Experiment -5

Student Name: Manjot Singh
UID: 23BCS12549
Branch: BE-CSE
Section/Group: KRG_2B
Semester: 5th
Date of Performance: 22/9/25
Subject Name: ADBMS
Subject Code: 23CSP-333

1. Problem Description/Aim:

Problem 1: Generate 1 million records per ID in 'transaction_data' using generate_series() and random() ,create a normal view and a materialized view 'sales_summary' with aggregated metric, (total_quantity_sold , total_sales, total_orders) , and compare their performance and execution time.

Procedure (Step-by-Step):

1. Create a large dataset:
   - Create a table names transaction_data (id , value) with 1 million records.
   take id 1 and 2, and for each id, generate 1 million records in value column.
   - Use Generate_series () and random() to populate the data.
2. Create a normal view and materialized view to for sales_summary, which includes total_quantity_sold, total_sales, and total_orders with aggregation.
3. Compare the performance and execution time of both.

Sample Output Description:

The transaction_data table has 2 million rows (1 million per ID) with random values. The normal view sales_summary computes aggregates on the fly, while the materialized view sales_summary_mv stores precomputed results. Queries on the materialized view are much faster, but it needs refreshing when data changes, whereas the normal view always shows up-to-date results.

Problem 2 : Create restricted views in the sales database to provide summarized, non sensitive data to the reporting team, and control access using DCL commands(GRANT and REVOKE).

Procedure (Step-by-Step):

1. Create restricted views-
   - Define views that show only **aggregated sales data** (e.g., total_sales, total_orders) without exposing sensitive columns like customer details or payment info.
2. Assign access to reporting team(or client)-
   -Use "GRANT SELECT ON view_name TO reporting_user; " to give access.

3. Revoke access if needed.
-Use "REVOKE SELECT ON view_name FROM reporting_user;" to remove access.
4. Verify access
- Reporting users can query the view but cannot access base tables directly, ensuring security.

Sample Output Description:

The result shows the restricted view providing summarized sales data only like
- Columns shown are - product_id,total_quantity_sold, total_sales, total_orders - Columns hidden are - Customer names, addresses, payment details

A reporting user querying the view sees something like :
- Product 101 - 5000 units sold, total sales Rs. 12,50,000,500 orders.
- Product 102 - 3200 units sold, total sales Rs. 8,60,000,320 orders.

When the user tries to query the base "sales_transactions" table directly, access is denied, enforcing security.

2. **Objective:** To design and implement secure, efficient data access mechanisms by creating large-scale transaction datasets, summarizing them through normal and materialized views for performance comparison, and enforcing restricted access to sensitive data using views and DCL commands.

3. SQL QUERY AND OUTPUTS -

-- PROBLEM 1

```
Create table TRANSACTION_DATA(id int,val decimal);
INSERT INTO TRANSACTION_DATA(ID,VAL)
SELECT 1,RANDOM()
FROM GENERATE_SERIES(1,1000000);

INSERT INTO TRANSACTION_DATA(ID,VAL)
SELECT 2,RANDOM()
FROM GENERATE_SERIES(1,1000000);
SELECT * FROM TRANSACTION_DATA;
CREATE or REPLACE VIEW SALES_SUMMARY AS
SELECT
ID,
```

```
COUNT(*) AS total_quantity_sold,
sum(val) AS total_sales,
count(distinct id) AS total_orders
FROM TRANSACTION_DATA GROUP
BY ID;

EXPLAIN ANALYZE
SELECT * FROM SALES_SUMMARY;

CREATE MATERIALIZED VIEW SALES_SUMM AS
SELECT
ID,
COUNT(*) AS total_quantity_sold,
sum(val) AS total_sales,
count(distinct id) AS total_orders
FROM TRANSACTION_DATA GROUP
BY ID;

EXPLAIN ANALYZE
SELECT * FROM SALES_SUMM;
```

-- PROBLEM 2

```
CREATE TABLE customer_data (
  transaction_id SERIAL PRIMARY KEY,
  customer_name VARCHAR(100),
  email VARCHAR(100),
  phone VARCHAR(15),
  payment_info VARCHAR(50),
  order_value DECIMAL,
  order_date DATE DEFAULT CURRENT_DATE
);

INSERT INTO customer_data (customer_name, email, phone, payment_info, order_value)
VALUES
('M', 'M@example.com', '9131094977', '1234-5678-9012-3456', 500),
('A', 'A@example.com', '9931094977', '1234-5678-9012-3456', 234),
```

```
('B', 'B@example.com', '9263444151', '9876-5432-1098-7654', 754),
('C', 'C@example.com', '9263444151', '9876-5432-1098-7654', 300);

CREATE OR REPLACE VIEW restricted_sales_data AS
SELECT
  customer_name,
  COUNT(*) AS total_orders,
  SUM(order_value) AS total_sales
FROM customer_data
GROUP BY customer_name;

SELECT * FROM restricted_sales_data;

CREATE USER client1 WITH PASSWORD 'password123';

GRANT SELECT ON restricted_sales_data TO client1;
REVOKE SELECT ON restricted_sales_data FROM client1;
```

Output:

Output 1:

```
Output:

CREATE TABLE
INSERT 0 1000
INSERT 0 1000
 id |         val
----+---------------------
  1 |     0.85769666100434
  1 |    0.156152411912014
  1 |    0.769247939305863
  1 |     0.94759144873942
  1 |    0.535303978648415
  1 |  0.00759400169134139
  1 |     0.99270863754365
  1 |    0.239119714886034
```

Output 2:

```
Output:

CREATE TABLE
INSERT 0 4
CREATE VIEW
 customer_name | total_orders | total_sales
---------------+--------------+-------------
 B             |            1 |         754
 C             |            1 |         300
 M             |            1 |         500
 A             |            1 |         234
(4 rows)


psql:commands.sql:28: ERROR:  permission denied to create role
```

Learning Outcomes:

- Successfully implemented sub-queries to extract top salary earners by department.
- Successfully implemented stored procedures in PostgreSQL.
- Practiced handling input and output parameters in procedures.
- Automated HR analytics queries for gender-based employee counts.
- Developed an order-processing system with real-time stock validation.
- Enhanced SQL procedural programming skills for enterprise applications.