UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Experiment No. 3

Student Name: Roshan Kumar Singh
UID: 25MCA20067
Branch: MCA(GEN)
Section/Group: 25MCA-1_A
Semester: 2nd
Date of Performance: 27/01/26
Subject Name: Technical Training-1
Subject Code: 25CAP-652

Aim:

To implement conditional decision-making logic in PostgreSQL using IF-ELSE constructs and CASE expressions for classification, validation, and rule-based data processing.

Objective:

- To understand conditional execution in SQL
- To implement decision-making logic using CASE expressions
- To simulate real-world rule validation scenarios
- To classify data based on multiple conditions
- To strengthen SQL logic skills required in interviews and backend systems

S/W Requirement:

PostgreSQL

Procedure:

Prerequisite Understanding

First create a table that stores:

- A unique identifier

- A schema or entity name
- A numeric count representing violations or issues

Populate the table with multiple records having different violation counts.

Step 1: Classifying Data Using CASE Expression

- Retrieve schema names and their violation counts.
- Use conditional logic to classify each schema into categories such as:
  - No Violation
  - Minor Violation
  - Moderate Violation
  - Critical Violation

Step 2: Applying CASE Logic in Data Updates

- Add a new column to store approval status.
- Update this column based on violation count using conditional rules such as:
  - Approved
  - Needs Review
  - Rejected

Step 3: Implementing IF–ELSE Logic Using PL/pgSQL

- Use a procedural block instead of a SELECT statement.
- Declare a variable representing violation count.
- Display different messages based on the value of the variable using IF–ELSE logic.

Step 4: Real-World Classification Scenario (Grading System)

- Create a table to store student names and marks.
- Classify students into grades based on their marks using conditional logic.

Step 5: Using CASE for Custom Sorting

- Retrieve schema details.
- Apply conditional priority while sorting records based on violation severity.

Code:

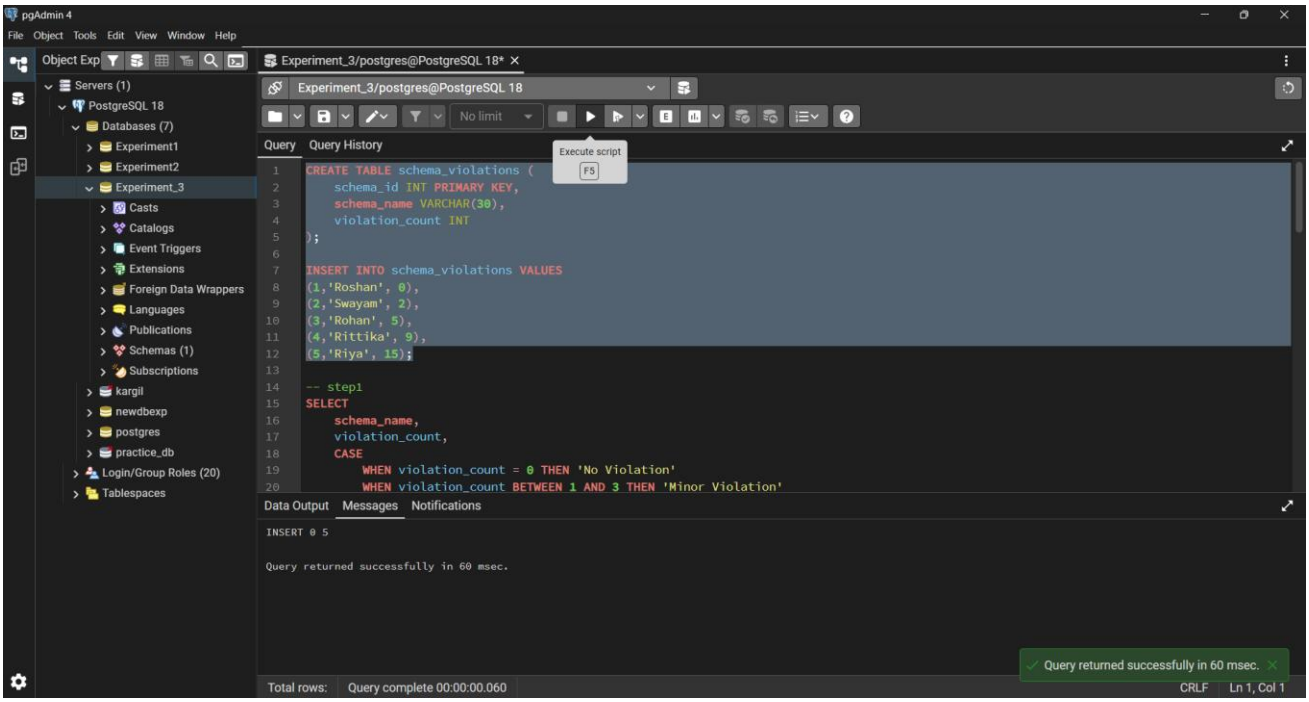
```
CREATE TABLE schema_violations (
   schema_id INT PRIMARY KEY,
   schema_name VARCHAR(30),
   violation_count INT
);

INSERT INTO schema_violations VALUES
(1,'Roshan', 0),
(2,'Swayam', 2),
(3,'Rohan', 5),
(4,'Rittika', 9),
(5,'Riya', 15);

-- step1
SELECT
   schema_name,
   violation_count,
   CASE
         WHEN violation_count = 0 THEN 'No Violation'
         WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
         WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation' ELSE
'Critical Violation' END AS violation_status
FROM schema_violations;


-- step2

ALTER TABLE schema_violations
```

```
ADD COLUMN approval_status VARCHAR(20);

UPDATE schema_violations
SET approval_status = CASE WHEN violation_count = 0 THEN 'Approved' WHEN
violation_count <= 5 THEN 'Needs Review' ELSE 'Rejected' END;

SELECT * FROM schema_violations;

--step 3

DO $$
DECLARE
   v_count INT := 6;
BEGIN
   IF v_count = 0 THEN
      RAISE NOTICE 'No violations detected';
   ELSIF v_count <= 5 THEN
      RAISE NOTICE 'Minor violations review required';
   ELSE
      RAISE NOTICE 'Critical violations access denied';
   END IF;
END $$;

--step 4
CREATE TABLE students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(30),
   marks INT
);

INSERT INTO students VALUES
(1,'Roshan',85),
(2,'Swayam',72),
(3,'Rohan',60),
(4,'Rittika',45),
(5,'Riya',30);


SELECT student_name,marks,
   CASE
      WHEN marks >= 80 THEN 'A'
```

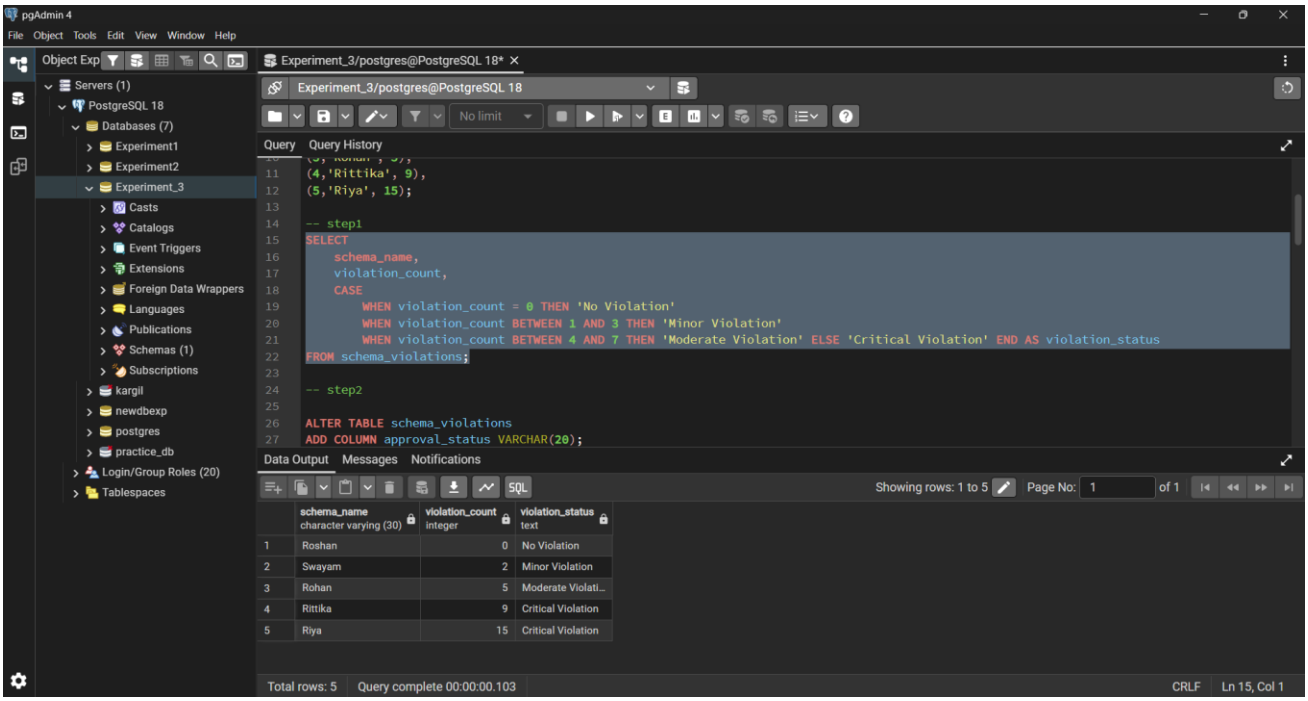
```
        WHEN marks >= 70 THEN 'B'
        WHEN marks >= 60 THEN 'C'
        WHEN marks >= 40 THEN 'D'
        ELSE 'F'
    END AS grade
FROM students;

--step 5

SELECT schema_name,violation_count FROM schema_violations
ORDER BY CASE
            WHEN violation_count = 0 THEN 1
            WHEN violation_count <= 3 THEN 2
            WHEN violation_count <= 7 THEN 3
            ELSE 4
            END;
```

Output:

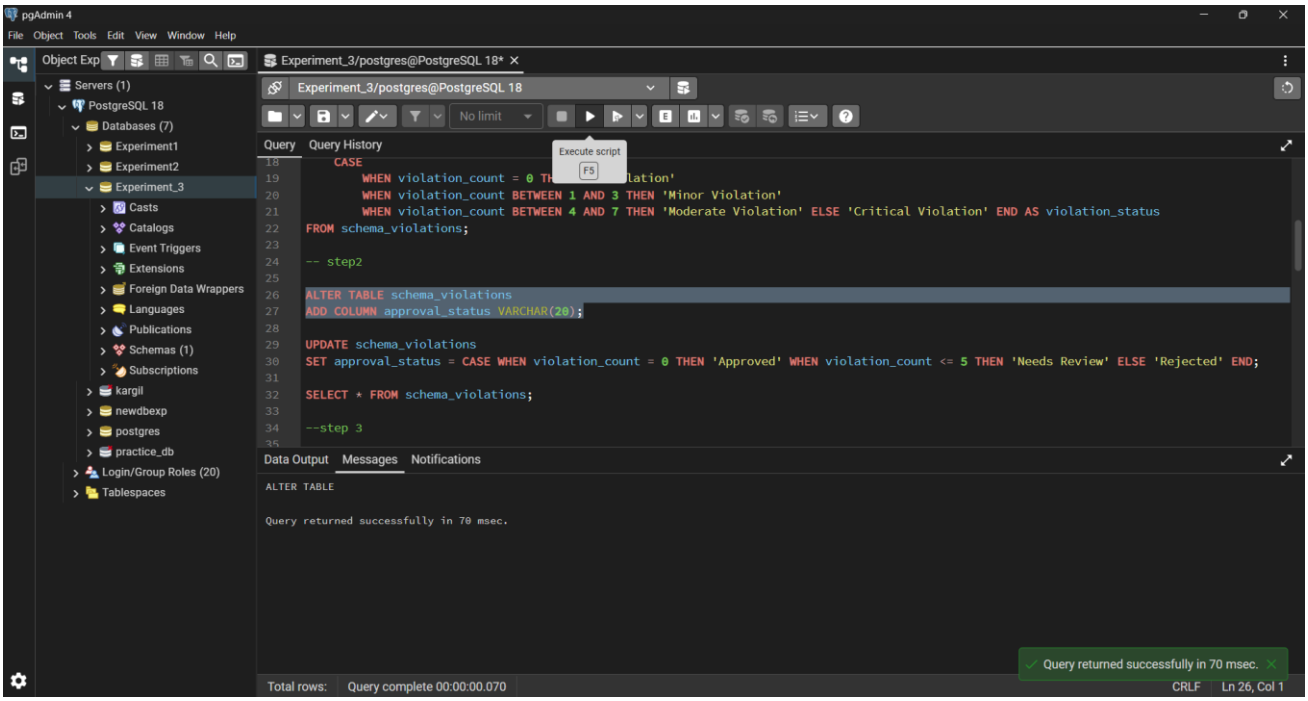
Prerequisite table creation



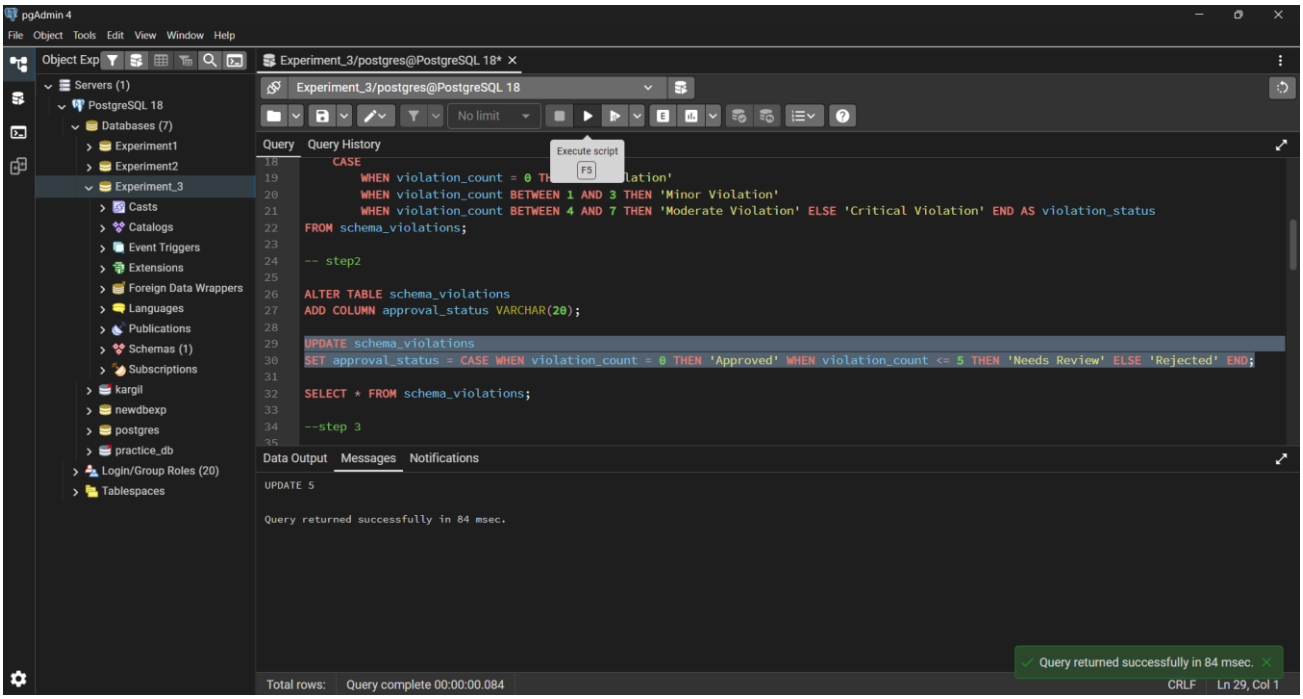
Step1



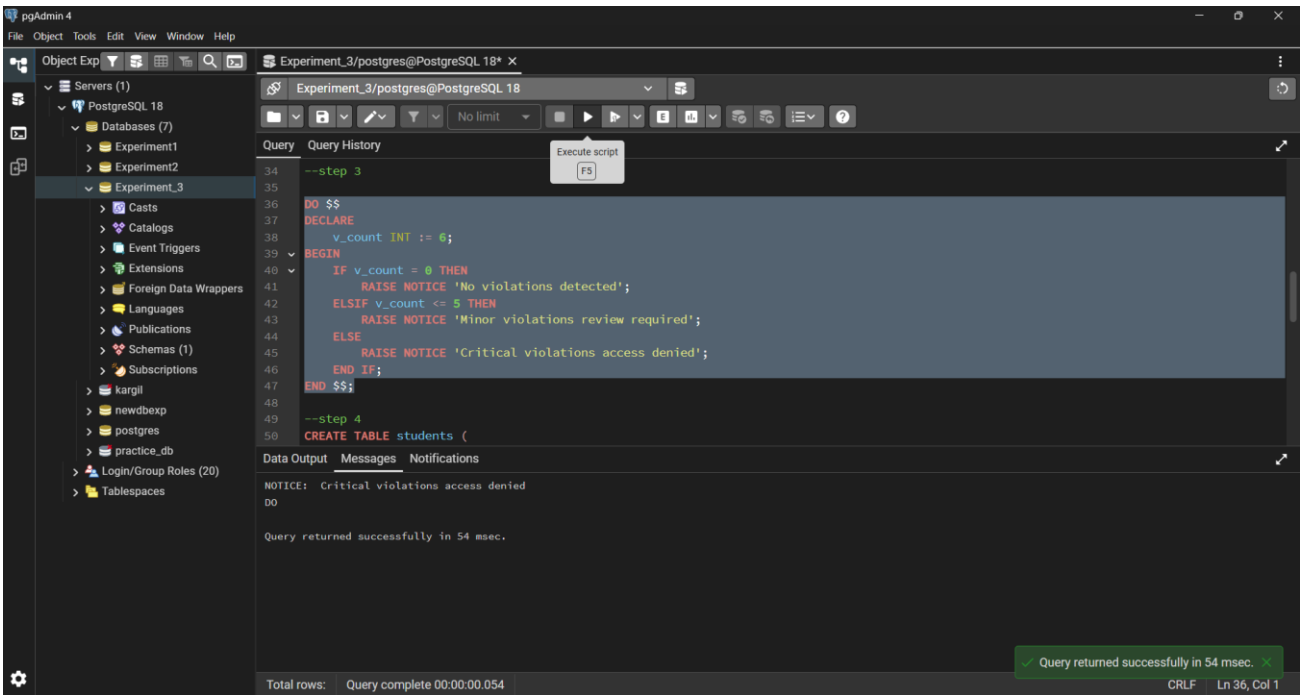
Step2 add column



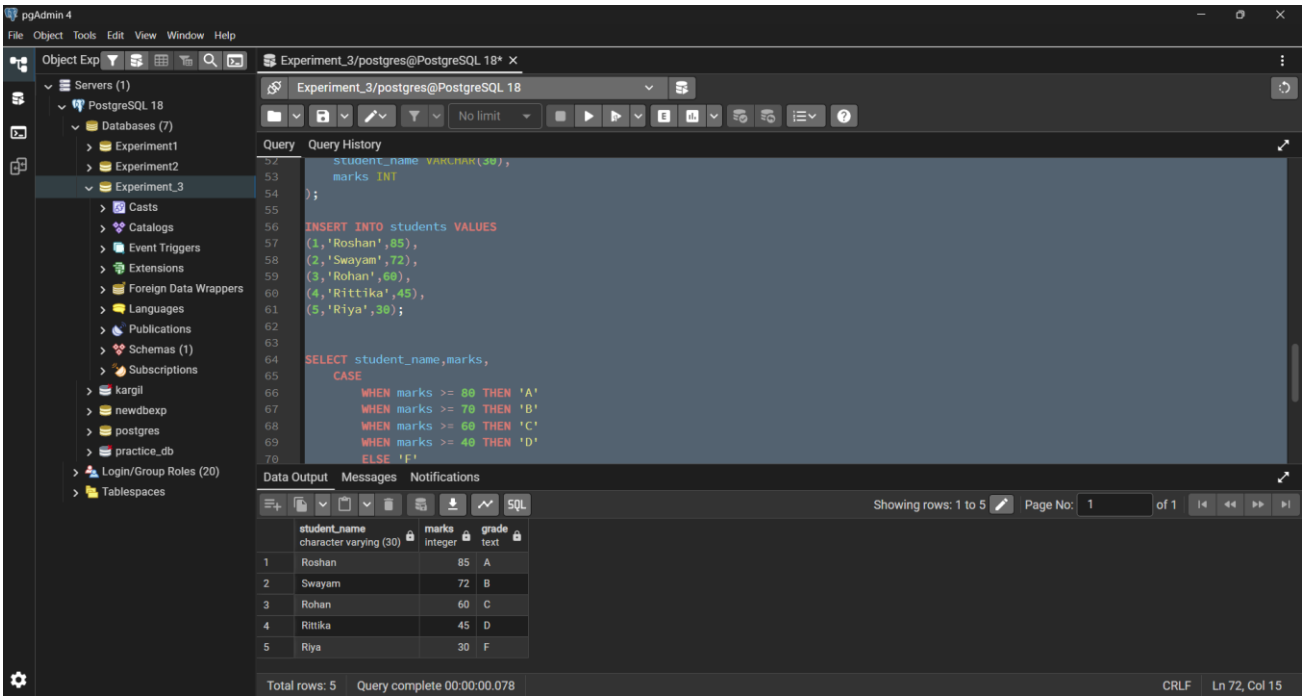
Step 2 update column



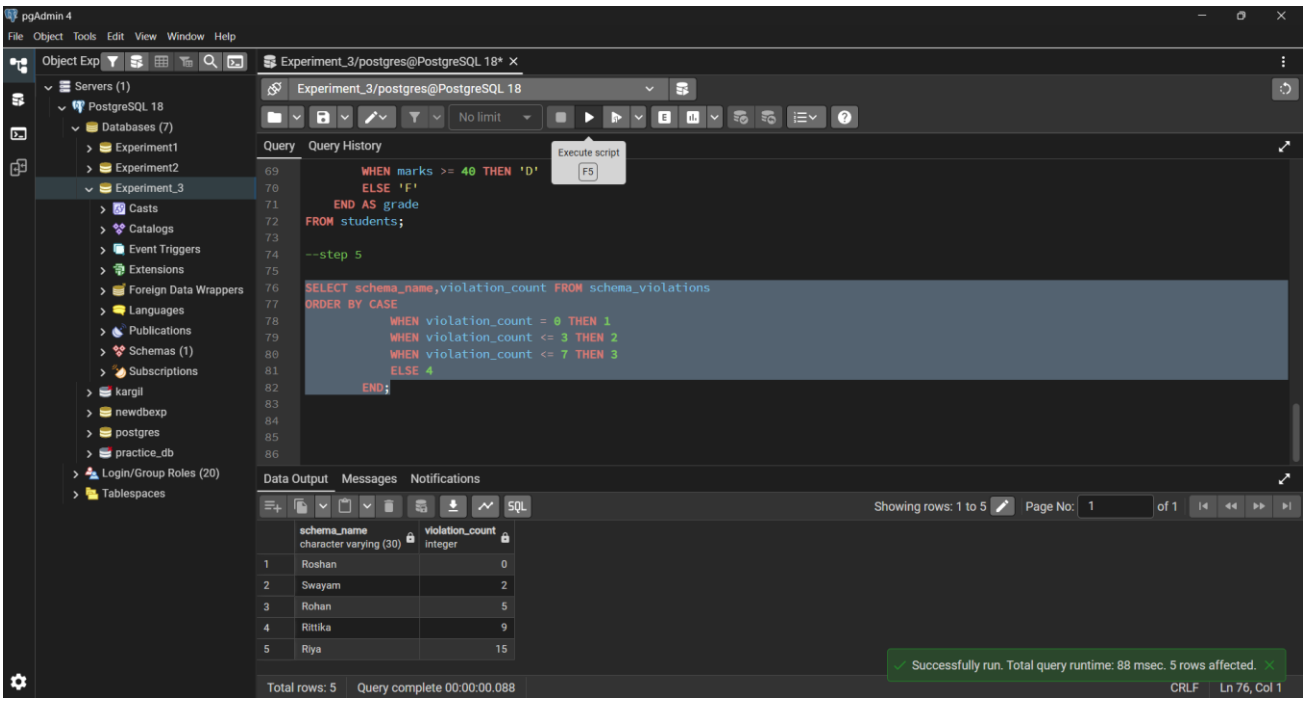
Step3



Step4


pgAdmin 4
File Object Tools Edit View Window Help
Experiment_3/postgres@PostgreSQL 18*
Query Query History
marks INT
);
INSERT INTO students VALUES
(1,'Roshan',85),
(2,'Swayam',72),
(3,'Rohan',60),
(4,'Rittika',45),
(5,'Riya',30);
SELECT student_name,marks,
CASE
WHEN marks >= 80 THEN 'A'
WHEN marks >= 70 THEN 'B'
WHEN marks >= 60 THEN 'C'
WHEN marks >= 40 THEN 'D'
ELSE 'F'
Data Output Messages Notifications
student_name character varying (30)
marks integer
grade text
1 Roshan 85 A
2 Swayam 72 B
3 Rohan 60 C
4 Rittika 45 D
5 Riya 30 F
Total rows: 5 Query complete 00:00:00.078
CRLF Ln 72, Col 15

Step5


pgAdmin 4
File Object Tools Edit View Window Help
Experiment_3/postgres@PostgreSQL 18*
Query Query History
Execute script
F5
WHEN marks >= 40 THEN 'D'
ELSE 'F'
END AS grade
FROM students;
--step 5
SELECT schema_name,violation_count FROM schema_violations
ORDER BY CASE
WHEN violation_count = 0 THEN 1
WHEN violation_count <= 3 THEN 2
WHEN violation_count <= 7 THEN 3
ELSE 4
END;
Data Output Messages Notifications
schema_name character varying (30)
violation_count integer
1 Roshan 0
2 Swayam 2
3 Rohan 5
4 Rittika 9
5 Riya 15
Successfully run. Total query runtime: 88 msec. 5 rows affected.
Total rows: 5 Query complete 00:00:00.088
CRLF Ln 76, Col 1

Learning Outcomes:

- Understood how conditional logic is implemented in PostgreSQL using CASE expressions and IF-ELSE constructs.
- Learnt how rule-based SQL logic helps in data classification and validation.
- Gained the ability to apply conditional decisions directly at the database level for backend systems.
- Clearly able to use CASE-based logic for analytics and compliance reporting scenarios.
- Developed confidence in writing interview-oriented SQL queries involving conditional decision-making.