

EXPERIMENT - 5

STUDENT NAME: ASHLIN JAMES

UID: 23BAI70722

BRANCH: BE-AIT-CSE

SECTION/GROUP: 23AIT_KRG-G1_A

SEMESTER: 5

DATE OF PERFORMANCE: 24 SEPT, 2025

SUBJECT NAME: ADBMS

SUBJECT CODE: 23CSP-333

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.

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.

THEORY:

Views: Logical representations of data derived from SQL queries:

- **Regular View:** A virtual table created from a stored SQL query. It doesn't hold data itself but dynamically fetches results whenever accessed. The performance depends on the query's complexity and the volume of underlying tables, which may lead to delays with heavy joins or aggregations.
- **Materialized View:** A database object that physically stores the query output. It greatly improves query performance, especially for analytical or reporting workloads, since results are pre-computed. However, the data may become outdated and requires manual or scheduled refresh to stay synchronized with source tables.

Data Control Language (DCL) and Database Security:

- **Access Control:** Restricting user privileges ensures that only authorized individuals can view or modify sensitive information.
- **Views as a Security Mechanism:** By defining views, administrators can provide customized perspectives of the data, exposing only necessary or masked fields, and preventing direct access to confidential records in base tables.
- **Data Control Language (DCL):** A category of SQL commands (such as GRANT and REVOKE) that govern permissions and roles, ensuring controlled and secure database usage.

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:

Setup 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., 'ASHLIN'). 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.

CODE:

--Medium Level Problem

```
CREATE TABLE transaction_data (  
  id INT,  
  value INT  
);  
  
INSERT INTO transaction_data (id, value)  
SELECT 1, (random() * 1000)::INT  
FROM generate_series(1, 500000);  
  
INSERT INTO transaction_data (id, value)  
SELECT 2, (random() * 1000)::INT  
FROM generate_series(1, 500000);  
  
SELECT * FROM transaction_data LIMIT 10;  
  
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;  
  
CREATE MATERIALIZED VIEW sales_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 sales_summary_mv;
```

```
EXPLAIN ANALYZE  
SELECT * FROM sales_summary_mv;
```

```
INSERT INTO transaction_data (id, value)  
SELECT 1, (random() * 1000)::INT  
FROM generate_series(1, 2500000);
```

```
INSERT INTO transaction_data (id, value)  
SELECT 2, (random() * 1000)::INT  
FROM generate_series(1, 2500000);
```

```
REFRESH MATERIALIZED VIEW sales_summary_mv;
```

```
SELECT * FROM sales_summary_view;
```

```
SELECT * FROM sales_summary_mv;
```

--Hard Level Problem

```
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', 'Karan Patel', '9876001122', 'karan.patel@example.com', 'Jaipur'),  
( 'C2', 'Meera Joshi', '9812233445', 'meera.joshi@example.com', 'Chandigarh'),  
( 'C3', 'Suresh Rao', '9922003344', 'suresh.rao@example.com', 'Mysore'),  
( 'C4', 'Alok Deshmukh', '9765432199', 'alok.deshmukh@example.com', 'Nagpur'),  
( 'C5', 'Divya Kapoor', '9090887766', 'divya.kapoor@example.com', 'Indore'),  
( 'C6', 'Rahul Bansal', '9123450098', 'rahul.bansal@example.com', 'Bhopal'),  
( 'C7', 'Shruti Nanda', '9345098761', 'shruti.nanda@example.com', 'Surat'),  
( 'C8', 'Mohit Arora', '9800765432', 'mohit.arora@example.com', 'Noida'),  
( 'C9', 'Nidhi Jain', '9012345677', 'nidhi.jain@example.com', 'Amritsar'),  
( 'C10', 'Aditya Malhotra', '9998877665', 'aditya.malhotra@example.com', 'Patna');
```

```
INSERT INTO product_catalog (product_id, product_name, brand, unit_price)  
VALUES  
( 'P1', 'Smartphone X100', 'X-Tech', 47500.00),  
( 'P2', 'Laptop Pro 15', 'ZenBook', 58500.00),  
( 'P3', 'Wireless Earbuds', 'AudioMax', 20000.00),  
( 'P4', 'Smartwatch Fit', 'WearableCo', 27600.00),  
( 'P5', 'Tablet 10.5', 'TabCorp', 41800.00),  
( 'P6', 'Gaming Console', 'GameON', 39600.00),  
( 'P7', 'Bluetooth Speaker', 'SoundWave', 14000.00),  
( 'P8', 'Digital Camera', 'PhotoPro', 49500.00),  
( 'P9', 'LED TV 55 inch', 'VisionX', 51000.00),  
( 'P10', 'Power Bank 20000mAh', 'ChargeFast', 10000.00);
```

```
INSERT INTO sales_orders (product_id, quantity, customer_id, discount_percent, order_date)  
VALUES  
( 'P1', 1, 'C1', 0.00, '2025-09-01'),  
( 'P2', 1, 'C2', 0.00, '2025-09-02'),  
( 'P3', 1, 'C3', 25.00, '2025-09-03'),  
( 'P4', 1, 'C4', 0.00, '2025-09-04'),  
( 'P5', 1, 'C5', 0.00, '2025-09-05'),  
( 'P6', 1, 'C1', 0.00, '2025-09-06'),  
( 'P7', 1, 'C2', 0.00, '2025-09-07'),  
( 'P8', 1, 'C3', 0.00, '2025-09-08'),  
( 'P9', 1, 'C6', 0.00, '2025-09-09'),  
( 'P10', 1, 'C7', 0.00, '2025-09-10'),  
( 'P1', 1, 'C8', 50.00, '2025-09-11'),  
( 'P2', 2, 'C9', 0.00, '2025-09-12'),  
( 'P3', 1, 'C10', 50.00, '2025-09-13');
```

```
CREATE OR REPLACE 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.quanyity) * 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 ASHLIN
LOGIN
PASSWORD 'As@123';


GRANT SELECT ON vW_ORDER_SUMMARY TO ASHLIN;

REVOKE SELECT ON vW_ORDER_SUMMARY FROM ASHLIN;
```

OUTPUT:

Data Output					Messages	Notifications
<div><div>≡+</div><div><div>▼</div></div><div><div>▼</div></div><div></div><div></div><div></div><div></div><div>SQL</div></div>						
	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
<div><div>≡+</div><div><div>▼</div></div><div><div>▼</div></div><div></div><div></div><div></div><div></div><div>SQL</div></div>						
	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
<div><div>≡+</div><div><div>▼</div></div><div><div>▼</div></div><div></div><div></div><div></div><div></div><div>SQL</div></div>						
	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=1)					
5	Planning Time: 0.297 ms					
6	Execution Time: 0.054 ms					

Data Output					Messages	Notifications
<div><div>≡+</div><div><div>▼</div></div><div><div>▼</div></div><div></div><div></div><div></div><div></div><div>SQL</div></div>						
	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=1)					
2	Planning Time: 0.189 ms					
3	Execution Time: 0.024 ms					

	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	47500.000000000000000000
2	2	2025-09-02	Laptop Pro 15	Priya Verma	58500.000000000000000000
3	3	2025-09-03	Wireless Earbuds	Ravi Kumar	15000.000000000000000000
4	4	2025-09-04	Smartwatch Fit	Neha Singh	27600.000000000000000000
5	5	2025-09-05	Tablet 10.5	Arjun Mehta	41800.000000000000000000
6	6	2025-09-06	Gaming Console	Amit Sharma	39600.000000000000000000
7	7	2025-09-07	Bluetooth Speaker	Priya Verma	14000.000000000000000000
8	8	2025-09-08	Digital Camera	Ravi Kumar	49500.000000000000000000
9	9	2025-09-09	LED TV 55 inch	Sneha Reddy	51000.000000000000000000
10	10	2025-09-10	Power Bank 20000mAh	Vikram Das	10000.000000000000000000
11	11	2025-09-11	Smartphone X100	Rohit Gupta	23750.000000000000000000
12	12	2025-09-12	Laptop Pro 15	Pooja Nair	117000.000000000000000000

LEARNING OUTCOMES:

- o Understanding Standard vs. Materialized Views:
Learners will clearly differentiate between a standard view, which is a dynamic logical query, and a materialized view, which is a stored snapshot of pre-computed data.
- o Performance Optimization:
The key take away is how materialized views significantly enhance query performance, especially for large or relatively static datasets. By caching aggregated results, they reduce the overhead of repeated expensive calculations.
- o Database Scalability:
The experiment highlights how materialized views contribute to building scalable database systems. Learners will see when to leverage them to offload heavy computations from live queries to efficient pre-processed structures.
- o Data Security and Abstraction:
Views act as a protective security layer. Learners will understand how to abstract sensitive details and present only summarized or non-sensitive data to selected users, ensuring confidentiality.
- o Implementing DCL for Access Control:
Hands-on practice with GRANT and REVOKE will teach learners how to manage permissions effectively, creating a strong foundation for database security.
- o Role-Based Access Control (RBAC):
By creating a role (such as ASHLIN) and assigning privileges to it instead of directly to individual users, learners will be introduced to RBAC, a professional security standard that simplifies permission management.