UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Worksheet 3

Student Name:Suyash **UID:25MCI10054**
Branch:MCA (AI&ML) **Section/Group:MAM-1 A**
Semester:2nd **Date of Performance:27/01/2026**
Subject Name:-DBMS Lab **Subject Code:**

1. Aim of the Session

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

2. Software Requirements

- PostgreSQL (Database Server)

3. Objective of the Session

- 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

4. Practical / Experiment Steps

- Create a sample table representing Violation
- Insert realistic records into the table
- Classifying Data Using CASE Expression
- Applying CASE Logic in Data Updates
- Implementing IF–ELSE Logic Using PL/pgSQL
- Real-World Classification Scenario (Grading System)
- Using CASE for Custom Sorting

5. Procedure of the Practical

(i) Start the system and log in to the computer.

(ii) Open PostgreSQL software.

iii) Create and select the database.

(iv) Create table using DDL command.

```
CREATE TABLE Violations (
   id INT PRIMARY KEY,
   Entity_Name VARCHAR(100),
   Violation_count INT
);
```

(v) Insert records into the table.

```
INSERT INTO Violations VALUES
(1, 'User_data', 13),
(2, 'Payment_data', 5),
(3, 'Audit_data', 22),
(4, 'Order_data', 0),
(5, 'Inventory_data', 0),
(6, 'Supplyer_data', 13);
```

(vi) Display all records.

```
SELECT*FROM Violations;
```

	id [PK] integer	entity_name character varying (100)	violation_count integer
1	1	User_data	13
2	2	Payment_data	5
3	3	Audit_data	22
4	4	Order_data	0
5	5	Inventory_data	0
6	6	Supplyer_data	13

(vii) Classifying Data Using CASE Expression

```
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;
```

	entity_name character varying (100)	violation_count integer	violation_status text
1	User_data	13	Moderate Violation
2	Payment_data	5	Minor Violation
3	Audit_data	22	Critical Violation
4	Order_data	0	No Violation
5	Inventory_data	0	No Violation
6	Supplyer_data	13	Moderate Violation

(viii) Applying CASE Logic in Data Updates

```
ALTER TABLE Violations
ADD COLUMN approval_status VARCHAR(30);
```

	id [PK] integer	entity_name character varying (100)	violation_count integer	approval_status character varying (30)
1	1	User_data	13	[null]
2	2	Payment_data	5	[null]
3	3	Audit_data	22	[null]
4	4	Order_data	0	[null]
5	5	Inventory_data	0	[null]
6	6	Supplyer_data	13	[null]

```
UPDATE Violations
SET approval_status =
  CASE
    WHEN violation_count = 0 THEN 'Approved'
    WHEN violation_count BETWEEN 1 AND 15 THEN 'Needs Review'
    ELSE 'Rejected'
  END;
```

	id [PK] integer	entity_name character varying (100)	violation_count integer	approval_status character varying (30)
1	1	User_data	13	Needs Review
2	2	Payment_data	5	Needs Review
3	3	Audit_data	22	Rejected
4	4	Order_data	0	Approved
5	5	Inventory_data	0	Approved
6	6	Supplyer_data	13	Needs Review

(ix) Implementing IF–ELSE Logic Using PL/pgSQL

```
DO $$
DECLARE
  v_violation_count INT := 12;   -- Change value to test output
```

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

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

Query returned successfully in 283 msec.
```

(x) Real-World Classification Scenario (Grading System)

- **Table creation**

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

- **Data Insertion**

```
INSERT INTO Grades (student_name, marks) VALUES
('Rohit', 95),
('Neha', 82),
```

('Sumit', 68),

('Priya', 91),

('Mehak', 56),

('Hemant', 45),

('Nikita', 77),

('Prince', 88),

('Atul', 35);

- **Grading**

```
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 Grades;
```

	student_name character varying (50)	marks integer	grade text
1	Rohit	95	A+ Grade
2	Neha	82	A Grade
3	Sumit	68	C Grade
4	Priya	91	A+ Grade
5	Mehak	56	D Grade
6	Hemant	45	D Grade
7	Nikita	77	B Grade
8	Prince	88	A Grade
9	Atul	35	Fail

(xi) Using CASE for Custom Sorting

```
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;
```

	entity_name character varying (100)	violation_count integer	violation_severity text
1	Audit_data	22	Critical Violation
2	User_data	13	Moderate Violation
3	Supplyer_data	13	Moderate Violation
4	Payment_data	5	Minor Violation
5	Order_data	0	No Violation
6	Inventory_data	0	No Violation

6. Learning Outcomes

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