University Institute of Engineering

Department of Computer Science & Engineering

Experiment: 7

Student Name: Jagrath **UID:24BDA70365**

Branch: CSE **Section/Group: AIT-KRG-GP2**

Semester: 4th **Date of Performance:27/02/2026**

Subject Name: DBMS

1. **Aim of the practical:** To design and implement a materialized view and to compare and analyze execution time and performance differences between simple views, complex views, and materialized views, thereby understanding their impact on query optimization and system performance.

2. Tool Used:

- **Database Management System:**
  - **PostgreSQL**
- **Database Administration Tool:**
  - **pgAdmin**

3. Objective:

To create simple views, complex views, and materialized views, and to evaluate their performance by comparing query execution time for each, highlighting the advantages of materialized views in enterprise-level applications.

4. Practical / Experimental Steps

Step 1: Create base tables (Employee, Department, Payroll).
Step 2: Insert sample data into tables.
Step 3: Create a simple view based on a single table.
Step 4: Create a complex view involving joins and aggregation.
Step 5: Create a materialized view storing precomputed results.
Step 6: Execute queries on all views.
Step 7: Compare execution time and analyze performance.

5. I / O Analysis

A) Create Table

```
-- SandDisk (Product table)
CREATE TABLE sanddisk (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price NUMERIC(10,2)
);


-- JTG (Customer table)
CREATE TABLE jtg (
    customer_id SERIAL PRIMARY KEY,
    customer_name VARCHAR(100),
    city VARCHAR(100)
);
```

```
-- PayPal (Transaction table)
CREATE TABLE paypal (
    transaction_id SERIAL PRIMARY KEY,
    product_id INT REFERENCES sanddisk(product_id),
    customer_id INT REFERENCES jtg(customer_id),
    quantity INT,
    transaction_date DATE,
    payment_amount NUMERIC(12,2)
);
```

```
CREATE TABLE

Query returned successfully in 138 msec.
```

B) Insert Sample Data

```
-- Insert Products
INSERT INTO sanddisk (product_name, category, price)
SELECT
   'Product_' || i,
   CASE WHEN i % 2 = 0 THEN 'Storage' ELSE 'Accessories' END,
   (RANDOM()*5000)::NUMERIC(10,2)
FROM generate_series(1, 10000) AS s(i);

-- Insert Customers
INSERT INTO jtg (customer_name, city)
SELECT
   'Customer_' || i,
   CASE WHEN i % 3 = 0 THEN 'Delhi'
        WHEN i % 3 = 1 THEN 'Mumbai'
        ELSE 'Bangalore'
   END
FROM generate_series(1, 20000) AS s(i);

-- Insert Transactions (Large Dataset)
INSERT INTO paypal (product_id, customer_id, quantity, transaction_date, payment_amount)
SELECT
   (RANDOM()*9999 + 1)::INT,
   (RANDOM()*19999 + 1)::INT,
   (RANDOM()*10 + 1)::INT,
   CURRENT_DATE - (RANDOM()*365)::INT,
```

```
    (RANDOM()*10000)::NUMERIC(12,2)
FROM generate_series(1, 1000000);
```

```
INSERT 0 1000000

Query returned successfully in 18 secs 209 msec.
```

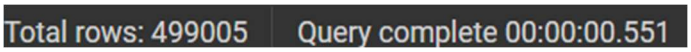
C) Create Views

```
CREATE VIEW simple_view AS
SELECT transaction_id, product_id, customer_id, payment_amount
FROM paypal;


CREATE VIEW complex_view AS
SELECT
    s.category,
    j.city,
    COUNT(p.transaction_id) AS total_transactions,
    SUM(p.payment_amount) AS total_revenue
FROM paypal p
JOIN sanddisk s ON p.product_id = s.product_id
JOIN jtg j ON p.customer_id = j.customer_id
GROUP BY s.category, j.city;


CREATE MATERIALIZED VIEW materialized_sales_summary AS
SELECT
    s.category,
    j.city,
    COUNT(p.transaction_id) AS total_transactions,
    SUM(p.payment_amount) AS total_revenue
```

```
FROM paypal p
JOIN sanddisk s ON p.product_id = s.product_id
JOIN jtg j ON p.customer_id = j.customer_id
GROUP BY s.category, j.city;
```

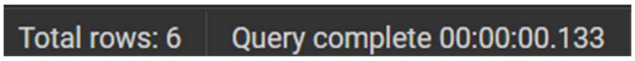
D) Compare Their Performance

```
SELECT * FROM simple_view WHERE payment_amount > 5000;
```




```
SELECT * FROM complex_view;
```


```
SELECT * FROM materialized_sales_summary;
```



6. Learning outcomes (What I have learnt):

- Understand the concept and working of materialized views in a database system.
- Differentiate between simple views, complex views, and materialized views.
- Create and refresh materialized views in PostgreSQL.
- Measure and compare query execution time for different types of views.
- Analyze performance benefits of materialized views in data-intensive applications.
- Apply materialized view concepts in real-world company scenarios such as SanDisk, JTG, and PayPal.