



## WORKSHEET 3

**Student Name: Aradhya Sharma**

**UID: 25MCC20042**

**Branch: MCA (CCD)**

**Section/Group: 25MCC-101A**

**Semester: 2**

**Date of Performance: 27 JAN 2026**

**Subject Name: Technical training**

**Subject Code: 25CAP-652**

### 1. AIM:

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

### 2. OBJECTIVES:

- 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

### 3. SOFTWARE REQUIREMENTS

To perform this experiment, the following software is required:

#### 1. Operating System

- Windows 10 / 11, Linux, or macOS

#### 2. Database Management System (DBMS)

- PostgreSQL (version 12 or higher)

#### 3. SQL Client / Interface

- pgAdmin 4 (for PostgreSQL)

**OR**

- Command Line Interface (psql)

#### 4. PROCEDURE FOR EXPERIMENT

##### Step 1: Database and Table Preparation

```
CREATE TABLE violation_schema (  
    schema_id INT PRIMARY KEY,  
    schema_name VARCHAR(20),  
    violation_count INT  
);
```

```
INSERT INTO violation_schema (schema_id, schema_name, violation_count) VALUES  
(1, 'Customer_Schema', 0),  
(2, 'Security_Schema', 2),  
(3, 'Product_Schema', 5),  
(4, 'Project_Schema', 1),  
(5, 'Operation_Schema', 0);
```

Output:

	schema_id [PK] integer	schema_name character varying (20)	violation_count integer
1	1	Customer_Schema	0
2	2	Security_Schema	2
3	3	Product_Schema	5
4	4	Project_Schema	1
5	5	Operation_Schema	0

## Step 2: Classification of data using case

SELECT

schema\_name,

violation\_count,

CASE

WHEN violation\_count = 0 THEN 'No Violation'




WHEN violation\_count BETWEEN 1 AND 2 THEN 'Minor Violation'

ELSE 'Critical Violation'

END AS violation\_Status

FROM violation\_schema;

Output:

	<b>schema_name</b> character varying (20) 	<b>violation_count</b> integer 	<b>violation_status</b> text 
1	Customer_Schema	0	No Violation
2	Security_Schema	2	Minor Violation
3	Product_Schema	5	Critical Violation
4	Project_Schema	1	Minor Violation
5	Operation_Schema	0	No Violation

### Step 3: Applying CASE Logic in Data Updates

-- Add new status column

```
ALTER TABLE violation_schema
```

```
ADD COLUMN status VARCHAR(30);
```

-- Update violation status

```
UPDATE violation_schema
```

```
SET status =
```

```
  CASE
```

```
    WHEN violation_count = 0 THEN 'Approved'
```

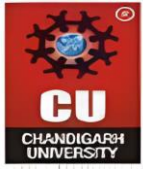
```
    WHEN violation_count BETWEEN 1 AND 2 THEN 'Needs Review'
```

```
    ELSE 'Rejected'
```

```
  END;
```

Output:

	schema_id [PK] integer	schema_name character varying (20)	violation_count integer	status character varying (30)
1	1	Customer_Schema	0	Approved
2	2	Security_Schema	2	Needs Review
3	3	Product_Schema	5	Rejected
4	4	Project_Schema	1	Needs Review
5	5	Operation_Schema	0	Approved



#### Step 4: Implementing IF–ELSE Logic Using PL/pgSQL

```
DO $$  
DECLARE  
    violation_count INT := 5;  
BEGIN  
  
    IF violation_count = 0 THEN  
        RAISE NOTICE 'Approved';  
    ELSIF violation_count BETWEEN 1 AND 2 THEN  
        RAISE NOTICE 'Needs Review';  
    ELSE  
        RAISE NOTICE 'Rejected';  
    END IF;  
END $$;
```

Output:

```
NOTICE:  Rejected  
DO  
  
Query returned successfully in 149 msec.
```

## Step 5: Real-World Grading System (Classification Scenario)

```
CREATE TABLE StudentGrades (
```

```
    uid INT PRIMARY KEY,
```

```
    student_name VARCHAR(50),
```

```
    marks INT
```

```
);
```

```
INSERT INTO StudentGrades (uid, student_name, marks) VALUES
```

```
(1, 'Hanna', 85),
```

```
(2, 'Jatin', 32),
```

```
(3, 'Gourish', 68),
```

```
(4, 'Vansh', 71),
```

```
(5, 'Akash', 56);
```

Output:

	uid [PK] integer	student_name character varying (50)	marks integer
1	1	Hanna	85
2	2	Jatin	32
3	3	Gourish	68
4	4	Vansh	71
5	5	Akash	56

### Step 5.1: Classify Students Using Conditional Logic

SELECT

student\_name,

marks,

CASE

WHEN marks >= 90 THEN 'A+ Grade'

WHEN marks BETWEEN 80 AND 89 THEN 'A Grade'

WHEN marks BETWEEN 70 AND 79 THEN 'B Grade'

WHEN marks BETWEEN 60 AND 69 THEN 'C Grade'

WHEN marks BETWEEN 40 AND 59 THEN 'D Grade'

ELSE 'Fail'

END AS Grade

FROM StudentGrades;

Output:

	student_name character varying (50) 🔒	marks integer 🔒	grade text 🔒
1	Hanna	85	A Grade
2	Jatin	32	Fail
3	Gourish	68	C Grade
4	Vansh	71	B Grade
5	Akash	56	D Grade

## Step 6: Using CASE for Custom Sorting

SELECT

schema\_name,

violation\_count,

CASE

WHEN violation\_count > 5 THEN 'Critical Violation'

WHEN violation\_count BETWEEN 3 AND 4 THEN 'Moderate Violation'

WHEN violation\_count BETWEEN 1 AND 2 THEN 'Minor Violation'

ELSE 'No Violation'

END AS violation\_status

FROM violation\_schema

ORDER BY

CASE

WHEN violation\_count > 5 THEN 1

WHEN violation\_count BETWEEN 3 AND 4 THEN 2

WHEN violation\_count BETWEEN 1 AND 2 THEN 3

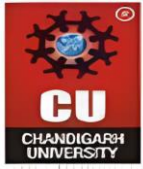
ELSE 4

END;

Output:

	schema_name character varying (20) 🔒	violation_count integer 🔒	violation_status text 🔒
1	Security_Schema	2	Minor Violation
2	Project_Schema	1	Minor Violation
3	Customer_Schema	0	No Violation
4	Product_Schema	5	No Violation
5	Operation_Schema	0	No Violation





UNIVERSITY INSTITUTE *of*  
**COMPUTING**  
*Asia's Fastest Growing University*



## Learning Outcome

This experiment demonstrates how conditional logic is implemented in PostgreSQL using **CASE expressions** and **IF–ELSE constructs**.

Students gain strong command over **rule-based SQL logic**, which is essential for:

- Backend systems
- Analytics
- Compliance reporting
- Placement and technical interviews