University Institute of Engineering

Department of Computer Science & Engineering

Experiment:5

Date of Experiment: 03-10-2025

1. Aim of the practical:

[MEDIUM]

1. Create a large dataset:
   a. Create a table names transaction_data (id , value) with 1 million records.
   b. take id 1 and 2, and for each id, generate 1 million records in value column
   c. 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.

[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: SQL Server Management Studio

3. Queries:

----------MEDIUM----------

```
-- Create base table
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);
-- View data
SELECT * FROM transaction_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;
-- Analyze performance of normal view
EXPLAIN ANALYZE
SELECT * FROM sales_summary_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;
```

```sql
-- Analyze performance of materialized view
EXPLAIN ANALYZE
SELECT * FROM sales_summary_mv;
-- Create another table
CREATE TABLE random_tabl (id INT, val DECIMAL);
-- Insert random data
INSERT INTO random_tabl
SELECT 1, random() FROM generate_series(1, 1000000);
INSERT INTO random_tabl
SELECT 2, random() FROM generate_series(1, 1000000);
-- Normal execution
SELECT id, AVG(val), COUNT(*)
FROM random_tabl
GROUP BY id;
-- Create materialized view
CREATE MATERIALIZED VIEW mv_random_tabl AS
SELECT id, AVG(val), COUNT(*)
FROM random_tabl
GROUP BY id;
-- Query materialized view
SELECT * FROM mv_random_tabl;
-- Refresh materialized view after data change
REFRESH MATERIALIZED VIEW mv_random_tabl;
```

----------HARD----------

```sql
-- ====================================
--Create Tables
CREATE TABLE customer_master (
    customer_id INT IDENTITY PRIMARY KEY,
    full_name NVARCHAR(100) NOT NULL
);
CREATE TABLE product_catalog (
    product_id INT IDENTITY PRIMARY KEY,
    product_name NVARCHAR(100) NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL
);
CREATE TABLE sales_orders (
    order_id INT IDENTITY PRIMARY KEY,
    order_date DATE NOT NULL,
    customer_id INT FOREIGN KEY REFERENCES customer_master(customer_id),
    product_id INT FOREIGN KEY REFERENCES product_catalog(product_id),
    quantity INT NOT NULL,
    discount_percent DECIMAL(5,2) DEFAULT 0
);
```

```sql
--Create Tables
CREATE TABLE customer_master (
    customer_id INT IDENTITY PRIMARY KEY,
    full_name NVARCHAR(100) NOT NULL
);
CREATE TABLE product_catalog (
    product_id INT IDENTITY PRIMARY KEY,
    product_name NVARCHAR(100) NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL
);
CREATE TABLE sales_orders (
    order_id INT IDENTITY PRIMARY KEY,
    order_date DATE NOT NULL,
    customer_id INT FOREIGN KEY REFERENCES customer_master(customer_id),
    product_id INT FOREIGN KEY REFERENCES product_catalog(product_id),
    quantity INT NOT NULL,
    discount_percent DECIMAL(5,2) DEFAULT 0
);
-- Insert Sample Data
-- Customers
INSERT INTO customer_master (full_name)
VALUES ('Alice Smith'), ('Bob Johnson'), ('Charlie Rose');
-- Products
INSERT INTO product_catalog (product_name, unit_price)
VALUES ('Laptop', 1000.00), ('Phone', 600.00), ('Tablet', 300.00);
-- Orders
INSERT INTO sales_orders (order_date, customer_id, product_id, quantity, discount_percent)
VALUES
('2023-09-01', 1, 1, 2, 10),
('2023-09-02', 2, 2, 1, 5),
('2023-09-03', 3, 3, 3, 0);
-- Create View vW_ORDER_SUMMARY
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;
-- Test the view
SELECT * FROM vW_ORDER_SUMMARY;
-- Create Login and User
-- Create Login at Server Level (run as sysadmin)
CREATE LOGIN ADvitiya WITH PASSWORD = 'StrongPassword123!';
```

```sql
-- Create Database User for the login
USE [Academic];
CREATE USER ALOK FOR LOGIN ADvitiya;
-- Grant SELECT permission on the view only
GRANT SELECT ON vW_ORDER_SUMMARY TO ALOK;
-- Create Employee Table
CREATE TABLE EMPLOYEE (
    empId INT PRIMARY KEY,
    name NVARCHAR(50) NOT NULL,
    dept NVARCHAR(50) NOT NULL
);
-- Insert Data
INSERT INTO EMPLOYEE VALUES
(1, 'Clark', 'Sales'),
(2, 'Dave', 'Accounting'),
(3, 'Ava', 'Sales');
-- Create View With CHECK OPTION
CREATE VIEW vW_STORE_SALES_DATA
AS
SELECT empId, name, dept
FROM EMPLOYEE
WHERE dept = 'Sales'
WITH CHECK OPTION;
-- View Content
SELECT * FROM vW_STORE_SALES_DATA;
-- This works
INSERT INTO vW_STORE_SALES_DATA (empId, name, dept)
VALUES (4, 'Riya', 'Sales');
-- ❌ This fails - violates CHECK OPTION
INSERT INTO vW_STORE_SALES_DATA (empId, name, dept)

VALUES (5, 'Aman', 'Admin');
```

4. Output:

[MEDIUM]

Output:

```
CREATE TABLE
INSERT 0 1000000
INSERT 0 1000000
CREATE VIEW
                                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Finalize GroupAggregate  (cost=26527.17..26580.34 rows=200 width=52) (actual time=2007.775.
   Group Key: transaction_data.id
   ->  Gather Merge  (cost=26527.17..26573.84 rows=400 width=52) (actual time=2007.761..2009
         Workers Planned: 2
         Workers Launched: 2
         ->  Sort  (cost=25527.14..25527.64 rows=200 width=52) (actual time=1998.241..1998.2
               Sort Key: transaction_data.id
               Sort Method: quicksort  Memory: 25kB
               Worker 0:  Sort Method: quicksort  Memory: 25kB
               Worker 1:  Sort Method: quicksort  Memory: 25kB
               ->  Partial HashAggregate  (cost=25517.50..25519.50 rows=200 width=52) (actua
                     Group Key: transaction_data.id
                     Batches: 1  Memory Usage: 40kB
                     Worker 0:  Batches: 1  Memory Usage: 40kB
                     Worker 1:  Batches: 1  Memory Usage: 40kB
                     ->  Parallel Seq Scan on transaction_data  (cost=0.00..17183.75 rows=83
 Planning Time: 0.265 ms
 Execution Time: 2009.753 ms
(18 rows)
```

Analyze performance of normal view

Output:

```
CREATE TABLE
INSERT 0 1000000
INSERT 0 1000000
SELECT 2
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
 Seq Scan on sales_summary_mv  (cost=0.00..20.20 rows=1020 width=52) (actual time=
 Planning Time: 0.057 ms
 Execution Time: 0.015 ms
(3 rows)
```

Analyze performance of materialized view

```
Output:

CREATE TABLE
INSERT 0 1000000
INSERT 0 1000000
 id |            avg             |  count
----+----------------------------+---------
  1 |  0.5001362743231808392690 | 1000000
  2 | 0.4998730053796865218092 8 | 1000000
(2 rows)

SELECT 2
 id |            avg             |  count
----+----------------------------+---------
  1 |  0.5001362743231808392690 | 1000000
  2 | 0.4998730053796865218092 8 | 1000000
(2 rows)

REFRESH MATERIALIZED VIEW
```

[HARD]

Results | Messages

	order_id	order_date	product_name	full_name	final_cost
1	1	2023-09-01	Laptop	Alice Smith	1800.00000000
2	2	2023-09-02	Phone	Bob Johnson	570.00000000
3	3	2023-09-03	Tablet	Charlie Rose	900.00000000

View summary of vW_ORDER_SUMMARy

Results | Messages

	empId	name	dept
1	1	Clark	Sales
2	3	Ava	Sales

Create View With CHECK OPTION

Learning outcomes (What I have learnt):

- How to generate large datasets using `generate_series()` and `random()` functions in PostgreSQL.
- Creating normal views with `CREATE OR REPLACE VIEW` to summarize transactional data.
- Creating and using materialized views with `CREATE MATERIALIZED VIEW` for performance optimization.
- Understanding the need to manually refresh materialized views using `REFRESH MATERIALIZED VIEW` after base data changes.
- Using `EXPLAIN ANALYZE` to analyze and compare query performance between normal and materialized views.