



## Experiment - 6

**Student Name:** Sarthak Arora

**UID:** 23BCS12984

**Branch:** BE-CSE

**Section/Group:** KRG\_2B

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

**Date of Performance:** 22/9/25

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

**Subject Code:** 23CSP-333

### 1. Problem Description/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.

**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):**

1. Create products table with columns: product\_id, product\_name, price, quantity\_remaining, quantity\_sold.
2. Create sales table with columns: sale\_id, product\_id,

- quantity, total\_price, sale\_date.
3. 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!!”
  4. 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.

**2. 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.

### 3. SQL QUERY AND OUTPUTS -

#### -----MEDIUM PROBLEM-----

```
CREATE TABLE employees
```

```
( emp_id SERIAL PRIMARY KEY,  
  emp_name VARCHAR(100),  
  gender VARCHAR(10)
```

```
);
```

```
-- Sample data
```

```
INSERT INTO employees (emp_name, gender) VALUES
```

```
('Amit', 'Male'),
```

```
('Priya', 'Female'),
```

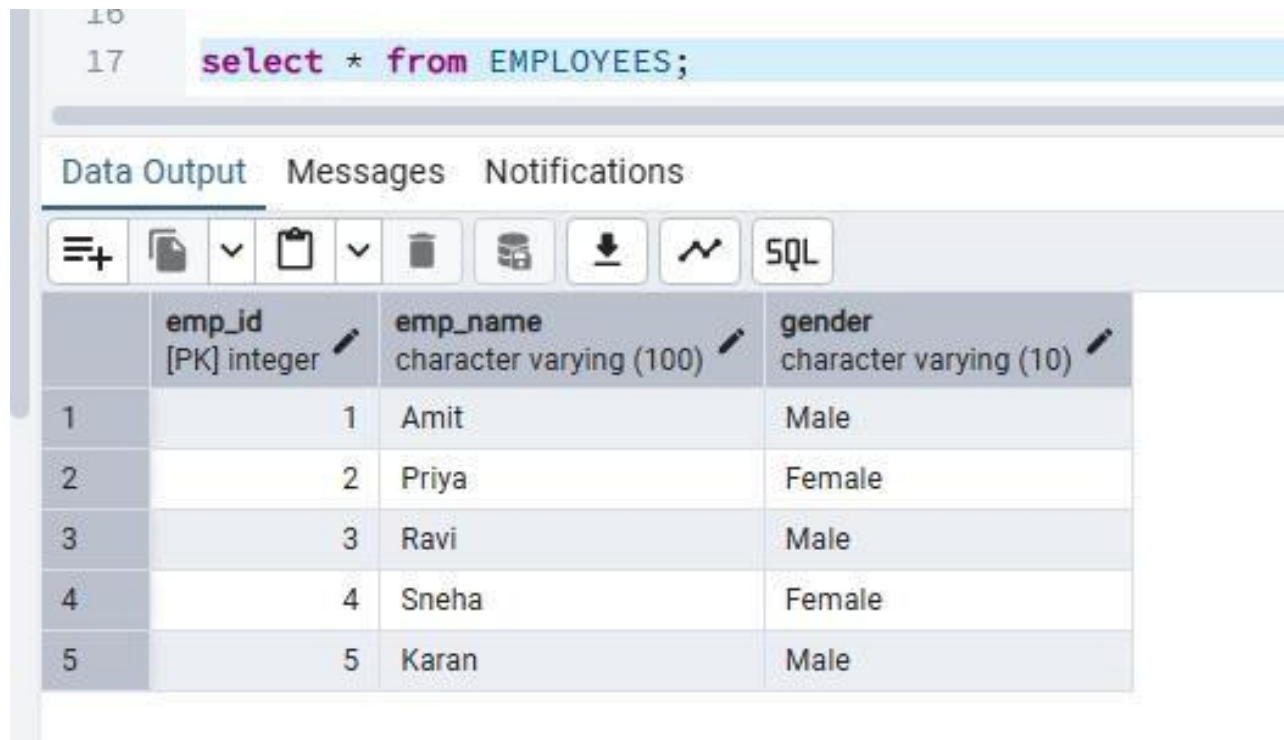
```
('Ravi', 'Male'),
```

```
('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
$$ DECL
RE
    result INT;
BEGIN
    CALL count_employees_by_gender('Male', result);
    RAISE NOTICE 'TOTAL EMPLOYEES OF GENDER Male ARE %', result;
END;
$$;
```



The screenshot shows a database IDE interface. At the top, a SQL query is entered in a text area: `select * from EMPLOYEES;`. Below the query area, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, displaying a table with 5 rows and 4 columns. The columns are labeled "emp\_id [PK] integer", "emp\_name character varying (100)", and "gender character varying (10)". The rows contain the following data:

	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.

## -----HARD PROBLEM-----

```
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()
);
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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

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); --PRODUCT SOLD SUCCESSFULLY AND  
QUANTITY\_REMAINING COLUMN SET TO -20 AND DATA LOGGED TO SALES  
TABLE

SELECT \* FROM SALES;

SELECT \* FROM PRODUCTS;

CALL PLACE\_ORDER(3,100); --INSUFFICIENT QUANTITY AVAILABLE

```

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

```

Data Output Messages Notifications

Showing rows: 1 to 1

	sale_id [PK] integer	product_id integer	quantity integer	total_price numeric (10,2)	sale_date timestamp without time zone
1	1	2	20	400000.00	2025-09-25 23:12:19.653032

```

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

--Here in above output, After selling 20 tablets (id==2) we are left with 10 and the selling data is logged into sales table.

```

102 SELECT * FROM PRODUCTS;
103 CALL PLACE_ORDER(3,100); --INSUFFICIENT QUANTITY AVAILABLE
104

```

Data Output Messages Notifications

NOTICE: Insufficient Quantity Available!  
CALL  
  
Query returned successfully in 158 msec.

ID ==3 means laptop are 20 only and we place order for 100 ...so we get notice - for insufficient quantity!!