



**Student Name:** Sahil Hans  
**Branch:** M.C.A(A.I & M.L)  
**Semester:** Second Sem  
**Subject Name:** TECHINCAL SKILLS

**UID:** 25MCI10088  
**Section/Group:** MAM-1 A  
**Date of Performance:** 12/01/2026  
**Subject Code:**

### WORKSHEET 3

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

**S/W Requirement:** Oracle Database Express Edition and pgAdmin

#### **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

#### **Practical / Experiment Steps**

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

Students should 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

Querry:

```
CREATE TABLE Violations (
```

```
    id INT PRIMARY KEY,  
    entity_name VARCHAR(100),  
    violation_count INT  
);
```

```
INSERT INTO Violations (id, entity_name, violation_count) VALUES
(1, 'Finance_Department', 12),
(2, 'HR_Department', 5),
(3, 'IT_Department', 20),
(4, 'Sales_Department', 0),
(5, 'Admin_Department', 0),
(6, 'Security_Team', 15);
```

Output:

	<b>id</b> [PK] integer	<b>entity_name</b> character varying (100)	<b>violation_count</b> integer
1	1	Finance_Department	12
2	2	HR_Department	5
3	3	IT_Department	20
4	4	Sales_Department	0
5	5	Admin_Department	0
6	6	Security_Team	15

## Step 2: 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

Querry:

```
SELECT
entity_name,
violation_count,
CASE
WHEN violation_count = 0 THEN 'No Violation'
```

```

WHEN violation_count BETWEEN 1 AND 5 THEN 'Minor Violation'
WHEN violation_count BETWEEN 6 AND 15 THEN 'Moderate Violation'
ELSE 'Critical Violation'
END AS Violation_Status
FROM Violations;

```

Output:

	entity_name character varying (100)	violation_count integer	violation_status text
1	Finance_Department	12	Moderate Violati...
2	HR_Department	5	Minor Violation
3	IT_Department	20	Critical Violation
4	Sales_Department	0	No Violation
5	Admin_Department	0	No Violation
6	Security_Team	15	Moderate Violati...

### Step 3: 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

Querry:

```

ALTER TABLE Violations
ADD COLUMN approval_status VARCHAR(30);

```

```
UPDATE Violations
```

```
SET approval_status =
```

```
CASE
```

```

WHEN violation_count = 0 THEN 'Approved'
WHEN violation_count BETWEEN 1 AND 5 THEN 'Needs Review'
WHEN violation_count BETWEEN 6 AND 15 THEN 'Needs Review'

```

ELSE 'Rejected'

END;

Output:

	<b><i>id</i></b> [PK] integer	<b><i>entity_name</i></b> character varying (100)	<b><i>violation_count</i></b> integer	<b><i>approval_status</i></b> character varying (30)
1	1	Finance_Department	12	Needs Review
2	2	HR_Department	5	Needs Review
3	3	IT_Department	20	Rejected
4	4	Sales_Department	0	Approved
5	5	Admin_Department	0	Approved
6	6	Security_Team	15	Needs Review

#### Step 4: 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.

Querry:

```

DO $$

DECLARE
    vViolationCount INT := 12; -- change this value to test

BEGIN
    IF vViolationCount = 0 THEN
        RAISE NOTICE 'Status: Approved — No violations found';
    ELSIF vViolationCount BETWEEN 1 AND 5 THEN
        RAISE NOTICE 'Status: Needs Review — Minor violations detected';
    ELSIF vViolationCount BETWEEN 6 AND 15 THEN
        RAISE NOTICE 'Status: Needs Review — Moderate violations detected';
    ELSE
        RAISE NOTICE 'Status: Rejected — Critical violations detected';
    END IF;
END;

```

END IF;

END \$\$;

Output:

```
NOTICE: Status: Needs Review – Moderate violations detected
DO

Query returned successfully in 62 msec.
```

### Step 5: 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.

Querry:

#### Step 5.1: Create Table

```
CREATE TABLE StudentGrades (
    student_id SERIAL PRIMARY KEY,
    student_name VARCHAR(50),
    marks INT
);
```

#### Insert Sample Data

```
INSERT INTO StudentGrades (student_name, marks) VALUES
('Aarav', 95),
('Neha', 82),
('Rohit', 68),
('Priya', 91),
('Karan', 56),
('Simran', 45),
('Aman', 77),
('Riya', 88),
('Vikas', 35);
```

Output:

	student_id [PK] integer ↴	student_name character varying (50) ↴	marks integer ↴
1	1	Aarav	95
2	2	Neha	82
3	3	Rohit	68
4	4	Priya	91
5	5	Karan	56
6	6	Simran	45
7	7	Aman	77
8	8	Riya	88
9	9	Vikas	35

### Step 5.2: 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	Aarav	95	A+ Gra...
2	Neha	82	A Grade
3	Rohit	68	C Grade
4	Priya	91	A+ Gra...
5	Karan	56	D Grade
6	Simran	45	D Grade
7	Aman	77	B Grade
8	Riya	88	A Grade
9	Vikas	35	Fail

### Step 6: Using CASE for Custom Sorting

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

Querry:

SELECT

entity\_name,

violation\_count,

CASE

WHEN violation\_count > 15 THEN 'Critical Violation'

WHEN violation\_count BETWEEN 6 AND 15 THEN 'Moderate Violation'

WHEN violation\_count BETWEEN 1 AND 5 THEN 'Minor Violation'

ELSE 'No Violation'

END AS Violation\_Severity

FROM Violations

ORDER BY

CASE

WHEN violation\_count > 15 THEN 1

WHEN violation\_count BETWEEN 6 AND 15 THEN 2

WHEN violation\_count BETWEEN 1 AND 5 THEN 3

ELSE 4

END;

Output:

	entity_name character varying (100)	violation_count integer	violation_severity text
1	IT_Department	20	Critical Violation
2	Finance_Department	12	Moderate Violation
3	Security_Team	15	Moderate Violation
4	HR_Department	5	Minor Violation
5	Sales_Department	0	No Violation
6	Admin_Department	0	No Violation

## 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