



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

## EXPERIMENT - 5

<b>Name</b> : Mohit Sengar	<b>UID</b> : 23BCS10331
<b>Branch:</b> CSE	<b>Section:</b> KRG1-B
<b>Semester:</b> 5	<b>Date of Performance:</b> 23/9/2025
<b>Subject:</b> ADBMS	<b>Subject Code:</b> 23CSP-333

### Question 1. Views: Performance Benchmarking : Normal View vs. Materialized View

#### 1. Create a large dataset:

- Create a table names `transaction_data` (`id` , `value`) with 1 million records.
- take `id` 1 and 2, and for each `id`, generate 1 million records in `value` column
- Use `Generate_series ()` and `random()` to populate the data.

#### 2. Create a normal view and materialized view to for `sales_summary`, which includes `total_quantity_sold`, `total_sales`, and `total_orders` with aggregation.

#### 3. Compare the performance and execution time of both.

#### Solution :

```
DROP TABLE IF EXISTS transaction_data;
```

```
DROP VIEW IF EXISTS sales_summary;
```

```
DROP MATERIALIZED VIEW IF EXISTS sales_summary_mat;
```

```
CREATE TABLE transaction_data (
```

```
    id INT,
```

```
    value NUMERIC
```

```
);
```

```
INSERT INTO transaction_data (id, value)
```

```
SELECT id_series, ROUND(random()*1000, 2)
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
FROM generate_series(1,2) AS id_series,  
     generate_series(1,1000000) AS gs;  
CREATE VIEW sales_summary AS  
SELECT  
    id,  
    COUNT(*) AS total_orders,  
    SUM(value) AS total_sales,  
    SUM(value)/COUNT(*) AS total_quantity_sold  
FROM transaction_data  
GROUP BY id;
```

```
CREATE MATERIALIZED VIEW sales_summary_mat AS  
SELECT  
    id,  
    COUNT(*) AS total_orders,  
    SUM(value) AS total_sales,  
    SUM(value)/COUNT(*) AS total_quantity_sold  
FROM transaction_data  
GROUP BY id;
```

```
CREATE INDEX idx_sales_summary_mat_id ON sales_summary_mat(id);  
EXPLAIN ANALYZE  
SELECT * FROM sales_summary;  
EXPLAIN ANALYZE  
SELECT * FROM sales_summary_mat;
```



**NORMAL VIEW:**

### MATERIALIZED VIEW :

## Question 2. Views: Securing Data Access with Views and Role-Based Permissions

**The company TechMart Solutions stores all sales transactions in a central database. A new reporting team has been formed to analyze sales but they should not have direct access to the base tables for security reasons. The database administrator has decided to:**



1. Create restricted views to display only summarized, non-sensitive data.
2. Assign access to these views to specific users using DCL commands (GRANT, REVOKE).

**Solution:**

```
DROP TABLE IF EXISTS sales;
```

```
CREATE TABLE sales (  
    sale_id SERIAL PRIMARY KEY,  
    customer_name VARCHAR(100),  
    product_name VARCHAR(100),  
    quantity INT,  
    price NUMERIC,  
    sale_date DATE  
);
```

```
INSERT INTO sales (customer_name, product_name, quantity, price, sale_date)  
VALUES  
( 'Alice', 'Laptop', 2, 800, '2025-01-01'),  
( 'Bob', 'Mouse', 5, 20, '2025-01-02'),  
( 'Charlie', 'Keyboard', 3, 50, '2025-01-03'),  
( 'Alice', 'Laptop', 1, 800, '2025-01-04');
```

```
DROP VIEW IF EXISTS sales_summary_view;  
CREATE VIEW sales_summary_view AS  
SELECT  
    product_name,  
    SUM(quantity) AS total_quantity_sold,  
    SUM(quantity * price) AS total_sales  
FROM sales  
GROUP BY product_name;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
DROP ROLE IF EXISTS reporting_team;
```

```
CREATE ROLE reporting_team LOGIN PASSWORD 'report123';
```

```
GRANT SELECT ON sales_summary_view TO reporting_team;
```

```
REVOKE ALL ON sales FROM reporting_team;
```

```
SELECT * FROM sales_summary_view;
```

```
SELECT * FROM sales;
```

## LEARNING OUTCOMES:

1. Understand how to use stored procedures to perform multiple operations such as checking stock, inserting orders, and updating inventory in one unit.
2. Learn how to track and manage inventory by updating remaining quantity and quantity sold after each transaction.
3. Automate the process of order placement, including validating stock availability and calculating total price dynamically.
4. Develop skills in handling errors and displaying meaningful messages for both successful and failed transactions.
5. Gain hands-on practice in creating tables, writing INSERT, UPDATE, and SELECT queries, and using conditional logic in PL/pgSQL.