DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

CU CHANDIGARH UNIVERSITY Discover. Learn. Empower.

# Experiment - 5

**Student Name: Anshu Sharma** **UID:** 23BAI70204

**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., 'ARMAAN'). 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:

```
                    -----Experiment 5 (Medium Level Solution)-----
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



--WITH NORMAL VIEW
CREATE OR REPLACE VIEW sales_summary_view AS
SELECT id,
COUNT(*) AS total_orders,
SUM(value) AS total_sales,
AVG(value) AS avg_transaction
FROM transaction_data
GROUP BY id;

SELECT * FROM sales_summary_view;

EXPLAIN ANALYZE
SELECT * FROM sales_summary_view;
```

```sql
--WITH MATERIALIZED VIEW

CREATE MATERIALIZED VIEW sale_summary_mv AS
SELECT id,
COUNT(*) AS total_orders,
SUM(value) AS total_sales,
AVG(value) AS avg_transaction
FROM transaction_data
GROUP BY id;


SELECT * FROM sale_summary_mv;

EXPLAIN ANALYZE
SELECT * FROM sale_summary_mv;

                -----Experiment 5 (Hard Level Solution)-----

CREATE TABLE customer_master (
customer_id VARCHAR(5) PRIMARY KEY,
full_name VARCHAR(50) NOT NULL,
phone VARCHAR(15), email
VARCHAR(50), city VARCHAR(30)
);


CREATE TABLE product_catalog (
product_id VARCHAR(5) PRIMARY KEY,
product_name VARCHAR(50) NOT NULL, brand
VARCHAR(30),
unit_price NUMERIC(10,2) NOT NULL
);


CREATE TABLE sales_orders ( order_id
SERIAL PRIMARY KEY,
product_id VARCHAR(5) REFERENCES product_catalog(product_id),
quantity INT NOT NULL,
customer_id VARCHAR(5) REFERENCES customer_master(customer_id),
discount_percent NUMERIC(5,2), order_date DATE NOT NULL
);



INSERT INTO customer_master (customer_id, full_name, phone, email, city) VALUES
('C1', 'Amit Sharma', '9876543210', 'amit.sharma@example.com', 'Delhi'),
('C2', 'Priya Verma', '9876501234', 'priya.verma@example.com', 'Mumbai'),
('C3', 'Ravi Kumar', '9988776655', 'ravi.kumar@example.com', 'Bangalore'),
('C4', 'Neha Singh', '9123456789', 'neha.singh@example.com', 'Kolkata'),
('C5', 'Arjun Mehta', '9812345678', 'arjun.mehta@example.com', 'Hyderabad'),
('C6', 'Sneha Reddy', '9090909090', 'sneha.reddy@example.com', 'Chennai'),
('C7', 'Vikram Das', '9123412345', 'vikram.das@example.com', 'Pune'),
('C8', 'Rohit Gupta', '9000000001', 'rohit.gupta@example.com', 'Lucknow'),
('C9', 'Pooja Nair', '9898989898', 'pooja.nair@example.com', 'Kochi'),
('C10', 'Ankit Yadav', '9345678901', 'ankit.yadav@example.com', 'Ahmedabad');
```

```
INSERT INTO product_catalog (product_id, product_name, brand, unit_price)
VALUES
('P1', 'Smartphone X100', 'Samsung', 25000.00),
('P2', 'Laptop Pro 15', 'Dell', 65000.00),
('P3', 'Wireless Earbuds', 'Sony', 5000.00),
('P4', 'Smartwatch Fit', 'Apple', 30000.00),
('P5', 'Tablet 10.5', 'Lenovo', 22000.00),
('P6', 'Gaming Console', 'Sony', 45000.00),
('P7', 'Bluetooth Speaker', 'JBL', 7000.00),
('P8', 'Digital Camera', 'Canon', 55000.00),
('P9', 'LED TV 55 inch', 'LG', 60000.00),
('P10', 'Power Bank 20000mAh', 'Mi', 2500.00);
```

```
INSERT INTO sales_orders (product_id, quantity, customer_id, discount_percent,
order_date) VALUES
('P1', 2, 'C1', 5.00, '2025-09-01'),
('P2', 1, 'C2', 10.00, '2025-09-02'),
('P3', 3, 'C3', 0.00, '2025-09-03'),
('P4', 1, 'C4', 8.00, '2025-09-04'),
('P5', 2, 'C5', 5.00, '2025-09-05'),
('P6', 1, 'C1', 12.00, '2025-09-06'),
('P7', 2, 'C2', 0.00, '2025-09-07'),
('P8', 1, 'C3', 10.00, '2025-09-08'),
('P9', 1, 'C6', 15.00, '2025-09-09'),
('P10', 4, 'C7', 0.00, '2025-09-10'),
('P1', 1, 'C8', 5.00, '2025-09-11'),
('P2', 2, 'C9', 10.00, '2025-09-12'),
('P3', 2, 'C10', 0.00, '2025-09-13'),
('P4', 1, 'C5', 8.00, '2025-09-14'),
('P5', 3, 'C6', 5.00, '2025-09-15'),
('P6', 1, 'C7', 12.00, '2025-09-16'),
('P7', 2, 'C8', 0.00, '2025-09-17'),
('P8', 1, 'C9', 10.00, '2025-09-18'),
('P9', 1, 'C10', 15.00, '2025-09-19'),
('P10', 5, 'C4', 0.00, '2025-09-20');
```

```
CREATE VIEW vW_ORDER_SUMMARY
AS
SELECT
O.order_id,
O.order_date,
P.product_name,
C.full_name,
(P.unit_price * O.quantity) - ((P.unit_price * O.quantity) * O.discount_percent
/ 100) AS final_cost
FROM customer_master AS C
JOIN sales_orders AS O
ON O.customer_id = C.customer_id
JOIN product_catalog AS P
ON P.product_id = O.product_id;
```

```
SELECT * FROM vW_ORDER_SUMMARY;
```

```
CREATE ROLE ARMAAN
LOGIN
PASSWORD 'armaan';
GRANT SELECT ON vW_ORDER_SUMMARY TO ARMAAN;
REVOKE SELECT ON vW_ORDER_SUMMARY FROM ARMAAN;
```

## 6. Output:

Data Output | Messages | Notifications

| | id integer | total_orders bigint | total_sales bigint | avg_transaction numeric |
|---|---|---|---|---|
| 1 | 1 | 1000000 | 499938967 | 499.9389670000000000 |
| 2 | 2 | 1000000 | 499858812 | 499.8588120000000000 |

Data Output | Messages | Notifications

| | id integer | total_orders bigint | total_sales bigint | avg_transaction numeric |
|---|---|---|---|---|
| 1 | 1 | 3000000 | 1499968727 | 499.9895756666666667 |
| 2 | 2 | 3000000 | 1500810967 | 500.2703223333333333 |

Data Output Messages Notifications

| | QUERY PLAN text |
|---|---|
| 1 | HashAggregate (cost=55.20..57.70 rows=200 width=52) (actual time=0.011..0.011 rows=0 loops=1) |
| 2 | Group Key: transaction_data.id |
| 3 | Batches: 1 Memory Usage: 40kB |
| 4 | -> Seq Scan on transaction_data (cost=0.00..32.60 rows=2260 width=8) (actual time=0.007..0.007 rows=0 loop... |
| 5 | Planning Time: 0.297 ms |
| 6 | Execution Time: 0.054 ms |

Data Output Messages Notifications

| | QUERY PLAN text |
|---|---|
| 1 | Seq Scan on sales_summary_mv (cost=0.00..20.20 rows=1020 width=52) (actual time=0.011..0.011 rows=0 loops=... |
| 2 | Planning Time: 0.189 ms |
| 3 | Execution Time: 0.024 ms |

Data Output Messages Notifications

| | order_id integer | order_date date | product_name character varying (50) | full_name character varying (50) | final_cost numeric |
|---|---|---|---|---|---|
| 1 | 1 | 2025-09-01 | Smartphone X100 | Amit Sharma | 47500.0000000000000000 |
| 2 | 2 | 2025-09-02 | Laptop Pro 15 | Priya Verma | 58500.0000000000000000 |
| 3 | 3 | 2025-09-03 | Wireless Earbuds | Ravi Kumar | 15000.0000000000000000000 |
| 4 | 4 | 2025-09-04 | Smartwatch Fit | Neha Singh | 27600.0000000000000000 |
| 5 | 5 | 2025-09-05 | Tablet 10.5 | Arjun Mehta | 41800.0000000000000000 |
| 6 | 6 | 2025-09-06 | Gaming Console | Amit Sharma | 39600.0000000000000000 |
| 7 | 7 | 2025-09-07 | Bluetooth Speaker | Priya Verma | 14000.0000000000000000000 |
| 8 | 8 | 2025-09-08 | Digital Camera | Ravi Kumar | 49500.0000000000000000 |
| 9 | 9 | 2025-09-09 | LED TV 55 inch | Sneha Reddy | 51000.0000000000000000 |
| 10 | 10 | 2025-09-10 | Power Bank 20000mAh | Vikram Das | 10000.0000000000000000000 |
| 11 | 11 | 2025-09-11 | Smartphone X100 | Rohit Gupta | 23750.0000000000000000 |
| 12 | 12 | 2025-09-12 | Laptop Pro 15 | Pooja Nair | 117000.000000000000 |
| 13 | 13 | 2025-09-13 | Wireless Earbuds | Ankit Yadav | 10000.0000000000000000000 |

## 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.

- **Database Scalability:** This experiment teaches a practical approach to building scalable database solutions. Learners will understand when to use a materialized view to offload computational work from a busy live database to a more efficient pre-calculated state
- **Data Security and Abstraction:** Learners will understand how to use views as a crucial security layer. They will learn how to abstract away sensitive data and expose only nonconfidential, aggregated information to specific user groups.
- **Implementing DCL for Access Control:** The experiment provides hands-on experience with Data Control Language (DCL) commands (GRANT, REVOKE). This teaches learners how to precisely manage user permissions on specific database objects, ensuring a robust security model.
- **Role-Based Access Control (RBAC):** By creating a user role (like 'ARMAAN') and granting permissions to the role rather than the individual, learners will be introduced to the concept of role-based access control, a standard security practice that simplifies user management in a professional database environment.