

EXPERIMENT - 6

STUDENT NAME: ASHLIN JAMES

UID: 23BAI70722

BRANCH: BE-AIT-CSE

SECTION/GROUP: 23AIT_KRG-G1_A

SEMESTER: 5

DATE OF PERFORMANCE: 24 SEPT, 2025

SUBJECT NAME: ADBMS

SUBJECT CODE: 23CSP-333

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.

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.

THEORY:

1. Stored Procedures

A stored procedure is a named collection of SQL statements stored and executed on the database server. Unlike ad-hoc queries, stored procedures provide several advantages:

- ☒ Performance: As they are precompiled and stored, execution is faster.
- ☒ Reusability: Common logic can be reused without rewriting queries.
- ☒ Security: Database access can be controlled by exposing procedures instead of raw tables.
- ☒ Maintainability: Business logic can be updated in one place rather than multiple queries.

2. Input and Output Parameters

- ☒ IN Parameters: Accept input from the caller (e.g., gender = 'Male').
- ☒ OUT Parameters: Return results back to the caller (e.g., employee_count = 6).
- ☒ Parameters help in designing flexible and reusable procedures.

3. RAISE NOTICE

- ☒ A PostgreSQL debugging and logging tool.
- ☒ Prints messages in the console to show intermediate or final outputs.
- ☒ Useful for confirming whether the procedure executed successfully or failed due to some condition.

4. Application in HR Analytics

- ☒ HR departments often require reports such as “How many female employees do we have in Bangalore?” or “How many male employees are there in the company?”
A stored procedure with gender as a parameter eliminates repetitive queries and makes analytics dynamic.

5. Application in Retail Automation

- ☒ Retailers need real-time inventory updates to prevent overselling.
- ☒ The procedure validates whether the required stock is available.
- ☒ If available → updates sales, adjusts inventory, and notifies success.
- ☒ If unavailable → rejects order immediately with a clear error message.
- ☒ This improves data accuracy, enhances customer trust, and reduces manual intervention.

6. Transactions

- ☒ A transaction ensures that a set of operations (insert, update, delete) either all succeed or all fail. In this experiment, both sales entry and inventory update must occur together. If either fails, the transaction rolls back, ensuring database consistency.

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 error handling capabilities.

CODE:

--Medium Level 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  
( 'Arjun', 'Male', 53000.00, 'Bengaluru'),  
( 'Meera', 'Female', 61000.00, 'Chennai'),  
( 'Karan', 'Male', 47000.00, 'Pune'),  
( 'Divya', 'Female', 56000.00, 'Delhi'),  
( 'Rohan', 'Male', 49000.00, 'Mumbai'),  
( 'Ananya', 'Female', 52000.00, 'Kolkata'),  
( 'Siddharth', 'Male', 48000.00, 'Hyderabad'),  
( 'Tanvi', 'Female', 63000.00, 'Ahmedabad'),  
( 'Vikram', 'Male', 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 Level 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', 'MacBook Pro 14"', 189999.00, 5, 0),  
( 'P002', 'Google Pixel 8 Pro', 79999.00, 12, 0),  
( 'P003', 'OnePlus 12', 69999.00, 10, 0),  
( 'P004', 'iPad Pro 11"', 64999.00, 6, 0),  
( 'P005', 'Bose QuietComfort 45 Headphones', 24999.00, 20, 0);
```

```
INSERT INTO sales (order_date, product_code, quantity_ordered, sale_price)  
VALUES  
( '2025-09-20', 'P001', 1, 189999.00),  
( '2025-09-21', 'P002', 2, 159998.00),  
( '2025-09-22', 'P003', 1, 69999.00),  
( '2025-09-23', 'P004', 1, 64999.00),  
( '2025-09-24', 'P005', 3, 74997.00);
```

```
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);

```

OUTPUT:

The first screenshot shows the 'Messages' tab in pgAdmin. It displays a 'NOTICE: PRODUCT SOLD...! Order placed successfully for 1 unit(s) of MacBook Pro 14". CALL' and a status message 'Query returned successfully in 69 msec.'

The second screenshot shows the 'Messages' tab with a different message: 'NOTICE: INSUFFICIENT QUANTITY...! Order cannot be processed for 10 unit(s) of MacBook Pro 14". CALL'. The status message is 'Query returned successfully in 37 msec.'

The third screenshot shows the 'Data Output' tab with a table titled 'p_employee_count' of type 'integer'. The table has one row with the value '6'.

p_employee_count integer	
1	6

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.