EXPERIMENT - 6

Name: Mohit Sengar	**UID**: 23BCS10331
Branch: CSE	**Section**: KRG1-B
Semester: 5	**Date of Performance**: 23/9/2025
Subject: ADBMS	**Subject Code**: 23CSP-333

Question 1. HR-Analytics: Employee count based on dynamic gender passing (Medium)

TechSphere Solutions, a growing IT services company with offices across India, wants to track and monitor gender diversity within its workforce. The HR department frequently needs to know the total number of employees by gender (Male or Female) .
To solve this problem, the company needs an automated database-driven solution that can instantly return the count of employees by gender through a stored procedure that:

1. Create a PostgreSQL stored procedure that:
2. Takes a gender (e.g., 'Male' or 'Female') as input.
3. Calculates the total count of employees for that gender.
4. Returns the result as an output parameter.
5. Displays the result clearly for HR reporting purposes.

Solution:

```
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(100),
    gender VARCHAR(10)
);


INSERT INTO employees (emp_name, gender) VALUES
('Aarav Sharma', 'Male'),
```

('Neha Verma', 'Female'),

('Rohan Gupta', 'Male'),

('Priya Singh', 'Female'),

('Kunal Mehta', 'Male');


```
CREATE OR REPLACE PROCEDURE get_employee_count_by_gender(IN p_gender VARCHAR,
OUT p_count INT)
LANGUAGE plpgsql
AS $$
BEGIN
   SELECT COUNT(*) INTO p_count
   FROM employees
   WHERE gender = p_gender;
   RAISE NOTICE 'Total number of % employees: %', p_gender, p_count;
END;
$$;
CALL get_employee_count_by_gender('Male', NULL);
DO $$
DECLARE
   result INT;
BEGIN
  CALL get_employee_count_by_gender('Female', result);
  RAISE NOTICE 'Captured result: % Female employees', result;
END $$;
```

OUTPUT:

```
Data Output    Messages    Notifications

NOTICE:   Total number of Male employees: 3
NOTICE:   Total number of Female employees: 2
NOTICE:   Captured result: 2 Female employees
DO


Query returned successfully in 70 msec.
```

Question 2. SmartStore Automated Purchase System

SmartShop is a modern retail company that sells electronic gadgets like smartphones, laptops and tablets. The company wants to automate its ordering and inventory management process.
Whenever a customer places an order, the system must:
1. Verify stock availability for the requested product and quantity.
2. If sufficient stock is available:
  - Log the order in the sales table with the ordered quantity and total price.
  - Update the inventory in the products table by reducing quantity_remaining and increasing quantity_sold.
  - Display a real-time confirmation message: "Product sold successfully!"
3. If there is insufficient stock, the system must:
  - Reject the transaction and display: Insufficient Quantity Available!"

Solution:

CREATE TABLE products (

   product_id SERIAL PRIMARY KEY,

   product_name VARCHAR(100),

   price NUMERIC(10,2),

```
    quantity_remaining INT,
    quantity_sold INT DEFAULT 0
);

INSERT INTO products (product_name, price, quantity_remaining) VALUES
('Smartphone', 20000.00, 10),
('Laptop', 55000.00, 5),
('Tablet', 30000.00, 8);

CREATE TABLE sales (
    sale_id SERIAL PRIMARY KEY,
    product_id INT REFERENCES products(product_id),
    quantity_ordered INT,
    total_price NUMERIC(10,2),
    sale_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE PROCEDURE place_order(IN p_product_id INT, IN p_quantity INT)
LANGUAGE plpgsql
AS $$
DECLARE
    v_price NUMERIC(10,2);
    v_remaining INT;
    v_total NUMERIC(10,2);
BEGIN
 SELECT price, quantity_remaining INTO v_price, v_remaining
    FROM products
    WHERE product_id = p_product_id;
```

```
    IF NOT FOUND THEN
        RAISE NOTICE 'Invalid product ID!';
        RETURN;
END IF;
    IF v_remaining >= p_quantity THEN
        -- Calculate total price
        v_total := v_price * p_quantity;


        INSERT INTO sales(product_id, quantity_ordered, total_price)
        VALUES (p_product_id, p_quantity, v_total);


    UPDATE products
        SET quantity_remaining = quantity_remaining - p_quantity,
            quantity_sold = quantity_sold + p_quantity
        WHERE product_id = p_product_id;


        RAISE NOTICE 'Product sold successfully!';
    ELSE
        RAISE NOTICE 'Insufficient Quantity Available!';
    END IF;
END;
$$;
CALL place_order(1, 2);
SELECT * FROM products;
SELECT * FROM sales;
```

OUTPUT:

Data Output   Messages   Notifications

```
NOTICE:  Product sold successfully!
CALL

Query returned successfully in 62 msec.
```

Data Output   Messages   Notifications

	name character varying (20)	quantity_remaining integer	quantity_sold integer
1	Laptop	5	2
2	IPhone	3	1
3	Oven	4	2

Data Output   Messages   Notifications

	sale_id [PK] integer	customer_name character varying (100)	product_name character varying (100)	quantity integer	price numeric	sale_date date
1	1	Alice	Laptop	2	800	2025-01-01
2	2	Bob	Mouse	5	20	2025-01-02
3	3	Charlie	Keyboard	3	50	2025-01-03
4	4	Alice	Laptop	1	800	2025-01-04

LEARNING OUTCOMES:

1. Understand how to use stored procedures to perform multiple operations such as checking stock, inserting orders, and updating inventory in one unit.

2. Learn how to track and manage inventory by updating remaining quantity and quantity sold after each transaction. 3. Automate the process of order placement, including validating stock availability and calculating total price dynamically.

4. Develop skills in handling errors and displaying meaningful messages for both successful and failed transactions.

5. Gain hands-on practice in creating tables, writing INSERT, UPDATE, and SELECT queries, and using conditional logic in PL/pgSQL.