P4 Assignment: DW Performance

### **Experiment 1 : Simple Aggregation**

Our first experiment is to create simple aggregations from the Superstore sales fact table.

In this we are trying to find the average sales for each of the category in order priorities.

A screenshot of a computer

Description automatically generated

The execution time of the above highlighted query is 0.095 seconds, which has been queried from the original table.

Next we will create a summary table : sales\_agg , here the aggregation results were stored which has only 2 columns that is order\_priority and avg\_sales.

Now let’s try running the same aggregate query against the summary table.

A screenshot of a computer

Description automatically generated

Now the execution time is 0.088, this has a slightly better time and hence there is a performance gain.

### **Experiment 2 : Try a Materialized View**

A materialized view stores pre-computed summary-level information based on the original data. The BUILD IMMEDIATE and REFRESH COMPLETE statements define the materialized view as an immediate build and specify that it should be refreshed completely with the latest data.

A screenshot of a computer

Description automatically generated

The output of the query :

A screenshot of a computer

Description automatically generated

### **Experiment 3: Query Rewrite**

Query rewrite allows the query optimizer to automatically use the materialized view when appropriate, effectively replacing queries against the original data with the materialized view. This can lead to significant performance improvements when querying data.

A screenshot of a computer

Description automatically generated

Then when you run the original query, the optimizer will select this materialized view automatically since the query rewrite is on as it is the optimal solution as shown in the below screenshot.

A screenshot of a computer

Description automatically generated

To test where the query rewrite is working, we have got the plan for the initial query , from the above image it is clear that the optimizer chose the materialized view to run this query as it is the optimal solution.

### **Experiment 4: Partitioning**

This SQL statement creates a table named yahoo\_axp\_part with range partitioning on the year column. The table contains various numeric columns related to stock trading data.

A screenshot of a computer program

Description automatically generated

Querying the partitioned table :

A screenshot of a computer

Description automatically generated

Range partitioning on the year column can potentially improve query performance by allowing the database to access only the relevant partitions when executing queries with a WHERE condition on the year column.

This below query gives the execution plan for a particular year that is 1991 , here only one partition was used. It can clearly seen

A screenshot of a computer

Description automatically generated

If we execute this same query on the table without partition, the execution time was : 0.207s.

A screenshot of a computer

Description automatically generated