



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment - 5

Student Name: Naman

UID: 23BCS11036

Branch: BE-CSE

Section/Group: KRG_1A

Semester: 5th

Date of Performance: 22/9/25

Subject Name: Advanced Database and Management System

Subject Code: 23CSP-333

Aim:

Medium-Problem:

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.

Hard-Problem

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).

1. 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;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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;
```

The screenshot shows a PostgreSQL pgAdmin interface. The top part contains a SQL query editor with the following code:

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

The bottom part shows a data viewer with the results of the query. The table has two columns: 'id' (integer) and 'val' (numeric). The data is as follows:

	id	val
	integer	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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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

Data Output Messages Notifications

The screenshot shows a PostgreSQL query tool interface. At the top, there's a toolbar with various icons for file operations like new, open, save, and export. Below the toolbar, a menu bar has 'Show' selected. The main area displays a table with three columns: id, total_quantity_sold, and total_sales. There are two rows of data: one with id 1, total_quantity_sold 2000000, and total_sales 1000226.201610874170319933640; and another with id 2, total_quantity_sold 1000000, and total_sales 499473.47586932728250459408. The table has a light gray header row and white data rows.

	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

The screenshot shows the EXPLAIN ANALYZE output for the SELECT query. It starts with a 'QUERY PLAN' header and then lists numbered steps. Step 1: GroupAggregate (cost=471514.97..509014.99 rows=2 width=52) (a Group Key: transaction_data.id). Step 2: Sort (cost=471514.97..479014.97 rows=3000000 width=15) (an Sort Key: transaction_data.id). Step 3: Seq Scan on transaction_data (cost=0.00..46224.00 rows=3000000 width=15). Step 4: Planning Time: 0.135 ms. Step 5: Execution Time: 4396.880 ms. The output is presented in a table-like structure with a light gray header row and white data rows.

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

Data Output Messages Notifications

The screenshot shows a PostgreSQL query tool interface. At the top, there's a toolbar with various icons for file operations like new, open, save, and export. Below the toolbar, a menu bar has 'Show' selected. The main area displays a table with three columns: id, total_quantity_sold, and total_sales. There are two rows of data: one with id 1, total_quantity_sold 1000000, and total_sales 500106.667545326356598143529; and another with id 2, total_quantity_sold 1000000, and total_sales 499473.47586932728250459408. The table has a light gray header row and white data rows.

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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

Data Output Messages Notifications

SQL

Showing rows: 1

QUERY PLAN	
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
('John', 'John@example.com', '9040122324', '1234-5678-9012-3456', 500),
('John', 'John@example.com', '9040122324', '1234-5678-9012-3456', 1000),
('Alice Singh', 'Alice@example.com', '9876543210', '9876-5432-1098-7654', 700),
('Alice Singh', 'Alice@example.com', '9876543210', '9876-5432-1098-7654', 300);
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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 MySQL Workbench interface. At the top, a message bar says "The session is idle and there is no current transaction." Below it is a toolbar with various icons. The main area has tabs for "Query" and "Query History". A code editor window displays the following SQL statements:

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

Below the code editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Messages" tab is active, showing the error:

ERROR: permission denied for view restricted_sales_data

SQL state: 42501

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

The screenshot shows a MySQL Workbench interface. The toolbar at the top includes icons for file operations, search, and various database management functions. Below the toolbar, tabs for 'Query' and 'Query History' are visible, with 'Query' being the active tab. The main pane displays a series of SQL statements:

```
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;
```

Below the SQL statements, there are tabs for 'Data Output', 'Messages', and 'Notifications', with 'Messages' being the active tab. The message content is:

ERROR: permission denied for view restricted_sales_data
SQL state: 42501