



EXPERIMENT – 03

Name: Pratham

UID: 25MCI10178

Branch: MCA- AI/ML

Section: 25MAM_1_A

Semester: 2nd

Date: 27/01/26

Subject Name: Technical Training

Subject Code: 25CAP-652

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
-

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

```
create table schema_Analysis
(
id int primary key,
schema_name varchar(50),
violation_score int
)
```



```
CREATE TABLE
```

```
| Query returned successfully in 217 msec.
```

```
insert into schema_Analysis values(1,'dept_table',2),  
(2,'log_table',4),  
(3,'emp_table',0),  
(4,'project_table',1),  
(5,'user_table',1),  
(6,'student_table',3)
```

```
select* from schema_Analysis
```

	id [PK] integer	schema_name character varying (50)	violation_score integer
1	1	dept_table	2
2	2	log_table	4
3	3	emp_table	0
4	4	project_table	1
5	5	user_table	1
6	6	student_table	3

```
ALTER TABLE schema_analysis  
ADD COLUMN violation_type VARCHAR(30);
```

```
UPDATE schema_Analysis  
SET violation_type =  
CASE  
WHEN violation_score = 0 THEN 'NO VIOLATION'  
WHEN violation_score BETWEEN 1 AND 2 THEN 'Minor Violation'  
WHEN violation_score BETWEEN 3 AND 4 THEN 'Moderate Violation'  
ELSE 'Critical Violation'  
END;
```



	id [PK] integer	schema_name character varying (50)	violation_score integer	violation_type character varying (30)
1	1	dept_table	2	Minor Violation
2	2	log_table	4	Moderate Violation
3	3	emp_table	0	NO VIOLATION
4	4	project_table	1	Minor Violation
5	5	user_table	1	Minor Violation
6	6	student_table	3	Moderate Violation

Step 2: Applying CASE Logic in Data Updates

```
ALTER TABLE schema_analysis  
ADD COLUMN status VARCHAR(30);
```

```
UPDATE schema_Analysis
```

```
SET status =
```

```
CASE
```

```
WHEN violation_score = 0 THEN 'Approved'
```

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

```
ELSE 'Rejected'
```

```
END;
```



	id [PK] integer	schema_name character varying (50)	violation_score integer	violation_type character varying (30)	status character varying (30)
1	1	dept_table	2	Minor Violation	Needs Review
2	2	log_table	4	Moderate Violation	Rejected
3	3	emp_table	0	NO VIOLATION	Approved
4	4	project_table	1	Minor Violation	Needs Review
5	5	user_table	1	Minor Violation	Needs Review
6	6	student_table	3	Moderate Violation	Rejected

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

```
DO $$  
DECLARE  
    violation_count INT := 3;  
BEGIN  
    IF violation_count = 0 THEN  
        RAISE NOTICE 'NO VIOLATION';  
    ELSIF violation_count BETWEEN 1 AND 2 THEN  
        RAISE NOTICE 'Minor Violation';  
    ELSIF violation_count BETWEEN 3 AND 4 THEN  
        RAISE NOTICE 'Moderate Violation';  
    ELSE  
        RAISE NOTICE 'Critical Violation';  
    END IF;  
END $$;
```



NOTICE: Moderate Violation

DO

Query returned successfully in 154 msec.

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

```
create table student (
    id int primary key ,
    name varchar(50),
    marks int
)
```

```
INSERT INTO student (id, name, marks) VALUES
(1, 'Amit', 85),
(2, 'Neha', 72),
(3, 'Rohan', 90),
(4, 'Simran', 65),
(5, 'Ankit', 40);
```

```
select * from student;
```

```
ALTER TABLE student
ADD COLUMN grade VARCHAR(2);
```

```
DO $$  
DECLARE  
    r RECORD;  
BEGIN  
    FOR r IN SELECT id, marks FROM student  
    LOOP  
        IF r.marks >= 90 THEN  
            UPDATE student SET grade = 'A' WHERE id = r.id;  
        ELSIF r.marks >= 75 THEN  
            UPDATE student SET grade = 'B' WHERE id = r.id;
```



```
ELSIF r.marks >= 50 THEN
    UPDATE student SET grade = 'C' WHERE id = r.id;
ELSE
    UPDATE student SET grade = 'F' WHERE id = r.id;
END IF;
END LOOP;
END $$;
```

	id [PK] integer	name character varying (50)	marks integer	grade character varying (2)
1	1	Amit	85	B
2	2	Neha	72	C
3	3	Rohan	90	A
4	4	Simran	65	C
5	5	Ankit	40	F

Step 5: Using CASE for Custom Sorting

```
SELECT
    schema_name,
    violation_score,
    violation_type
FROM schema_Analysis
ORDER BY
    CASE
        WHEN violation_type = 'Critical Violation' THEN 1
        WHEN violation_type = 'Moderate Violation' THEN 2
        WHEN violation_type = 'Minor Violation' THEN 3
    END
```



ELSE 4

END, schema_name;

	schema_name character varying (50)	violation_score integer	violation_type character varying (30)
1	log_table	4	Moderate Violation
2	student_table	3	Moderate Violation
3	dept_table	2	Minor Violation
4	project_table	1	Minor Violation
5	user_table	1	Minor Violation
6	emp_table	0	NO VIOLATION

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