

## Worksheet -3

**Student Name:** SANCHIT KATOCH

**Branch:** MCA

**Semester:** 2<sup>nd</sup>

**Subject Name:** TECHNICAL TRAINING I

**UID:** 25MCA20059

**Section/Group:** 25MCA-1-A

**Date of Performance:** 27/01/2026

**Subject Code:** 25CAP-652

### 1. Aim/Overview of the practical:

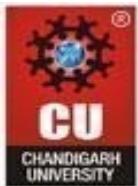
To implement conditional decision-making logic in PostgreSQL using IF-ELSE constructs and CASE expressions for classification, validation, and rule-based data processing.

### 2. 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 back-end systems.

### 3. S/W Requirement:

- Oracle Database Express Edition
- pgAdmin.



## Procedure:

### Start PostgreSQL Environment

#### 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.

### 4.Code:

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

```
INSERT INTO schemaViolations VALUES
(1,'Sanchit', 0),
(2,'Mandeep', 2),
(3,'Digam', 4),
(4,'Vanshaj', 7),
(5,'Divanshu', 8);
```

```
-- 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 schemaViolations;
```

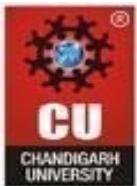
```
-- step2
```

```
ALTER TABLE schemaViolations
ADD COLUMN approval_status VARCHAR(20);
```

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

```
SELECT * FROM schemaViolations;
```

```
--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,'Sanchit', 88),  
(2,'Mandeep', 80),  
(3,'Digam', 72),  
(4,'Vanshaj',45),  
(5,'Divanshu', 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'
```

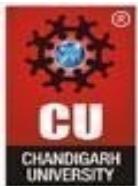
```
END AS grade
```

```
FROM students;
```

```
--step 5
```

```
SELECT schema_name,violation_count FROM schemaViolations  
ORDER BY CASE
```

```
    WHEN violation_count = 0 THEN 1  
    WHEN violation_count <= 3 THEN 2  
    WHEN violation_count <= 7 THEN 3
```

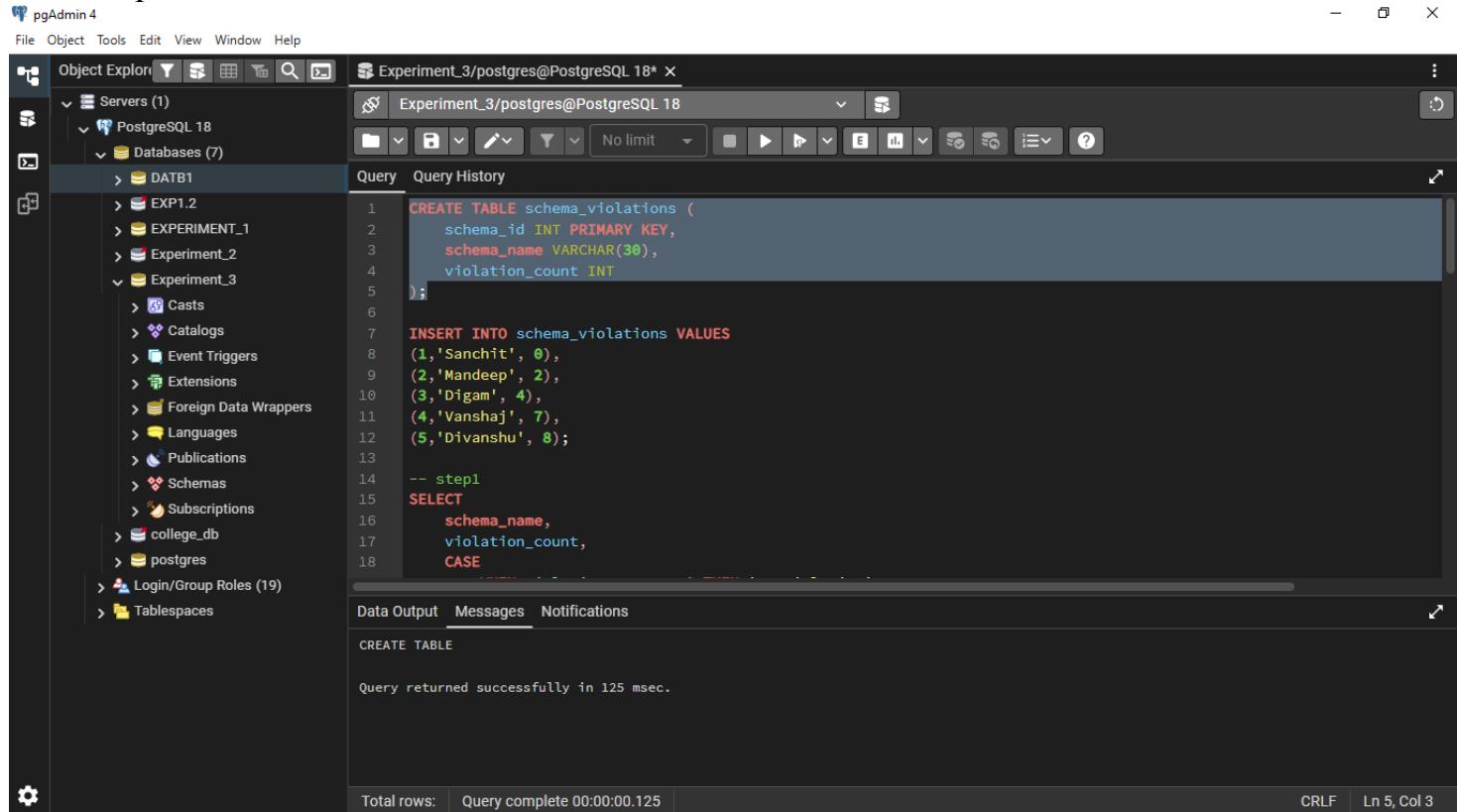


ELSE 4

END;

## 5.Output:

### Prerequisite table creation



The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer pane displays a tree structure of database objects. In the main Query tab, a SQL script is being run against the 'Experiment\_3/postgres@PostgreSQL 18' connection. The script creates a table 'schema\_violations' with three columns: 'schema\_id' (INT PRIMARY KEY), 'schema\_name' (VARCHAR(30)), and 'violation\_count' (INT). It then inserts five rows of data into the table. Finally, it performs a SELECT query to retrieve the schema name, violation count, and a CASE expression for each row.

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

INSERT INTO schema_violations VALUES
(1,'Sanchit', 0),
(2,'Mandeep', 2),
(3,'Digam', 4),
(4,'Vanshaj', 7),
(5,'Divanshu', 8);

-- step1
SELECT
    schema_name,
    violation_count,
    CASE
```

Messages tab content: CREATE TABLE  
Query returned successfully in 125 msec.

Step1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer    Query    Query History

Servers (1) PostgreSQL 18 Databases (7)

- DATB1
- EXP1.2
- EXPERIMENT\_1
- Experiment\_2
- Experiment\_3
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas
  - Subscriptions
- college\_db
- postgres
- Login/Group Roles (19)
- Tablespaces

```

13  -- step1
14  SELECT
15    schema_name,
16    violation_count,
17    CASE
18      WHEN violation_count = 0 THEN 'No Violation'
19      WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
20      WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation' ELSE 'Critical Violation' END AS violation_st
21  FROM schemaViolations;
22
23  -- step2
24
25  ALTER TABLE schemaViolations
26  ADD COLUMN approval_status VARCHAR(20);
27
28  UPDATE schemaViolations
29
30

```

Data Output Messages Notifications

INSERT 0 5

Query returned successfully in 99 msec.

Total rows: 0 Query complete 00:00:00.099 CRLF Ln 22, Col 24

## Step2 add – update - alter column

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer    Query    Query History

Servers (1) PostgreSQL 18 Databases (7)

- DATB1
- EXP1.2
- EXPERIMENT\_1
- Experiment\_2
- Experiment\_3
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas
  - Subscriptions
- college\_db
- postgres
- Login/Group Roles (19)
- Tablespaces

```

13  -- step1
14  SELECT
15    schema_name,
16    violation_count,
17    CASE
18      WHEN violation_count = 0 THEN 'No Violation'
19      WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
20      WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation' ELSE 'Critical Violation' END AS violation_status
21  FROM schemaViolations;
22
23  -- step2
24
25  ALTER TABLE schemaViolations
26  ADD COLUMN approval_status VARCHAR(20);
27
28  UPDATE schemaViolations
29  SET approval_status = CASE WHEN violation_count = 0 THEN 'Approved' WHEN violation_count <= 5 THEN 'Needs Review' ELSE 'Rejected' END;
30

```

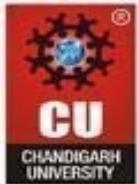
Execute script F5

Data Output Messages Notifications

UPDATE 5

Query returned successfully in 115 msec.

Total rows: 0 Query complete 00:00:00.115 CRLF Ln 31, Col 135



### Step3

The screenshot shows the pgAdmin 4 interface. On the left is the object browser tree, which includes a 'Servers' node with one entry ('PostgreSQL 18'), a 'Databases' node with seven entries ('DATB1', 'EXP1.2', 'EXPERIMENT\_1', 'Experiment\_2', 'Experiment\_3', 'college\_db', and 'postgres'), and a 'Login/Group Roles' node. The main window displays a query editor titled 'Experiment\_3/postgres@PostgreSQL 18\*'. The query itself is a PL/pgSQL script:

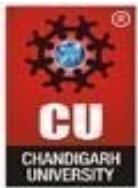
```
DO $$  
DECLARE  
    v_count INT := 6;  
BEGIN  
    IF v_count = 0 THEN  
        RAISE NOTICE 'No violations detected';  
    ELSIF v_count <= 5 THEN  
        RAISE NOTICE 'Minor violations review required';  
    ELSE  
        RAISE NOTICE 'Critical violations access denied';  
    END IF;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

Below the query editor, the 'Data Output' tab is active, showing the output of the query:

```
NOTICE: Critical violations access denied  
DO  
Query returned successfully in 157 msec.
```

At the bottom right of the pgAdmin window, there is a green status bar message: 'Query returned successfully in 157 msec.'

### Step4



pgAdmin 4

File Object Tools Edit View Window Help

Servers (1) PostgreSQL 18 Databases (7)

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

INSERT INTO students VALUES
(1, 'Sanchit', 88),
(2, 'Mandeep', 80),
(3, 'Digam', 72),
(4, 'Vanshaj', 45),
(5, 'Divanshu', 30);
```

Total rows: 5 Query complete 00:00:00.100

Execute script F5

Query History Data Output Messages Notifications

Query returned successfully in 100 msec.

✓ Query returned successfully in 100 msec. CRLF Ln 64, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Servers (1) PostgreSQL 18 Databases (7)

```
(5, 'Divanshu', 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;
```

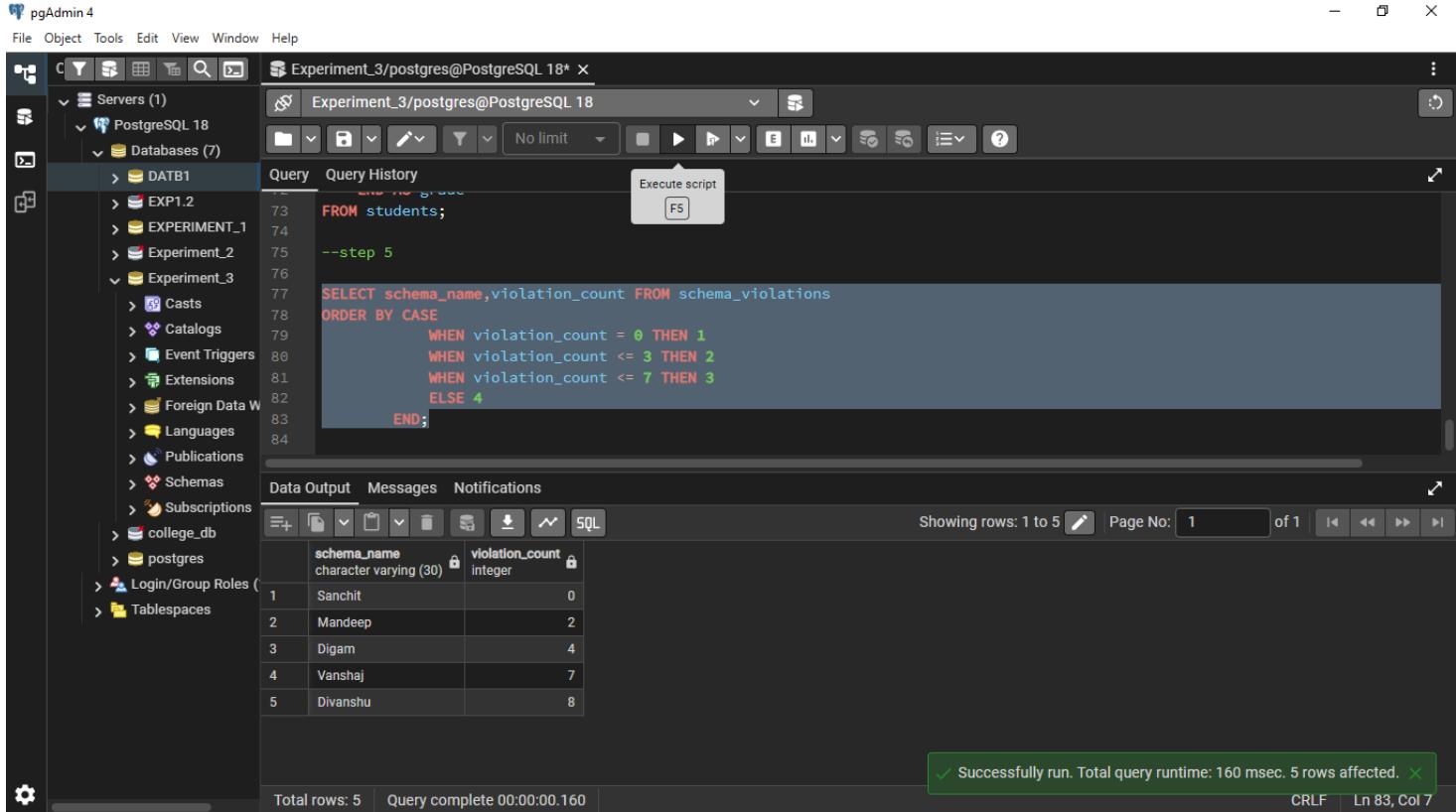
Total rows: 5 Query complete 00:00:00.146

Showing rows: 1 to 5 Page No: 1 of 1

	student_name	marks	grade
1	Sanchit	88	A
2	Mandeep	80	A
3	Digam	72	B
4	Vanshaj	45	D
5	Divanshu	30	F

✓ Successfully run. Total query runtime: 146 msec. 5 rows affected. CRLF Ln 74, Col 1

## Step5



```
FROM students;
--step 5
SELECT schema_name,violation_count FROM schemaViolations
ORDER BY CASE
    WHEN violation_count = 0 THEN 1
    WHEN violation_count <= 3 THEN 2
    WHEN violation_count <= 7 THEN 3
    ELSE 4
END;
```

schema_name	violation_count
Sanchit	0
Mandeep	2
Digam	4
Vanshaj	7
Divanshu	8

Successfully run. Total query runtime: 160 msec. 5 rows affected.

## 6.Learning Outcomes:

- \* Understood how conditional logic is implemented in PostgreSQL and 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 statements
- \* Clearly able to use CASE-based logic for analytics and compliance reporting scenarios.