

University Institute of Engineering

Department of Computer Science & Engineering

EXPERIMENT: 5

NAME: AAYUSH AMAN UID: 23BCS14046

BRANCH: BE-CSE SECTION / GROUP: KRG 3B

SEMESTER: 5TH SUBJECT CODE: 23CSP-339

SUBJECT NAME: ADBMS

1. Aim of the practical:

Performance Benchmarking: Normal View vs. Materialized View [MEDIUM]

• Create a large dataset:

- o Create a table names transaction data (id, value) with 1 million records.
- o take id 1 and 2, and for each id, generate 1 million records in value column
- O Use Generate series () and random() to populate the data.
- Create a normal view and materialized view to for sales_summary, which includes total quantity sold, total sales, and total orders with aggregation.
- Compare the performance and execution time of both.

Views: Securing Data Access with Views and Role-Based Permissions [HARD]

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).
- 2. Tools Used: PgAdmin, PostgresSQL

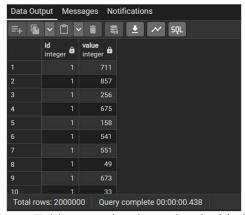
3. Code:

```
-- Database: krg_3b
-- Medium
CREATE TABLE transaction_data (
    id INT,
    value INT
);
```

-- For id = 1
INSERT INTO transaction_data (id, value)
SELECT 1, random() * 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



Output: Table created and populated with 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;

EXPLAIN ANALYZE SELECT * FROM sales_summary_view;

1	Finalize GroupAggregate (cost=26516.7226517.37 rows=2 width=52) (actual time=173.561180.059 rows=2.00 loops=1)
2	Group Key: transaction_data.id
3	Buffers: shared hit=8864
4	-> Gather Merge (cost=26516.7226517.30 rows=5 width=52) (actual time=173.519180.015 rows=6.00 loops=1)
5	Workers Planned: 2
6	Workers Launched: 2
7	Buffers: shared hit=8864
8	-> Sort (cost=25516.6925516.70 rows=2 width=52) (actual time=151.276151.277 rows=2.00 loops=3)
9	Sort Key: transaction_data.id
10	Sort Method: quicksort Memory: 25kB
11	Buffers: shared hit=8864
12	Worker 0: Sort Method: quicksort Memory: 25kB
13	Worker 1: Sort Method: quicksort Memory: 25kB
14	-> Partial HashAggregate (cost=25516.6625516.68 rows=2 width=52) (actual time=151.258151.259 rows=2.00 loops=3)
15	Group Key: transaction_data.id
16	Batches: 1 Memory Usage: 32kB
17	Buffers: shared hit=8850
18	Worker 0: Batches: 1 Memory Usage: 32kB
19	Worker 1: Batches: 1 Memory Usage: 32kB
20	-> Parallel Seq Scan on transaction_data (cost=0.0017183.33 rows=833333 width=8) (actual time=0.00832.261 rows=666666.67 loo
21	Buffers: shared hit=8850
Total	rows: 25 Query complete 00:00:00.207

Output: Result with Normal View

```
--WITH MATERIALIZED 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;
```

EXPLAIN ANALYZE SELECT * FROM sales_summary_mv;

Data	Output Messages Notifications	
=+	<u> </u>	Sho
	QUERY PLAN text	•
1	Seq Scan on sales_summary_mv (cost=0.0020.20 rows=1020 width=52) (actual time=0.0120.013 rows=2.00 loo	
2	Buffers: shared hit=1	
3	Planning:	
4	Buffers: shared hit=13	
5	Planning Time: 0.666 ms	
6	Execution Time: 0.023 ms	
Total	l rows: 6 Query complete 00:00:00.052	

Output: Results with Materialized View

-- Hard

```
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;

--1. CREATE USER

CREATE ROLE Aman

LOGIN

PASSWORD 'Aman123';

GRANT SELECT ON vW ORDER SUMMARY TO Aman;

REVOKE SELECT ON vW ORDER SUMMARY FROM Aman;

4. Learning Outcomes:

- 1. Materialized views improve query performance by storing precomputed results, especially useful for large datasets.
- 2. Normal views always reflect live data, while materialized views require manual refresh to stay updated.
- 3. EXPLAIN ANALYZE helps compare execution efficiency between views and materialized views.
- 4. Materialized views are ideal for reporting and dashboards where speed matters more than real-time accuracy.
- 5. PostgreSQL functions like generate_series() and random() are effective for simulating large-scale test data.