Experiment - 6

Student Name: Naman **UID:** 23BCS11036
Branch: BE-CSE **Section/Group:** KRG_1A
Semester: 5th **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.

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.

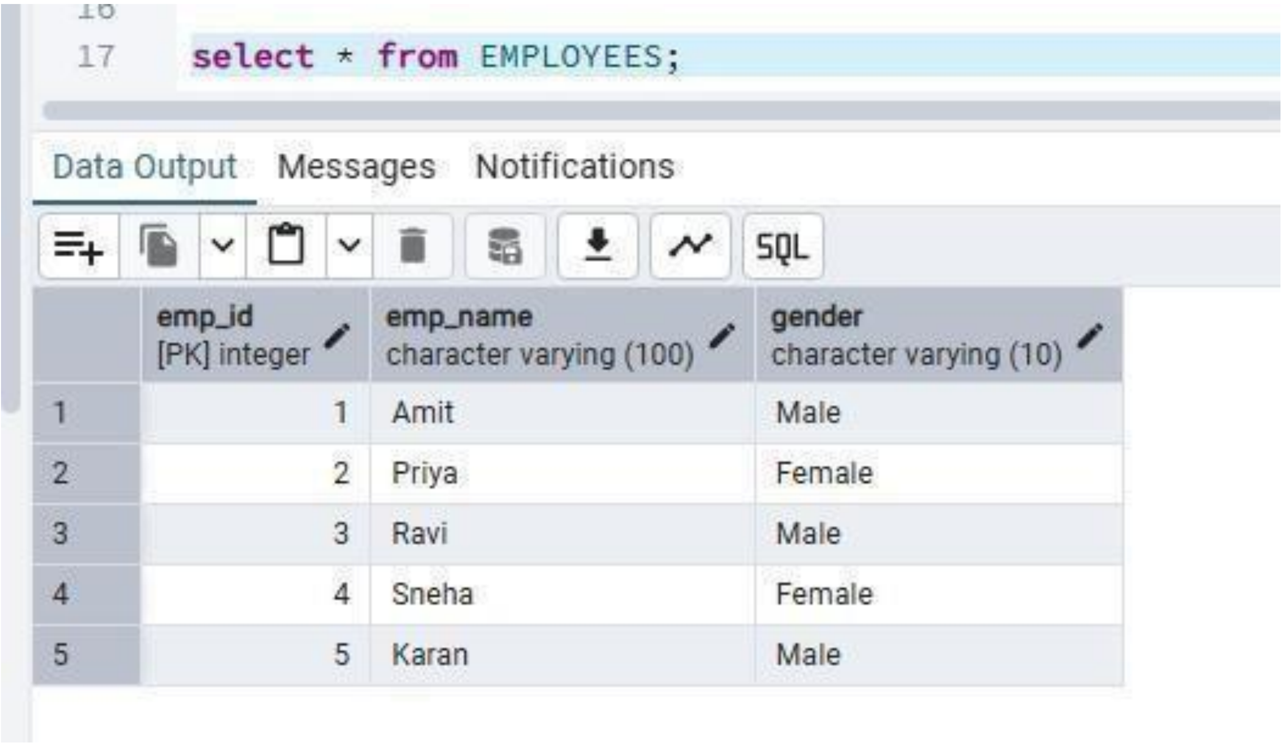
2. **Objective:** The objective is to automate critical business operations using PostgreSQL stored procedures. For HR, it tracks gender diversity by returning the total count of employees by gender. For retail, it manages orders by validating stock, logging sales, updating inventory, and providing real-time 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---

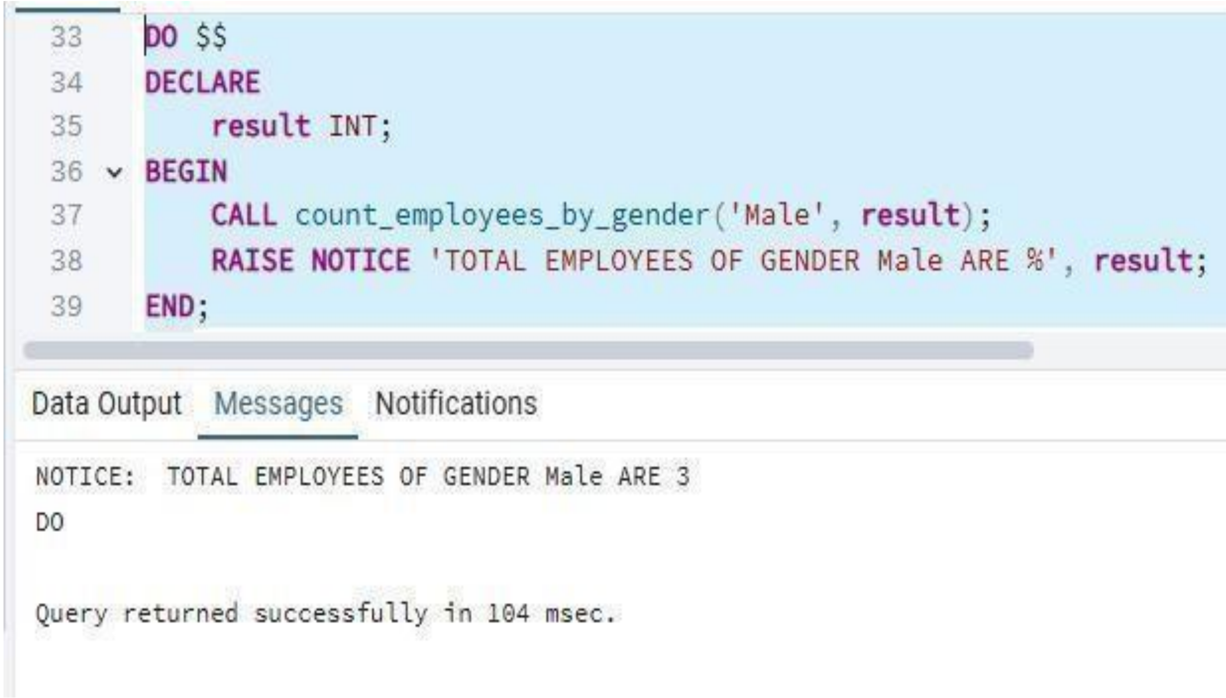
```
CREATE TABLE employees ( emp_id
   SERIAL        PRIMARY        KEY,
   emp_name VARCHAR(100), gender
   VARCHAR(10)
);

INSERT INTO employees (emp_name, gender) VALUES ('Amit', 'Male'),
('Priya', 'Female'),
('Ravi', 'Male'),
('Sneha', 'Female'),
('Karan', 'Male');
```

```
select * from EMPLOYEES;
CREATE OR REPLACE PROCEDURE
      count_employees_by_gender( IN input_gender VARCHAR, OUT
              total_count int
)
LANGUAGE plpgsql AS
$$
BEGIN
     SELECT COUNT(*) INTO total_count FROM
     employees
     WHERE gender = input_gender; END;
$$;

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



```
select * from EMPLOYEES;
```

	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



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

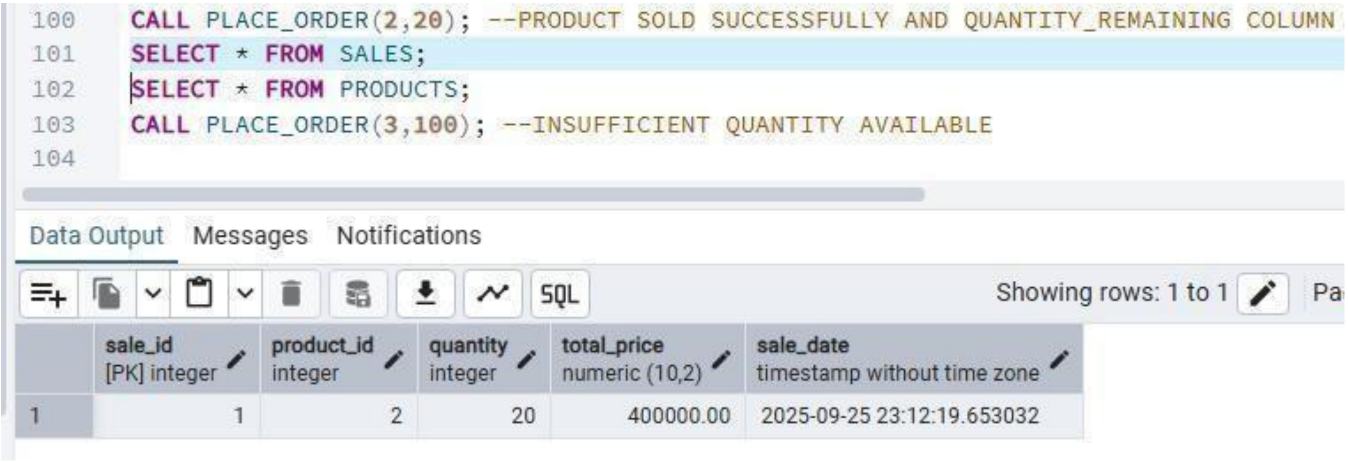
```
NOTICE:  TOTAL EMPLOYEES OF GENDER Male ARE 3
DO

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()
);


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
```

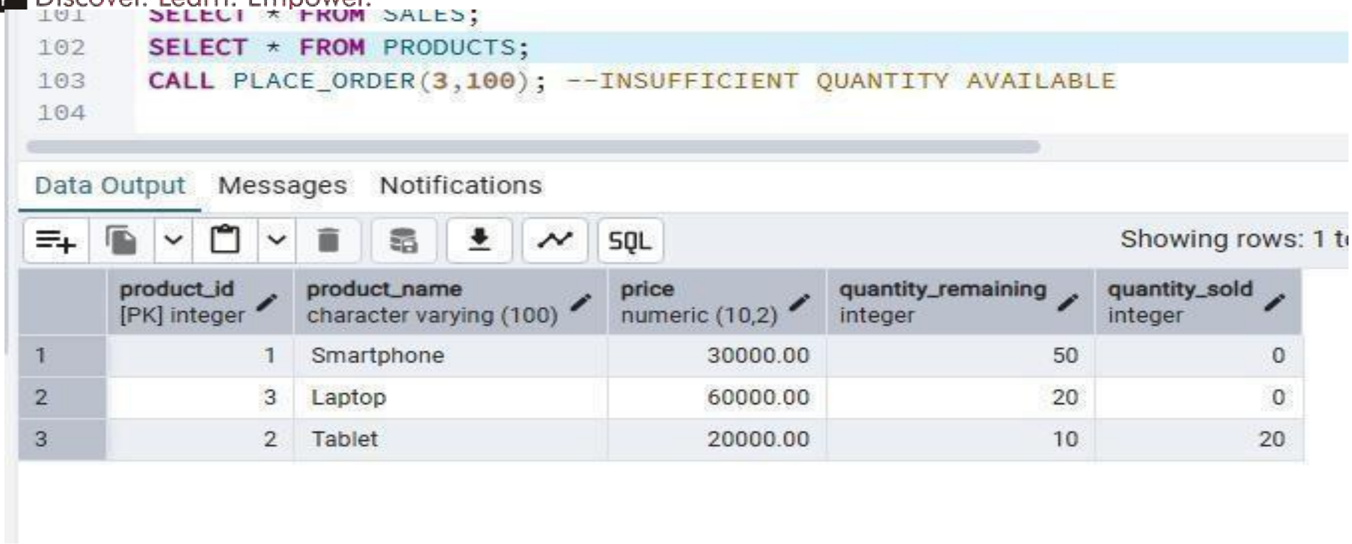
```
        RAISE NOTICE 'Product ID % does not exist!', p_product_id; ELSIF
    available_stock >= p_quantity THEN
        -- LOGGING THE ORDER
        INSERT INTO sales (product_id, quantity, total_price)
        VALUES (p_product_id, p_quantity, p_quantity * product_price);

        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
```



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

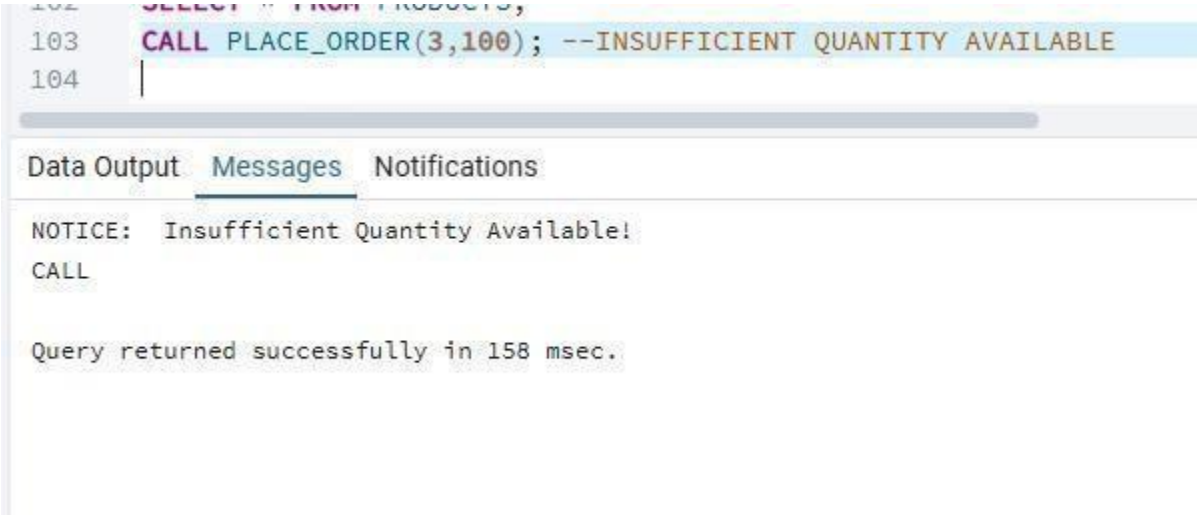
	sale_id [PK] integer	product_id integer	quantity integer	total_price numeric (10,2)	sale_date timestamp without time zone
1	1	2	20	400000.00	2025-09-25 23:12:19.653032



```
SELECT * FROM SALES;
SELECT * FROM PRODUCTS;
CALL PLACE_ORDER(3,100); --INSUFFICIENT QUANTITY AVAILABLE
```

	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.



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

```
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!!