

Assignment -9

1. Query profile

Step 1: Enable Query Profiling

Use the Web UI to open the Query History tab.

Select a query to view its Query Profile.

SQL Text[View Results](#)

Query ID: 01bb9df9-0000-ee84-0009-1cfa000276da

```
SELECT * FROM SALES
WHERE QUANTITY >15;
```



Step 2: Analyze the Profile

Look for bottlenecks such as high scan percentages or large intermediate results.

Optimize queries by:

Adding filters (e.g., WHERE clauses).

```
-- BEFORE
SELECT SUM(amount) FROM sales;

-- AFTER
SELECT SUM(amount) FROM sales
WHERE sale_date >= '2023-01-01';
```

Using proper indexes or clustering keys.

```
ALTER TABLE sales CLUSTER BY (sale_date);
```

Minimizing unnecessary joins or subqueries.

```
-- BEFORE
SELECT customer_name
FROM customers
WHERE customer_id IN (
    SELECT customer_id
    FROM sales
    WHERE amount > 500
);

-- AFTER
SELECT DISTINCT c.customer_name
FROM customers c
JOIN sales s ON c.customer_id = s.customer_id
WHERE s.amount > 500;
```

So here are some main techniques used in case of bottlenecks for improving the performance:

Technique	When to Use	Benefit
Add WHERE filters	High scan %	Smaller scans
Clustering keys	Large tables, filtered queries	Partition pruning
Avoid joins/subqueries	Large intermediate results	Faster processing
Limit columns	Query pulls too much data	Saves bandwidth
Result cache	Repeated queries	Zero-cost re-runs

2. Experiment with Clustering Keys for Large Tables

Step 1: Create a Large Dataset

```
CREATE TABLE sales_info (  
    sale_id INT AUTOINCREMENT,  
    sale_date DATE,  
    customer_id INT,  
    amount DECIMAL(10,2)  
);  
  
INSERT INTO sales_info (sale_date, customer_id, amount)  
SELECT DATEADD(DAY, UNIFORM(1, 3650, RANDOM()), '2015-01-01'),  
    UNIFORM(1, 10000, RANDOM()),  
    UNIFORM(100, 1000, RANDOM())  
FROM TABLE(GENERATOR(ROWCOUNT => 1000000));
```

I have manually generated random values for 10 lakh rows in order to get a large dataset.

Step 2: Add a Clustering Key

Choose a column frequently used in WHERE clauses.

Add a clustering key :

```
ALTER TABLE sales_info CLUSTER BY (sale_date);
```

Step 3: Monitor Clustering Performance

Check clustering depth:

```
| SELECT SYSTEM$CLUSTERING_INFORMATION('sales_info');
```

```
A SYSTEM$CLUSTERING_INFORMATION('SALES_INFO')
{
  "cluster_by_keys" : "LINEAR(sale_date)",
  "notes" : "Clustering key columns contain high cardinality key
  SALE_DATE which might result in expensive re-clustering. Consider
  reducing the cardinality of clustering keys. Please refer to
  https://docs.snowflake.net/manuals/user-guide/tables-clustering-
  keys.html for more information.",
  "total_partition_count" : 1,
  "total_constant_partition_count" : 0,
  "average_overlaps" : 0.0,
  "average_depth" : 1.0,
  "partition_depth_histogram" : {
    "000000" : 0,
    "000001" : 1,
    "000002" : 0,
    "000003" : 0,
    "000004" : 0,
    "000005" : 0,
    "000006" : 0,
    "000007" : 0,
    "000008" : 0,
    "000009" : 0,
    "000010" : 0,
    "000011" : 0,
    "000012" : 0,
    "000013" : 0,
    "000014" : 0,
    "000015" : 0,
    "000016" : 0
  },
  "clustering_errors" : [ ]
}
```

3. Monitor and Analyze Cost Usage Metrics

Step 1: Check Warehouse Usage

Query warehouse metering data:

```
SELECT WAREHOUSE_NAME, SUM(CREDITS_USED) AS TOTAL_CREDITS
FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
WHERE START_TIME > CURRENT_TIMESTAMP() - INTERVAL '7 DAY'
GROUP BY WAREHOUSE_NAME;
```


	A WAREHOUSE_NAME	# TOTAL_CREDITS
1	CLOUD_SERVICES_ONLY	0.000753059
2	COMPUTE_WH	7.641894172
3	test_warehouse	5.947146945

The *SUM(CREDITS_USED)* value represents the total compute credits consumed by each warehouse over the last 7 days.

Step 2: Optimize Warehouse Costs

Suspend unused warehouses using:

```
ALTER WAREHOUSE my_warehouse SUSPEND;
```


SYSTEM\$STREAMLIT_NOTEB...
Standard
X-Small
Suspended