DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CU CHANDIGARH UNIVERSITY Discover. Learn. Empower.

Experiment - 6

Student Name: Gursharan Singh	**UID** 23BAI70554
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:
To develop an automated database solution for TechSphere Solutions to monitor gender diversity. This will be achieved by creating a PostgreSQL stored procedure that takes a specific gender ('Male' or 'Female') as an input parameter, calculates the total count of employees for that gender, and returns the result as an output parameter for immediate HR reporting. This will allow the HR department to instantly and accurately retrieve the required diversity metrics.

HARD LEVEL PROBLEM:
The aim is to automate SmartShop's ordering and inventory management by creating a robust system that, upon a customer's request, verifies stock availability. If the stock is sufficient, the system will execute a single, atomic transaction to log the sale, update the product inventory, and confirm the transaction in real-time. Conversely, if the stock is insufficient, the system will reject the transaction and immediately inform the user, ensuring the integrity of both sales and inventory data.

2. Objective:

- Develop a PostgreSQL stored procedure to calculate and return the count of employees for a specified gender.
- Enable the HR department to instantly retrieve accurate gender diversity metrics for reporting.
- Automate the ordering process to verify stock availability for a customer's request.
- Execute a single, atomic transaction to log successful sales and update inventory levels in real-time.
- Implement a system to reject transactions with insufficient stock and provide immediate feedback to the user.
- Ensure the integrity of both sales and inventory data through a transactional approach.

3. Theory:

A **stored procedure** is a pre-compiled set of SQL statements and control flow logic stored in the database. Unlike a simple SQL query, a stored procedure can accept input parameters, perform conditional logic (like IF/ELSE), and return values. This makes them ideal for encapsulating complex, repeatable business logic. In the context of the TechSphere Solutions problem, a stored procedure provides an efficient and reusable way to solve a recurring HR task. Instead of writing a new SELECT and COUNT query every time, the HR team can simply call a pre-built procedure with the required gender as an argument. This approach improves data retrieval speed by reducing network traffic and, more importantly, ensures that the calculation is consistently and accurately performed every time, a crucial aspect of reliable reporting.

In database management, a **transaction** is a single unit of work that contains one or more operations. Transactions are essential for maintaining data integrity, especially in applications like inventory management where multiple related operations must succeed or fail together. This is governed by the ACID properties:

- **Atomicity:** This is the most critical concept for the SmartShop problem. Atomicity ensures that a transaction is treated as a single, indivisible unit. It either completes fully, with all its operations succeeding, or it fails completely, with none of its operations taking effect. This prevents partial updates that could lead to data inconsistencies—for example, a sale being logged without a corresponding reduction in inventory.
- **Consistency:** The database state remains valid before and after the transaction.
- **Isolation:** Concurrent transactions do not interfere with one another.
- **Durability:** Once a transaction is committed, its changes are permanent.

By using a transactional stored procedure, SmartShop can ensure that an order is only logged if the stock can be successfully updated. If the stock check fails, the transaction is rejected, and the database remains in its original, consistent state. This guarantees that the inventory count is always accurate and that no stock is "sold" if it's not physically available, which is vital for real-time business operations.

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 error- handling capabilities.

5. Code:

```
---------------------------------------------- EXflERIMENT 06 (MEDIUM LEVEL) ------------------------------------------------

CREATE  TABLE  employee_info  (  id
SERIAL  flRIMARY  KEY,  name  VARC
                  H         N     N
AR(50) OT ULL, gender VARC AR(10) OT
                  H        N    N
ULL, salary UMERIC(10,2) OT ULL,
               N           N       N
city VARC AR(50) OT ULL
               H          N     N
);

INSERT INTO employee_info (name, gender, salary, city)
VALUES
('Alok', 'Male', 50000.00, 'Delhi'),
('flriya', '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, 'flune'),
('Ritu', 'Male', 62000.00, 'Ahmedabad'),
('Amit', 'Female', 51000.00, 'Jaipur');



CREATE OR REflLACE flROCEDURE sp_get_employees_by_gender(
                               H
IN p_gender VARC AR(50),
OUT p_employee_count I T  N
)
LANGUAGE plpgsql
AS $$
BEGIN
```

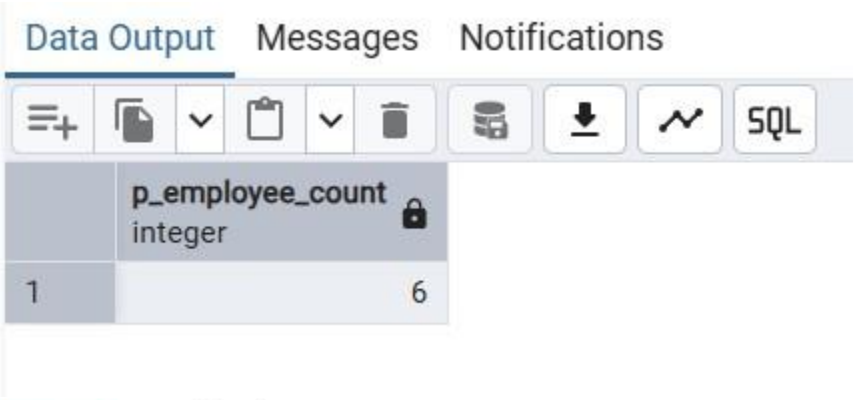
```
SELECT COUNT(id)
VNTO p_employee_count
FROM employee_info
W          HERE gender = p_gender;


RAISE ONICE 'Total employees with gender %: %', p_gender, p_employee_count; E D;
$$N;


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

-- EXflERIMENT 06 (HARD LEVEL) --

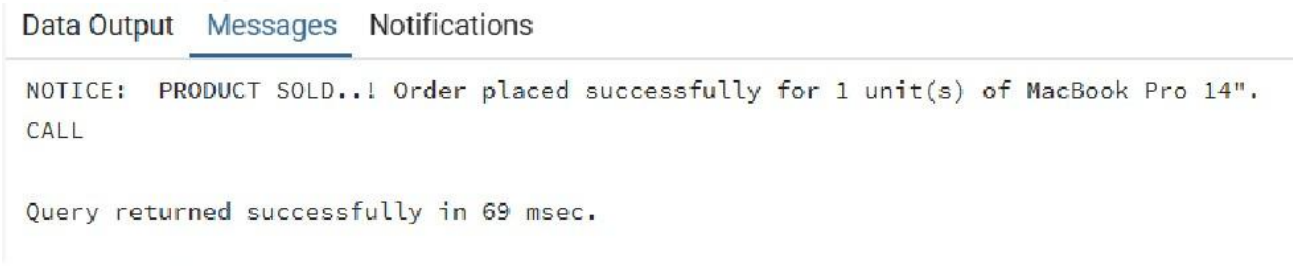
```
CREATE TABLE products (
product_code   VARC HAR(10)   flRIMARY   KEY,
product_name  VARC AHR(100)  OT  NULL,  Nprice N
UMERIC(10,2)  OT  ULL,  Nquantity_remaining  I  T
OT ULL,             N    N      N
quantity_sold I T DNEFAULT 0
);
CREATE   TABLE   sales   (   order_id   SERIAL
flRIMARY  KEY,  order_date  DATE  OT  ULL,
product_code    VARCN AR(N10)    OT    ULL,
quantity_ordered  I  T  HOT  UNLL,  sNale_price N
UMERIC(10,2) OT ULL,  N    N     N
FOREIGN KEY (product_code) REFNERE CNES products(product_code)
);                                          N

X NSERT INTO products (product_code, product_name, price, quantity_remaining, quantity_sold)
VALUES
('fl001', 'ifl O EH1N5 flRO MAX', 109999.00, 10, 0),
('fl002', 'Samsung Galaxy S23 Ultra', 99999.00, 8, 0),
('fl003', 'iflAD AIR', 55999.00, 5, 0),
('fl004', 'MacBook flro 14"', 189999.00, 3, 0),
('fl005', 'Sony WH-1000XM5 Headphones', 29999.00, 15, 0);

INSERT INTO sales (order_date, product_code, quantity_ordered, sale_price) VALUES
('2025-09-15', 'fl001', 1, 109999.00),
('2025-09-16', 'fl002', 2, 199998.00),
('2025-09-17', 'fl003', 1, 55999.00),
('2025-09-18', 'fl005', 2, 59998.00),
('2025-09-19', 'fl001', 1, 109999.00);

SELECT * FROM flRODUCTS;
SELECT * FROM SALES;


CREATE OR REflLACE flROCEDURE pr_buy_products(
IN p_product_name VARCHAR,
IN p_quantity INT
)
```

```
LANGUAGE plpgsql
AS $$ DECLARE
v_product_code    VARC  H  AR(20);
v_price FLOAT; v_count I T; BEGI
           N
      N

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


IF v_count > 0 T E    H N


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));
UflDATE products
SET quantity_remaining = quantity_remaining - p_quantity,
quantity_sold   =   quantity_sold   +   p_quantity   W H ERE
product_code = v_product_code;


RAISE  N OTICE  'flRODUCT  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;
 N   ED IF;
 N   ED;
$$;


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

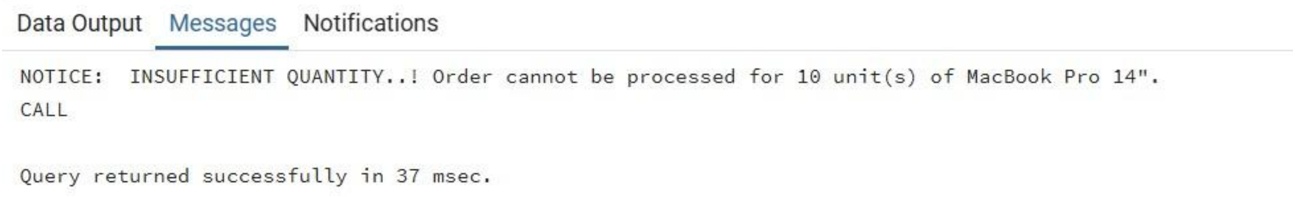
6. Output:



	p_employee_count integer
1	6



```
NOTICE:  PRODUCT SOLD..! Order placed successfully for 1 unit(s) of MacBook Pro 14".
CALL

Query returned successfully in 69 msec.
```



```
NOTICE:  INSUFFICIENT QUANTITY..! Order cannot be processed for 10 unit(s) of MacBook Pro 14".
CALL

Query returned successfully in 37 msec.
```

7. Learning Outcomes:

- **Encapsulation of Business Logic:** Learners will understand how stored procedures can be used to encapsulate complex business logic (e.g., counting employees by a specific criteria) into a single, reusable function. This promotes modularity and maintainability in database design.
- **Procedural Language Fundamentals:** This experiment provides a practical introduction to procedural languages like PL/pgSQL, including the use of DECLARE, BEGIN...END, input and output parameters, and the RAISE NOTICE command for debugging and reporting.
- **Efficient Reporting:** Learners will grasp how stored procedures can create a more efficient reporting layer. Instead of requiring a full query with aggregations from the client, the client only needs to make a simple CALL to the procedure, which handles the computation on the database server, reducing network overhead.
- **Encapsulation of Business Logic:** Learners will understand how stored procedures can be used to encapsulate complex business logic (e.g., counting employees by a specific criteria) into a single, reusable function. This promotes modularity and maintainability in database design.
- **Transactional Integrity (ACID Properties):** The primary learning outcome is a deep understanding of atomicity and transactional integrity. Learners will see firsthand how a single, cohesive transaction prevents data inconsistencies that could arise from partial updates, such as logging a sale without updating the inventory.
- **Real-time Inventory Management:** This experiment demonstrates a core function of e-commerce systems. Learners will understand how to use stored procedures to handle

real-time inventory checks and updates, ensuring that stock levels are always accurate and preventing overselling.

- **Error Handling and User Feedback:** The use of IF/ELSE logic and RAISE NOTICE provides a clear example of how to handle different outcomes (success vs. failure) within a stored procedure and provide direct, real-time feedback to the user, a critical component of any user-facing application.