



EXPERIMENT 4

Student Name: Shreyansh Gupta

Branch: AIT-CSE (AIML)

Semester: 4

Subject Name: Database Management System

UID: 24BAI70658

Section/Group: 24AIT_KRG-1/G2

Subject Code: 24CSH-298

EXPERIMENT – 04

Implementation of Conditional Logic using IF–ELSE and CASE Statements in PostgreSQL

Aim

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

Tools Used

- 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 systems

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.

EXPERIMENT 4

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

Experiment / Practical Steps

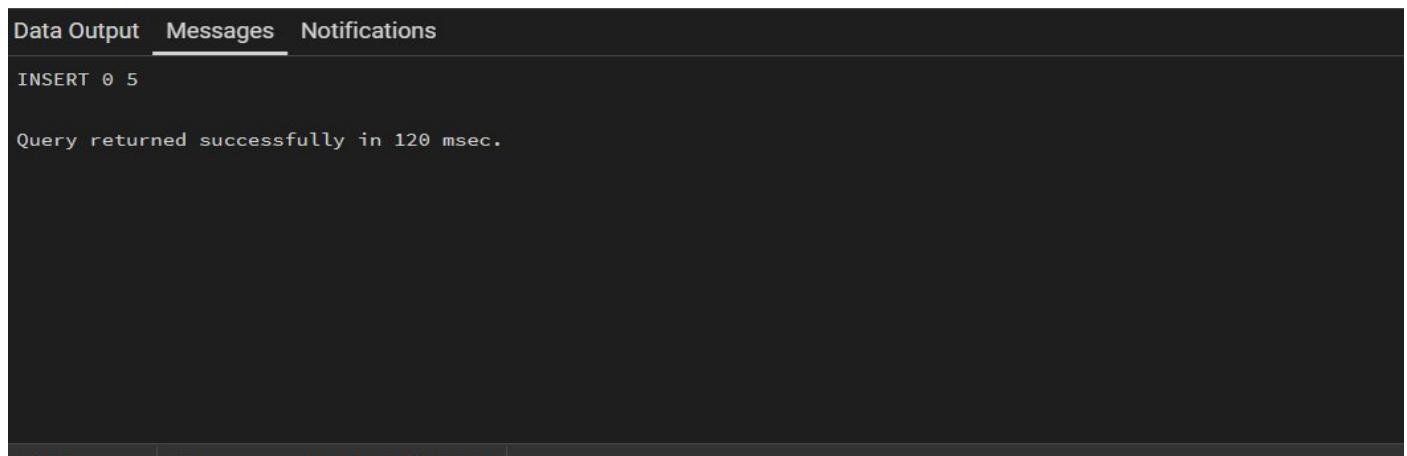
Prerequisite Setup

```
CREATE TABLE schema_violations (
    id SERIAL PRIMARY KEY,
    schema_name VARCHAR(50),
    violation_count INT
);
```

```
INSERT INTO schema_violations (schema_name, violation_count) VALUES
('Finance', 0),
```

EXPERIMENT 4

```
('HR', 2),  
(('Sales', 5),  
(('Security', 9),  
(('Admin', 1);
```



The screenshot shows a dark-themed application window with a tab bar at the top labeled "Data Output", "Messages", and "Notifications". The "Messages" tab is selected, indicated by a blue underline. Below the tabs, the text "INSERT 0 5" is displayed, followed by the message "Query returned successfully in 120 msec."

Step 1: Classifying Data Using CASE Expression

```
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;
```

EXPERIMENT 4

Data Output Messages Notifications

≡+ File Save Open Print Export SQL Showing rows: 1 to 5 Edit

	schema_name character varying (50)	violation_count integer	violation_status text
1	Finance	0	No Violation
2	HR	2	Minor Violation
3	Sales	5	Moderate Violati...
4	Security	9	Critical Violation
5	Admin	1	Minor Violation

Total rows: 5 | Query complete 00:00:00.386

Step 2: Applying CASE Logic in Data Updates

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

```
UPDATE schema_violations
```

```
SET approval_status =
```

```
CASE
```

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

```
WHEN violation_count BETWEEN 1 AND 5 THEN 'Needs Review'
```

```
ELSE 'Rejected'
```

```
END;
```

Data Output Messages Notifications

UPDATE 5

Query returned successfully in 135 msec.

Total rows: 5 | Query complete 00:00:00.135

EXPERIMENT 4

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

```
DO $$  
DECLARE  
    v_count INT := 4;  
BEGIN  
    IF v_count = 0 THEN  
        RAISE NOTICE 'System is clean.';  
    ELSIF v_count <= 5 THEN  
        RAISE NOTICE 'System has minor issues.';  
    ELSE  
        RAISE NOTICE 'System has critical violations!';  
    END IF;  
END $$;
```

```
NOTICE: System has minor issues.  
DO  
  
Query returned successfully in 138 msec.
```

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

```
CREATE TABLE students (  
    name VARCHAR(50),  
    marks INT  
);
```

EXPERIMENT 4

INSERT INTO students VALUES

```
('Aryan', 92),
('Riya', 75),
('Karan', 61),
('Megha', 48);
```

SELECT name, marks,

CASE

WHEN marks >= 90 THEN 'A Grade'

WHEN marks >= 70 THEN 'B Grade'

WHEN marks >= 50 THEN 'C Grade'

ELSE 'Fail'

END AS grade

FROM students;

Data Output Messages Notifications

Showing rows: 1 to 4

	name character varying (50)	marks integer	grade text
1	Aryan	92	A Grade
2	Riya	75	B Grade
3	Karan	61	C Grade
4	Megha	48	Fail

Step 5: Using CASE for Custom Sorting

SELECT schema_name, violation_count

EXPERIMENT 4

```
FROM schemaViolations
ORDER BY
CASE
    WHEN violation_count = 0 THEN 1
    WHEN violation_count BETWEEN 1 AND 3 THEN 2
    WHEN violation_count BETWEEN 4 AND 7 THEN 3
    ELSE 4
END;
```

Data Output Messages Notifications

Showing rows: 1 to 5

	id [PK] integer	schema_name character varying (50)	violation_count integer	approval_status character varying (20)
1	1	Finance	0	Approved
2	2	HR	2	Needs Review
3	3	Sales	5	Needs Review
4	4	Security	9	Rejected
5	5	Admin	1	Needs Review

Course 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

Result

Conditional logic using CASE and IF-ELSE was successfully implemented for classification, approval automation, grading systems, and custom sorting in PostgreSQL.