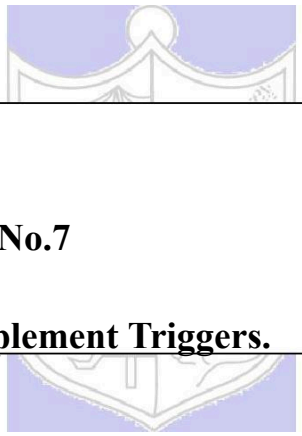


Experiment No.7

Title: To Implement Triggers.

Batch:SY-IT(B3) **Roll No.:16010423076** **Experiment No.:7**

Title: To implement Triggers for given database.

Resources needed: PostgreSQL 9.3

Theory

Pre Lab/ Prior Concepts:

Trigger:

A trigger is a statement that the system executes automatically as a side effect of a modification to the database. Triggers are used to ensure some types of integrity.

To design a trigger mechanism you should:

1. Specify when a trigger is to be executed. This is broken up into an event that causes a trigger to be checked and a condition that must be satisfied for trigger execution to proceed
2. Specify the actions to be taken when the trigger executes.

Generalized Model:

Triggers are based on the Event- condition- Action (ECA) Model. A rule in the ECA model has three components.

1. The event(s) that triggers the rule.
2. The condition that determines whether the rule action should be executed. If no action is specified, the action will be executed once the event occurs.
3. The action to be taken. It could be a sequence of SQL statements, a DB transaction or an external program that will be executed automatically.

When to use Trigger:

In many cases, it is convenient to specify the type of action to be taken when certain event occurs and when certain conditions are satisfied.

It may be useful to specify a condition that, if violated, causes some user to be informed of the violation.

For example:-

A manager may want to be informed, if an employee's travel expenses exceed a certain limit by receiving a message whenever this occurs.

The condition is thus used to monitor the database.

CREATE TRIGGER statement is used to implement such action in SQL. Consider the triggers for following cases:-

1. Trigger for insertion:-

This trigger executes whenever condition is satisfied, during the insertion statement. If no condition is satisfied, then it executed for every insertion statement, for the relation specified.

Example:-

In DB, we have created a trigger for insertion, on the relation Employee. If salaries < 1500 then print 'Unsuccessful' else print 'Successful'.

2. Trigger for Updation:-

This trigger executes for updating of a column name(s) of a particular relation. A condition may or may not be specified.

Example:-

In our DB, we have created a trigger for updating, on the relation Employee. If salary is updated to unacceptable amount, then print unsuccessful and rollback else print successful.

3. Trigger for Deletion:-

This trigger executes during the deletion statement. If a particular condition is satisfied, the system may allow or not allow the deletion of certain tuples,

Example:-

If SSN from Employee =101, then print 'Unsuccessful' and rollback, else print 'Successful'.

In Postgresql we need to create a function to execute all action statements of trigger and call this function in create trigger statement for certain event.

Example.

In employeee table dnum is a foreign key. So the trigger writen here will increment total_emp of department table by one and total_sal of department by salary of newly inserted employee's salary.

```
CREATE or replace  FUNCTION inse_function() RETURNS trigger As $emp_update$

begin
     UPDATE dept set total_emp=total_emp + 1,
         total_sal=total_sal+ new.salary where dept.dno=new.dno;
   RETURN new;

END
$emp_update$ LANGUAGE plpgsql;


CREATE TRIGGER emp_update AFTER INSERT
      ON emp_table FOR each ROW
      EXECUTE PROCEDURE inse_function();
```

Procedure / Approach /Algorithm / Activity Diagram

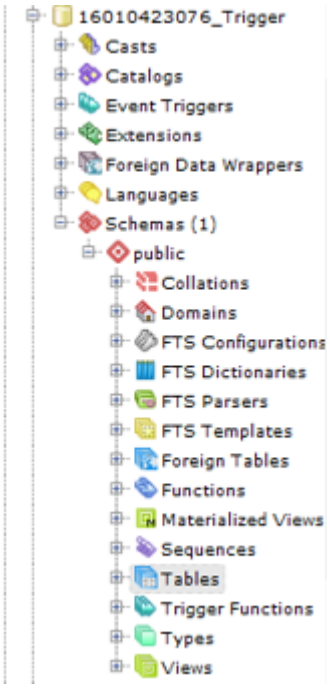
1. Create new database for your application
2. Apply required integrity constraints on tables in your database

To design a trigger.

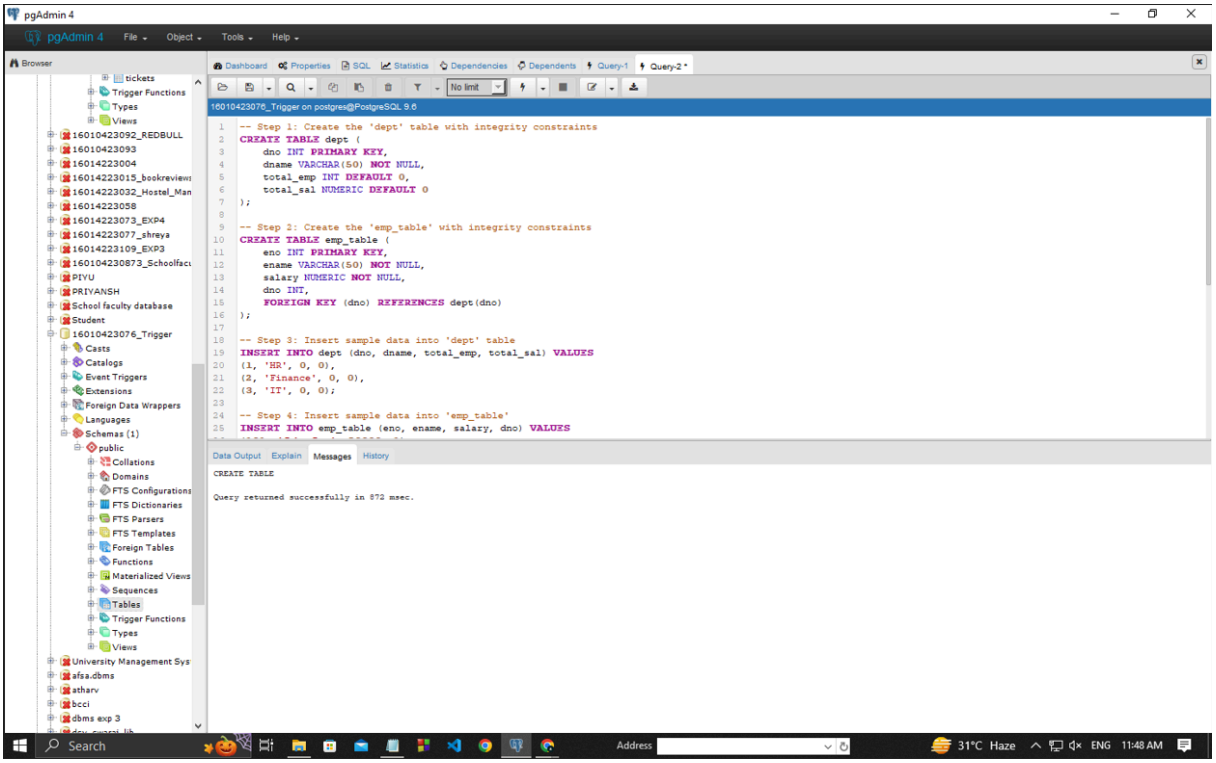
1. Identify events in the database.
2. Specify conditions under which the trigger is to be executed.
3. Specify actions to be taken when trigger is executed.

Results: (Program printout with output / Document printout as per the format)

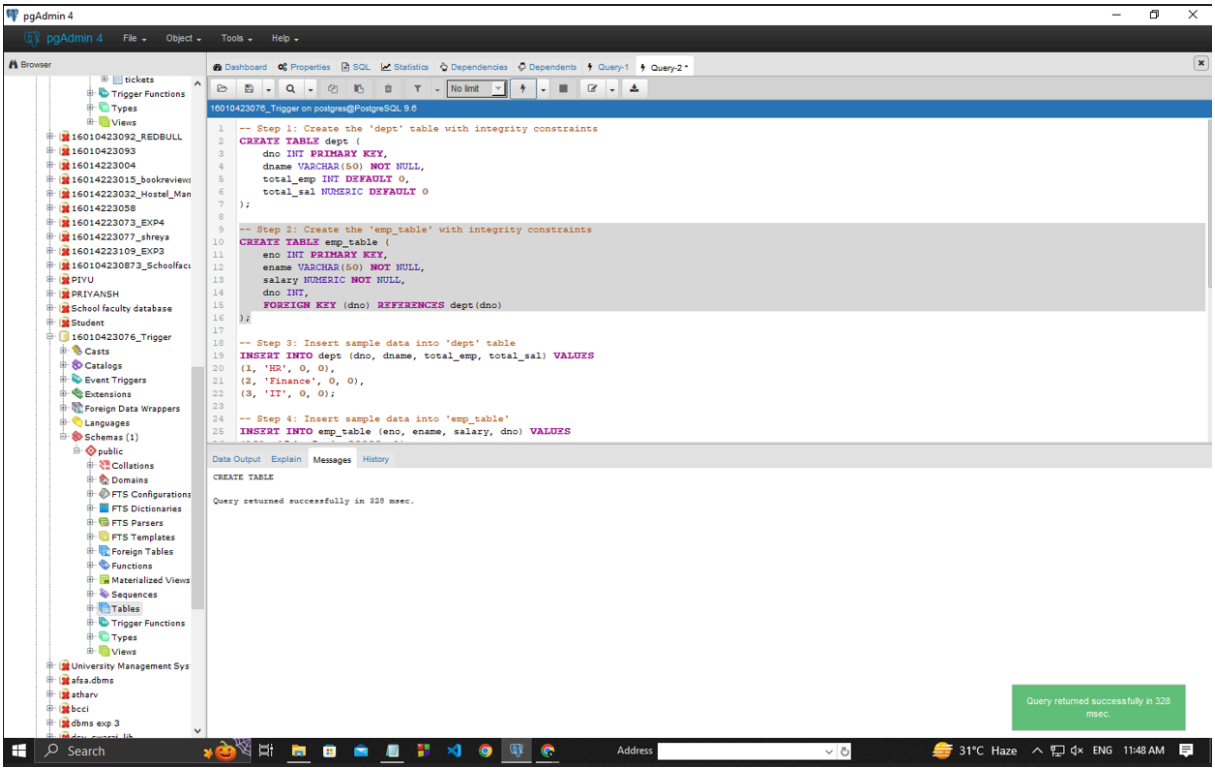
Created a new Database with the name 16010423076_Trigger :



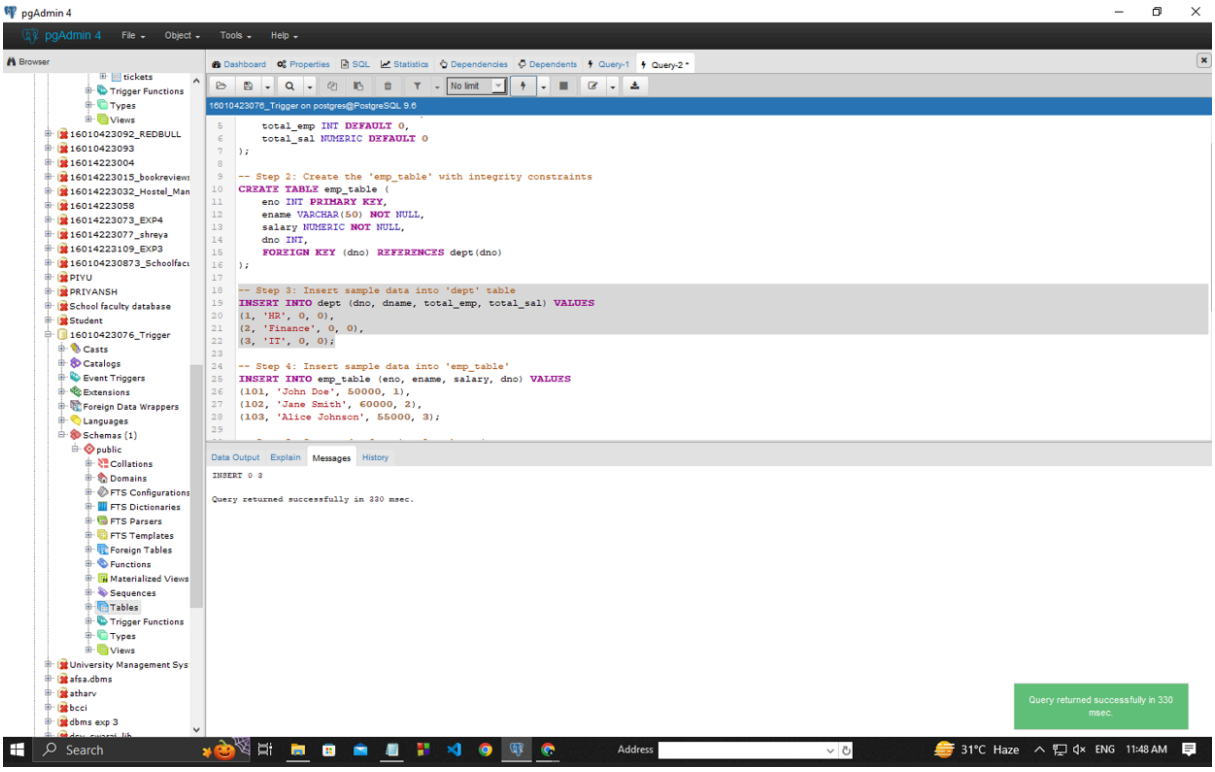

```
-- Step 1: Create the 'dept' table with integrity constraints
CREATE TABLE dept (
    dno INT PRIMARY KEY,
    dname VARCHAR(50) NOT NULL,
    total_emp INT DEFAULT 0,
    total_sal NUMERIC DEFAULT 0
);
```



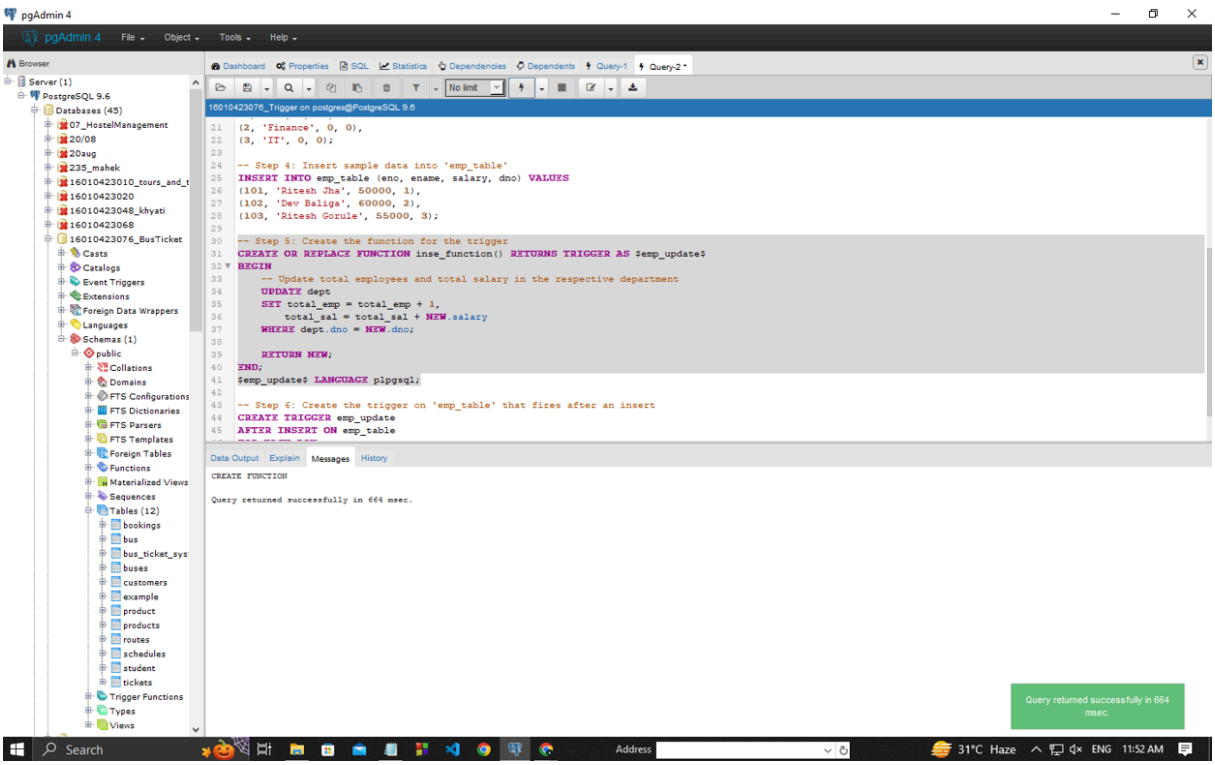

```
-- Step 2: Create the 'emp_table' with integrity constraints
CREATE TABLE emp_table (
    eno INT PRIMARY KEY,
    ename VARCHAR(50) NOT NULL,
    salary NUMERIC NOT NULL,
    dno INT,
    FOREIGN KEY (dno) REFERENCES dept(dno)
);
```



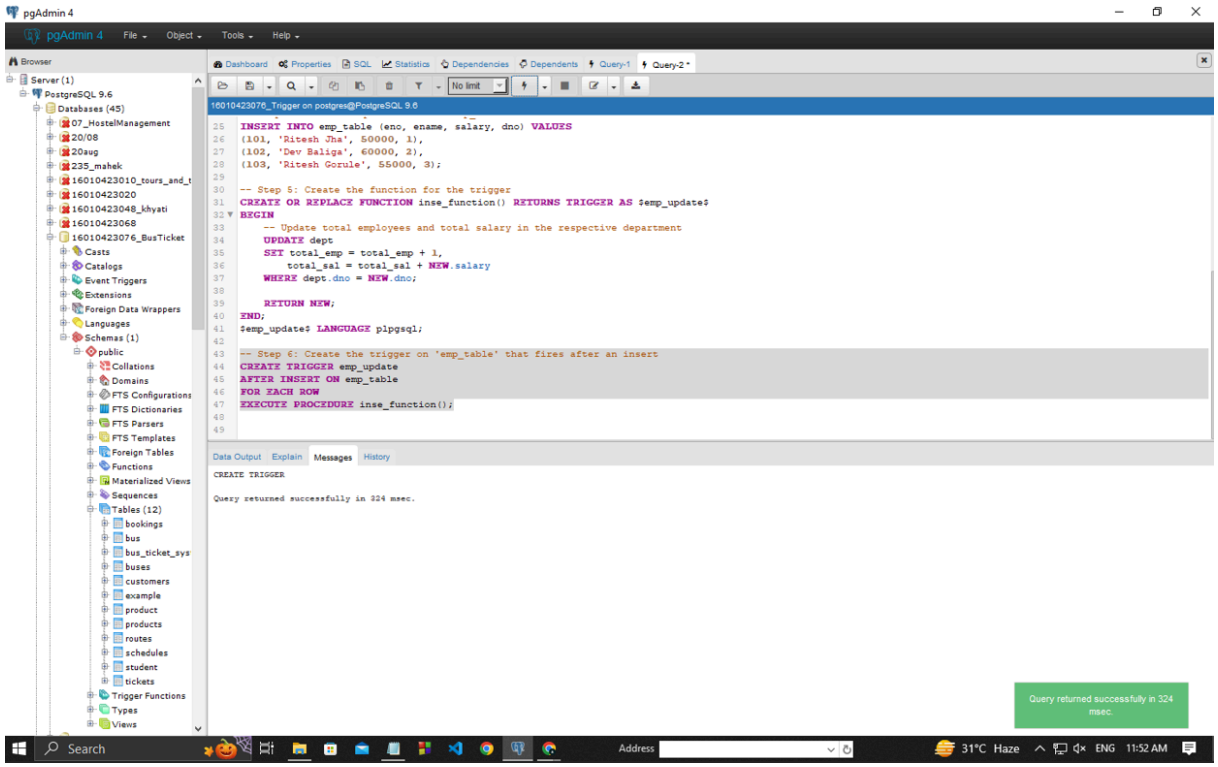
-- Step 3: Insert sample data into 'dept' table
INSERT INTO dept (dno, dname, total_emp, total_sal) VALUES
(1, 'HR', 0, 0),
(2, 'Finance', 0, 0),
(3, 'IT', 0, 0);



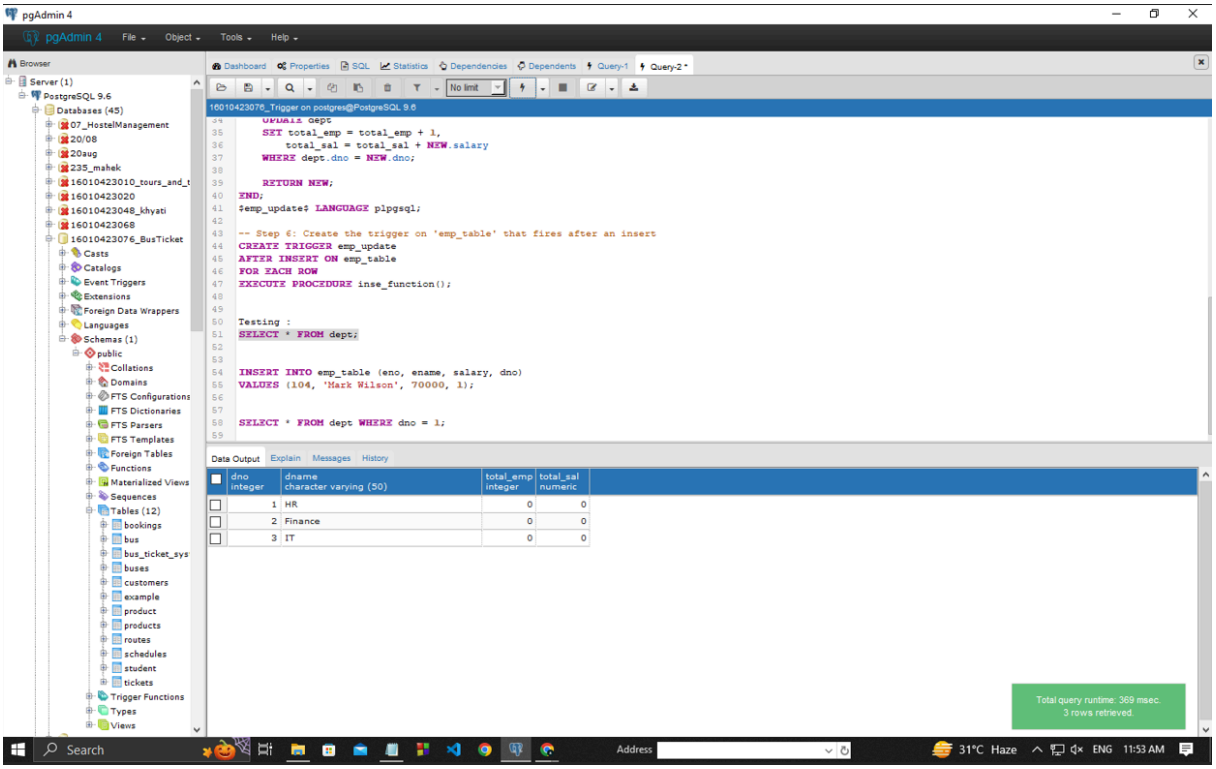
-- Step 4: Insert sample data into 'emp_table'
INSERT INTO emp_table (eno, ename, salary, dno) VALUES
(101, 'Ritesh Jha', 50000, 1),
(102, 'Dev Baliga', 60000, 2),
(103, 'Ritesh Gorule', 55000, 3);



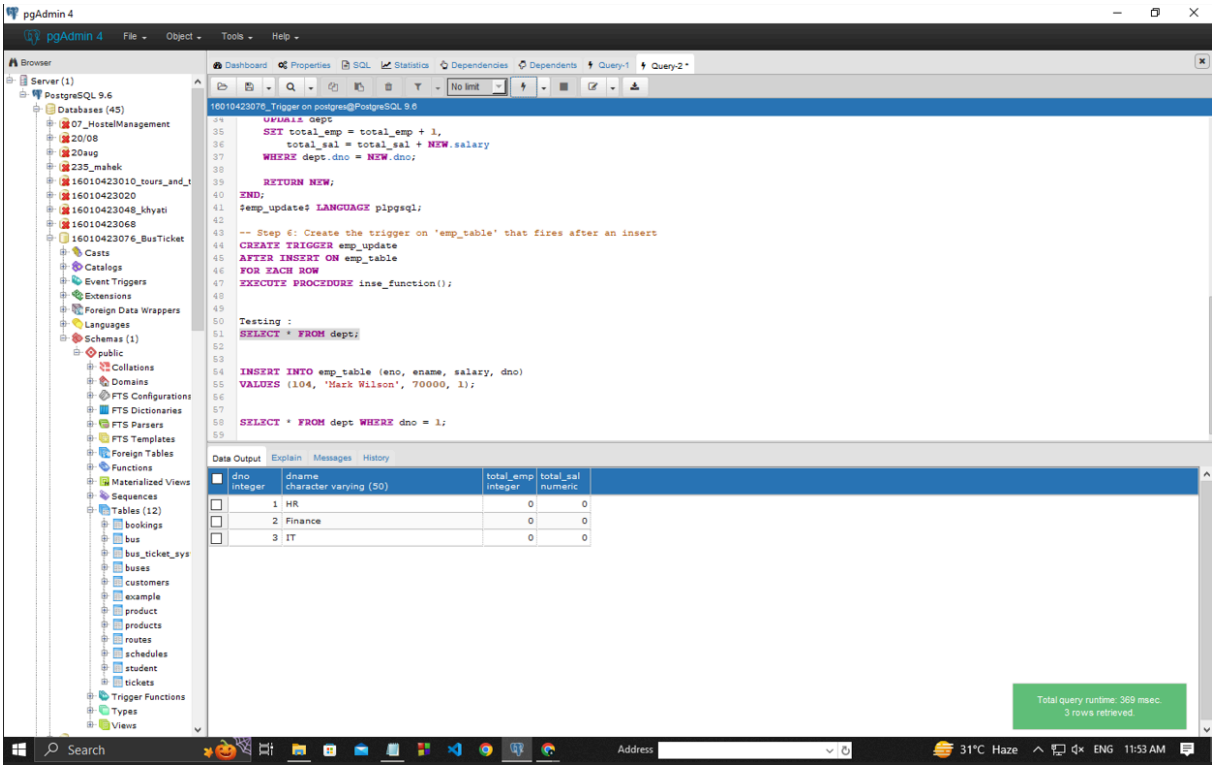

```
-- Step 5: Create the function for the trigger
CREATE OR REPLACE FUNCTION inse_function() RETURNS TRIGGER AS
$emp_update$
BEGIN
    -- Update total employees and total salary in the respective department
    UPDATE dept
    SET total_emp = total_emp + 1,
        total_sal = total_sal + NEW.salary
    WHERE dept.dno = NEW.dno;

    RETURN NEW;
END;
$emp_update$ LANGUAGE plpgsql;
```



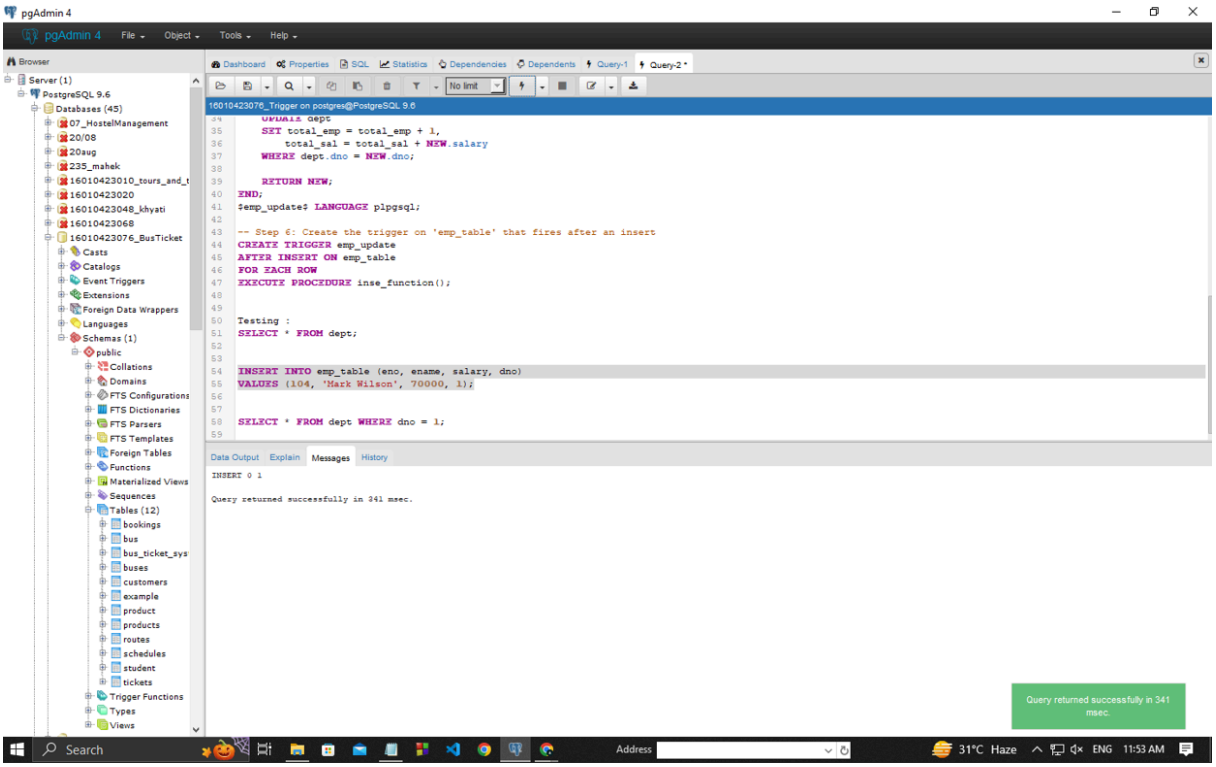
-- Step 6: Create the trigger on 'emp_table' that fires after an insert
CREATE TRIGGER emp_update
AFTER INSERT ON emp_table
FOR EACH ROW
EXECUTE PROCEDURE inse_function();



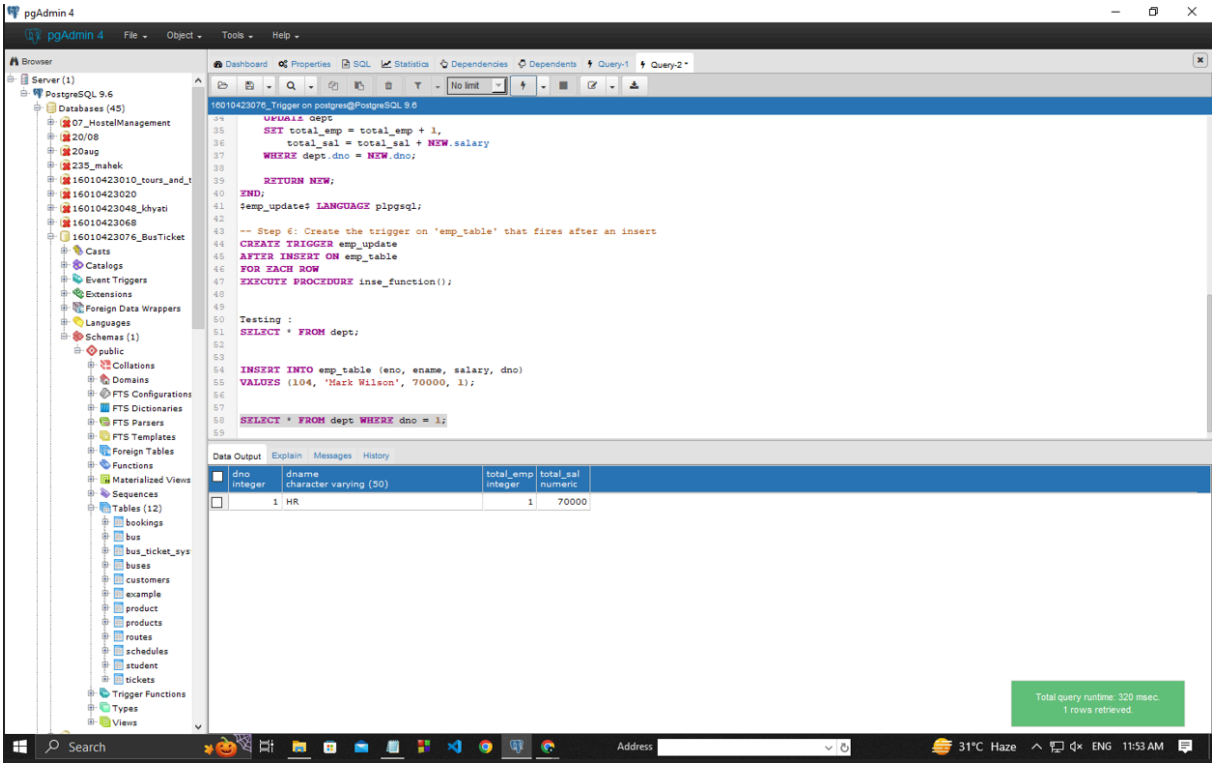
-- Testing :
SELECT * FROM dept;



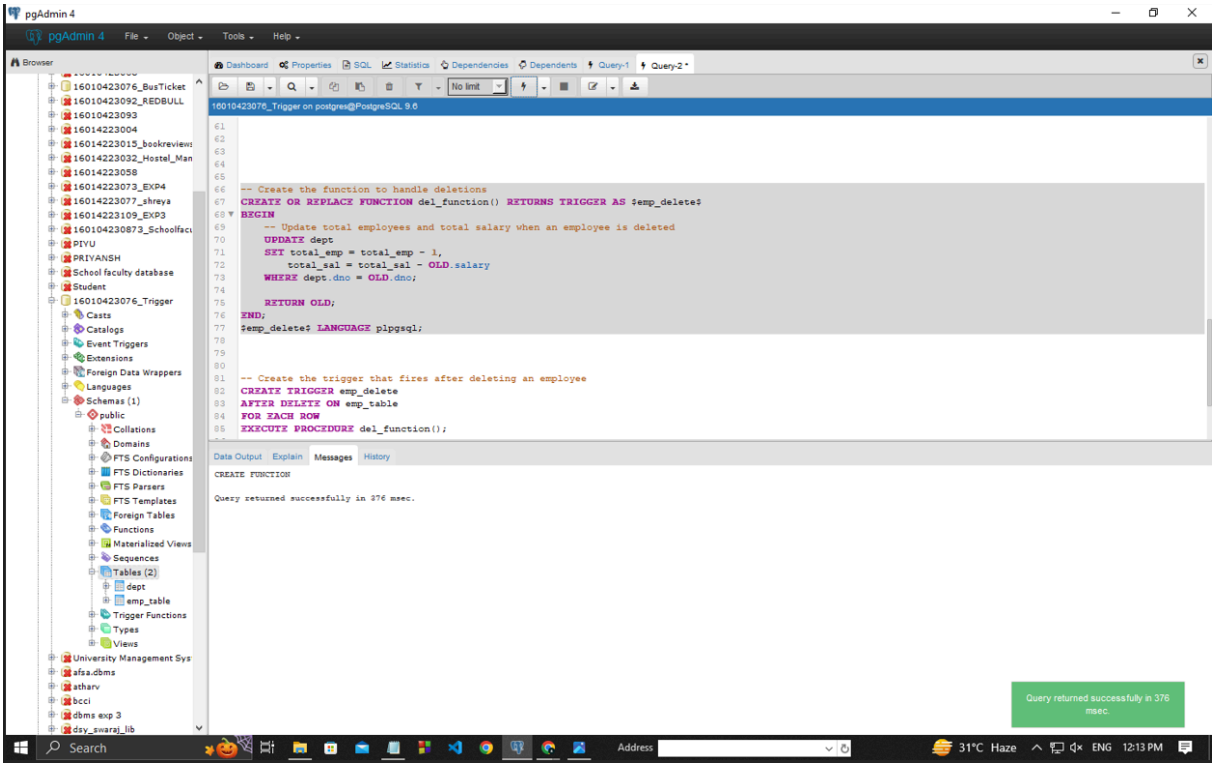
INSERT INTO emp_table (eno, ename, salary, dno)
VALUES (104, Mark Wilson, 70000, 1);



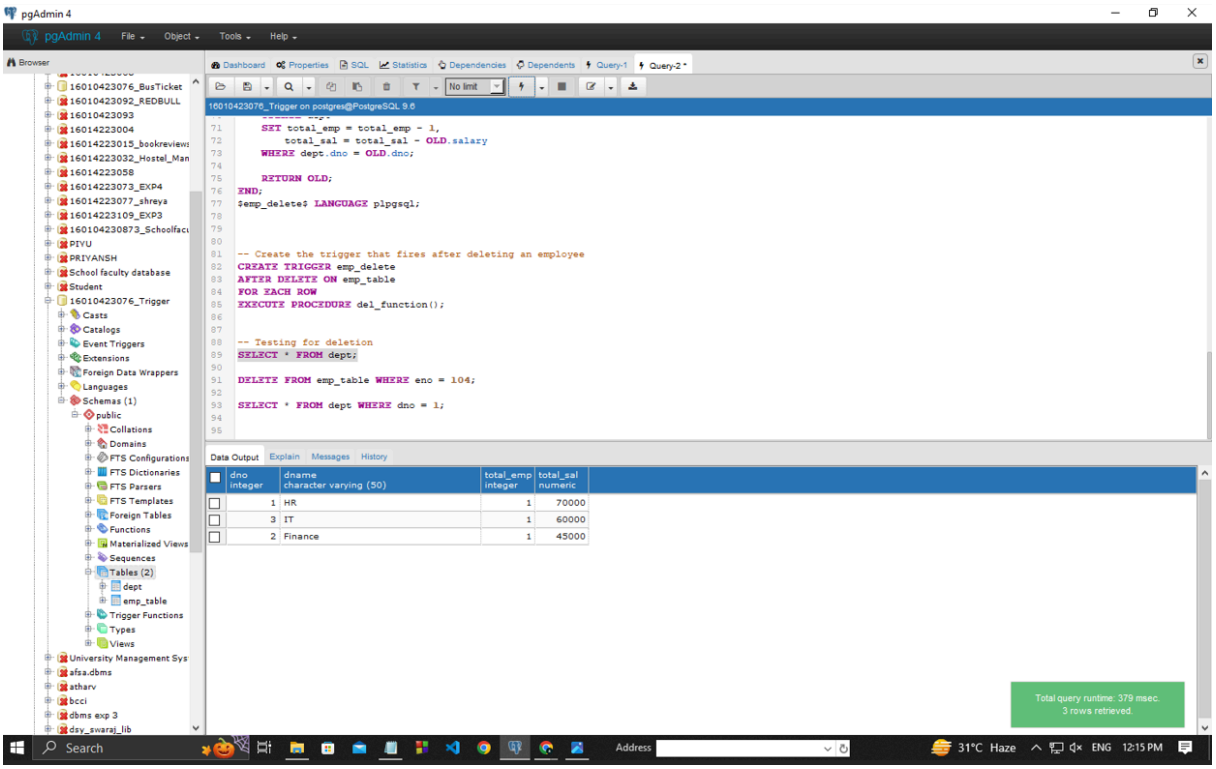
SELECT * FROM dept WHERE dno = 1;



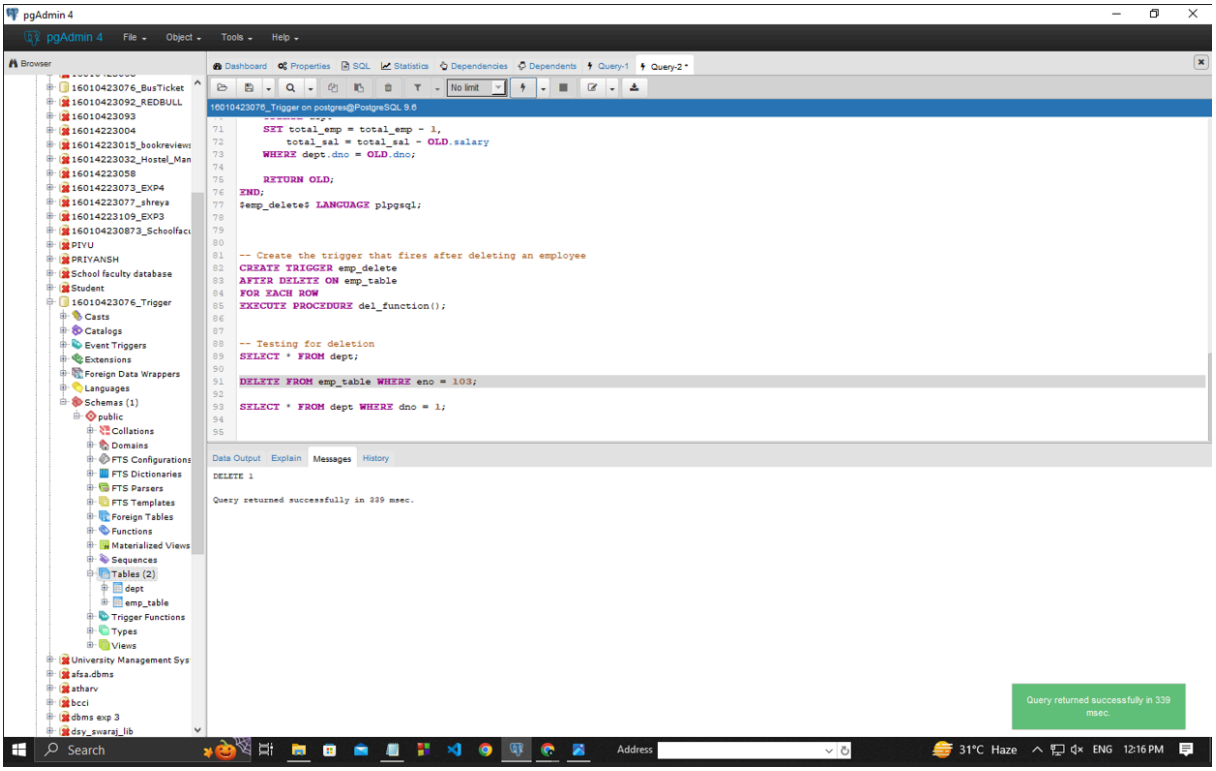

```
-- Create the function to handle deletions
CREATE OR REPLACE FUNCTION del_function() RETURNS TRIGGER AS
$emp_delete$
BEGIN
    -- Update total employees and total salary when an employee is deleted
    UPDATE dept
    SET total_emp = total_emp - 1,
        total_sal = total_sal - OLD.salary
    WHERE dept.dno = OLD.dno;

    RETURN OLD;
END;
$emp_delete$ LANGUAGE plpgsql;
```



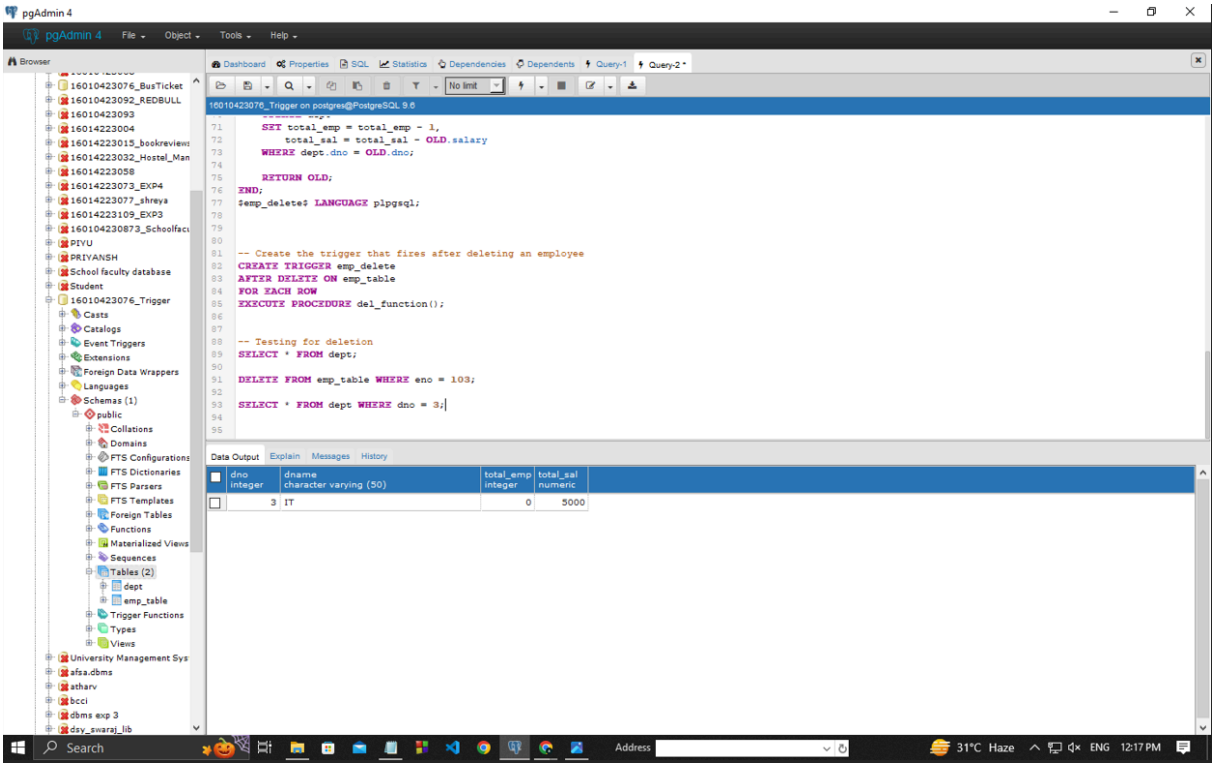

```
-- Create the trigger that fires after deleting an employee
CREATE TRIGGER emp_delete
AFTER DELETE ON emp_table
FOR EACH ROW
EXECUTE PROCEDURE del_function();


-- Testing for deletion
SELECT * FROM dept;
```



DELETE FROM emp_table WHERE eno = 103;



SELECT * FROM dept WHERE dno = 3;



Questions:

1 Explain any one real-time application of trigger

Ans)

- **Scenario:** In a banking environment, tracking transaction changes is essential for compliance and auditing.
- **Trigger Creation:** A database trigger is set up on the transactions table to automatically execute during INSERT, UPDATE, or DELETE operations.
- **Action:** The trigger logs transaction details—such as amount, date, user ID, and transaction type—into an audit table, ensuring all changes are captured without manual intervention.
- **Benefits:**
    - **Real-time Monitoring:** Captures data instantly as transactions occur, providing up-to-date auditing information.
    - **Data Integrity:** Ensures consistent logging of all changes, reducing the risk of missing or fraudulent transactions.

    - **Regulatory Compliance:** Assists the bank in meeting legal requirements for financial record-keeping.
- Overall, this use of triggers enhances the security and reliability of the banking system by automating the auditing process.

Conclusion :

I learned how to create and use database triggers to automatically update related tables in response to changes. Specifically, I saw how triggers can maintain accurate counts of employees and total salaries in a department when new employees are added or removed. This not only simplifies data management but also ensures data integrity and real-time updates, which are crucial in a relational database. Overall, this experience helped me appreciate the power of triggers in automating tasks and maintaining consistency within the database.