

Experiment -6

Student Name: Nishant Rejra UID 23BCS12961

Branch: BE-CSE Section/Group: KRG_2B

Semester: 5th Date of Performance: 22/9/25

Subject Name: Advanced Database and Management System

SubjectCode: 23CSP-333

1. Problem Description/Aim:

Medium-Problem Title: Gender Diversity Tracking-Create a PostgreSQL stored procedure to track gender diversity in the workforce. The procedure takes a gender as input and returns the total number of employees of that gender, providing HR with instant and secure reporting.

Procedure (Step-by-Step):

- 1. Create a table employees with columns like emp id, emp name and gender.
- 2. Insert sample data with varying genders.
- 3. Create a stored procedure 'count_employees_by_gender' that: Takes a gender as input.
 - Counts the number of employees with that gender.
 - Returns the result as an OUT parameter.
- 4. Call the procedure in a DO block to capture and display the result.

Sample Output Description:

- Input: 'Male' --- Output: 3
- Input: 'Female'----Output: 2
- -HR sees results instantly without accessing full employee data.

Hard-Problem Title: Order Placement and Inventory Management-Automate the ordering process in a retail company. The procedure validates stock availability, logs sales, updates inventory, and provides real-time confirmation or rejection messages.

Procedure (Step-by-Step):

- 1. Create products table with columns: product_id, product_name, price, quantity_remaining, quantity_sold.
- 2. Create sales table with columns: sale_id, product_id, quantity, total_price, sale_date.
- 3. Create a stored procedure place_order that: Takes product_id and quantity as input.

Discover. Learn. Empower.

- Checks if quantity_remaining is sufficient.
- If yes:
- Logs the sale in sales table.
- Updates products(decrease quantity_remaining, increase quantity_sold).
- Display "Product sold successfully!!".
- If no:
- Display "Insufficient quantity available!!"
- 4. Call the procedure for different orders to validate functionality.

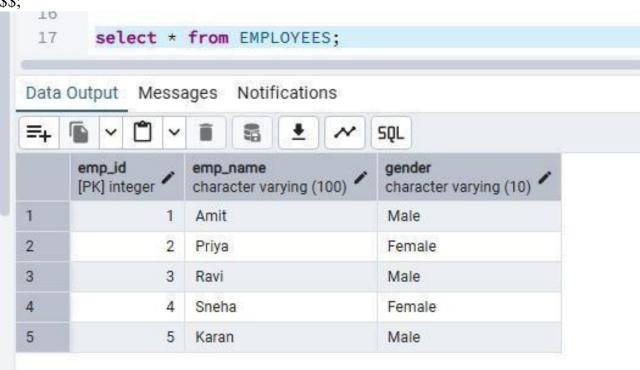
Sample Output Description:

- Order 5 units of Smartphone (stock available): "Product sold successfully!".
- Order 100 units of Tablet (insufficient stock): "Insufficient Quantity Available!". Inventory updates automatically for successful orders.
- 2. Objective: The objective is to automate critical business operations using PostgreSQL stored procedures. For HR, it tracks gender diversity by returning the total count of employees by gender. For retail, it manages orders by validating stock, logging sales, updating inventory, and providing real-time confirmation or rejection messages. This ensures efficiency, accuracy, and real-time insights in both workforce and inventory management.

3. SQL QUERY AND OUTPUTS -

```
-----MEDIUM PROBLEM-----
CREATE TABLE employees (emp id
  SERIAL
             PRIMARY
              VARCHAR(100),
  emp name
  gender VARCHAR(10)
);
-- Sample data
INSERT INTO employees (emp name, gender) VALUES
('Amit', 'Male'),
('Priya', 'Female'),
('Ravi', 'Male'),
('Sneha', 'Female'),
('Karan', 'Male');
select * from EMPLOYEES; ----CREATING
A PROCEDURE----
CREATE OR REPLACE PROCEDURE
  count employees by gender (IN input gender VARCHAR,
```

```
OUT total count int
LANGUAGE plpgsql
AS $$
BEGIN
  SELECT COUNT(*) INTO total count
  FROM employees
  WHERE gender = input gender;
END;
$$;
---CALLING THE PROCEDURE----DO
$$ DECLA
RE
  result INT;
BEGIN
  CALL count employees by gender('Male', result);
  RAISE NOTICE 'TOTAL EMPLOYEES OF GENDER Male ARE %', result;
END;
$$;
```



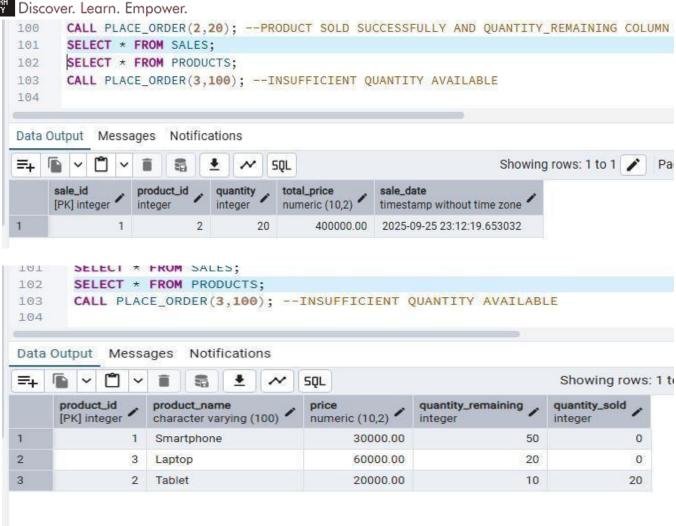
```
Discover. Learn. Empower.
 33
       DO $$
 34
       DECLARE
           result INT;
 35
 36 V BEGIN
           CALL count_employees_by_gender('Male', result);
 37
           RAISE NOTICE 'TOTAL EMPLOYEES OF GENDER Male ARE %', result;
 38
       END;
 39
Data Output Messages Notifications
NOTICE: TOTAL EMPLOYEES OF GENDER Male ARE 3
D0
Query returned successfully in 104 msec.
```

-----HARD PROBLEM -----

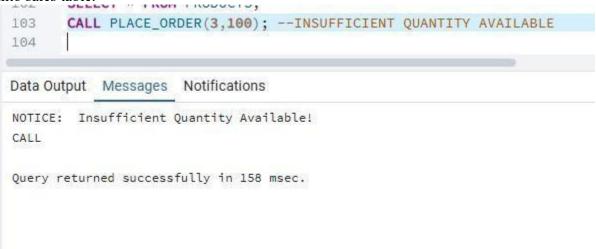
```
CREATE TABLE products (product id
  SERIAL
               PRIMARY
                              KEY.
  product name VARCHAR(100), price
  NUMERIC(10,2),
  quantity remaining INT,
  quantity sold INT DEFAULT 0
);
INSERT INTO products (product name, price, quantity remaining) VALUES
('Smartphone', 30000, 50),
('Tablet', 20000, 30),
('Laptop', 60000, 20);
CREATE TABLE sales ( sale id
  SERIAL PRIMARY KEY,
  product id INT REFERENCES products(product id),
  quantity INT, total price NUMERIC(10,2),
  sale date TIMESTAMP DEFAULT NOW()
);
CREATE OR REPLACE PROCEDURE
  place order(IN p product id INT, IN
  p quantity INT
```

```
LANGUAGE plpgsql
AS $$
DECLARE
  available stock
                           INT;
  product price NUMERIC(10,2);
BEGIN
  SELECT quantity remaining, price
  INTO available stock, product price
  FROM products
  WHERE product id = p product id;
  IF available stock IS NULL THEN
    RAISE NOTICE 'Product ID % does not exist!', p product id;
  ELSIF available stock >= p quantity THEN
    -- LOGGING THE ORDER
    INSERT INTO sales (product id, quantity, total price)
    VALUES (p product id, p quantity, p quantity * product price);
    UPDATE products
    SET quantity remaining = quantity remaining - p quantity,
    quantity sold = quantity sold + p quantity WHERE
    product_id = p_product_id;
    RAISE NOTICE 'Product sold successfully!';
  ELSE
    RAISE NOTICE 'Insufficient Quantity Available!'; END
  IF;
END:
$$;
CALL PLACE ORDER(2,20); -- PRODUCT SOLD SUCCESSFULLY AND
QUANTITY REMAINING COLUMN SET TO -20 AND DATA LOGGED TO SALES
TABLE
SELECT * FROM SALES;
SELECT * FROM PRODUCTS;
CALL PLACE ORDER(3,100); -- INSUFFICIENT QUANTITY AVAILABLE
```

DEPARTMENTOF COMPUTERSCIENCE&ENGINEERING



--Here in above output, After selling 20 tablets (id==2) we are left with 10 and the selling data is logged into sales table.



ID ==3 means laptop are 20 only and we place order for 100 ...so we get notice - for insufficient quantity!!