DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

CU CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

# Experiment - 5

**Student Name:** Priyanka

**UID:** 23BAI70303

**Branch:** BE-AIT-CSE

**Section/Group:** 23AIT_KRG-G1_A

**Semester:** 5th

**Date of Performance:** 24 Sept, 2025

**Subject Name:** ADBMS

**Subject Code:** 23CSP-333

## 1. Aim:

**MEDIUM LEVEL PROBLEM:**

Demonstrate the performance benefits of materialized views by creating a large-scale dataset of two million records in a transaction_data table, then establishing both a standard view and a materialized view for a sales summary, and finally, comparing their query execution times to prove that the materialized view, with its pre-computed results, offers superior performance.

**HARD LEVEL PROBLEM:**

To enhance data security for the TechMart Solutions reporting team by creating restricted, summary-based views on the central sales database, and then to use DCL commands (GRANT, REVOKE) to manage access, ensuring the team can analyze non-sensitive data without having direct access to the base tables.

## 2. Objective:

- Demonstrate the performance superiority of materialized views on a large-scale sales dataset (2 million records).
- Enhance data security for the reporting team by creating restricted, summary-based views on the database.
- Implement strict access controls using DCL commands (GRANT, REVOKE) to ensure the team can only access non-sensitive, summarized data.

## 3. Theory:

**Views:** Virtual tables based on a SQL query result:

- **Standard View:** A stored query executed in real-time. It does not store data on disk. Performance is dependent on the complexity of the underlying query and the size of the base tables, making it slow for complex aggregations on large datasets.
- **Materialized View:** A physical table that stores pre-computed query results. It offers a significant performance boost for read-heavy operations by eliminating the need to re-run complex queries. The data is "stale" until the view is manually refreshed.

**Data Control Language (DCL) and Security:**

- **Data Security:** Limiting user access to only necessary information is crucial.
- **Views as a Security Layer:** Views are used to filter and summarize data. They can expose only non-sensitive, aggregated information, protecting the underlying base tables from direct access.
- **Data Control Language (DCL):** A subset of SQL used to manage database permissions.

## 4. Procedure:

### Medium Level Solution:

- **Set up the environment:** First, you must execute the SQL commands to create a transaction_data table and populate it with two million records. This step simulates a large-scale dataset for a real-world scenario.
- **Create views:** Next, define and create two views on this data. The first is a standard view, which will perform its aggregation on demand. The second is a materialized view, which will pre-compute and store the results.
- **Compare performance:** Use the EXPLAIN ANALYZE command on a SELECT statement for both views. This command will provide detailed information on query execution, including the time taken. By comparing the results, you'll see a significant difference in execution time, with the materialized view being much faster due to its pre-calculated data.

### Hard Level Solution:

- **Set up the environment:** Begin by creating and populating the three base tables: customer_master, product_catalog, and sales_orders. These tables represent the raw, sensitive data.
- **Create a restricted view:** Define a new view called vW_ORDER_SUMMARY. This view will perform joins and calculations but will deliberately exclude sensitive customer information, such as phone numbers and emails, providing a sanitized summary of order data.
- **Manage user access:** Use **DCL** (Data Control Language) commands to control who can access this view. Create a new user role (e.g., 'SHRESHTA'). Then, use the GRANT command to assign SELECT (read) permissions on the vW_ORDER_SUMMARY view to this new user. This ensures the user can query the public view but has no access to the underlying, private tables.

## 5. Code:

Medium:

```
1.CREATE TABLE transaction_data (
   id INT,
   value INT
);
-- For id = 1
INSERT INTO transaction_data (id, value)
SELECT 1, random() * 1000  -- simulate transaction amounts 0-1000
FROM generate_series(1, 1000000);

-- For id = 2
INSERT INTO transaction_data (id, value)
SELECT 2, random() * 1000
FROM generate_series(1, 1000000);
SELECT *FROM transaction_data

2. CREATE TABLE Sales (
   order_id INT,
   product_id INT,
   quantity INT,
   price DECIMAL(10,2),
   order_date DATE
);

INSERT INTO Sales VALUES
(1, 101, 2, 500.00, '2025-09-01'),
(2, 102, 1, 1200.00, '2025-09-02'),
(3, 101, 5, 500.00, '2025-09-03'),
(4, 103, 3, 300.00, '2025-09-04');

CREATE VIEW sales_summary AS
SELECT
   SUM(quantity) AS total_quantity_sold,
   SUM(quantity * price) AS total_sales
```

FROM Sales;
SELECT * FROM sales_summary;

## 6. Output:

| | id integer | value integer |
|---|---|---|
| 1 | 1 | 328 |
| 2 | 1 | 777 |
| 3 | 1 | 706 |
| 4 | 1 | 139 |
| 5 | 1 | 681 |
| 6 | 1 | 810 |
| 7 | 1 | 771 |
| 8 | 1 | 122 |
| 9 | 1 | 329 |
| 10 | 1 | 755 |
| 11 | 1 | 51 |
| 12 | 1 | 55 |
| 13 | 1 | 449 |
| 14 | 1 | 64 |
| 15 | 1 | 680 |

| | id integer | value integer |
|---|---|---|
| 1 | 2 | 578 |
| 2 | 2 | 405 |
| 3 | 2 | 431 |
| 4 | 2 | 44 |
| 5 | 2 | 800 |
| 6 | 2 | 687 |
| 7 | 2 | 497 |
| 8 | 2 | 710 |
| 9 | 2 | 648 |

Total rows: 1000000 Query complete 00:00:00.425

| | total_quantity_sold bigint | total_sales numeric |
|---|---|---|
| 1 | 11 | 5600.00 |

**Normal view:**

```
Successfully run. Total query runtime: 58 msec.
1 rows affected.
```

**Materializsed View:**

```
Successfully run. Total query runtime: 99 msec.
1 rows affected.
```

## 7. Learning Outcomes:

- **Understanding the difference between standard and materialized views:** Learners will grasp that a standard view is a logical query, while a materialized view is a physical snapshot of the data.
- **Performance Optimization:** The most significant takeaway is recognizing how materialized views can drastically improve read-query performance, especially on large, static datasets. Learners will understand that by pre-computing and caching aggregated results, materialized views eliminate the need for expensive, on-the-fly calculations.