



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment - 5

**Name:** Ankush

**UID:** 23BCS12742

**Branch:** BE-CSE

**Section/Group:** KRG\_3B

**Semester:** 5<sup>th</sup>

**Date of Performance:** 19/9/25

**Subject Name:** Advanced Database and Management System

**Subject Code:** 23CSP-333

### 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 upto-date results.

### Source Code

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



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	id integer 🔒	val 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

Investigate

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

Data Output Messages Notifications

SQL Showir

	id integer	total_quantity_sold bigint	total_sales numeric	total_orders 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 text
1	GroupAggregate (cost=471514.97..509014.99 rows=2 width=52) (a
2	Group Key: transaction_data.id
3	-> Sort (cost=471514.97..479014.97 rows=3000000 width=15) (ac
4	Sort Key: transaction_data.id
5	Sort Method: external merge Disk: 73504kB
6	-> Seq Scan on transaction_data (cost=0.00..46224.00 rows=3
7	Planning Time: 0.135 ms
8	Execution Time: 4396.880 ms

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

Data Output Messages Notifications

SQL Show

	id integer	total_quantity_sold bigint	total_sales numeric	total_orders bigint
1	1	1000000	500106.667545326356598143529	1
2	2	1000000	499473.47586932728250459408	1

**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.

Revoke access if needed.

Use “REVOKE SELECT ON view\_name FROM reporting\_user;” to remove access.

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.

Assign access to reporting team(or client)-

-Use “GRANT SELECT ON view\_name TO reporting\_user; “ to give access.

2. Revoke access if needed.

-Use “REVOKE SELECT ON view\_name FROM reporting\_user;” to remove access.

3. 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.

## Source Code

```
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
);
```

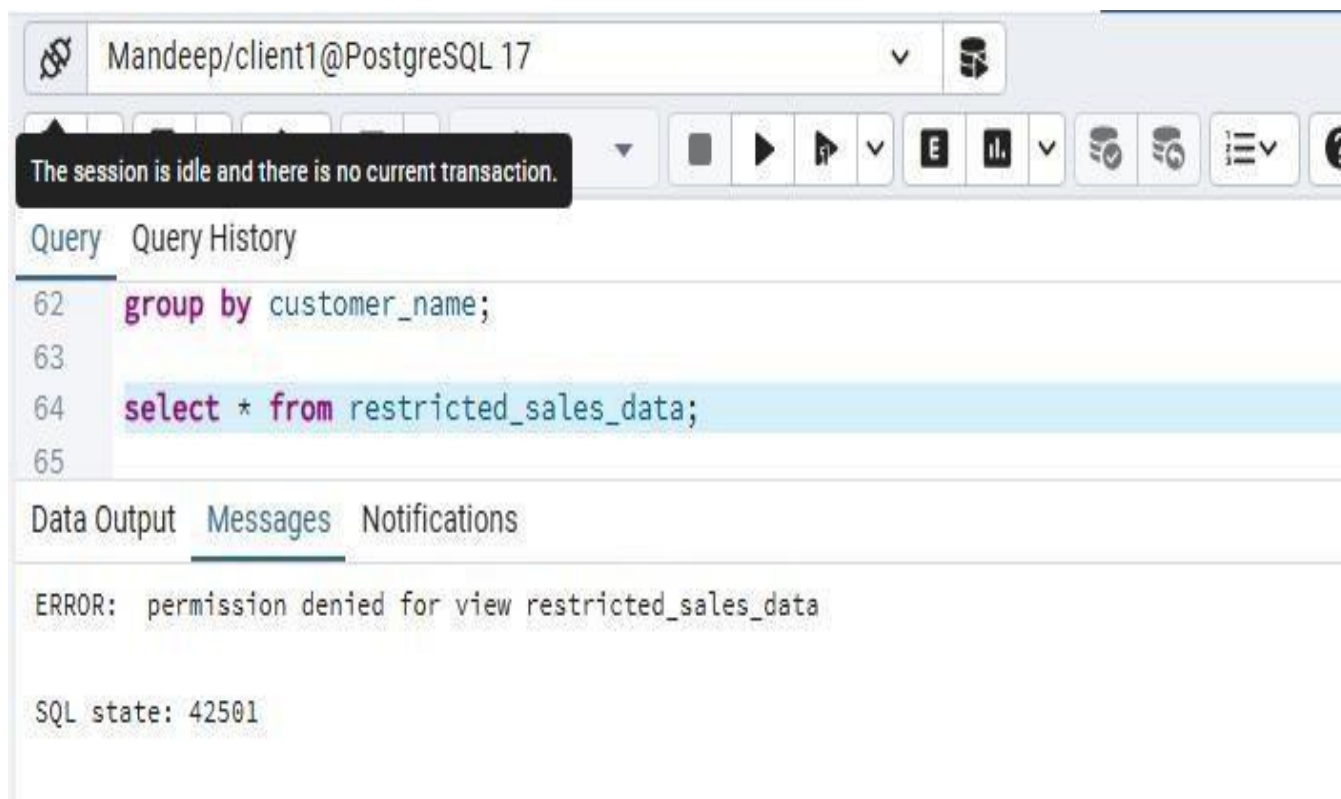
-- Insert sample data

```
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;
```



The screenshot shows a PostgreSQL client interface with the username 'Mandeep/client1@PostgreSQL 17'. A message at the top states: 'The session is idle and there is no current transaction.' Below this, the 'Query' tab is active, displaying SQL code with line numbers 62 to 65. Line 62 is 'group by customer\_name;', line 63 is empty, line 64 is 'select \* from restricted\_sales\_data;', and line 65 is empty. The 'Data Output' tab is also visible. Below the tabs, an error message is displayed: 'ERROR: permission denied for view restricted\_sales\_data' followed by 'SQL state: 42501'.

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

ERROR: permission denied for view restricted\_sales\_data  
SQL state: 42501



Mandeep/postgres@PostgreSQL 17

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

Query Query History

```

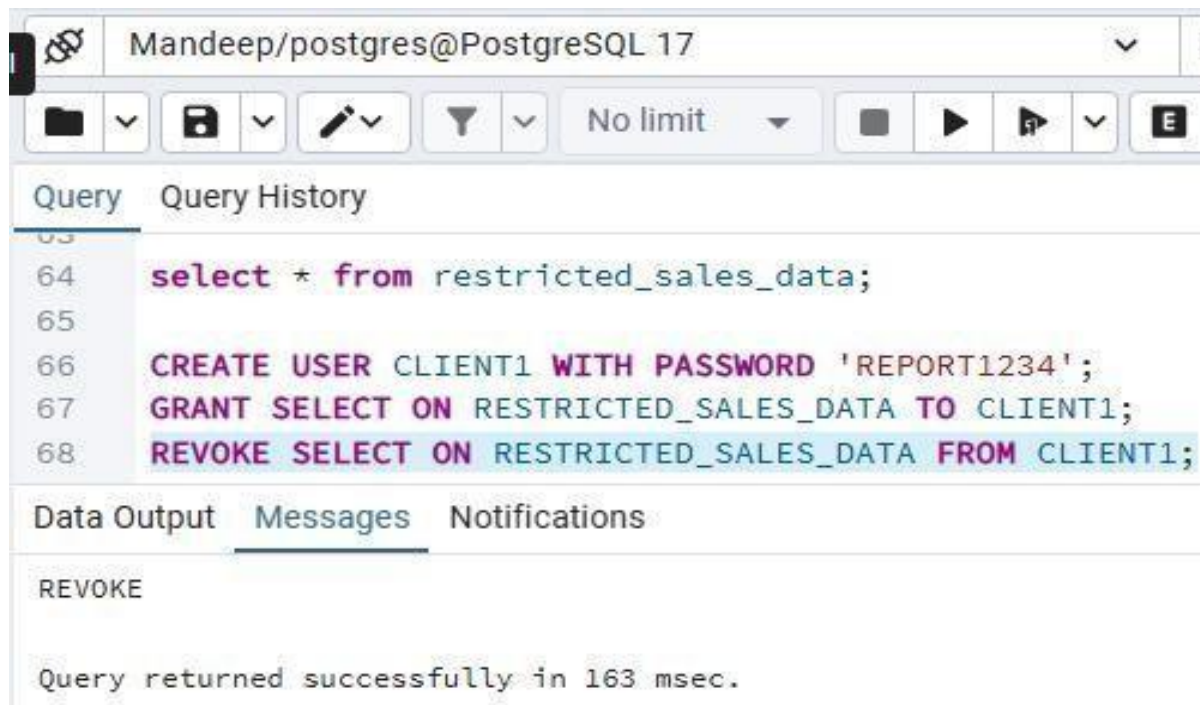
62 group by customer_name;
63
64 select * from restricted_sales_data;
65

```

Data Output Messages Notifications

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



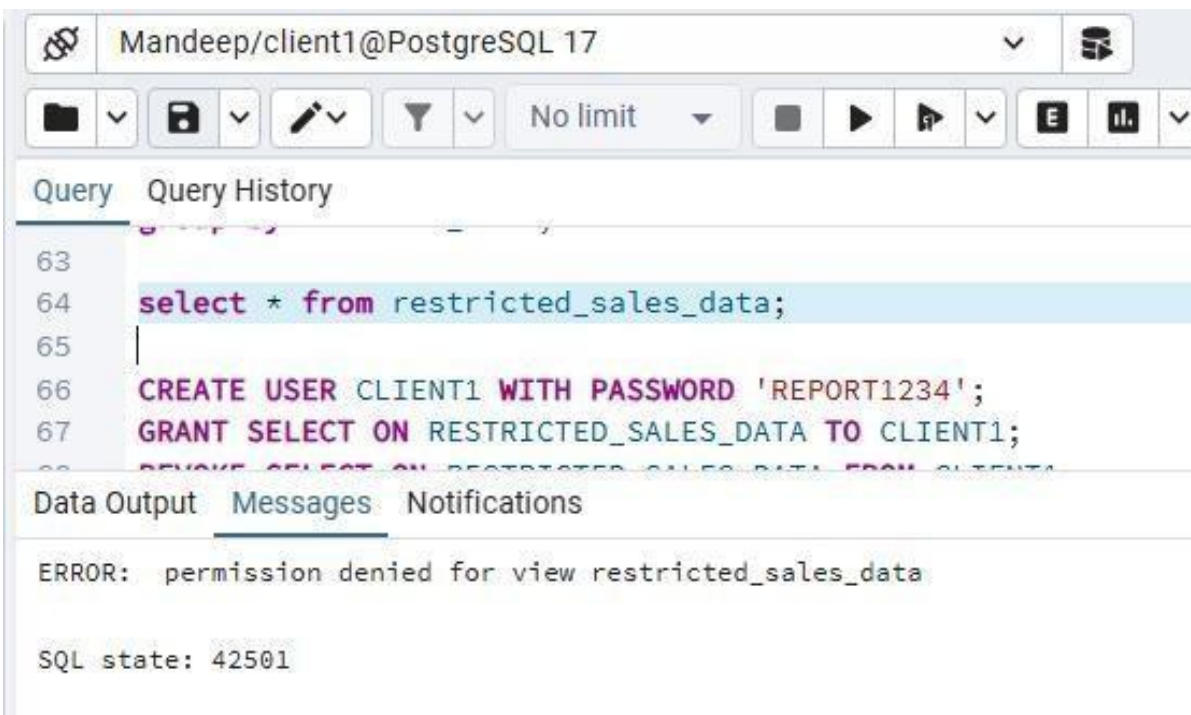


A screenshot of the PostgreSQL Query Editor interface. The title bar shows 'Mandeep/postgres@PostgreSQL 17'. The interface includes a toolbar with icons for file operations, query execution, and a 'No limit' dropdown. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL script with line numbers 64 through 68. The script contains a SELECT statement followed by CREATE, GRANT, and REVOKE statements. The 'REVOKE' statement is highlighted in blue. Below the script, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the text 'REVOKE' and a confirmation message: 'Query returned successfully in 163 msec.'

```
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;
```

REVOKE

Query returned successfully in 163 msec.



A screenshot of the PostgreSQL Query Editor interface, similar to the one above. The title bar shows 'Mandeep/client1@PostgreSQL 17'. The 'Query' tab is active, displaying a SQL script with line numbers 63 through 68. The script contains a SELECT statement followed by CREATE, GRANT, and REVOKE statements. The 'REVOKE' statement is highlighted in blue. Below the script, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing an error message: 'ERROR: permission denied for view restricted\_sales\_data' and 'SQL state: 42501'.

```
63
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;
```

ERROR: permission denied for view restricted\_sales\_data

SQL state: 42501