Experiment 6

Student Name: Diksha
Branch: CSE
Semester: 5th
Subject Name: ADBMS

UID: 23BCS10994
Section/Group: KRG_2B
Date : 22/09/25
Subject Code: 23CSP-333

1. Aim:

A) Medium Level Problem:
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

B) Hard Level Problem:
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:

Medium-Level Problem: Gender Diversity Tracking

- Procedure Creation: Develop a PostgreSQL stored procedure to track gender diversity in the workforce.
- Parameterized Input: Accept gender as an input parameter (e.g., 'Male', 'Female', 'Other').
- Data Retrieval: Count the total number of employees corresponding to the input gender.
- Instant Reporting: Provide HR with real-time, secure reporting without exposing unnecessary data.
- Efficiency & Security: Ensure the procedure runs efficiently on large datasets while protecting sensitive employee information.

Hard-Level Problem: Order Placement and Inventory Management

- Automated Order Processing: Create a stored procedure to automate retail orders.
- Stock Validation: Check inventory availability before confirming an order.
- Sales Logging: Record each order in a sales table for tracking and analytics.

- Inventory Update: Update stock levels immediately after order confirmation to prevent overselling.
- Real-Time Feedback: Provide instant confirmation or rejection messages to the user.

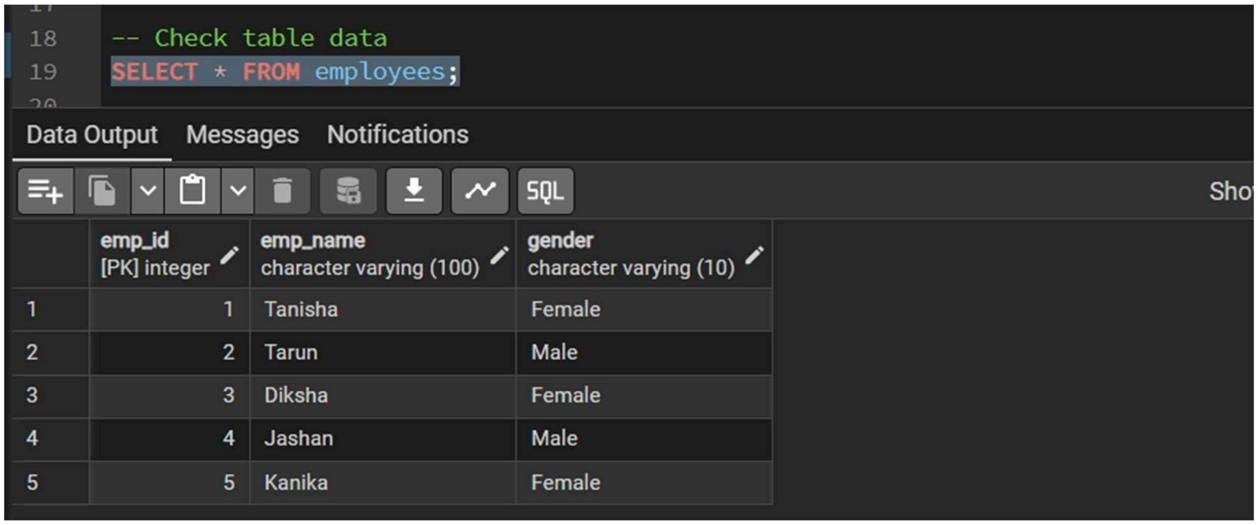
3. ADBMS script and output:

Medium Level:

```
CREATE TABLE employees (

emp_id SERIAL PRIMARY

KEY,     emp_name

VARCHAR(100),     gender

VARCHAR(10)

);

INSERT INTO employees (emp_name, gender) VALUES

('Tanisha', 'Female'),

('Tarun', 'Male'),

('Diksha', 'Female'),

('Jashan', 'Male'),

('Kanika', 'Female');

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
```

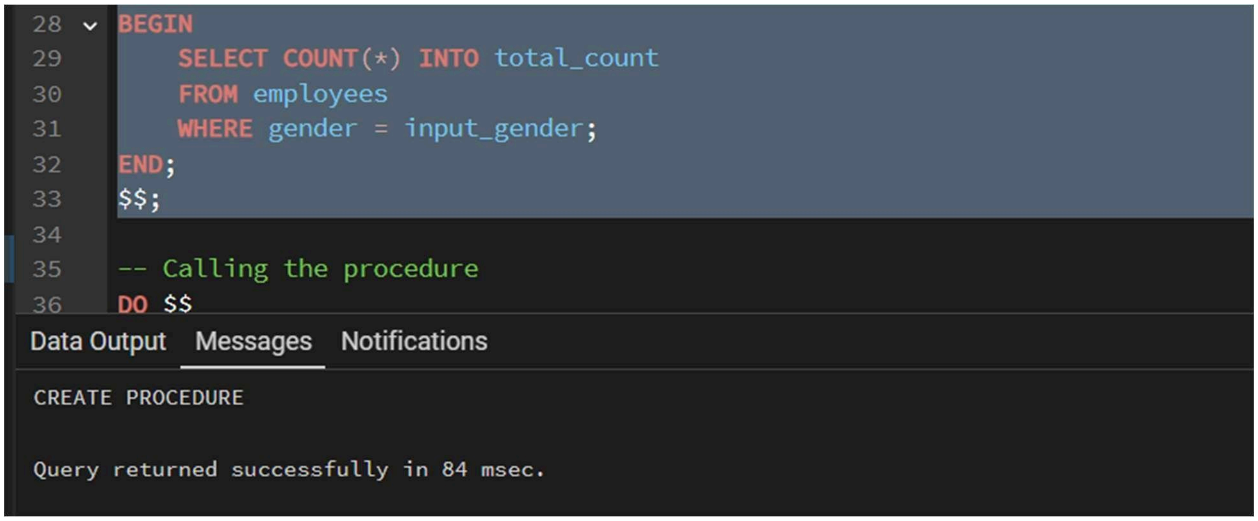
```
    FROM employees
    WHERE gender = input_gender;
END;
$$;
-- Calling the procedure
DO $$
DECLARE
    result INT;
BEGIN
    CALL count_employees_by_gender('Male', result);
    RAISE NOTICE 'Total employees of gender Male are %', result;
END;
$$;
```

Output:



```
-- Check table data
SELECT * FROM employees;
```

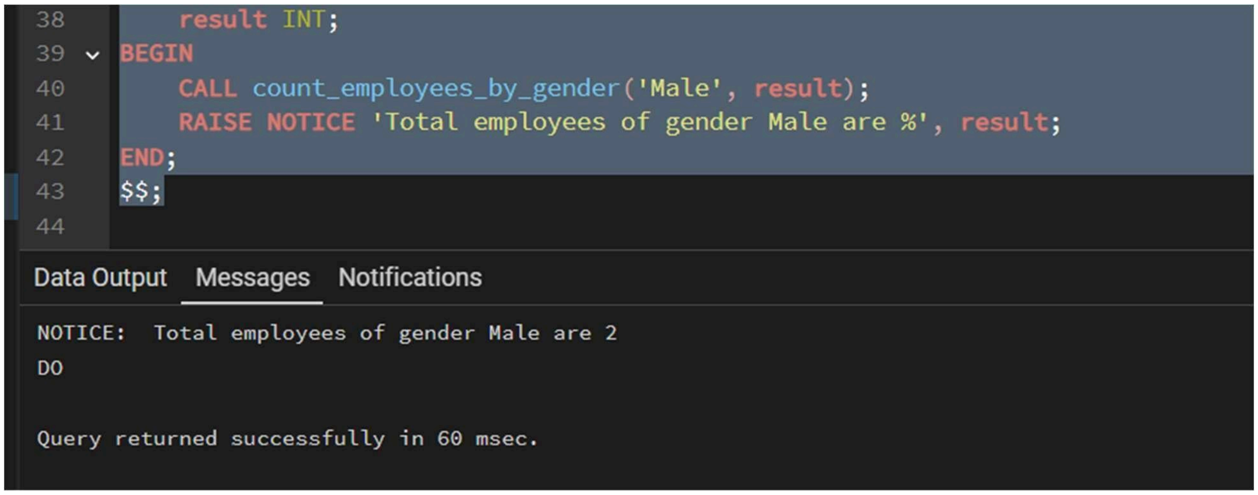
	emp_id [PK] integer	emp_name character varying (100)	gender character varying (10)
1	1	Tanisha	Female
2	2	Tarun	Male
3	3	Diksha	Female
4	4	Jashan	Male
5	5	Kanika	Female



```
BEGIN
    SELECT COUNT(*) INTO total_count
    FROM employees
    WHERE gender = input_gender;
END;
$$;

-- Calling the procedure
DO $$
```

Data Output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 84 msec.
```



```
    result INT;
BEGIN
    CALL count_employees_by_gender('Male', result);
    RAISE NOTICE 'Total employees of gender Male are %', result;
END;
$$;

```

Data Output Messages Notifications

```
NOTICE:  Total employees of gender Male are 2
DO

Query returned successfully in 60 msec.
```

Hard Level:

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

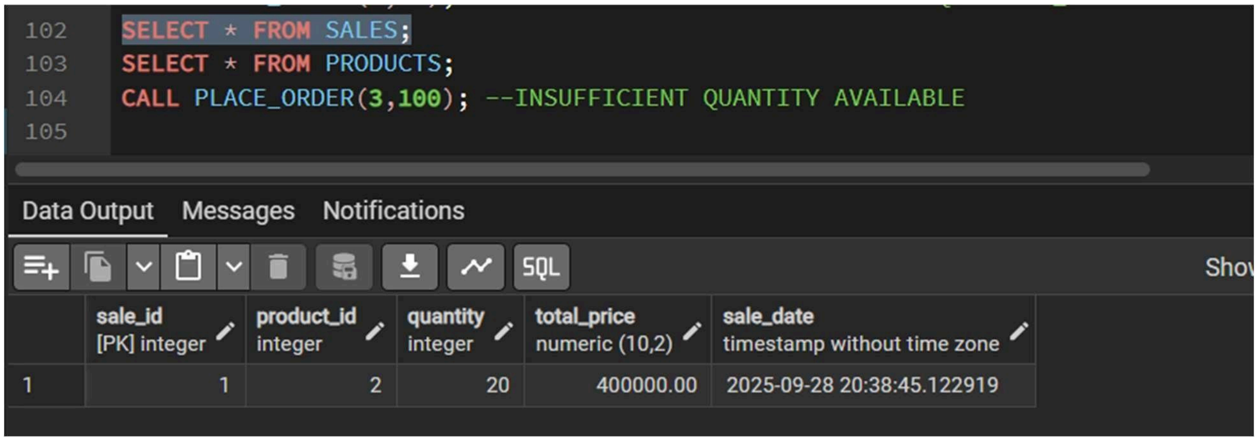
('Smartphone', 30000, 50),

('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)
```

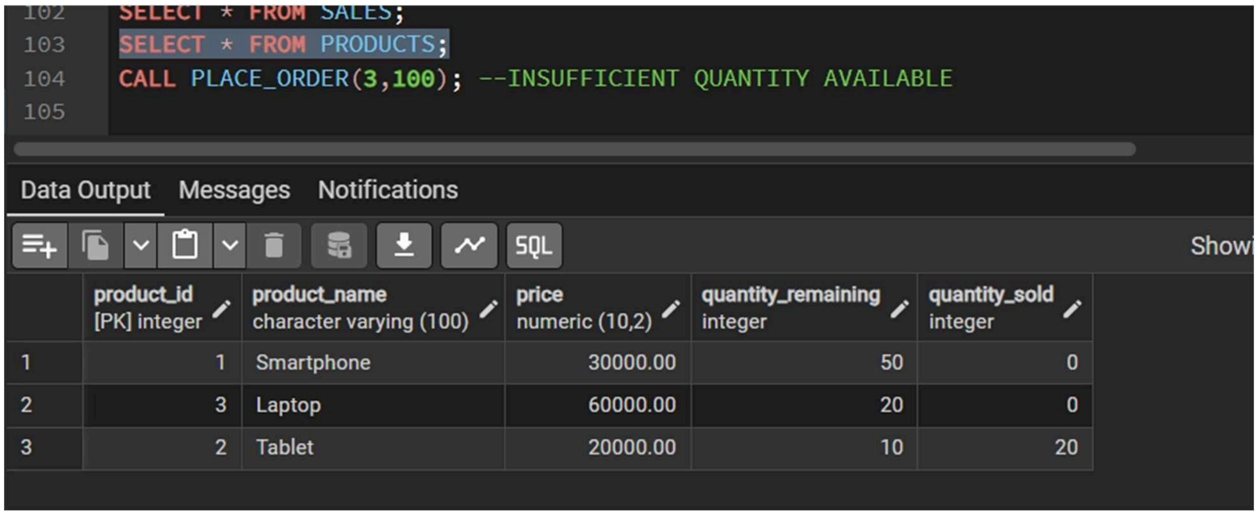
```
        VALUES (p_product_id, p_quantity, p_quantity * product_price);
UPDATE products
        SET quantity_remaining = quantity_remaining - p_quantity,
quantity_sold = quantity_sold + p_quantity
        WHERE product_id = p_product_id;

        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
```

Output:



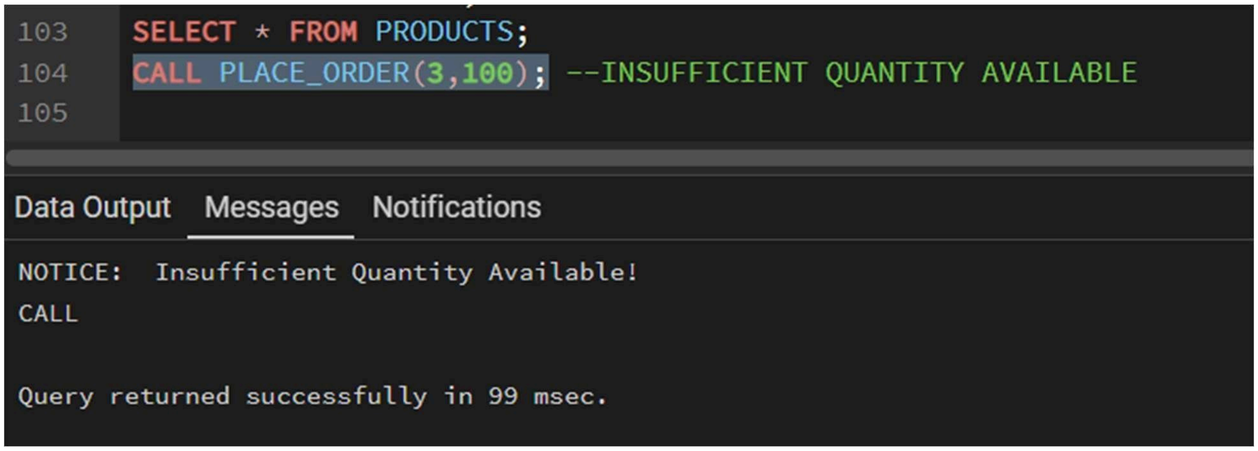
```
SELECT * FROM SALES;
SELECT * FROM PRODUCTS;
CALL PLACE_ORDER(3,100); --INSUFFICIENT QUANTITY AVAILABLE
```

	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	2025-09-28 20:38:45.122919



```
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



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

```
NOTICE:  Insufficient Quantity Available!
CALL

Query returned successfully in 99 msec.
```