



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 2.1

Student Name: Gurshaan Singh

UID: 23BCS14195

Branch: CSE

Section/Group: KRG – 1B

Semester: 5th

Date of Performance: 24/09/25

Subject Name: ADBMS

Subject Code: 23CSP-333

1. **Aim:** Views: Performance Benchmarking : Normal View vs. Materialized View
2. **Requirements(Hardware/Software):** MySQL, PostgreSQL, Oracle, or SQL Server
3. **DBMS script and output:**

Problem A

Problem Title: Performance Benchmarking : Normal View vs. Materialized View



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.

```
CREATE TABLE TRANSACTION_DATA (  
    ID NUMBER,  
    VALUE NUMBER  
);  
  
INSERT INTO TRANSACTION_DATA (ID, VALUE)  
SELECT 1, ROUND(DBMS_RANDOM.VALUE(0,100),2)  
FROM DUAL  
CONNECT BY LEVEL <= 1000000;  
  
INSERT INTO TRANSACTION_DATA (ID, VALUE)  
SELECT 2, ROUND(DBMS_RANDOM.VALUE(0,100),2)  
FROM DUAL  
CONNECT BY LEVEL <= 1000000;  
  
COMMIT;  
  
CREATE OR REPLACE VIEW SALES_SUMMARY AS  
SELECT  
    ID,  
    COUNT(*) AS TOTAL_QUANTITY_SOLD,  
    SUM(VALUE) AS TOTAL_SALES,  
    COUNT(DISTINCT ID) AS TOTAL_ORDERS  
FROM TRANSACTION_DATA  
GROUP BY ID;  
  
CREATE MATERIALIZED VIEW SALES_SUMMARY_MV  
BUILD IMMEDIATE  
REFRESH COMPLETE ON DEMAND  
AS  
SELECT  
    ID,  
    COUNT(*) AS TOTAL_QUANTITY_SOLD,  
    SUM(VALUE) AS TOTAL_SALES,  
    COUNT(DISTINCT ID) AS TOTAL_ORDERS  
FROM TRANSACTION_DATA  
GROUP BY ID;  
  
SELECT * FROM SALES_SUMMARY;  
SELECT * FROM SALES_SUMMARY_MV;  
  
EXEC DBMS_MVIEW.REFRESH('SALES_SUMMARY_MV');
```

View created.

0.02 seconds

☒ Autocommit Rows 10   Save Run

```
CREATE TABLE TRANSACTION_DATA (  
    ID NUMBER,  
    VALUE NUMBER  
);  
  
INSERT INTO TRANSACTION_DATA (ID, VALUE)  
SELECT 1, ROUND(DBMS_RANDOM.VALUE(0,100),2)  
FROM DUAL  
CONNECT BY LEVEL <= 1000000;  
  
INSERT INTO TRANSACTION_DATA (ID, VALUE)  
SELECT 2, ROUND(DBMS_RANDOM.VALUE(0,100),2)  
FROM DUAL  
CONNECT BY LEVEL <= 1000000;
```

Results Explain Describe Saved SQL History

1000000 row(s) inserted.

Problem B:

Problem Title: Securing Data Access with Views and Role-Based Permissions

Procedure (Step-by-Step):

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:

Create **restricted views** to display only summarized, non-sensitive data.

2. Assign access to these views to specific users using **DCL commands** (GRANT, REVOKE).

```
CREATE TABLE TRANSACTION_DATA (  
    ID NUMBER,  
    VALUE NUMBER  
);
```

```
INSERT INTO TRANSACTION_DATA (ID, VALUE)
SELECT 1, ROUND(DBMS_RANDOM.VALUE(0,100),2)
FROM DUAL
CONNECT BY LEVEL <= 100000;
```

```
INSERT INTO TRANSACTION_DATA (ID, VALUE)
SELECT 2, ROUND(DBMS_RANDOM.VALUE(0,100),2)
FROM DUAL
CONNECT BY LEVEL <= 100000;
```

```
COMMIT;
```

```
CREATE OR REPLACE VIEW SALES_SUMMARY AS
SELECT
  ID,
  COUNT(*) AS TOTAL_QUANTITY_SOLD,
  SUM(VALUE) AS TOTAL_SALES,
  COUNT(DISTINCT ID) AS TOTAL_ORDERS
FROM TRANSACTION_DATA
GROUP BY ID;
```

```
CREATE USER REPORT_USER IDENTIFIED BY Report123;
```

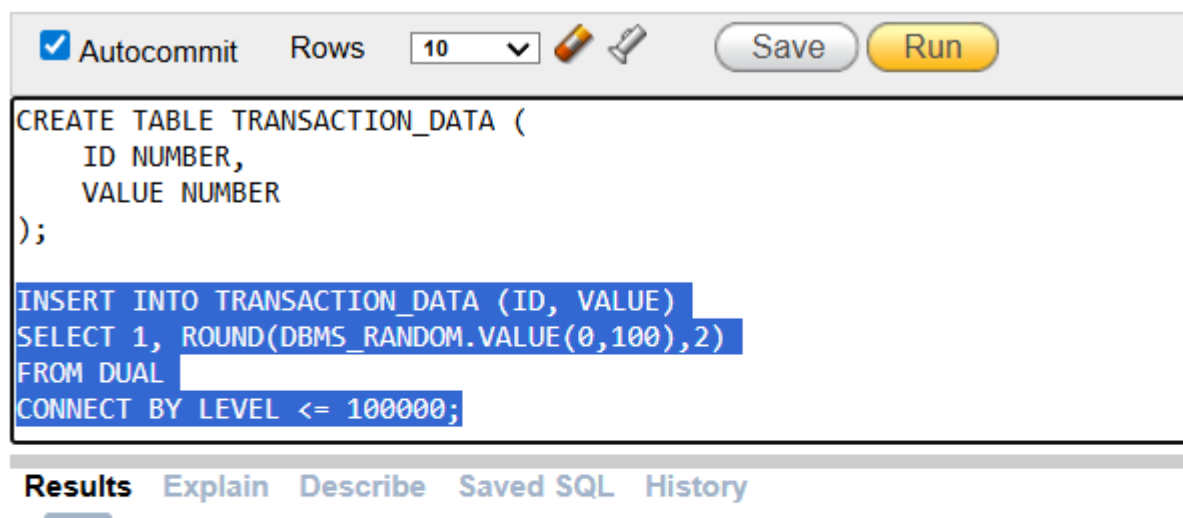
```
GRANT CREATE SESSION TO REPORT_USER;
```

```
GRANT SELECT ON SALES_SUMMARY TO REPORT_USER;
```

```
REVOKE SELECT ON TRANSACTION_DATA FROM REPORT_USER;
```

```
SELECT * FROM SALES_SUMMARY;
```

```
EXEC DBMS_MVIEW.REFRESH('SALES_SUMMARY');
```



The screenshot shows a SQL IDE window with a toolbar at the top containing 'Autocommit' (checked), 'Rows' (set to 10), and 'Save' and 'Run' buttons. The main text area contains the following SQL code:



```
CREATE TABLE TRANSACTION_DATA (
  ID NUMBER,
  VALUE NUMBER
);

INSERT INTO TRANSACTION_DATA (ID, VALUE)
SELECT 1, ROUND(DBMS_RANDOM.VALUE(0,100),2)
FROM DUAL
CONNECT BY LEVEL <= 100000;
```

Below the code area is a tabbed interface with 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History' tabs. The 'Results' tab is active, displaying the output of the SQL execution.

100000 row(s) inserted.

0.19 seconds

☒ Autocommit Rows 10   Save Run

```
CREATE OR REPLACE VIEW SALES_SUMMARY AS
SELECT
  ID,
  COUNT(*) AS TOTAL_QUANTITY_SOLD,
  SUM(VALUE) AS TOTAL_SALES,
  COUNT(DISTINCT ID) AS TOTAL_ORDERS
FROM TRANSACTION_DATA
GROUP BY ID;
```

Results Explain Describe Saved SQL History

View created.

0.00 seconds

| Results Explain Describe Saved SQL History | | | | |
|---|---------------------|-------------|--------------|--|
| ID | TOTAL_QUANTITY_SOLD | TOTAL_SALES | TOTAL_ORDERS | |
| 1 | 100000 | 5001755.13 | 1 | |
| 2 | 100000 | 5008213.59 | 1 | |

2 rows returned in 0.01 seconds [Download](#)

4. Learning Outcomes :

- Understand how to create **views in Oracle** to restrict access to sensitive data.
- Learn to **aggregate transactional data** using SQL for reporting purposes.
- Apply **DCL commands (GRANT and REVOKE)** to control user access to database objects.
- Gain knowledge of **creating and managing users** with specific privileges in Oracle.
- Demonstrate the ability to **securely provide summarized data** to reporting teams without exposing base tables.