Experiment 6

Student Name: Gurshaan Singh	**UID: 23BCS14195**
Branch: CSE	**Section/Group: KRG – 1B**
Semester: 5th	**Date of Performance: 24/09/25**
Subject Name: ADBMS	**Subject Code: 23CSP-333**

1. **Aim**: Stored Procedures in Database
2. **Requirements(Hardware/Software):** MySQL, PostgreSQL, Oracle, or SQL Server
3. **DBMS script and output:**

Problem A:

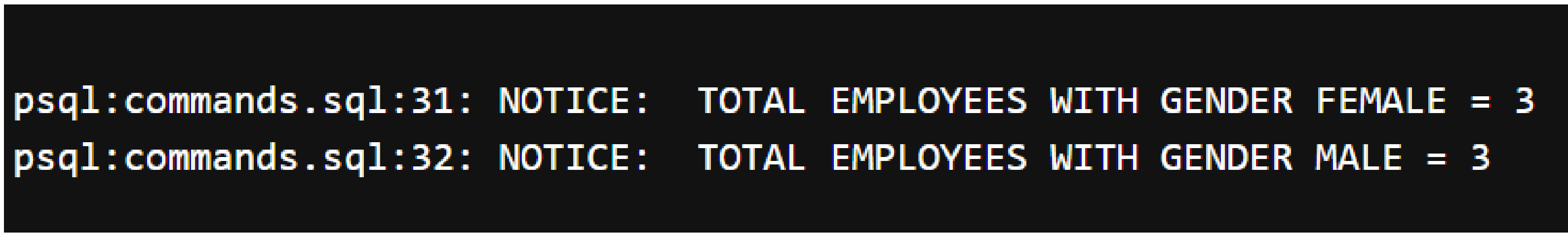
Problem Title: Employee count based on dynamic gender passing

Procedure (Step-by-Step):

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.

```
CREATE TABLE EMPLOYEES (

    EMP_ID SERIAL PRIMARY KEY,

    EMP_NAME TEXT,

    GENDER TEXT

);


INSERT INTO EMPLOYEES (EMP_NAME, GENDER) VALUES

('RAHUL', 'MALE'),

('PRIYA', 'FEMALE'),

('AMIT', 'MALE'),

('NEHA', 'FEMALE'),

('ARJUN', 'MALE'),

('SNEHA', 'FEMALE');


CREATE OR REPLACE PROCEDURE GET_EMPLOYEE_COUNT_BY_GENDER(

    IN IN_GENDER TEXT,

    OUT OUT_COUNT INT

)

LANGUAGE plpgsql

AS $$

BEGIN

    SELECT COUNT(*)

    INTO OUT_COUNT

    FROM EMPLOYEES

    WHERE GENDER = IN_GENDER;
    RAISE NOTICE 'TOTAL EMPLOYEES WITH GENDER % = %', IN_GENDER, OUT_COUNT;

END;

$$;



CALL GET_EMPLOYEE_COUNT_BY_GENDER('FEMALE', NULL);

CALL GET_EMPLOYEE_COUNT_BY_GENDER('MALE', NULL);
```



```
psql:commands.sql:31: NOTICE:  TOTAL EMPLOYEES WITH GENDER FEMALE = 3
psql:commands.sql:32: NOTICE:  TOTAL EMPLOYEES WITH GENDER MALE = 3
```

Problem B:

Problem Title: SmartStore Automated Purchase System

Procedure (Step-by-Step):

SmartShop is a modern retail company that sells electronic gadgets like smartphones, tablets, and laptops. 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.

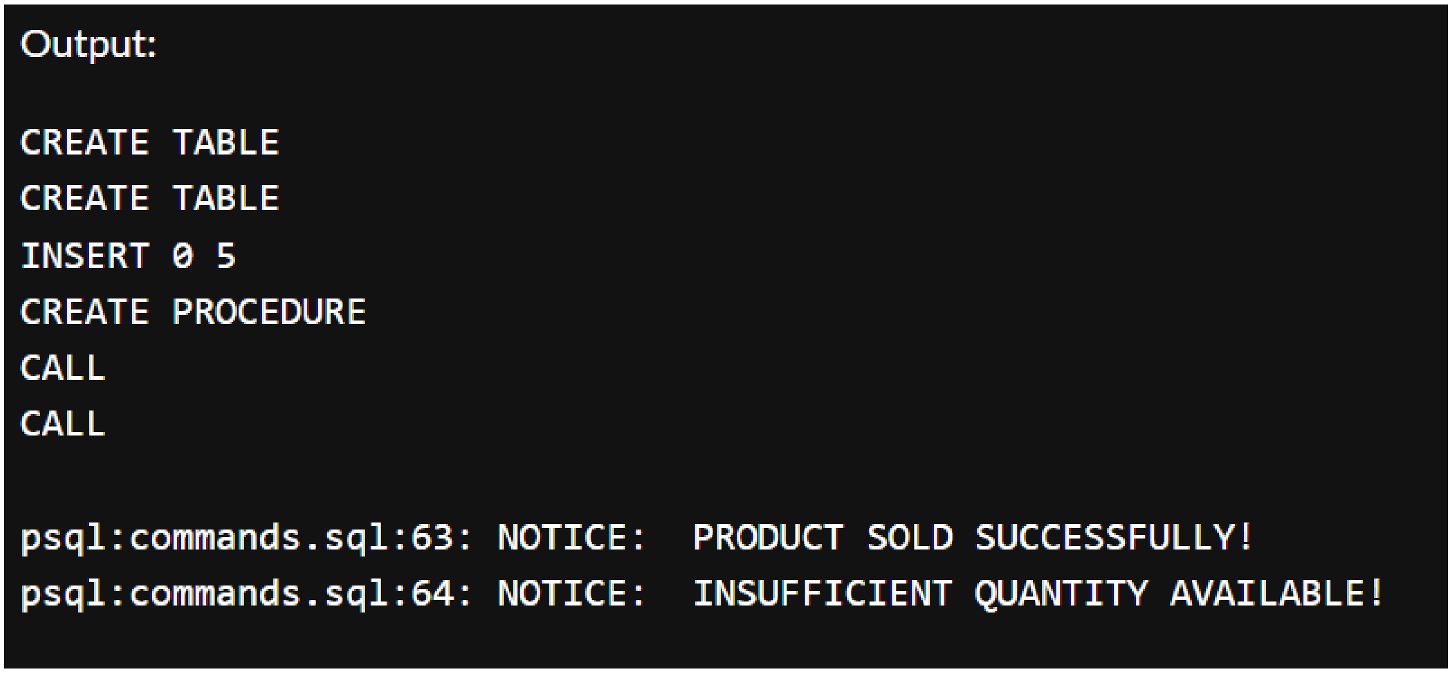
2. If sufficient stock is available:
   - **Log the order** in the sales table with the ordered quantity and total price.

   - **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!"

```
CREATE TABLE PRODUCTS (

    PRODUCT_ID SERIAL PRIMARY KEY,

    PRODUCT_NAME TEXT,

    PRICE NUMERIC(10,2),

    QUANTITY_REMAINING INT,

    QUANTITY_SOLD INT DEFAULT 0

);


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

);


INSERT INTO PRODUCTS (PRODUCT_NAME, PRICE, QUANTITY_REMAINING) VALUES

('SAMSUNG GALAXY', 20000, 50),

('APPLE IPHONE', 60000, 30),

('LENOVO LAPTOP', 45000, 20),

('DELL LAPTOP', 55000, 25),

('MI TABLET', 15000, 40);


CREATE OR REPLACE PROCEDURE PLACE_ORDER(

    IN IN_PRODUCT_ID INT,

    IN IN_QUANTITY INT
)
LANGUAGE plpgsql
AS $$
DECLARE
    AVAILABLE_QTY INT;
    PRODUCT_PRICE NUMERIC(10,2);
    TOTAL_COST NUMERIC(10,2);
BEGIN
    SELECT QUANTITY_REMAINING, PRICE
    INTO AVAILABLE_QTY, PRODUCT_PRICE
    FROM PRODUCTS
    WHERE PRODUCT_ID = IN_PRODUCT_ID;

    IF AVAILABLE_QTY IS NULL THEN
        RAISE NOTICE 'INVALID PRODUCT ID!';
        RETURN;
    END IF;

    IF AVAILABLE_QTY >= IN_QUANTITY THEN
        TOTAL_COST := PRODUCT_PRICE * IN_QUANTITY;

        INSERT INTO SALES (PRODUCT_ID, QUANTITY_ORDERED, TOTAL_PRICE)
        VALUES (IN_PRODUCT_ID, IN_QUANTITY, TOTAL_COST);

        UPDATE PRODUCTS
        SET QUANTITY_REMAINING = QUANTITY_REMAINING - IN_QUANTITY,
          QUANTITY_SOLD = QUANTITY_SOLD + IN_QUANTITY
        WHERE PRODUCT_ID = IN_PRODUCT_ID;

        RAISE NOTICE 'PRODUCT SOLD SUCCESSFULLY!';
    ELSE
        RAISE NOTICE 'INSUFFICIENT QUANTITY AVAILABLE!';
    END IF;
END;
$$;
```



Output:

```
CREATE TABLE
CREATE TABLE
INSERT 0 5
CREATE PROCEDURE
CALL
CALL

psql:commands.sql:63: NOTICE:  PRODUCT SOLD SUCCESSFULLY!
psql:commands.sql:64: NOTICE:  INSUFFICIENT QUANTITY AVAILABLE!
```

4. Learning Outcomes :

1. Understand how to **create and call a stored procedure** in PostgreSQL using CALL.
2. Learn the difference between **IN and OUT parameters** and how to pass values between SQL and a procedure.
3. Gain experience using **aggregate functions (COUNT)** inside procedures.
4. Learn to display messages using **RAISE NOTICE** for reporting and feedback.
5. Apply stored procedures to **real-world HR use cases** like diversity reporting.