DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

Experiment - 6

Student Name: SHRESHTA

UID: 23BAI70375

Branch: BE-AIT-CSE

Section/Group: 23AIT_KRG-G1_A

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 employee_info (
id SERIAL PRIMARY KEY, name
VARCHAR(50) NOT NULL, gender
VARCHAR(10) NOT NULL, salary
NUMERIC(10,2) NOT NULL, city
VARCHAR(50) NOT NULL
);

INSERT INTO employee_info (name, gender, salary, city)
VALUES
('Alok', 'Male', 50000.00, 'Delhi'),
('Priya', 'Male', 60000.00, 'Mumbai'),
('Rajesh', 'Female', 45000.00, 'Bangalore'),
```

```
('Sneha', 'Male', 55000.00, 'Chennai'),
('Anil', 'Male', 52000.00, 'Hyderabad'),
('Sunita', 'Female', 48000.00, 'Kolkata'),
('Vijay', 'Male', 47000.00, 'Pune'),
('Ritu', 'Male', 62000.00, 'Ahmedabad'),
('Amit', 'Female', 51000.00, 'Jaipur');



CREATE OR REPLACE PROCEDURE sp_get_employees_by_gender(
IN p_gender VARCHAR(50),
OUT p_employee_count INT
)
LANGUAGE plpgsql
AS $$
BEGIN

SELECT COUNT(id)
INTO p_employee_count
FROM employee_info
WHERE gender = p_gender;


RAISE NOTICE 'Total employees with gender %: %', p_gender, p_employee_count;
END;
$$;


CALL sp_get_employees_by_gender('Male', NULL);
```

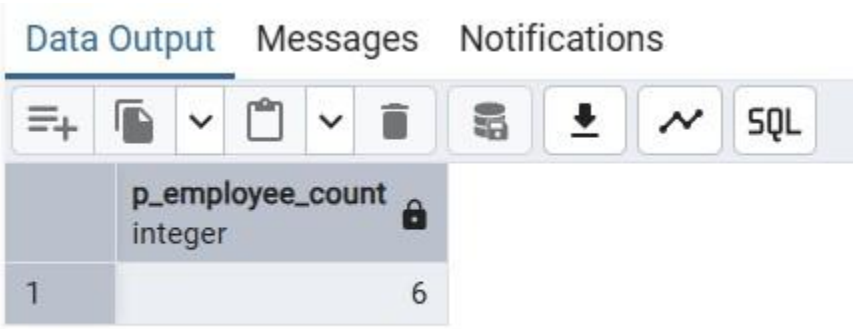
HARD PROBLEM

```
CREATE TABLE products (
product_code VARCHAR(10) PRIMARY KEY,
product_name VARCHAR(100) NOT NULL, price
NUMERIC(10,2) NOT NULL,
quantity_remaining INT NOT NULL,
quantity_sold INT DEFAULT 0
);

CREATE TABLE sales ( order_id
SERIAL PRIMARY KEY, order_date
DATE NOT NULL, product_code
VARCHAR(10) NOT NULL,
quantity_ordered INT NOT NULL,
sale_price NUMERIC(10,2) NOT NULL,
FOREIGN KEY (product_code) REFERENCES products(product_code)
);

INSERT INTO products (product_code, product_name, price, quantity_remaining,
quantity_sold)
VALUES
('P001', 'iPHONE 13 PRO MAX', 109999.00, 10, 0),
('P002', 'Samsung Galaxy S23 Ultra', 99999.00, 8, 0),
('P003', 'iPAD AIR', 55999.00, 5, 0),
('P004', 'MacBook Pro 14"', 189999.00, 3, 0),
('P005', 'Sony WH-1000XM5 Headphones', 29999.00, 15, 0);
```

```
INSERT INTO sales (order_date, product_code, quantity_ordered, sale_price)
VALUES
('2025-09-15', 'P001', 1, 109999.00),
('2025-09-16', 'P002', 2, 199998.00),
('2025-09-17', 'P003', 1, 55999.00),
('2025-09-18', 'P005', 2, 59998.00),
('2025-09-19', 'P001', 1, 109999.00);

SELECT * FROM PRODUCTS;
SELECT * FROM SALES;


CREATE OR REPLACE PROCEDURE pr_buy_products(
IN p_product_name VARCHAR,
IN p_quantity INT
)
LANGUAGE plpgsql
AS $$ DECLARE
v_product_code VARCHAR(20);
v_price FLOAT; v_count INT;
BEGIN


SELECT COUNT(*)
INTO v_count
FROM products
WHERE product_name = p_product_name
AND quantity_remaining >= p_quantity;


IF v_count > 0 THEN


SELECT product_code, price
INTO v_product_code, v_price
FROM products
WHERE product_name = p_product_name;


INSERT INTO sales (order_date, product_code, quantity_ordered, sale_price)
VALUES (CURRENT_DATE, v_product_code, p_quantity, (v_price * p_quantity));
UPDATE products
SET quantity_remaining = quantity_remaining - p_quantity, quantity_sold
= quantity_sold + p_quantity
WHERE product_code = v_product_code;


RAISE NOTICE 'PRODUCT SOLD..! Order placed successfully for % unit(s) of %.',
p_quantity, p_product_name;

ELSE

RAISE NOTICE 'INSUFFICIENT QUANTITY..! Order cannot be processed for % unit(s)
of %.', p_quantity, p_product_name;
END IF;
END;
```

```
$$;

CALL pr_buy_products ('MacBook Pro 14"', 1);
```

6. Output:



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.