

Experiment No. 3

Student Name: Roshan Kumar Singh

Branch: MCA(GEN)

Semester: 2nd

Subject Name: Technical Training-1

UID: 25MCA20067

Section/Group: 25MCA-1_A

Date of Performance: 27/01/26

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'
```

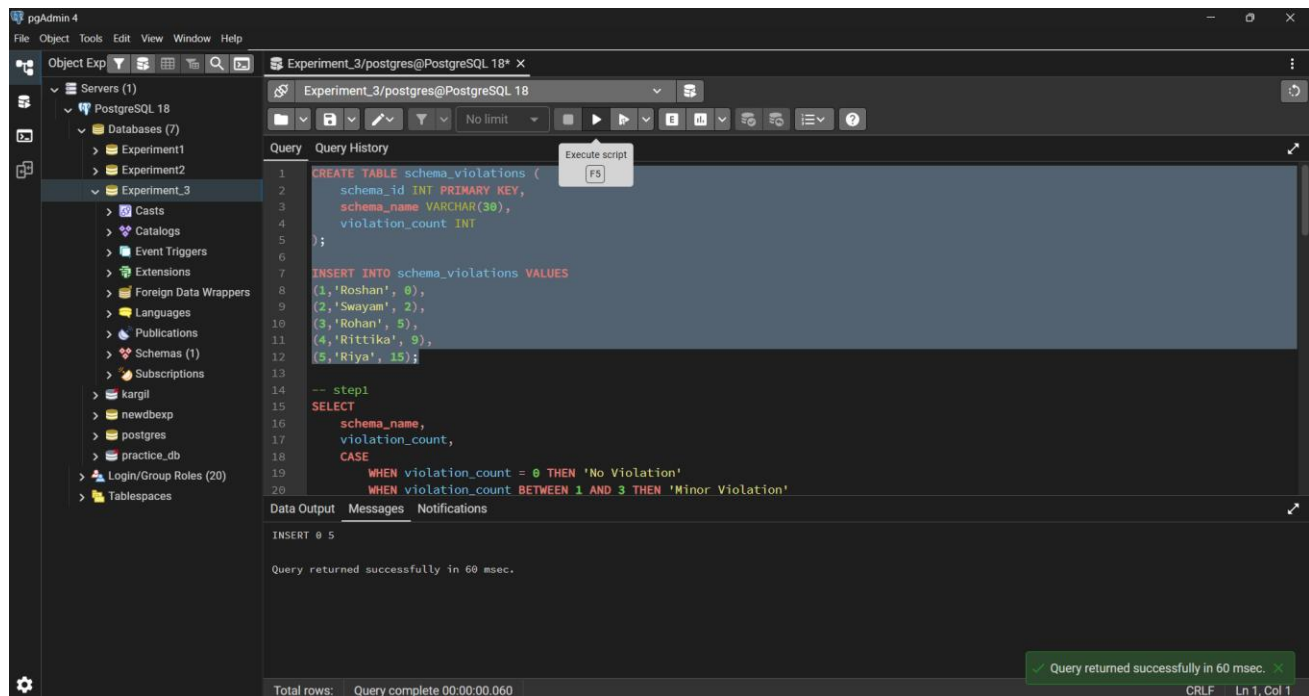
```
WHEN marks >= 70 THEN 'B'  
WHEN marks >= 60 THEN 'C'  
WHEN marks >= 40 THEN 'D'  
ELSE 'F'  
END AS grade  
FROM students;
```

--step 5

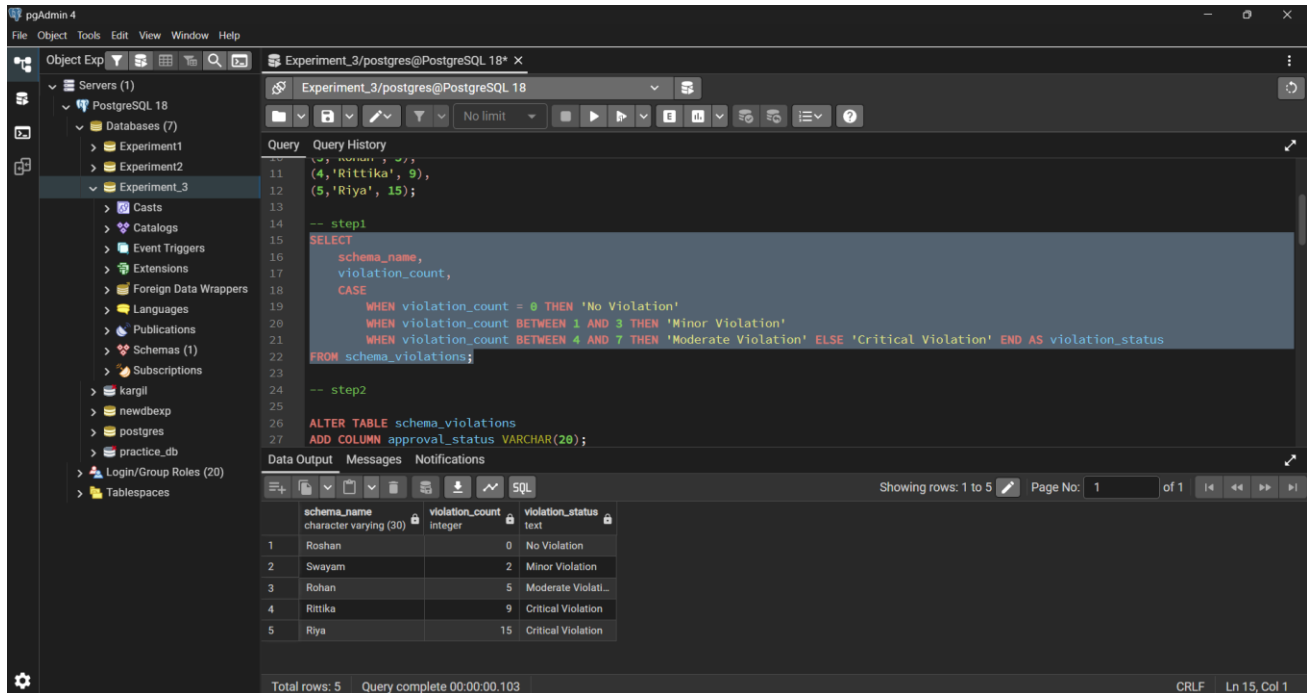
```
SELECT schema_name,violation_count FROM schema_violations  
ORDER BY CASE  
    WHEN violation_count = 0 THEN 1  
    WHEN violation_count <= 3 THEN 2  
    WHEN violation_count <= 7 THEN 3  
    ELSE 4  
END;
```

Output:

Prerequisite table creation



Step1



pgAdmin 4

Object Explorer: Servers (1) > PostgreSQL 18 > Databases (7) > Experiment3 > Experiment_3 > schema_violations

Query Editor: Experiment_3/postgres@PostgreSQL 18

```

11 (4,'Rittika', 9),
12 (5,'Riya', 15);
13
14 -- step1
15 SELECT
16     schema_name,
17     violation_count,
18     CASE
19         WHEN violation_count = 0 THEN 'No Violation'
20         WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
21         WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation' ELSE 'Critical Violation' END AS violation_status
22 FROM schema_violations;
23
24 -- step2
25 ALTER TABLE schema_violations
26 ADD COLUMN approval_status VARCHAR(20);
27

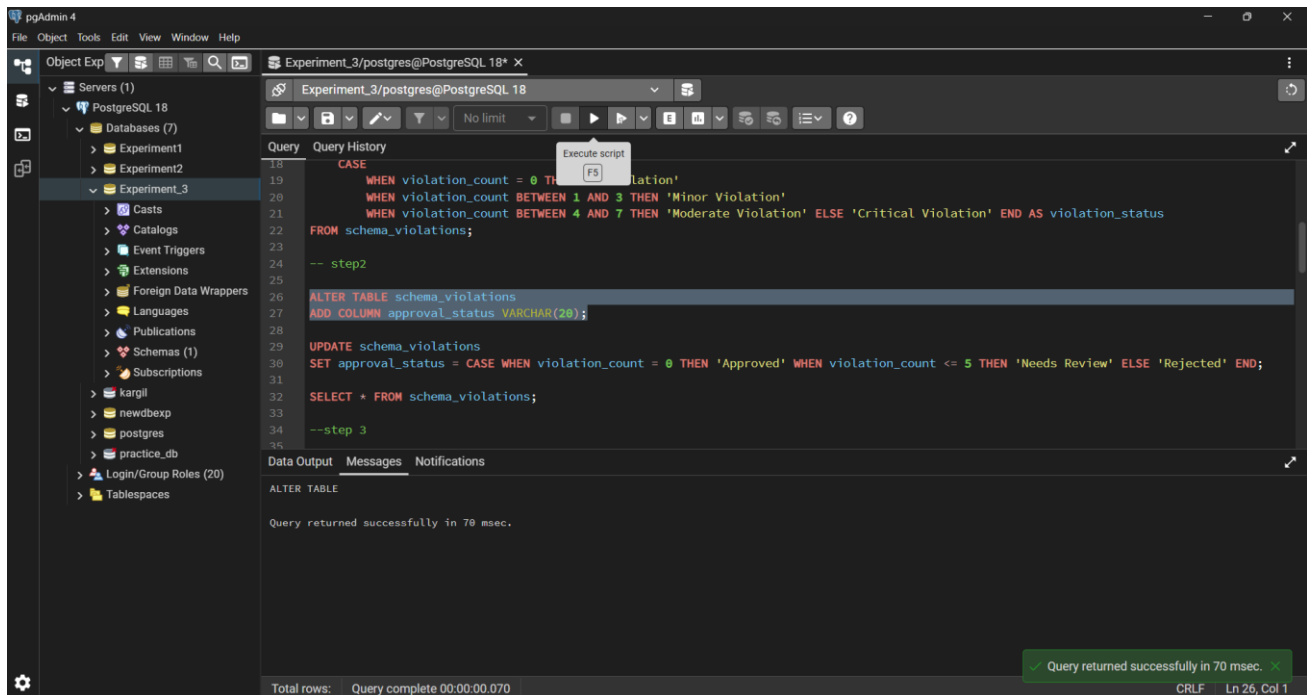
```

Data Output: Showing rows: 1 to 5 | Page No: 1 of 1

schema_name	violation_count	violation_status
Roshan	0	No Violation
Swayam	2	Minor Violation
Rohan	5	Moderate Violation
Rittika	9	Critical Violation
Riya	15	Critical Violation

Total rows: 5 | Query complete 00:00:00.103

Step2 add column



pgAdmin 4

Object Explorer: Servers (1) > PostgreSQL 18 > Databases (7) > Experiment3 > Experiment_3 > schema_violations

Query Editor: Experiment_3/postgres@PostgreSQL 18

```

18 CASE
19     WHEN violation_count = 0 THEN 'No Violation'
20     WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
21     WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation' ELSE 'Critical Violation' END AS violation_status
22 FROM schema_violations;
23
24 -- step2
25 ALTER TABLE schema_violations
26 ADD COLUMN approval_status VARCHAR(20);
27
28 UPDATE schema_violations
29 SET approval_status = CASE WHEN violation_count = 0 THEN 'Approved' WHEN violation_count <= 5 THEN 'Needs Review' ELSE 'Rejected' END;
30
31 SELECT * FROM schema_violations;
32
33 --step 3
34

```

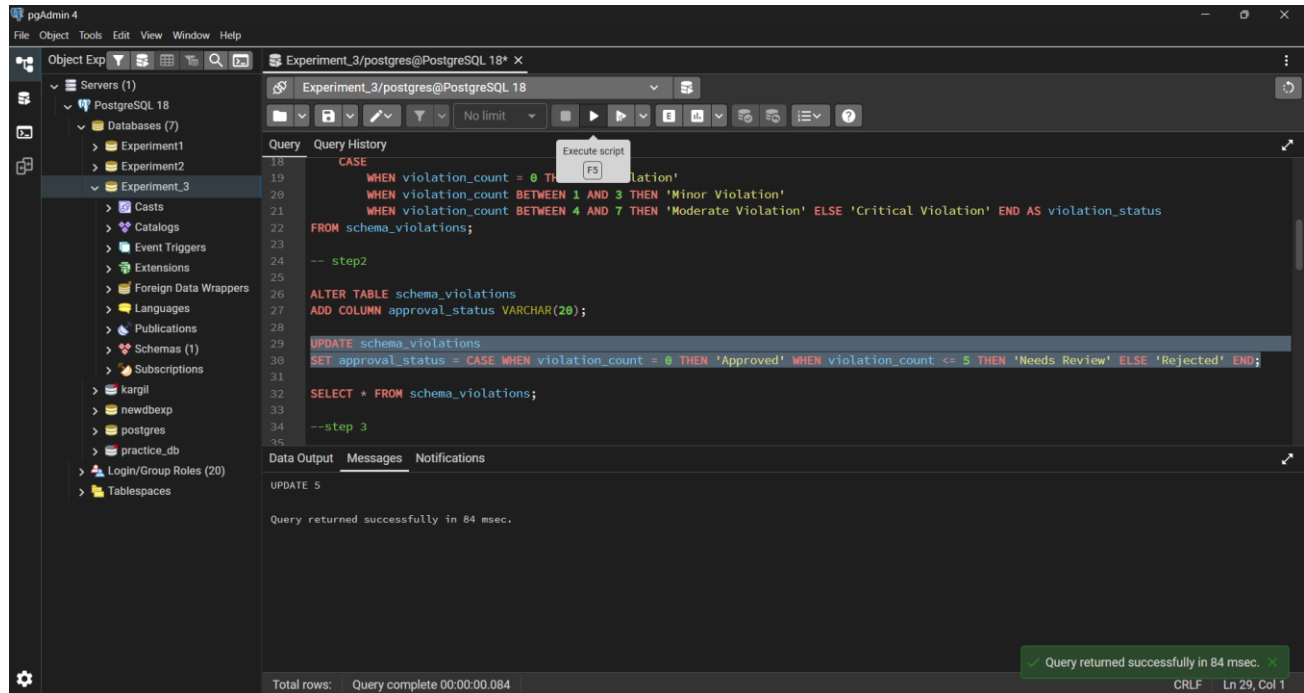
Data Output: ALTER TABLE

Query returned successfully in 70 msec.

Total rows: | Query complete 00:00:00.070

Query returned successfully in 70 msec.

Step 2 update column



```

18  CASE
19      WHEN violation_count = 0 THEN 'No Violation'
20      WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
21      WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation' ELSE 'Critical Violation' END AS violation_status
22  FROM schema_violations;
23
24  -- step2
25
26  ALTER TABLE schema_violations
27  ADD COLUMN approval_status VARCHAR(20);
28
29  UPDATE schema_violations
30  SET approval_status = CASE WHEN violation_count = 0 THEN 'Approved' WHEN violation_count <= 5 THEN 'Needs Review' ELSE 'Rejected' END;
31
32  SELECT * FROM schema_violations;
33
34  --step 3
35

```

Data Output Messages Notifications

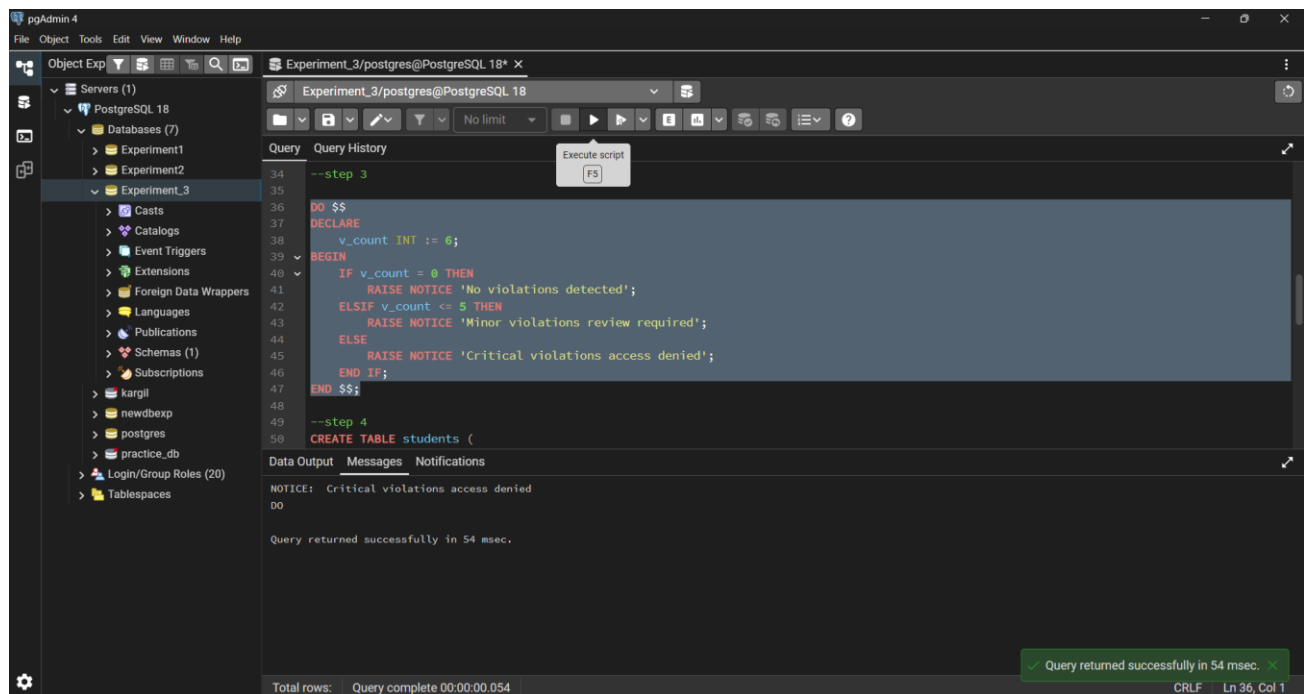
UPDATE 5

Query returned successfully in 84 msec.

Total rows: Query complete 00:00:00.084

Query returned successfully in 84 msec.

Step3



```

34  --step 3
35
36  DO $$
37  DECLARE
38      v_count INT := 0;
39  BEGIN
40      IF v_count = 0 THEN
41          RAISE NOTICE 'No violations detected';
42      ELSIF v_count <= 5 THEN
43          RAISE NOTICE 'Minor violations review required';
44      ELSE
45          RAISE NOTICE 'Critical violations access denied';
46      END IF;
47  END $$;
48
49  --step 4
50  CREATE TABLE students (

```

Data Output Messages Notifications

NOTICE: Critical violations access denied

DO

Query returned successfully in 54 msec.

Total rows: Query complete 00:00:00.054

Query returned successfully in 54 msec.

Step4

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Object Explorer' pane shows a tree structure of database objects under the 'Experiment_3/postgres@PostgreSQL 18' connection. The 'Databases' folder is expanded, showing 'Experiment1', 'Experiment2', and 'Experiment_3'. Under 'Experiment_3', 'Casts', 'Catalogs', 'Event Triggers', 'Extensions', 'Foreign Data Wrappers', 'Languages', 'Publications', 'Schemas (1)', and 'Subscriptions' are listed. The 'Schemas' folder is expanded, showing 'kargil', 'newdbexp', 'postgres', 'practice_db', 'Login/Group Roles (20)', and 'Tablespaces'.

The main pane shows the 'Query History' tab for the 'Experiment_3/postgres@PostgreSQL 18' connection. The query is as follows:

```
Query
Query History
52 student_name VARCHAR(30),
53 marks INT
54 );
55
56 INSERT INTO students VALUES
57 (1,'Roshan',85),
58 (2,'Swayam',72),
59 (3,'Rohan',60),
60 (4,'Ritika',45),
61 (5,'Riya',30);
62
63
64 SELECT student_name,marks,
65 CASE
66 WHEN marks >= 80 THEN 'A'
67 WHEN marks >= 70 THEN 'B'
68 WHEN marks >= 60 THEN 'C'
69 WHEN marks >= 40 THEN 'D'
70 ELSE 'F';
```

The 'Data Output' tab shows the results of the query:

student_name	marks	grade
Roshan	85	A
Swayam	72	B
Rohan	60	C
Ritika	45	D
Riya	30	F

The status bar at the bottom indicates 'Total rows: 5' and 'Query complete 00:00:00.078'. The bottom right corner shows 'CRLF' and 'Ln 72, Col 15'.

Step5

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the server hierarchy: Servers (1) > PostgreSQL 18 > Databases (7) > Experiment1 > Experiment2 > Experiment3. The main pane shows the 'Query' editor with a PL/pgSQL function. The function checks for schema violations in the 'schema_violations' table and returns the results. The query is executed successfully, and the results are displayed in a table with 5 rows. The status bar shows 'Successfully run. Total query runtime: 88 msec. 5 rows affected.'

Query:

```

--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;
```

Results:

	schema_name	violation_count
1	Roshan	0
2	Swayam	2
3	Rohan	5
4	Ritika	9
5	Riya	15

Status: Successfully run. Total query runtime: 88 msec. 5 rows affected.

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.