

Batch: E-2 Roll No.: 16010123325

Experiment No. 5

Title: Queries based Views and Triggers

Objective: To be able to use SQL view and triggers.

Expected Outcome of Experiment:

CO3: Utilize SQL for Relational Database Operations

Books/ Journals/ Websites referred:

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g. Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Silberchatz, Sudarshan : "Database Systems Concept", 5th Edition , McGraw Hill
4. Elmasri and Navathe, "Fundamentals of database Systems", 4th Edition, PEARSON Education.

Resources used: Postgresql

Theory

View

Views are pseudo-tables. That is, they are not real tables; nevertheless appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, you can use them to restrict table access so that the users see only specific rows or columns of a table.

A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can only see limited data instead of complete table.

- Summarize data from various tables, which can be used to generate reports.

Since views are not ordinary tables, you may not be able to execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a RULE to correct this problem of using DELETE, INSERT or UPDATE on a view.

Syntax

```
CREATE [TEMP | TEMPORARY] VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

Ex:

```
CREATE VIEW COMPANY-VIEW AS
SELECT ID, NAME, AGE
FROM COMPANY;
```

Display view:

```
select * from Company-View
```

Insert record into view:

```
Insert into Company-View values (123,'alpha', 10)
select * from Company
```

Dropping Views

Syntax: DROP VIEW view_name;

Triggers

A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. By using SQL triggers, developers can automate tasks, ensure data consistency, and keep accurate records of database activities.

For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated.

Key Features of SQL Triggers:

Automatic Execution: Triggers fire automatically when the defined event occurs (e.g., INSERT, UPDATE, DELETE).

Event-Driven: Triggers are tied to specific events that take place within the database.

Table Association: A trigger is linked to a specific table or view, and operates whenever changes are made to the table's data.

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....
```

```
];
```

- **trigger_name:** The name of the trigger to be created.
- **BEFORE | AFTER:** Specifies whether the trigger is fired before or after the triggering event (INSERT, UPDATE, DELETE).
- **{INSERT | UPDATE | DELETE}:** Specifies the operation that will activate the trigger.
- **table_name:** The name of the table the trigger is associated with.
- **FOR EACH ROW:** Indicates that the trigger is row-level, meaning it executes once for each affected row. (You can optionally specify FOR EACH ROW after table name.)
- **)**
- **trigger_body:** The SQL statements to be executed when the trigger is fired.

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....  
];
```

To use the PostgreSQL CREATE TRIGGER statement to create a trigger.

To create a new trigger in PostgreSQL, you follow these steps:

- First, create a trigger function using CREATE FUNCTION statement.
- Second, bind the trigger function to a table by using CREATE TRIGGER statement.

Example :

Suppose that when the name of an employee changes, you want to log it in a separate table called **employee_audits**.

```
employee  

emp_id | fname | lname | age | salary  

-----+-----+-----+-----+-----  

    1 | John | Doe | 30 | 50000.00  

    2 | Jane | Smith | 25 | 60000.00  

    3 | Alice | Johnson | 35 | 70000.00  

(3 rows)  

Employee_audit  

emp_id | fname | ChangeOn |  

-----+-----+-----+
```

```
CREATE OR REPLACE FUNCTION log_last_name_changes()  

RETURNS TRIGGER AS $$  

BEGIN  

INSERT INTO employee_audits(employee_id,last_name,changed_on)  

VALUES(OLD.id,OLD.last_name,now());  

RETURN NEW;  

END;  

$$ language plpgsql;
```

```
CREATE TRIGGER last_name_changes  

BEFORE UPDATE  

ON employees
```

FOR EACH ROW

```
EXECUTE PROCEDURE log_last_name_changes();  
select * from employee;
```

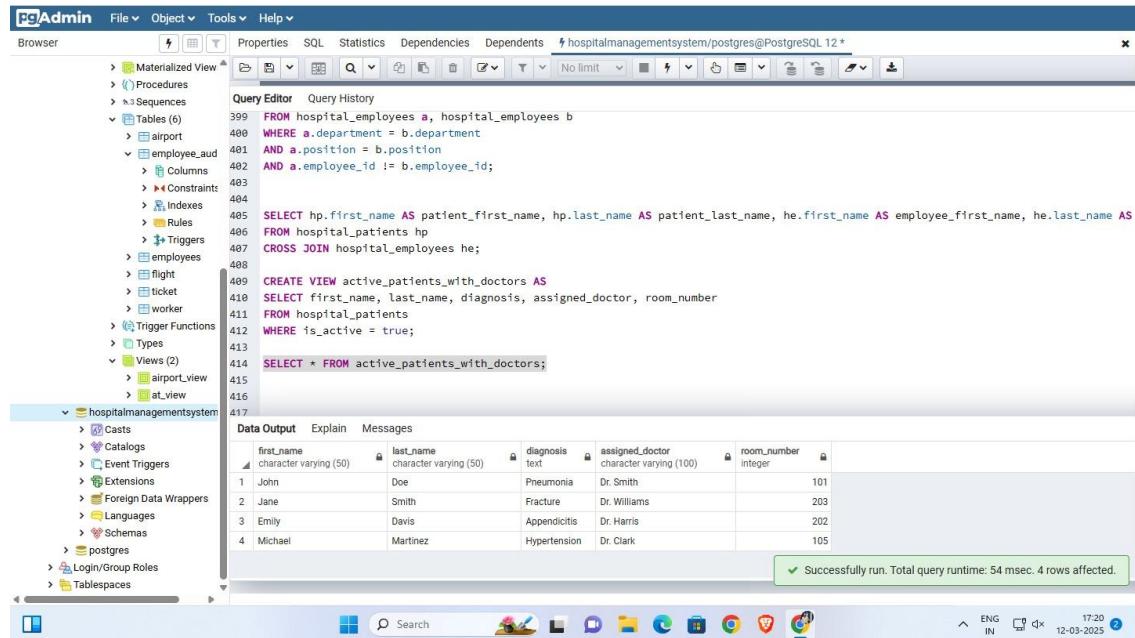
```
UPDATE employee  
SET lname = 'Kulkarni'  
WHERE ID = 1
```

```
select * from semployee  
select * from employee_audit
```

Implementation Screenshots (Problem Statement, Query and Screenshots of Results):

Operations on view

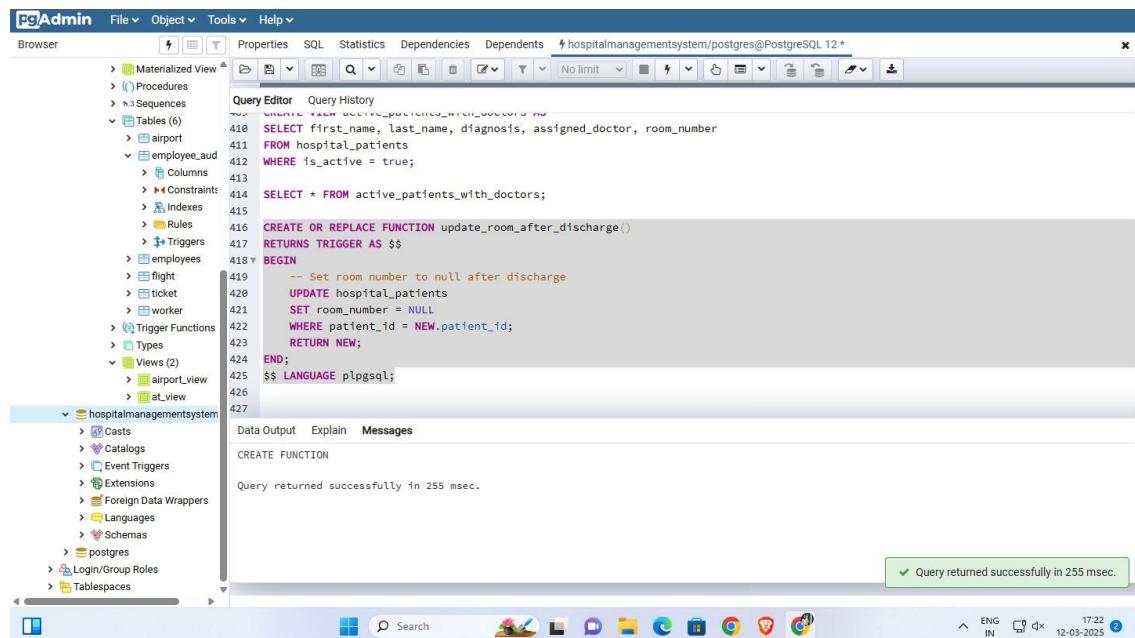
1) Selecting Active Patients with Doctors



```

pgAdmin File Object Tools Help
Browser
  > Materialized View
  > Procedures
  > Sequences
  > Table (6)
    > airport
    > employee_aud
      > Columns
      > Constraints
      > Indexes
      > Rules
      > Triggers
    > employees
    > flight
    > ticket
    > worker
  > Trigger Functions
  > Types
  > Views (2)
    > airport_view
    > at_view
  hospitalmanagementsystem
    > Casts
    > Catalogs
    > Event Triggers
    > Extensions
    > Foreign Data Wrappers
    > Languages
    > Schemas
    > postres
    > Login/Group Roles
    > Tablespaces
Query Editor Query History
399 FROM hospital_employees a, hospital_employees b
400 WHERE a.department = b.department
401 AND a.position = b.position
402 AND a.employee_id != b.employee_id;
403
404 SELECT hp.first_name AS patient_first_name, hp.last_name AS patient_last_name, he.first_name AS employee_first_name, he.last_name AS
405 FROM hospital_patients hp
406 CROSS JOIN hospital_employees he;
407
408 CREATE VIEW active_patients_with_doctors AS
409 SELECT first_name, last_name, diagnosis, assigned_doctor, room_number
410 FROM hospital_patients
411 WHERE is_active = true;
412
413
414 SELECT * FROM active_patients_with_doctors;
415
416
417
Data Output Explain Messages
first_name last_name diagnosis assigned_doctor room_number
1 John Doe Pneumonia Dr. Smith 101
2 Jane Smith Fracture Dr. Williams 203
3 Emily Davis Appendicitis Dr. Harris 202
4 Michael Martinez Hypertension Dr. Clark 105
  
```

Successfully run. Total query runtime: 54 msec. 4 rows affected.



```

pgAdmin File Object Tools Help
Browser
  > Materialized View
  > Procedures
  > Sequences
  > Tables (6)
    > airport
    > employee_aud
      > Columns
      > Constraints
      > Indexes
      > Rules
      > Triggers
    > employees
    > flight
    > ticket
    > worker
  > Trigger Functions
  > Types
  > Views (2)
    > airport_view
    > at_View
  hospitalmanagementsystem
    > Casts
    > Catalogs
    > Event Triggers
    > Extensions
    > Foreign Data Wrappers
    > Languages
    > Schemas
    > postres
    > Login/Group Roles
    > Tablespaces
Query Editor Query History
CREATE VIEW active_patients_with_doctors AS
SELECT first_name, last_name, diagnosis, assigned_doctor, room_number
FROM hospital_patients
WHERE is_active = true;
SELECT * FROM active_patients_with_doctors;
CREATE OR REPLACE FUNCTION update_room_after_discharge()
RETURNS TRIGGER AS $$
BEGIN
  -- Set room number to null after discharge
  UPDATE hospital_patients
  SET room_number = NULL
  WHERE patient_id = NEW.patient_id;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
Data Output Explain Messages
CREATE FUNCTION
Query returned successfully in 255 msec.
  
```

Query returned successfully in 255 msec.

The screenshot shows the pgAdmin 4 application window. The left sidebar displays the database structure under 'Browser' for the 'hospitalmanagementsystem' database, including Schemas, Tables, Views, and other objects like Functions, Procedures, and Triggers. The main area is the 'Query Editor' showing a SQL script for creating a function and a trigger. The function 'update_room_after_discharge()' updates the 'room_number' to null for patients after discharge. The trigger 'after_patient_discharge' calls this function after an update on the 'hospital_patients' table. The 'Messages' tab at the bottom indicates the query was successful.

```
CREATE OR REPLACE FUNCTION update_room_after_discharge()
RETURNS TRIGGER AS $$
BEGIN
    -- Set room number to null after discharge
    UPDATE hospital_patients
    SET room_number = NULL
    WHERE patient_id = NEW.patient_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_patient_discharge
AFTER UPDATE ON hospital_patients
FOR EACH ROW
WHEN (OLD.is_active = true AND NEW.is_active = false)
EXECUTE FUNCTION update_room_after_discharge();
```

Data Output Explain Messages

CREATE TRIGGER

Query returned successfully in 81 msec.

✓ Query returned successfully in 81 msec.

The screenshot shows the pgAdmin 4 application window. The left sidebar displays the database browser with the following tree structure:

- Browser
 - Materialized View
 - Procedures
 - Sequences
 - Tables (6)
 - airport
 - employee_aud
 - employees
 - flight
 - ticket
 - workers
 - Trigger Functions
 - Types
 - Views (2)
 - airport_view
 - at_view
- hospitalmanagementsystem
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Schemas
- postgres
- Login/Group Roles
- Tablespaces

The main panel features a toolbar at the top with various icons for file operations, search, and database management. Below the toolbar is the Query Editor tab, which is active. The query text is as follows:

```
420 UPDATE hospital_patients
421   SET room_number = NULL
422   WHERE patient_id = NEW.patient_id;
423   RETURN NEW;
424 END;
425 $$ LANGUAGE plpgsql;
426
427 CREATE TRIGGER after_patient_discharge
428 AFTER UPDATE ON hospital_patients
429 FOR EACH ROW
430 WHEN (OLD.is_active = true AND NEW.is_active = false)
431 EXECUTE FUNCTION update_room_after_discharge();
432
433 UPDATE hospital_patients
434   SET is_active = false
435 WHERE patient_id = 3; -- Example: Discharging Robert Johnson
436
437
```

Below the query editor, there are three tabs: Data Output, Explain, and Messages. The Data Output tab shows the result of the update query:

```
UPDATE 1
```

The Messages tab shows the message: "Query returned successfully in 54 msec."

The bottom right corner of the window shows the system tray with icons for battery, signal, volume, and date/time (17:23, 12-30-2025).

PGAdmin File Object Tools Help

Browser Properties SQL Statistics Dependencies Dependents hospitalmanagementsystem/postgres@PostgreSQL 12*

```

> Materialized View
> Procedures
> Sequences
> Tables (6)
>   airport
>   employee_aud
>   Columns
>   Constraints
>   Indexes
>   Rules
>   Triggers
>   employees
>   flight
>   ticket
>   worker
>   Trigger Functions
>   Types
>   Views (2)
>     airport_view
>     at_view

```

```

424 WHERE patient_id = NEW.patient_id;
425      RETURN NEW;
426 END;
427 $$ LANGUAGE plpgsql;
428
429 CREATE TRIGGER after_patient_discharge
430 AFTER UPDATE ON hospital_patients
431 FOR EACH ROW
432 WHEN (OLD.is_active = true AND NEW.is_active = false)
433 EXECUTE FUNCTION update_room_after_discharge();
434
435 UPDATE hospital_patients
436 SET is_active = false
437 WHERE patient_id = 3; -- Example: Discharging Robert Johnson
438
439 SELECT * FROM hospital_patients WHERE patient_id = 3;
440
441
442
443
444
445
446
447

```

Data Output Explain Messages

patientId	first_name	last_name	date_of_birth	gender	admission_date	discharge_date	diagnosis	treat
1	Robert	Johnson	1992-09-09	Male	2024-01-15	2024-01-20	Diabetes	Insul

Successfully run. Total query runtime: 80 msec. 1 rows affected.

ENGLISH IN 17:24 12-03-2025

PGAdmin File Object Tools Help

Browser Properties SQL Statistics Dependencies Dependents hospitalmanagementsystem/postgres@PostgreSQL 12*

```

> Materialized View
> Procedures
> Sequences
> Tables (6)
>   airport
>   employee_aud
>   Columns
>   Constraints
>   Indexes
>   Rules
>   Triggers
>   employees
>   flight
>   ticket
>   worker
>   Trigger Functions
>   Types
>   Views (2)
>     airport_view
>     at_view

```

```

429 FOR EACH ROW
430 WHEN (OLD.is_active = true AND NEW.is_active = false)
431 EXECUTE FUNCTION update_room_after_discharge();
432
433 UPDATE hospital_patients
434 SET is_active = false
435 WHERE patient_id = 3; -- Example: Discharging Robert Johnson
436
437 SELECT * FROM hospital_patients WHERE patient_id = 3;
438
439 CREATE VIEW archived_patients AS
440 SELECT first_name, last_name, diagnosis, treatment_plan, assigned_doctor, room_number
441 FROM hospital_patients
442 WHERE is_active = false;
443
444
445
446
447

```

Data Output Explain Messages

CREATE VIEW

Query returned successfully in 80 msec.

Query returned successfully in 80 msec.

ENGLISH IN 17:25 12-03-2025

pgAdmin File Object Tools Help

Browser

```

> Materialized View
> Procedures
> Sequences
> Tables (6)
  > airport
  > employee_aud
    > Columns
    > Constraints
    > Indexes
    > Rules
    > Triggers
  > employees
  > flight
  > ticket
  > worker
  > Trigger Functions
  > Types
  > Views (2)
    > airport_view
    > at_view

```

```

<#> FOR EACH ROW
430 WHEN (OLD.is_active = true AND NEW.is_active = false)
431 EXECUTE FUNCTION update_room_after_discharge();
432
433 UPDATE hospital_patients
434 SET is_active = false
435 WHERE patient_id = 3; -- Example: Discharging Robert Johnson
436
437 SELECT * FROM hospital_patients WHERE patient_id = 3;
438
439 CREATE VIEW archived_patients AS
440   SELECT first_name, last_name, diagnosis, treatment_plan, assigned_doctor, room_number
441   FROM hospital_patients
442   WHERE is_active = false;
443
444 SELECT * FROM archived_patients;
445
446

```

hospitalmanagementsystem

```

> Casts
> Catalogs
> Event Triggers
> Extensions
> Foreign Data Wrappers
> Languages
> Schemas
> postres
> Login/Group Roles
> Tablespace

```

Data Output Explain Messages

first_name	last_name	diagnosis	treatment_plan	assigned_doctor	room_number
Robert	Johnson	Diabetes	Insulin and lifestyle ...	Dr. Lee	305

Successfully run. Total query runtime: 59 msec. 1 rows affected.

ENG IN 17:25 12-03-2025

pgAdmin File Object Tools Help

Browser

```

> Materialized View
> Procedures
> Sequences
> Tables (6)
  > airport
  > employee_aud
    > Columns
    > Constraints
    > Indexes
    > Rules
    > Triggers
  > employees
  > flight
  > ticket
  > worker
  > Trigger Functions
  > Types
  > Views (2)
    > airport_view
    > at_view

```

```

438
439 CREATE VIEW archived_patients AS
440   SELECT first_name, last_name, diagnosis, treatment_plan, assigned_doctor, room_number
441   FROM hospital_patients
442   WHERE is_active = false;
443
444 SELECT * FROM archived_patients;
445
446 CREATE OR REPLACE FUNCTION update_insurance_provider()
447 RETURNS TRIGGER AS $$
448 BEGIN
449   IF NEW.admission_date < '2024-01-01' THEN
450     NEW.insurance_provider := 'Old Insurance Plan';
451   ELSE
452     NEW.insurance_provider := 'New Insurance Plan';
453   END IF;
454   RETURN NEW;
455 END;
456 $$ LANGUAGE plpgsql;

```

hospitalmanagementsystem

```

> Casts
> Catalogs
> Event Triggers
> Extensions
> Foreign Data Wrappers
> Languages
> Schemas
> postres
> Login/Group Roles
> Tablespace

```

Data Output Explain Messages

CREATE FUNCTION

Query returned successfully in 41 msec.

Query returned successfully in 41 msec.

ENG IN 17:26 12-03-2025

TRIGGER

PGAdmin File Object Tools Help

Browser Properties SQL Statistics Dependencies Dependents hospitalmanagementsystem/postgres@PostgreSQL 12*

Query Editor Query History

```

445 CREATE OR REPLACE FUNCTION update_insurance_provider()
446   RETURNS TRIGGER AS $$
447 BEGIN
448   IF NEW.admission_date < '2024-01-01' THEN
449     NEW.insurance_provider := 'Old Insurance Plan';
450   ELSE
451     NEW.insurance_provider := 'New Insurance Plan';
452   END IF;
453   RETURN NEW;
454 END;
455 $$ LANGUAGE plpgsql;
456
457 CREATE TRIGGER before_patient_admission
458 BEFORE INSERT OR UPDATE ON hospital_patients
459 FOR EACH ROW
460 EXECUTE FUNCTION update_insurance_provider();
461
462
463

```

Data Output Explain Messages

CREATE TRIGGER

Query returned successfully in 87 msec.

17:26 12-03-2025

PGAdmin File Object Tools Help

Browser Properties SQL Statistics Dependencies Dependents hospitalmanagementsystem/postgres@PostgreSQL 12*

Query Editor Query History

```

449 IF NEW.admission_date < '2024-01-01' THEN
450   NEW.insurance_provider := 'Old Insurance Plan';
451 ELSE
452   NEW.insurance_provider := 'New Insurance Plan';
453 END IF;
454 RETURN NEW;
455 $$ LANGUAGE plpgsql;
456
457 CREATE TRIGGER before_patient_admission
458 BEFORE INSERT OR UPDATE ON hospital_patients
459 FOR EACH ROW
460 EXECUTE FUNCTION update_insurance_provider();
461
462
463
464 INSERT INTO hospital_patients (first_name, last_name, date_of_birth, gender, admission_date, discharge_date, diagnosis, treatment_plan)
465 VALUES ('Lucy', 'Williams', '1995-05-10', 'Female', '2023-12-15', NULL, 'Asthma', 'Inhalers and monitoring', 'Dr. Morgan', '555-678-9
466
467

```

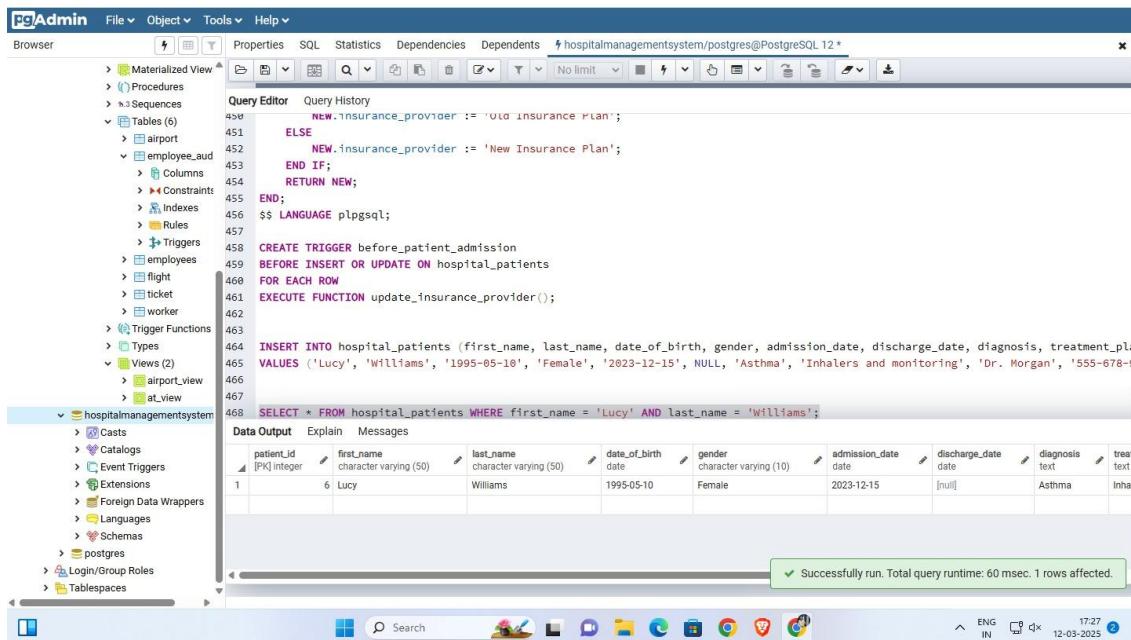
Data Output Explain Messages

INSERT 0 1

Query returned successfully in 87 msec.

✓ Query returned successfully in 87 msec.

17:27 12-03-2025



```

450     NEW.insurance_provider := 'Old Insurance Plan';
451
452     ELSE
453         NEW.insurance_provider := 'New Insurance Plan';
454     END IF;
455
456     $$ LANGUAGE plpgsql;
457
458     CREATE TRIGGER before_patient_admission
459     BEFORE INSERT OR UPDATE ON hospital_patients
460     FOR EACH ROW
461     EXECUTE FUNCTION update_insurance_provider();
462
463
464     INSERT INTO hospital_patients (first_name, last_name, date_of_birth, gender, admission_date, discharge_date, diagnosis, treatment_plan)
465     VALUES ('Lucy', 'Williams', '1995-05-10', 'Female', '2023-12-15', NULL, 'Asthma', 'Inhalers and monitoring', 'Dr. Morgan', '555-678-9');
466
467
468     SELECT * FROM hospital_patients WHERE first_name = 'Lucy' AND last_name = 'Williams';

```

patientId	first_name	last_name	date_of_birth	gender	admission_date	discharge_date	diagnosis	treatment_plan
1	6 Lucy	Williams	1995-05-10	Female	2023-12-15	[null]	Asthma	Inhalers and monitoring

Successfully run. Total query runtime: 60 msec. 1 rows affected.

Conclusion:

The above experiment highlights working with SQL queries based on views and triggers on our database.

Post Lab Questions:

1. What is a view in SQL, and how does it differ from a table?

A view in SQL is a virtual table that is based on the result of a SQL query. Unlike a table, a view does not store data permanently; instead, it dynamically retrieves data from the underlying tables whenever accessed. Views are useful for simplifying complex queries, restricting access to certain columns, and improving security.

2. Write a query to create a view that displays only Fname, Lname, and Salary from the employee table.

```
CREATE VIEW EmployeeView AS
SELECT Fname, Lname, Salary
FROM employee;
```

3. Write a query to drop the trigger trg_city_update from the employee table.

```
DROP TRIGGER trg_city_update ON employee;
```