



### Worksheet No- 3

Name - Arnab Lala

UID – 25MCI10232

Branch – MCA

Section – MAM-1(A)

Semester – 2<sup>nd</sup>

Date of performance – 27<sup>th</sup> Jan,2026

Subject - Technical Training

Subject Code – 25CAP-652\_25MAM\_KAR-1\_A

### 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.

### Software Requirements

- PostgreSQL

### 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 system

### Theory

In real-world database systems, data often needs to be validated, categorized, or transformed based on business rules. Conditional logic allows the database to make decisions dynamically instead of relying solely on application-layer logic.

PostgreSQL supports conditional logic mainly through:

- CASE Expressions (used inside SELECT, UPDATE, INSERT)
- IF–ELSE constructs (used inside PL/pgSQL blocks such as functions and procedures)

CASE Expression

- Evaluates conditions sequentially
- Returns a value based on the first true condition
- Can be used in SELECT, UPDATE, ORDER BY, and WHERE clauses

### Types of CASE

- Simple CASE → compares expressions
- Searched CASE → evaluates boolean conditions

Conditional logic is heavily used in:

- Data classification (grades, salary slabs)
- Violation detection
- Status mapping
- Business rule enforcement

Companies like Amazon, SAP, Oracle, and Adobe frequently test CASE-based logic in SQL interviews.

### Experiment / Practical Steps

#### Prerequisite Understanding

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.

#### Step 1: Classifying Data Using CASE Expression

##### Task for Students:

- Retrieve schema names and their violation counts.
- Use conditional logic to classify each schema into categories such as:
  - No Violation
  - Minor Violation
  - Critical Violation

##### Learning Focus:

- Using **searched CASE**
- Sequential condition checking
- Real-world compliance reporting logic

#### Step 2: Applying CASE Logic in Data Updates

##### Task for Students:

- Add a new column to store approval status.
- Update this column based on violation count using conditional rules such as:

- Approved
- Needs Review
- Rejected

**Learning Focus:**

- Automating decisions inside the database
- Reducing application-side logic
- Using CASE inside UPDATE statements

**Step 3: Implementing IF–ELSE Logic Using PL/pgSQL**

**Task for Students:**

- 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.

**Learning Focus:**

- Understanding procedural SQL
- ELSE-IF ladder execution
- Backend validation logic in stored procedures

**Step 4: Real-World Classification Scenario (Grading System)**

**Task for Students:**

- Create a table to store student names and marks.
- Classify students into grades based on their marks using conditional logic.

**Learning Focus:**

- Common interview use case
- Data categorization
- Rule-based evaluation

**Step 5: Using CASE for Custom Sorting**

**Task for Students:**

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

**Learning Focus:**

- Advanced CASE usage
- Custom ordering logic and Dashboard and reporting scenarios

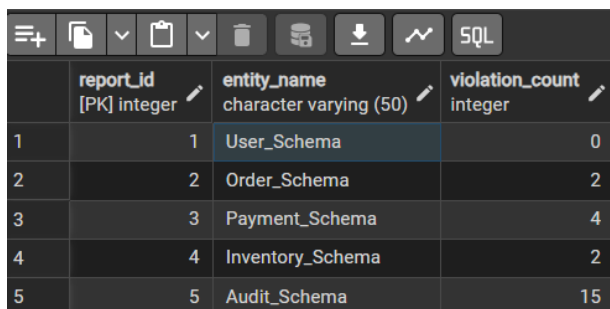
## Practical / Experiment Steps

```
CREATE TABLE SCHEMA_ANALYSIS(  
  report_id INT PRIMARY KEY,  
  entity_name VARCHAR(50) NOT NULL,  
  violation_count INT NOT NULL  
);
```

```
INSERT INTO SCHEMA_ANALYSIS VALUES
```

```
(1, 'User_Schema', 0),  
(2, 'Order_Schema', 2),  
(3, 'Payment_Schema', 4),  
(4, 'Inventory_Schema', 2),  
(5, 'Audit_Schema', 15);
```

```
SELECT * FROM SCHEMA_ANALYSIS;
```



	report_id [PK] integer	entity_name character varying (50)	violation_count integer
1	1	User_Schema	0
2	2	Order_Schema	2
3	3	Payment_Schema	4
4	4	Inventory_Schema	2
5	5	Audit_Schema	15

## -- EXAMPLE 1 : CLASSIFYING DATA USING CASE EXPRESSION

```
SELECT *,
```

```
CASE
```

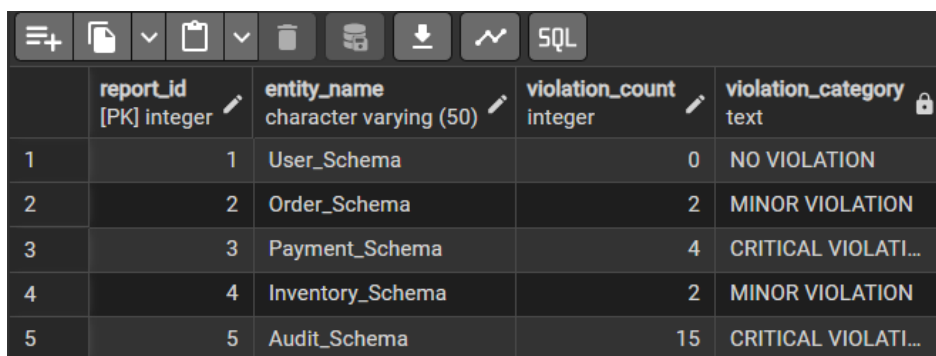
```
  WHEN violation_count = 0 THEN 'NO VIOLATION'
```

```
  WHEN violation_count BETWEEN 1 AND 2 THEN 'MINOR VIOLATION'
```

```
  ELSE 'CRITICAL VIOLATION'
```

```
END AS VIOLATION_CATEGORY
```

```
FROM SCHEMA_ANALYSIS;
```




	report_id [PK] integer	entity_name character varying (50)	violation_count integer	violation_category text
1	1	User_Schema	0	NO VIOLATION
2	2	Order_Schema	2	MINOR VIOLATION
3	3	Payment_Schema	4	CRITICAL VIOLATI...
4	4	Inventory_Schema	2	MINOR VIOLATION
5	5	Audit_Schema	15	CRITICAL VIOLATI...

## -- Example 2: Applying CASE Logic in Data Updates

```
ALTER TABLE SCHEMA_ANALYSIS
```

```
ADD COLUMN approval_status VARCHAR(20);
```



	report_id [PK] integer	entity_name character varying (50)	violation_count integer	approval_status character varying (20)
1	1	User_Schema	0	[null]
2	2	Order_Schema	2	[null]
3	3	Payment_Schema	4	[null]
4	4	Inventory_Schema	2	[null]
5	5	Audit_Schema	15	[null]

```
SELECT * FROM SCHEMA_ANALYSIS;
```

```
UPDATE SCHEMA_ANALYSIS
```

```
SET approval_status =
```


```
CASE
```

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

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

```
    ELSE 'Rejected'
```

```
END;
```



	report_id [PK] integer	entity_name character varying (50)	violation_count integer	approval_status character varying (20)
1	1	User_Schema	0	Approved
2	2	Order_Schema	2	Review
3	3	Payment_Schema	4	Rejected
4	4	Inventory_Schema	2	Review
5	5	Audit_Schema	15	Rejected

## -- Example 3: Implementing IF-ELSE Logic Using PL/pgSQL

```
DO $$
```

```
DECLARE
```

```
v_violation_count INT := 0; -- change value to test
```

```
BEGIN
```

```
    IF v_violation_count = 0 THEN
```

```
        RAISE NOTICE 'Status: Approved (No Violations)';
```

```
    ELSIF v_violation_count BETWEEN 1 AND 2 THEN
```

```
RAISE NOTICE 'Status: Review (Minor Violations)';

ELSE

    RAISE NOTICE 'Status: Rejected (Critical Violations)';

END IF;
END $$;
```

```
NOTICE:  Status: Approved
DO

Query returned successfully in 124 msec.
```

	report_id [PK] integer	entity_name character varying (50)	violation_count integer	approval_status character varying (20)
1	1	User_Schema	0	Approved
2	2	Order_Schema	2	Review
3	3	Payment_Schema	4	Rejected
4	4	Inventory_Schema	2	Review
5	5	Audit_Schema	15	Rejected

## 5. 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