Experiment No. 3

Student Name: Shubham Agarwal **UID: 25MCI10091**
Branch: MCA (AI & ML) **Section/Group: 25MAM-1/A**
Semester: 2nd **Date of Performance:27/01/2026**
Subject Name: Technical Training Lab **Subject Code: 25CAH-653**

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.

Software Requirements:

- PostgreSQL Database Server
- pgAdmin 4
- Windows Operating System

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

Practical Experiment Steps:

schema_analysis Table Creation :

```
CREATE TABLE schema_analysis (
    schema_id SERIAL PRIMARY KEY,
    schema_name VARCHAR(100) NOT NULL,
    violation_score INT NOT NULL
);
```

```
-- Insert data with varying violation scores
INSERT INTO schema_analysis (schema_name, violation_score) VALUES
('user_profiles', 0),
('product_catalog', 1),
('order_processing', 5),
('payment_gateway', 2),
('inventory_management', 3),
('new_users',4),
('business_profile',2);
```

	schema_id [PK] integer	schema_name character varying (100)	violation_score integer
1	1	user_profiles	0
2	2	product_catalog	1
3	3	order_processing	5
4	4	payment_gateway	2
5	5	inventory_management	3
6	6	new_users	4
7	7	business_profile	2

Step 1: Classifying Data Using CASE Expression

Task: Retrieve schema names and classify violation levels.

```
SELECT
   schema_id,
   schema_name,
   violation_score,
   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 AS violation_category
FROM schema_analysis
ORDER BY violation_score;
```

	schema_id [PK] integer	schema_name character varying (100)	violation_score integer	violation_category text
1	1	user_profiles	0	No Violation
2	2	product_catalog	1	Minor Violation
3	4	payment_gateway	2	Minor Violation
4	7	business_profile	2	Minor Violation
5	5	inventory_management	3	Moderate Violation
6	6	new_users	4	Moderate Violation
7	3	order_processing	5	Critical Violation

Step 2: Applying CASE Logic in Data Updates

Task: Add and update approval_status column.

```
ALTER TABLE schema_analysis
ADD COLUMN voilation_category varchar(50);

SELECT * FROM schema_analysis;
```

	schema_id [PK] integer	schema_name character varying (100)	violation_score integer	voilation_category character varying (50)
1	1	user_profiles	0	[null]
2	2	product_catalog	1	[null]
3	3	order_processing	5	[null]
4	4	payment_gateway	2	[null]
5	5	inventory_management	3	[null]
6	6	new_users	4	[null]
7	7	business_profile	2	[null]

```
UPDATE schema_analysis SET voilation_category = (
 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
)
```

SELECT * FROM schema_analysis;

	schema_id [PK] integer	schema_name character varying (100)	violation_score integer	voilation_category character varying (50)
1	1	user_profiles	0	No Violation
2	2	product_catalog	1	Minor Violation
3	3	order_processing	5	Critical Violation
4	4	payment_gateway	2	Minor Violation
5	5	inventory_management	3	Moderate Violation
6	6	new_users	4	Moderate Violation
7	7	business_profile	2	Minor Violation

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

Task: Direct DO block for violation checking.

```
DO $$
DECLARE
   rec RECORD;
   violation_count INT;
   schema_status VARCHAR(35);
BEGIN
   FOR rec IN SELECT schema_id, schema_name, violation_score FROM schema_analysis LOOP
      violation_count := rec.violation_score;

      IF violation_count = 0 THEN
         schema_status := 'Approved';
      ELSIF violation_count BETWEEN 1 AND 2 THEN
         schema_status := 'Need Review';
      ELSIF violation_count BETWEEN 3 AND 4 THEN
         schema_status := 'Review (Urgent)';
      ELSE
         schema_status := 'Rejected';
      END IF;

      RAISE NOTICE 'Schema: % , Status: %',
               rec.schema_name, schema_status;
   END LOOP;
END $$;
```

```
NOTICE:  Schema: user_profiles , Status: Approved
NOTICE:  Schema: product_catalog , Status: Need Review
NOTICE:  Schema: order_processing , Status: Rejected
NOTICE:  Schema: payment_gateway , Status: Need Review
NOTICE:  Schema: inventory_management , Status: Review (Urgent)
NOTICE:  Schema: new_users , Status: Review (Urgent)
NOTICE:  Schema: business_profile , Status: Need Review
DO

Query returned successfully in 41 msec.
```

Step 4: Real-World Classification - Grading System

student_grades Table Creation :

```
CREATE TABLE student_grades (
    student_id SERIAL PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    total_marks INT NOT NULL CHECK (total_marks >= 0 AND total_marks <= 500),
    subject VARCHAR(50)
);
--record insertion with total_marks range(0-500) -> 5 Subjects
INSERT INTO student_grades (student_name, total_marks, subject) VALUES
('Rahul Sharma', 465, 'Science Stream'),
('Priya Singh', 412, 'Science Stream'),
('Amit Kumar', 345, 'Science Stream'),
('Neha Gupta', 285, 'Science Stream'),
('Vikash Yadav', 165, 'Science Stream'),
('Anjali Verma', 448, 'Commerce'),
('Rohit Patel', 378, 'Commerce'),
('Sneha Roy', 298, 'Commerce');
```

	student_id [PK] integer	student_name character varying (100)	total_marks integer	subject character varying (50)
1	1	Rahul Sharma	465	Science Stream
2	2	Priya Singh	412	Science Stream
3	3	Amit Kumar	345	Science Stream
4	4	Neha Gupta	285	Science Stream
5	5	Vikash Yadav	165	Science Stream
6	6	Anjali Verma	448	Commerce
7	7	Rohit Patel	378	Commerce
8	8	Sneha Roy	298	Commerce

Grading Students Based Upon the Total_Marks using CASE statements :

```
SELECT
   student_name,
   total_marks,
   CASE
      WHEN total_marks >= 400 THEN 'A+'
      WHEN total_marks >= 350 THEN 'A'
      WHEN total_marks >= 300 THEN 'B'
      WHEN total_marks >= 250 THEN 'C'
      WHEN total_marks >= 200 THEN 'D'
      ELSE 'F'
   END AS grades
FROM student_grades
ORDER BY total_marks DESC;
```

	student_name character varying (100)	total_marks integer	grades text
1	Rahul Sharma	465	A+
2	Anjali Verma	448	A+
3	Priya Singh	412	A+
4	Rohit Patel	378	A
5	Amit Kumar	345	B
6	Sneha Roy	298	C
7	Neha Gupta	285	C
8	Vikash Yadav	165	F

Step 5: CASE for Custom Sorting (Priority)

```
SELECT * FROM schema_analysis;
SELECT
   schema_name,
   violation_score,
         voilation_category,
   CASE
      WHEN violation_score > 4 THEN 1
      WHEN violation_score BETWEEN 3 and 4 THEN 2
      WHEN violation_score BETWEEN 1 and 2 THEN 3
         ELSE 4
   END AS priority
FROM schema_analysis ORDER BY priority, violation_score DESC;
```

	schema_name character varying (100)	violation_score integer	voilation_category character varying (50)	priority integer
1	order_processing	5	Critical Violation	1
2	new_users	4	Moderate Violation	2
3	inventory_management	3	Moderate Violation	2
4	payment_gateway	2	Minor Violation	3
5	business_profile	2	Minor Violation	3
6	product_catalog	1	Minor Violation	3
7	user_profiles	0	No Violation	4

I/O Analysis

Input :

- schema_analysis table with violation scores(0-5)
- Student grades data (0-500 marks)

Output :

- Violation classification categories
- Approval status updates
- Procedural validation messages
- Student grading system
- Priority-based schema sorting

Learning Outcomes

- learned CASE expressions for data classification
- Implemented procedural IF-ELSE logic in PL/pgSQL
- Automated business rule enforcement through UPDATE CASE
- Developed custom priority sorting for dashboard reporting
- Prepared for SQL interview questions on conditional logic