



## Experiment - 6

**Student Name:** GUNJAN

**UID:** 23BCS13605

**Branch:** BE-CSE

**Section/Group:** KRG\_2A

**Semester:** 5<sup>th</sup>

**Date of Performance:** 25/9/25

**Subject Name:** Advanced Database and Management System

**Subject Code:** 23CSP-333

### 1. 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.

#### **Source Code**

```
CREATE TABLE employees
( emp_id SERIAL PRIMARY
KEY, emp_name VARCHAR(100),
gender VARCHAR(10)
);
```



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

-- 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;

\$\$;

```
16
17 select * from EMPLOYEES;
```

Data Output Messages Notifications

	emp_id [PK] integer	emp_name character varying (100)	gender character varying (10)
1	1	Amit	Male
2	2	Priya	Female
3	3	Ravi	Male
4	4	Sneha	Female
5	5	Karan	Male

```
33 DO $$
34 DECLARE
35     result INT;
36 BEGIN
37     CALL count_employees_by_gender('Male', result);
38     RAISE NOTICE 'TOTAL EMPLOYEES OF GENDER Male ARE %', result;
39 END;
```

Data Output Messages Notifications

NOTICE: TOTAL EMPLOYEES OF GENDER Male ARE 3  
DO

Query returned successfully in 104 msec.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 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):

- Create products table with columns: product\_id, product\_name, price, quantity\_remaining, quantity\_sold.
- Create sales table with columns: sale\_id, product\_id, quantity, total\_price, sale\_date.
- Create a stored procedure place\_order that:
  - ✦ Takes product\_id and quantity as input.
  - ✦ 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!!”
- 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.

**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.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Source Code

```
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
```



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

```
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);  
SELECT * FROM SALES; SELECT * FROM  
PRODUCTS;  
CALL PLACE_ORDER(3,100);
```

```

100 CALL PLACE_ORDER(2,20); --PRODUCT SOLD SUCCESSFULLY AND QUANTITY_REMAINING COLUMN
101 SELECT * FROM SALES;
102 SELECT * FROM PRODUCTS;
103 CALL PLACE_ORDER(3,100); --INSUFFICIENT QUANTITY AVAILABLE
104
101 SELECT * FROM SALES;
102 SELECT * FROM PRODUCTS;
103 CALL PLACE_ORDER(3,100); --INSUFFICIENT QUANTITY AVAILABLE
104

```

Data Output Messages Notifications

Showing rows: 1 to 3

	product_id [PK] integer	product_name character varying (100)	price numeric (10,2)	quantity_remaining integer	quantity_sold integer
1	1	Smartphone	30000.00	50	0
2	3	Laptop	60000.00	20	0
3	2	Tablet	20000.00	10	20