## Experiment - 5

**Student Name:** Archit Kaushal　　　　　　**UID:** 23BCS10313
**Branch:** BE-CSE　　　　　　　　　　　　**Section/Group:** KRG_2B
**Semester:** 5th　　　　　　　　　　　　**Date of Performance:** 22/9/25
**Subject Name:** Advanced Database and Management System
**Subject Code:** 23CSP-333

1. **Problem Description/Aim:**

   **Medium-Problem Title:** **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 metrics (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.

   **Hard-Problem Title: 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 -**
-------------------------------MEDIUM LEVEL PROBLEM-------------------------------------
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()
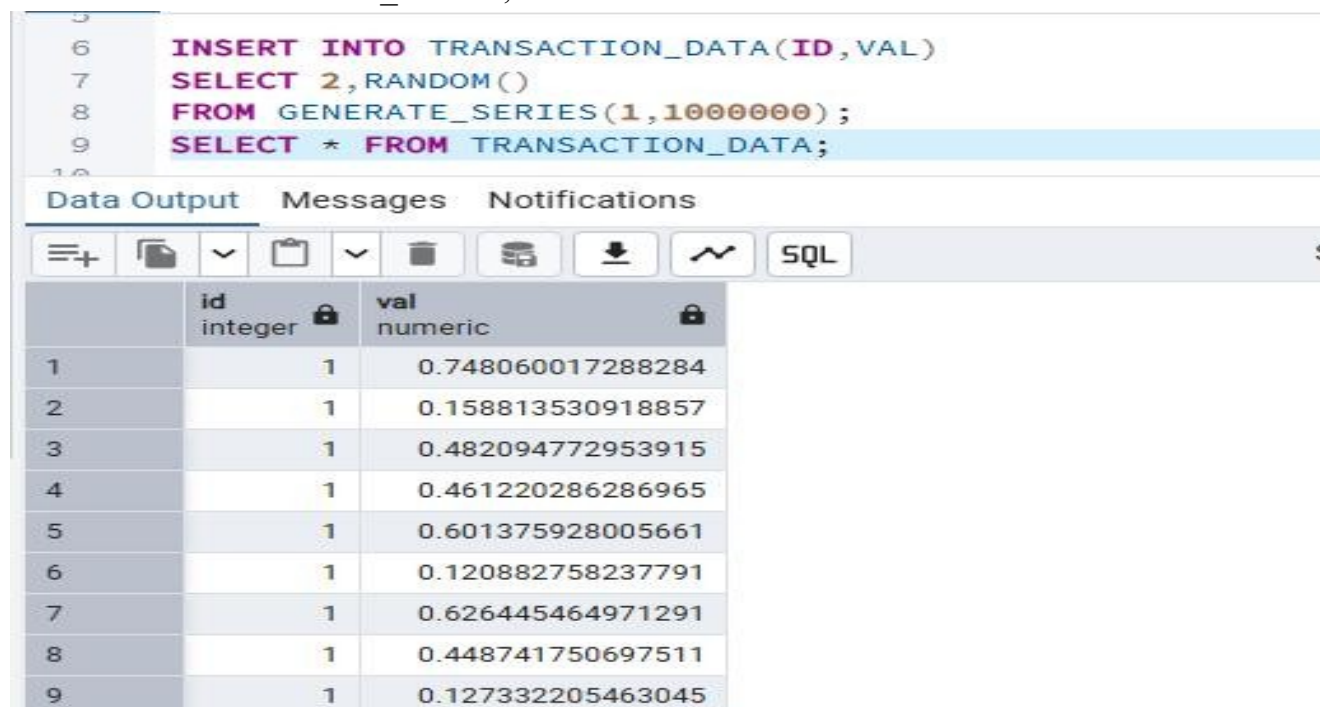FROM GENERATE_SERIES(1,1000000);
SELECT * FROM TRANSACTION_DATA;

```sql
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;
```

```sql
6    INSERT INTO TRANSACTION_DATA(ID,VAL)
7    SELECT 2,RANDOM()
8    FROM GENERATE_SERIES(1,1000000);
9    SELECT * FROM TRANSACTION_DATA;
```

Data Output    Messages    Notifications

SQL

| | id<br>integer | val<br>numeric |
|---|---|---|
| 1 | 1 | 0.748060017288284 |
| 2 | 1 | 0.158813530918857 |
| 3 | 1 | 0.482094772953915 |
| 4 | 1 | 0.461220286286965 |
| 5 | 1 | 0.601375928005661 |
| 6 | 1 | 0.120882758237791 |
| 7 | 1 | 0.626445464971291 |
| 8 | 1 | 0.448741750697511 |
| 9 | 1 | 0.127332205463045 |

```
21    SELECT * FROM SALES_SUMMARY; /*Simple view */
```

Data Output    Messages    Notifications

≡+  ▣ ⌄  ▯ ⌄  🗑  🗄  ⬇  ∿  SQL                                              Showir

| id<br>integer | total_quantity_sold<br>bigint | total_sales<br>numeric | total_orders<br>bigint |
|---|---|---|---|
| 1 | 1 | 2000000 | 1000226.201610874170319933640 | 1 |
| 2 | 2 | 1000000 | 499473.47586932728250459408 | 1 |

```
20    EXPLAIN ANALYZE
21    SELECT * FROM SALES_SUMMARY; /*Simple view *
```

Data Output    Messages    Notifications

≡+  ▣ ⌄  ▯ ⌄  🗑  🗄  ⬇  ∿  SQL

| QUERY PLAN<br>text |
|---|
| GroupAggregate (cost=471514.97..509014.99 rows=2 width=52) (a |
| Group Key: transaction_data.id |
| -> Sort (cost=471514.97..479014.97 rows=3000000 width=15) (ac |
| Sort Key: transaction_data.id |
| Sort Method: external merge Disk: 73504kB |
| -> Seq Scan on transaction_data (cost=0.00..46224.00 rows=3 |
| Planning Time: 0.135 ms |
| Execution Time: 4396.880 ms |

```
33    SELECT * FROM SALES_SUMM; /*Materialized view*/
```

Data Output    Messages    Notifications

≡+  ▣ ⌄  ▯ ⌄  🗑  🗄  ⬇  ∿  SQL                                              Show

| id<br>integer | total_quantity_sold<br>bigint | total_sales<br>numeric | total_orders<br>bigint |
|---|---|---|---|
| 1 | 1 | 1000000 | 500106.66754532635659814352 | 1 |
| 2 | 2 | 1000000 | 499473.47586932728250459408 | 1 |

```
32  EXPLAIN ANALYZE
33  SELECT * FROM SALES_SUMM; /*Materialized view*/
```

Data Output    Messages    Notifications

Showing rows: 1

| | QUERY PLAN<br>text | |
|---|---|---|
| 1 | Seq Scan on sales_summ  (cost=0.00..20.20 rows=1020 width=52) (actual time=0.017..0.018 rows=2 loops=... | |
| 2 | Planning Time: 0.063 ms | |
| 3 | Execution Time: 0.032 ms | |

OUTPUT -

As we can see that the execution time using the materialized view is very less as compared to the simple view's execution time.

--------------------------------------HARD  PROBLEM --------------------------------------

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

-- Insert sample data
INSERT INTO customer_data (customer_name, email, phone, payment_info, order_value)
VALUES
('Mandeep Kaur', 'mandeep@example.com', '9040122324', '1234-5678-9012-3456', 500),
('Mandeep Kaur', 'mandeep@example.com', '9040122324', '1234-5678-9012-3456', 1000),
('Jaskaran Singh', 'jaskaran@example.com', '9876543210', '9876-5432-1098-7654', 700),
('Jaskaran Singh', 'jaskaran@example.com', '9876543210', '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 'REPORT1234';
GRANT SELECT ON RESTRICTED_SALES_DATA TO CLIENT1;
REVOKE SELECT ON RESTRICTED_SALES_DATA FROM CLIENT1;
```
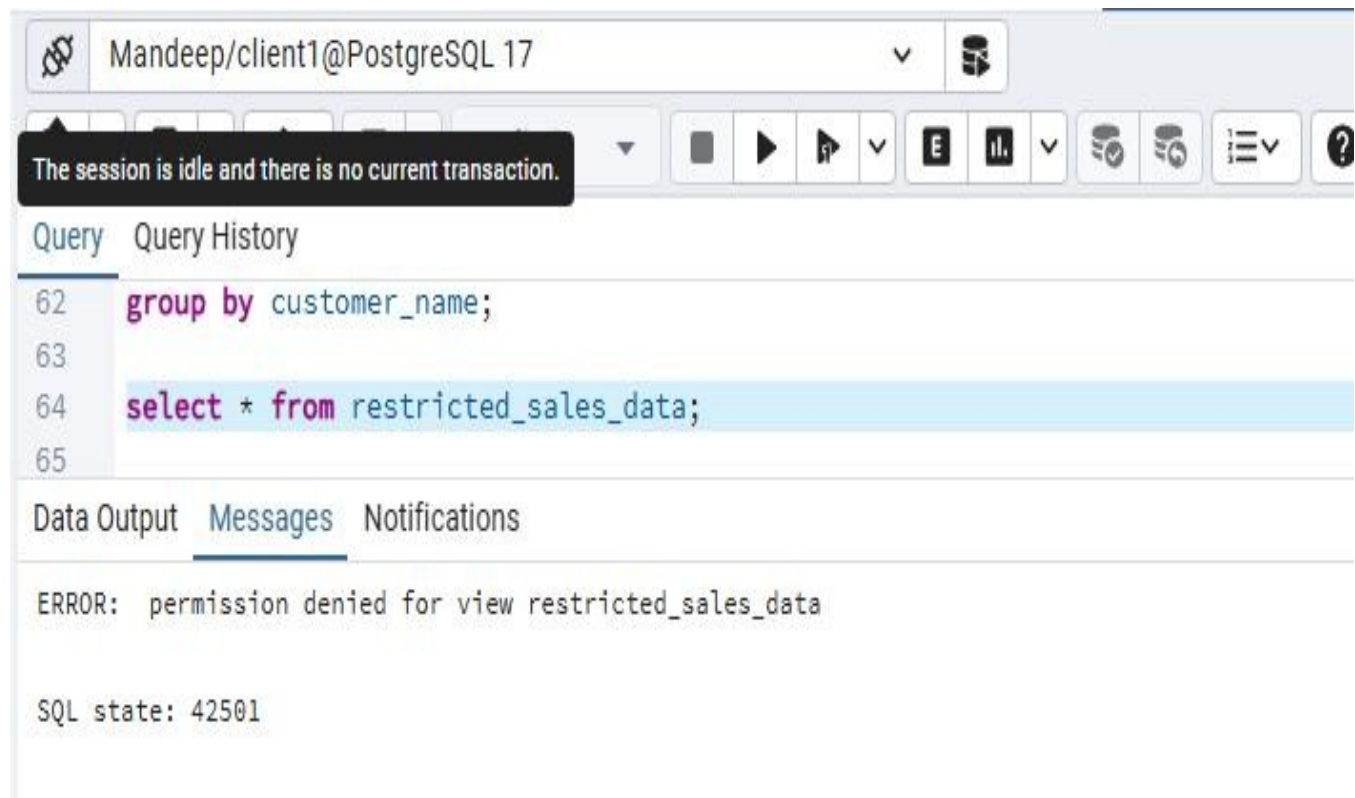
Mandeep/client1@PostgreSQL 17

The session is idle and there is no current transaction.

Query    Query History

```
62    group by customer_name;
63
64    select * from restricted_sales_data;
65
```

Data Output    Messages    Notifications

ERROR:   permission denied for view restricted_sales_data

SQL state: 42501

Mandeep/postgres@PostgreSQL 17

No limit

Query    Query History

```
65
66    CREATE USER CLIENT1 WITH PASSWORD 'REPORT1234';
67    GRANT SELECT ON RESTRICTED_SALES_DATA TO CLIENT1;
68    REVOKE SELECT ON RESTRICTED_SALES_DATA FROM CLIENT;
```

Data Output    Messages    Notifications
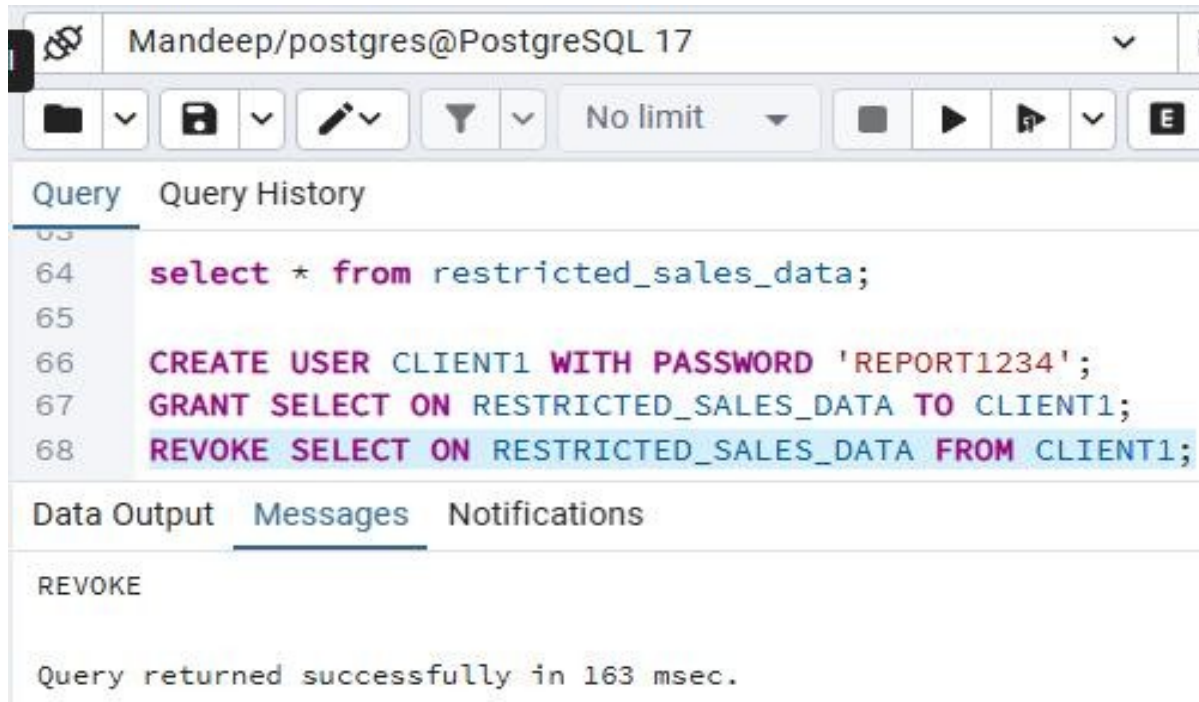
GRANT

Query returned successfully in 154 msec.

---

Mandeep/client1@PostgreSQL 17

No limit

Query    Query History

```
62    group by customer_name;
63
64    select * from restricted_sales_data;
65
```

Data Output    Messages    Notifications

SQL

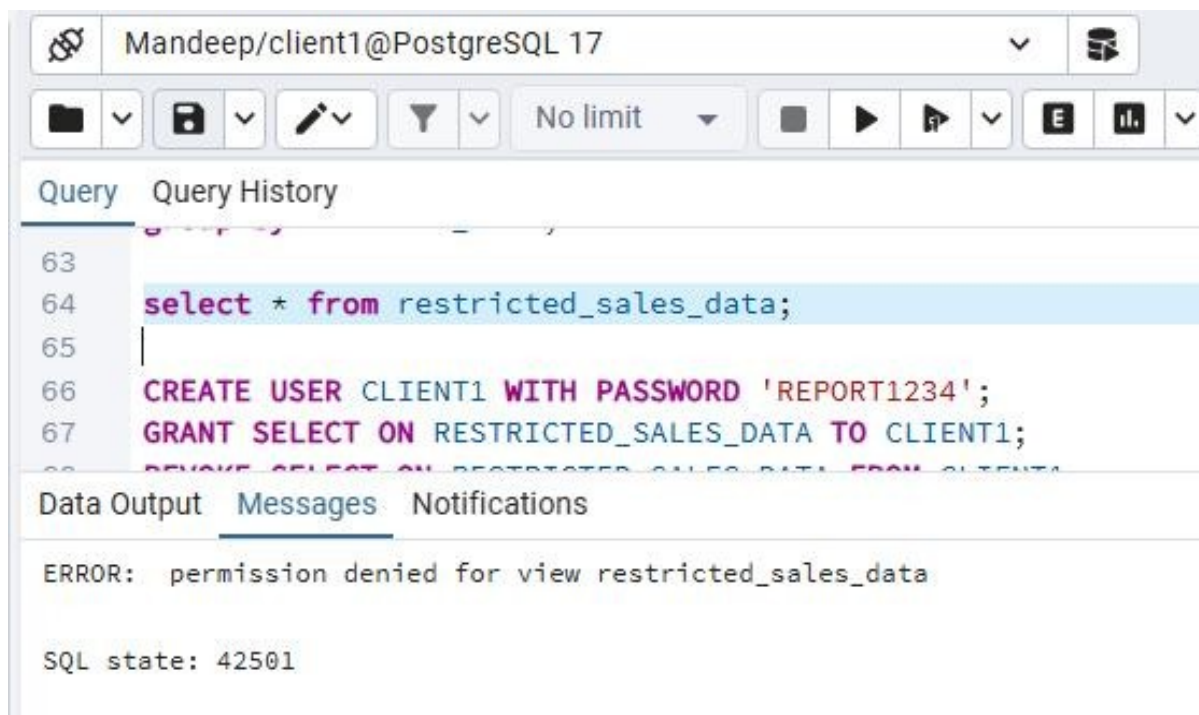| customer_name character varying (100) 🔒 | total_orders bigint 🔒 | total_sales numeric 🔒 |
|---|---|---|
| 1 | Jaskaran Singh | 2 | 1000 |
| 2 | Mandeep Kaur | 2 | 1500 |

Mandeep/postgres@PostgreSQL 17

No limit

Query    Query History

```
64      select * from restricted_sales_data;
65
66      CREATE USER CLIENT1 WITH PASSWORD 'REPORT1234';
67      GRANT SELECT ON RESTRICTED_SALES_DATA TO CLIENT1;
68      REVOKE SELECT ON RESTRICTED_SALES_DATA FROM CLIENT1;
```

Data Output    Messages    Notifications

REVOKE

Query returned successfully in 163 msec.

Mandeep/client1@PostgreSQL 17

No limit

Query    Query History

```
63      g   -   - r   ,                       ,
64      select * from restricted_sales_data;
65      |
66      CREATE USER CLIENT1 WITH PASSWORD 'REPORT1234';
67      GRANT SELECT ON RESTRICTED_SALES_DATA TO CLIENT1;
```

Data Output    Messages    Notifications

ERROR:   permission denied for view restricted_sales_data

SQL state: 42501