DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CU CHANDIGARH UNIVERSITY Discover. Learn. Empower.

Experiment - 6

Student Name: Aastha	**UID:** 23BAI70432
Branch: BE-AIT-CSE	**Section/Group:** 23AIT_KRG-G2_B
Semester: 5th	**Date of Performance:** 24 Sept, 2025
Subject Name: ADBMS	**Subject Code:** 23CSP-333

1. Aim:

MEDIUM LEVEL PROBLEM:

HR ANALYTICS:
To create a PostgreSQL stored procedure that dynamically counts the total number of employees based on a given gender. This allows HR departments to instantly generate reports on workforce diversity and track gender representation efficiently.

HARD LEVEL PROBLEM:

SMARTSTORE AUTOMATED PURCHASE SYSTEM:
To automate product ordering and inventory management in a retail database. The procedure ensures stock validation before processing orders, updates inventory accurately, logs sales transactions, and provides real-time confirmation messages to customers.

2. Objective:

For HR Analytics:

- Learn how to define and execute stored procedures in PostgreSQL.
- Enable dynamic input handling to count employees by gender.
- Provide HR with instant and accurate workforce analytics.
- Understand the use of IN and OUT parameters and result display using RAISE NOTICE.

For SmartStore System:

- Implement database-driven automation for retail operations.

- Check product stock availability before order processing.
- Update inventory (quantity_remaining, quantity_sold) correctly to prevent errors.
- Log transactions in a sales table for accountability.
- Provide feedback messages to users in real-time to improve the ordering experience.

3. Theory:

1. Stored Procedures

A **stored procedure** is a precompiled set of SQL statements stored in the database that can perform operations like querying, updating, or inserting data. Advantages include:

- Reusability: The procedure can be executed multiple times without rewriting SQL queries.
- Security: Users can execute procedures without direct access to tables.
- Efficiency: Reduces network traffic and increases performance by executing multiple SQL statements as one unit.

2. Input and Output Parameters

- **IN parameter:** Accepts input data from the user (e.g., gender, product_id).
- **OUT parameter:** Returns output data after processing (e.g., total employee count).

3. RAISE NOTICE

- A PostgreSQL command used to display messages during procedure execution.
- Useful for logging information or providing real-time feedback without writing to a table.

4. Application in HR Analytics

- HR often needs quick insights into workforce demographics.
- A stored procedure with a gender parameter avoids repetitive query writing and allows for **dynamic reporting**.

5. Application in Retail Automation

- SmartShop wants **real-time automation** in sales and inventory.
- The stored procedure validates stock before processing the order:
  - If sufficient: logs sale, updates inventory, displays confirmation.
  - If insufficient: rejects the order and shows an error.
- This ensures **data integrity**, **avoids overselling**, and enhances **customer satisfaction**.

6. Transactions

- Ensures that inventory updates and sales logging occur as a single atomic operation.
- If any step fails, the database rolls back changes to maintain consistency.

4. Procedure:

Medium Level Solution:

- **Setup:** Create an employee_info table and populate it with sample data, including employee names, genders, and other details.
- **Procedure Creation:** Develop a stored procedure named sp_get_employees_by_gender. This procedure takes a gender as an input parameter and an integer output parameter.
- **Business Logic:** Inside the procedure, a SELECT COUNT query counts all employees that match the input gender. The result is then stored in the output parameter.
- **Execution:** The procedure is called with a specific gender value (e.g., 'Male'), and a RAISE NOTICE command is used to print the final count, demonstrating a simple yet powerful automated reporting feature.

Hard Level Solution:

- **Setup:** Establish a database schema with products and sales tables to represent inventory and order history, respectively. Insert sample data into both tables.
- **Procedure Creation:** Create a stored procedure named pr_buy_products that accepts the product name and quantity as input.
- **Transactional Logic:** The procedure first checks if the requested quantity is available in the products table.
- **Conditional Processing:**
- **If sufficient stock:** The procedure executes a series of steps within a transaction: it inserts a new record into the sales table, updates the products table to reflect the reduced inventory (quantity_remaining) and increased sales (quantity_sold), and then prints a success message.
- **If insufficient stock:** The procedure immediately prints an "INSUFFICIENT QUANTITY" message without logging a sale or altering the inventory tables.
- **Execution:** Test the procedure with different values to demonstrate both a successful sale (when sufficient stock is available) and a failed transaction (when the quantity is too high), showcasing its transactional integrity and errorhandling capabilities.

5. Code:

MEDIUM PROBLEM

```
CREATE TABLE Employeess (
   emp_id SERIAL PRIMARY KEY,
   emp_name VARCHAR(50),
   gender VARCHAR(10)
);
INSERT INTO Employeess (emp_name, gender) VALUES
('Alice', 'Female'),
('Bob', 'Male'),
('Charlie', 'Male'),
('Diana', 'Female');
drop procedure if exists get_employee_count_by_gender
```

```
CREATE OR REPLACE PROCEDURE get_employee_count_by_gender(
    IN input_gender VARCHAR,
    OUT total_count INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT COUNT(*)
    INTO total_count
    FROM Employeess
    WHERE gender = input_gender;

    RAISE NOTICE 'Gender: %, Total Employees: %', input_gender, total_count;
END;
$$;
CALL get_employee_count_by_gender('Male', total_count => 0);
CALL get_employee_count_by_gender('Female', total_count => 0);
```

HARD PROBLEM

```
CREATE TABLE Items (
    item_id SERIAL PRIMARY KEY,
    item_name VARCHAR(50),
    price DECIMAL(10,2),
    quantity_remaining INT,
    quantity_sold INT DEFAULT 0
);

-- Orders Table
CREATE TABLE Orders (
    order_id SERIAL PRIMARY KEY,
    item_id INT REFERENCES Items(item_id),
    quantity_ordered INT,
    total_price DECIMAL(10,2),
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE OR REPLACE PROCEDURE process_order(
    IN input_item_id INT,
    IN input_quantity INT
)
LANGUAGE plpgsql
AS $$
DECLARE
    available_stock INT;
    item_price DECIMAL(10,2);
    total_cost DECIMAL(10,2);
BEGIN
    -- Check available stock and price
    SELECT quantity_remaining, price
    INTO available_stock, item_price
    FROM Items
    WHERE item_id = input_item_id;
    -- If enough stock available
    IF available_stock >= input_quantity THEN
        total_cost := item_price * input_quantity;

        -- Insert order into Orders table
        INSERT INTO Orders (item_id, quantity_ordered, total_price)
        VALUES (input_item_id, input_quantity, total_cost);

        -- Update inventory in Items table
        UPDATE Items
        SET quantity_remaining = quantity_remaining - input_quantity,
            quantity_sold = quantity_sold + input_quantity
        WHERE item_id = input_item_id;

        RAISE NOTICE 'Product sold successfully!';
    ELSE
        RAISE NOTICE 'Insufficient Quantity Available!';
    END IF;
END;
$$;
CALL process_order(1, 2);
```

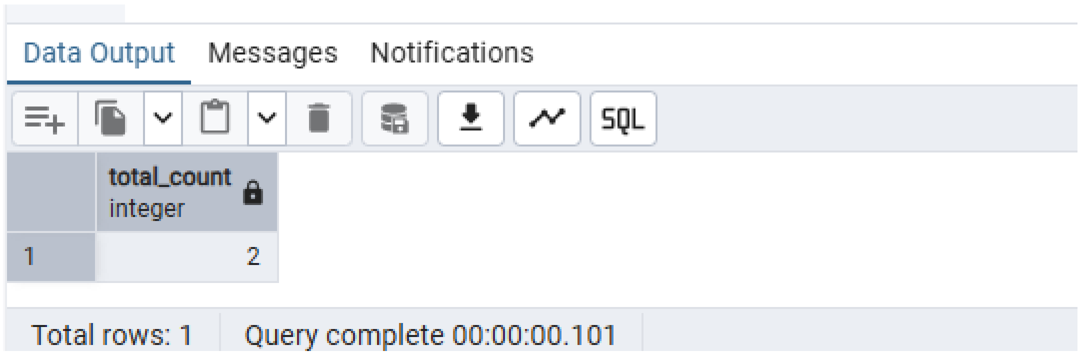
6. Output:

Medium:

```
CREATE PROCEDURE

Query returned successfully in 180 msec.
```

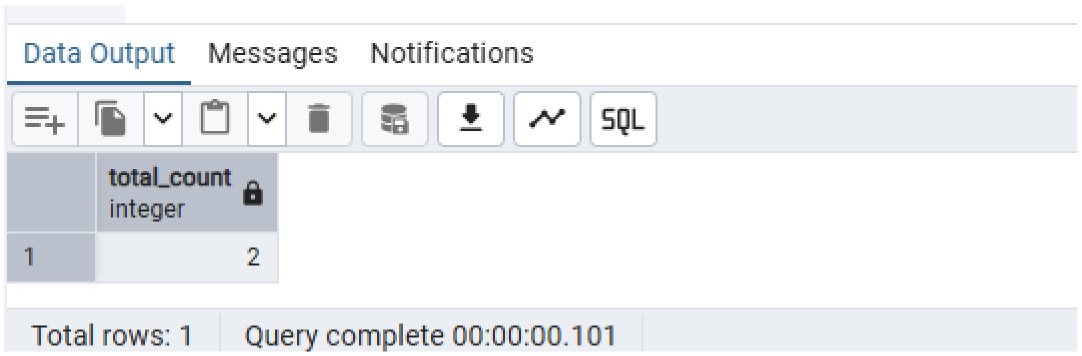
For males:



	total_count integer
1	2

```
Total rows: 1    Query complete 00:00:00.101
```

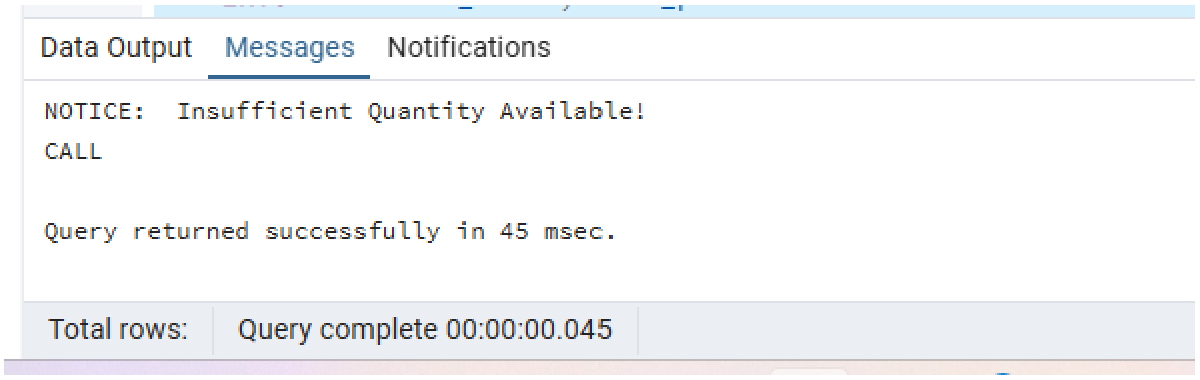
For Females:



	total_count integer
1	2

Total rows: 1 Query complete 00:00:00.101

Hard:



```
NOTICE:  Insufficient Quantity Available!
CALL

Query returned successfully in 45 msec.
```

Total rows: Query complete 00:00:00.045

7. Learning Outcomes:

Stored Procedure Implementation:

- Learned how to create, execute, and manage stored procedures in PostgreSQL.
- Understood the use of IN and OUT parameters for dynamic input and output handling.

Dynamic Querying:

- Gained the ability to write procedures that count records based on dynamic input, such as gender.
- Learned how to avoid repetitive queries by automating common HR analytics tasks.

Result Display:

- Learned to use RAISE NOTICE for real-time feedback in pgAdmin.
- Understood how to present calculated results clearly for reporting purposes.

Database Management Skills:
- Practiced working with tables, inserting data, and validating results.
- Developed analytical skills for HR reporting and workforce diversity tracking.

Transaction Automation:
- Learned to automate retail operations using stored procedures.
- Understood how to validate stock before processing orders.

Inventory Management:
- Gained experience in updating multiple tables (products and sales) in a single procedure.
- Learned how to maintain data integrity by adjusting quantity_remaining and quantity_sold.

Conditional Logic in Procedures:
- Learned to implement IF-ELSE logic to handle sufficient and insufficient stock scenarios.
- Practiced providing real-time notifications to the user.

Dynamic Input Handling:
- Developed the skill to take dynamic product name and quantity as input for automated processing.
- Learned to calculate total sale price dynamically using stored values.

Practical Application:
- Understood how database procedures can simulate real-world business operations, like inventory control and order management.
- Enhanced ability to solve complex database problems with procedural programming.