshots of Results):** |
| **Implementation Code:**  DROP TABLE IF EXISTS customer\_log;  CREATE TABLE customer\_log (  log\_id SERIAL PRIMARY KEY,  customer\_id INT,  action\_type VARCHAR(10),  action\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  CREATE OR REPLACE FUNCTION log\_customer\_action() RETURNS TRIGGER AS $$  BEGIN  IF TG\_OP = 'INSERT' THEN  INSERT INTO customer\_log (customer\_id, action\_type) VALUES (NEW.customer\_id, 'INSERT');  ELSIF TG\_OP = 'UPDATE' THEN  INSERT INTO customer\_log (customer\_id, action\_type) VALUES (NEW.customer\_id, 'UPDATE');  ELSIF TG\_OP = 'DELETE' THEN  INSERT INTO customer\_log (customer\_id, action\_type) VALUES (OLD.customer\_id, 'DELETE');  END IF;  RETURN NULL;  END;  $$ LANGUAGE plpgsql;  -- insertion trigger  DROP TRIGGER IF EXISTS after\_customer\_insert ON customer;  CREATE TRIGGER after\_customer\_insert  AFTER INSERT ON customer  FOR EACH ROW  EXECUTE FUNCTION log\_customer\_action();  -- update trigger  DROP TRIGGER IF EXISTS after\_customer\_update ON customer;  CREATE TRIGGER after\_customer\_update  AFTER UPDATE ON customer  FOR EACH ROW  EXECUTE FUNCTION log\_customer\_action();  -- deletion trigger  DROP TRIGGER IF EXISTS after\_customer\_delete ON customer;  CREATE TRIGGER after\_customer\_delete  AFTER DELETE ON customer  FOR EACH ROW  EXECUTE FUNCTION log\_customer\_action();  -- inserting  INSERT INTO customer (customer\_id, customer\_name, customer\_email, customer\_phone)  VALUES  (600, 'Ketaki Mahajan', 'ketaki.mahajan@gmail.com', '1234567890'),  (601, 'Sumeet Karani', 'sumeet.karani@gmail.com', '4684213278'),  (602, 'Tejas Kamath', 'tejas.kamath@gmail.com', '4684253278');  -- deleting  DELETE FROM customer  WHERE customer\_id IN (600, 601, 602);  -- updating  UPDATE customer  SET customer\_email = 'new.email@gmail.com'  WHERE customer\_id = 600;  SELECT \* FROM customer;  SELECT \* FROM customer\_log;  -- counting number of triggers  SELECT COUNT(\*)  FROM information\_schema.triggers;  -- dropping update trigger  DROP TRIGGER IF EXISTS after\_customer\_update ON customer;  -- checking if update trigger still exists  SELECT EXISTS (  SELECT 1  FROM information\_schema.triggers  WHERE trigger\_name = 'after\_customer\_update'  );  **Implementations Screenshots:**  Creating customer\_log –    Creating trigger function –    Creating insert, delete and update trigger –    Inserting records into customer table –    Checking customer\_log after inserting –    Deleting records from customer table –    Checking customer\_log after deleting –    Updating records in customer table –    Checking customer\_log after updating –    Counting the number of triggers –    Dropping the update trigger –    Checking if the update trigger still exists – |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Write a trigger to count number of new tuples inserted using each insert statement.**  CREATE OR REPLACE FUNCTION count\_inserted\_tuples()  RETURNS TRIGGER AS $$  DECLARE  inserted\_count INTEGER;  BEGIN  -- Count the number of newly inserted tuples  SELECT COUNT(\*)  INTO inserted\_count  FROM NEW;  -- Display the count of inserted tuples  RAISE NOTICE 'Number of tuples inserted: %', inserted\_count;  -- Continue with the insertion  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER count\_insert\_trigger  AFTER INSERT ON employees  FOR EACH STATEMENT  EXECUTE FUNCTION count\_inserted\_tuples();   **Trigger is a special type of \_\_\_\_\_\_\_\_\_ procedure.****Stored****Function****View****Table****Triggers can be enabled or disabled with the \_\_\_\_\_\_\_\_\_ statement.****ALTER TABLE statement****DROP TABLE statement****DELETE TABLE statement****None of the mentioned** |

|  |
| --- |
| **Conclusion:** |
| In conclusion, delving into database fundamentals such as triggers has equipped us with essential skills for efficient data handling and analysis. |

**Signature of faculty in-charge with Date:**