

Context

| | |
|----------------------------|---------|
| EXPLAIN Plans | 2 – 4 |
| Data Spilling | 5 |
| Use of the Data Cache | 5 – 6 |
| Micro-Partition Pruning | 6 |
| Query History | 7 |
| Create Warehouse | 8 – 10 |
| Resource Monitors | 10 – 11 |
| Query acceleration service | 12 – 13 |
| Materialized Views | 14 |
| Non Materialized Views | 14 – 15 |
| SELECT Commands | 15 – 19 |
| Persisted Query Results | 20 |

EXPLAIN Plans

1. In Snowflake, you can use the EXPLAIN command to obtain the query execution plan, which provides insights into how Snowflake intends to execute a query.
2. This can help in understanding and optimizing query performance.
3. Create below two tables.

Query: -- Create the customers table

```
CREATE OR REPLACE TABLE customers (  
  customer_id INT,  
  customer_name STRING,  
  city STRING,  
  country STRING,  
  signup_date DATE  
);
```

-- Insert sample data into the customers table

```
INSERT INTO customers (customer_id, customer_name, city, country, signup_date) VALUES  
(1, 'John Doe', 'New York', 'USA', '2023-01-15'),  
(2, 'Jane Smith', 'Los Angeles', 'USA', '2023-02-20'),  
(3, 'Carlos Ortega', 'Madrid', 'Spain', '2023-03-10'),  
(4, 'Maria Garcia', 'Barcelona', 'Spain', '2023-04-05'),  
(5, 'Liu Wei', 'Shanghai', 'China', '2023-05-25');
```

-- Create the orders table

```
CREATE OR REPLACE TABLE orders (  
  order_id INT,  
  customer_id INT,  
  order_date DATE,  
  order_amount DECIMAL(10,2)  
);
```

-- Insert sample data into the orders table

```
INSERT INTO orders (order_id, customer_id, order_date, order_amount) VALUES  
(101, 1, '2023-01-20', 150.00),  
(102, 1, '2023-02-15', 200.00),  
(103, 2, '2023-02-25', 300.00),  
(104, 3, '2023-03-20', 450.00),  
(105, 4, '2023-04-15', 100.00),  
(106, 5, '2023-05-30', 250.00);
```

4. Check the queries.

```
1 -- Create the customers table
2 CREATE OR REPLACE TABLE customers (
3     customer_id INT,
4     customer_name STRING,
5     city STRING,
6     country STRING,
7     signup_date DATE
8 );
9
10 -- Insert sample data into the customers table
11 INSERT INTO customers (customer_id, customer_name, city, country, signup_date) VALUES
12 (1, 'John Doe', 'New York', 'USA', '2023-01-15'),
13 (2, 'Jane Smith', 'Los Angeles', 'USA', '2023-02-20'),
14 (3, 'Carlos Ortega', 'Madrid', 'Spain', '2023-03-10'),
15 (4, 'Maria Garcia', 'Barcelona', 'Spain', '2023-04-05'),
16 (5, 'Liu Wei', 'Shanghai', 'China', '2023-05-25');
17
18 -- Create the orders table
19 CREATE OR REPLACE TABLE orders (
```

| | number of rows inserted |
|---|-------------------------|
| 1 | 6 |

Query Details

- Query duration: 1.9s
- Rows: 1

5. Simple Select Query.

Query: EXPLAIN

SELECT * FROM customers;

```
1 EXPLAIN
2 SELECT * FROM customers;
```

| | step | id | parentOperators | operation | objects | alias | expressions |
|---|------|------|-----------------|-------------|-----------------------------------|-------|--|
| 1 | null | null | null | GlobalStats | null | null | null |
| 2 | 1 | 0 | null | Result | null | null | CUSTOMERS.CUSTOMER_ID, CUSTOMERS.CUSTOMER_NAME, CUSTOMERS.CITY, CUSTOMERS.COUNTRY, CUSTOMERS.SIGNUP_DATE |
| 3 | 1 | 1 | [0] | TableScan | SAMPLE_SNOW."Snowflake".CUSTOMERS | null | CUSTOMER_ID, CUSTOMER_NAME, CITY, COUNTRY, SIGNUP_DATE |

Query Details

- Query duration: 85ms
- Rows: 3
- Query ID: 01b5112b-0001-2587-0...

6. Query with Filter Conditions.

Query: EXPLAIN

SELECT * FROM orders WHERE order_amount > 200.00;

```
1 EXPLAIN
2 SELECT * FROM orders WHERE order_amount > 200.00;
```

| | id | parentOperators | operation | objects | alias | expressions |
|---|------|-----------------|-------------|--------------------------------|-------|---|
| 1 | null | null | GlobalStats | null | null | null |
| 2 | 0 | null | Result | null | null | ORDERS.ORDER_ID, ORDERS.CUSTOMER_ID, ORDERS.CITY, ORDERS.COUNTRY, ORDERS.SIGNUP_DATE, ORDERS.ORDER_AMOUNT |
| 3 | 1 | [0] | Filter | null | null | ORDERS.ORDER_AMOUNT > 200 |
| 4 | 2 | [1] | TableScan | SAMPLE_SNOW."Snowflake".ORDERS | null | ORDER_ID, CUSTOMER_ID, CITY, COUNTRY, SIGNUP_DATE, ORDER_AMOUNT |

Query Details

- Query duration: 63ms
- Rows: 4
- Query ID: 01b5112c-0001-2586-0...

7. The TABLE SCAN reads all rows, and the FILTER reduces the number of rows to those that match the condition (order_amount > 200.00).
8. Join Query.

Query: EXPLAIN
 SELECT c.customer_name, o.order_id, o.order_amount
 FROM customers c
 JOIN orders o ON c.customer_id = o.customer_id
 WHERE c.country = 'USA';

Query Details

Query duration: 92ms

Rows: 7

Query ID: 01b5112e-0001-2587-0...

| step | id | parentOperators | operation | objects | alias | expressions |
|------|------|-----------------|-------------|-----------|-------------------------------------|---|
| 1 | null | null | GlobalStats | null | null | null |
| 2 | 1 | 0 | Result | null | null | C.CUSTOMER_NAME, O.ORDER_ID, O.ORDER_AMOUNT |
| 3 | 1 | 1 | InnerJoin | null | null | joinKey: (C.CUSTOMER_ID = O.CUSTOMER_ID) |
| 4 | 1 | 2 | Filter | null | null | C.COUNTRY = 'USA' |
| 5 | 1 | 3 | TableScan | SAMPLE_ C | CUSTOMER_ID, CUSTOMER_NAME, COUNTRY | |
| 6 | 1 | 4 | JoinFilter | null | null | joinKey: (C.CUSTOMER_ID = O.CUSTOMER_ID) |
| 7 | 1 | 5 | TableScan | SAMPLE_ O | ORDER_ID, CUSTOMER_ID, ORDER_AMOUNT | |

9. The join operation combines the relevant rows from both tables based on the customer_id and filters out customers from the USA.
10. Aggregation Query.

Query: EXPLAIN
 SELECT customer_id, SUM(order_amount) AS total_spent
 FROM orders
 GROUP BY customer_id;

Query Details

Query duration: 25ms

Rows: 7

Query ID: 01b5112f-0001-2586-0...

| step | id | parentOperators | operation | objects | alias | expressions |
|------|------|-----------------|-------------|--------------------------------|-------|---|
| 1 | null | null | GlobalStats | null | null | null |
| 2 | 1 | 0 | Result | null | null | ORDERS.CUSTOMER_ID, ORDER_AMOUNT |
| 3 | 1 | 1 | Aggregate | null | null | aggExprs: [SUM(ORDERS.CUSTOMER_ID, ORDER_AMOUNT)] |
| 4 | 1 | 2 | TableScan | SAMPLE_SNOW."Snowflake".ORDERS | null | CUSTOMER_ID, ORDER_AMOUNT |

11. The aggregation groups the orders by customer_id and calculates the total amount spent by each customer.

Data Spilling

1. Data spilling in Snowflake occurs when the amount of data processed by a query exceeds the available memory in the query processing tier.
2. Snowflake automatically manages this by temporarily writing excess data to disk.
3. This is transparent to the user but can affect query performance if spilling occurs frequently.
4. Create a large dataset for demonstration.

Query: CREATE OR REPLACE TABLE large_table AS
SELECT SEQ4() AS id, RPAD('x', 1000, 'x') AS big_column
FROM TABLE(Generator(rowcount => 1000000));

The screenshot shows the Snowflake query editor interface. At the top, there's a dropdown menu set to "SAMPLE_SNOW" and "Snowflake". Below it, the query text is: `1 CREATE OR REPLACE TABLE large_table AS`
`2 SELECT SEQ4() AS id, RPAD('x', 1000, 'x') AS big_column`
`3 FROM TABLE(Generator(rowcount => 1000000));`

Below the query editor, there's a "Results" tab. The results table has one row with the status: "Table LARGE_TABLE successfully created." To the right of the results, there's a "Query Details" panel showing "Query duration".

5. Query that might cause spilling (intentionally forcing large memory consumption).

Query: SELECT COUNT(*) FROM large_table;

The screenshot shows the Snowflake query editor interface. At the top, there's a dropdown menu set to "SAMPLE_SNOW" and "Snowflake". Below it, the query text is: `1 SELECT COUNT(*) FROM large_table;`

Below the query editor, there's a "Results" tab. The results table has one row with the count: "1000000". To the right of the results, there's a "Query Details" panel showing "Query duration" as "483ms".


Use of the Data Cache

1. Snowflake utilizes a data cache to improve query performance by storing frequently accessed data in memory.
2. This reduces the need to access data from disk repeatedly, speeding up query execution times for recurring queries.
3. Perform a query that might benefit from caching.

Query: SELECT COUNT(*) FROM large_table;

The screenshot shows the Snowflake query editor interface. At the top, there's a dropdown menu set to "SAMPLE_SNOW" and "Snowflake". Below it, the query text is: `1 SELECT COUNT(*) FROM large_table;`

Below the query editor, there's a "Results" tab. The results table has one row with the count: "1000000". To the right of the results, there's a "Query Details" panel showing "Query duration".

- 
- The screenshot shows a SQL query editor interface. At the top, there's a dropdown menu with "SAMPLE_SNOW" and "Snowflake" selected, and a "Settings" button. Below this, the SQL query is entered: `1 SELECT COUNT(*) FROM large_table;`. The results pane at the bottom shows a table with one column labeled **COUNT(*)** and one row with the value **1000000**. To the right of the results table, there's a "Query Details" section showing "Query duration" as 4 seconds. The interface includes a "Results" tab and a "Chart" tab, and a search icon is visible in the top right of the results pane.
- | COUNT(*) |
|----------|
| 1000000 |
- Query Details
- Query duration 4

- Query: `SELECT * FROM large_table WHERE id = 1000;`

[illegible]

1. Snowflake uses micro-partitions to store and manage data efficiently.
2. Micro-partition pruning is the process where Snowflake optimizes query performance by skipping unnecessary micro-partitions based on query filters, thus reducing the amount of data scanned.
3. Query with micro-partition pruning.

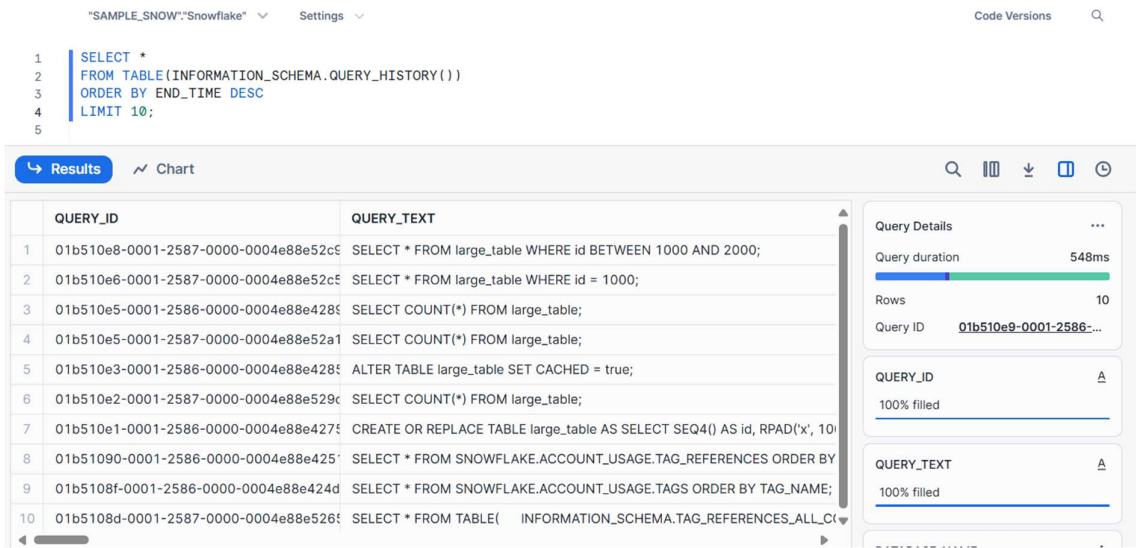
4. In this example, Snowflake will only scan the micro-partitions containing data relevant to the id range specified (1000 to 2000), ignoring other micro-partitions.

[illegible]

Query History

- 1. Snowflake keeps a history of queries executed within your account, which can be useful for auditing, performance analysis, and troubleshooting.
- 2. Users can access their query history to review past queries and their outcomes.
- 3. View recent queries in the account.

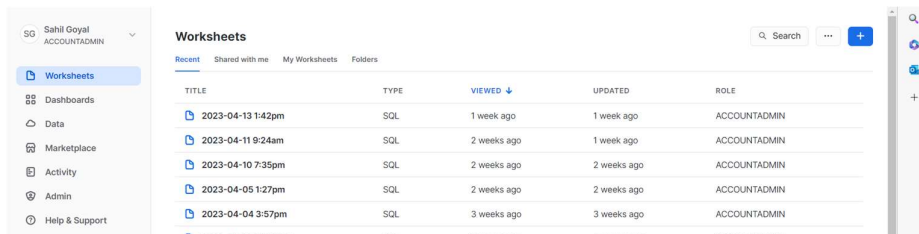
Query: `SELECT * FROM TABLE(INFORMATION_SCHEMA.QUERY_HISTORY())
ORDER BY END_TIME DESC
LIMIT 10;`



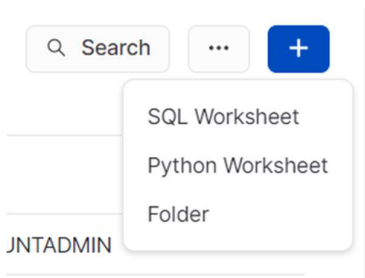
- 4. This query retrieves the 10 most recent queries executed in the Snowflake account, showing details such as start time, end time, duration, and status.

Create Warehouse

1. Login into Snowflake and click on the plus symbol to open the worksheet as shown below.



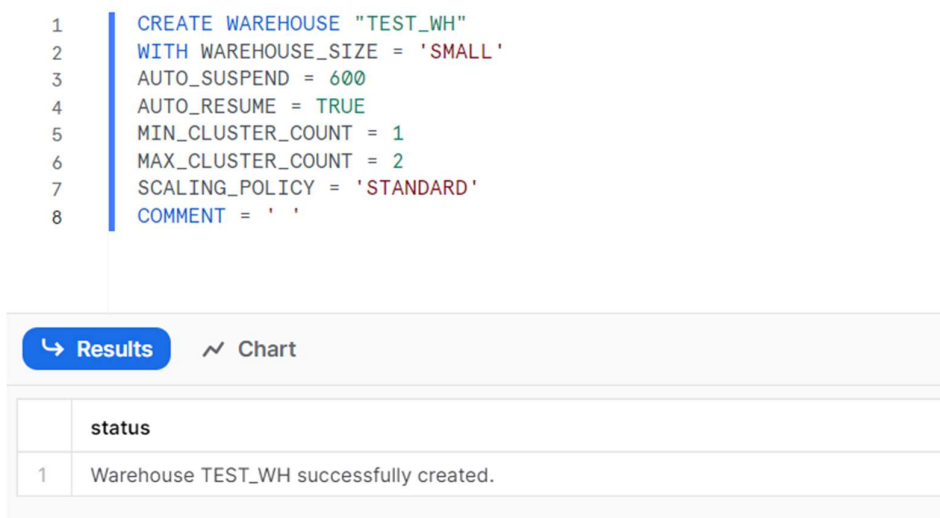
2. Now choose SQL Worksheet.



3. Use the following query to create a warehouse TEST_WH.

```
Query: CREATE WAREHOUSE "TEST_WH"  
WITH WAREHOUSE_SIZE = 'SMALL'  
AUTO_SUSPEND = 600  
AUTO_RESUME = TRUE  
MIN_CLUSTER_COUNT = 1  
MAX_CLUSTER_COUNT = 2  
SCALING_POLICY = 'STANDARD'  
COMMENT = ''
```

4. Click Run to execute the query. The result will be displayed in the Results panel as the warehouse "TEST_WH" was successfully created.



5. To alter/modify the warehouse, use the following query and run it.

Query: ALTER WAREHOUSE "TEST_WH"
SET WAREHOUSE_SIZE = 'SMALL'
AUTO_SUSPEND = 1200
AUTO_RESUME = TRUE
MIN_CLUSTER_COUNT = 1
MAX_CLUSTER_COUNT = 1
SCALING_POLICY = 'STANDARD'
COMMENT = ''

```
1 ALTER WAREHOUSE "TEST_WH"  
2 SET WAREHOUSE_SIZE = 'SMALL'  
3 AUTO_SUSPEND = 1200  
4 AUTO_RESUME = TRUE  
5 MIN_CLUSTER_COUNT = 1  
6 MAX_CLUSTER_COUNT = 1  
7 SCALING_POLICY = 'STANDARD'  
8 COMMENT = ''
```

| Results | | Chart |
|---------|----------------------------------|-------|
| status | | |
| 1 | Statement executed successfully. | |

6. To view all listed warehouses, the user can use the following SQL. It brings details of all listed warehouses.

Query: SHOW WAREHOUSES

```
1 SHOW WAREHOUSES
```

Results

Chart

| | name | state | type | size | min_cluster_count | ... | max_cluster_count | start |
|---|------------|-----------|----------|---------|-------------------|-----|-------------------|-------|
| 1 | COMPUTE_WH | SUSPENDED | STANDARD | X-Small | 1 | | 1 | |
| 2 | TEST_WH | STARTED | STANDARD | Small | 1 | | 1 | |

7. To suspend a warehouse, use the following SQL.

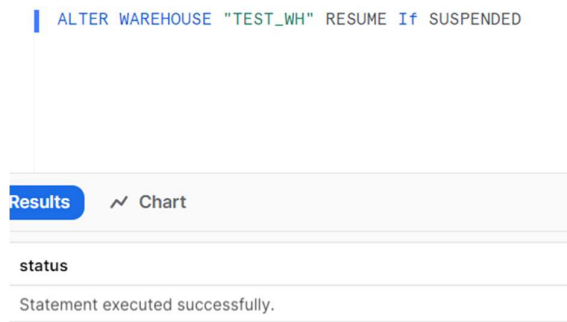
Query: ALTER WAREHOUSE TEST_WH SUSPEND

```
1 ALTER WAREHOUSE TEST_WH SUSPEND
```

| Results | | Chart |
|---------|----------------------------------|-------|
| status | | |
| 1 | Statement executed successfully. | |

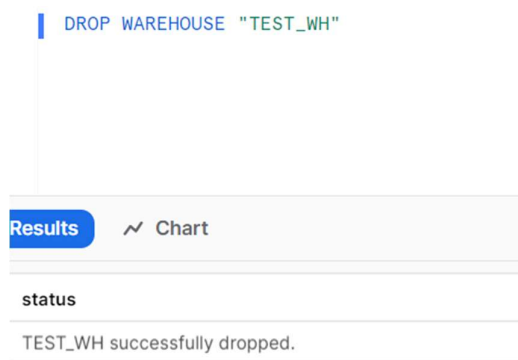
8. To resume a warehouse, use the following SQL.

Query: ALTER WAREHOUSE "TEST_WH" RESUME If SUSPENDED



9. To delete a warehouse, use the following SQL.

Query: DROP WAREHOUSE "TEST_WH"



Resource Monitors

1. Resource Monitors can also be created using the CREATE RESOURCE MONITOR command.
2. The below image shows an example of a resource monitor with a default schedule.

Query: CREATE RESOURCE MONITOR "RM_DEMO" WITH CREDIT_QUOTA = 100

TRIGGERS

ON 90 PERCENT DO SUSPEND

ON 95 PERCENT DO SUSPEND_IMMEDIATE

ON 70 PERCENT DO NOTIFY

ON 80 PERCENT DO NOTIFY;



3. Once the resource monitor is created, warehouses can be assigned to it as shown below.

Query: ALTER WAREHOUSE "COMPUTE_WH" SET RESOURCE_MONITOR = "RM_DEMO";

The screenshot shows the Snowflake SQL editor with the query: `ALTER WAREHOUSE "COMPUTE_WH" SET RESOURCE_MONITOR = "RM_DEMO";`. The results pane displays a single row with the status "Statement executed successfully." and a query duration of 67ms.

| status |
|----------------------------------|
| Statement executed successfully. |

4. Email Notifications for Non-Admin users cannot be enabled directly from the Web interface. It can only be enabled through SQL statements as shown below.

Query: CREATE RESOURCE MONITOR "RM_USER_ALERT" WITH CREDIT_QUOTA = 100
NOTIFY_USERS = ('SNOWFLAKE97098')
TRIGGERS
ON 90 PERCENT DO SUSPEND
ON 95 PERCENT DO SUSPEND_IMMEDIATE
ON 70 PERCENT DO NOTIFY
ON 80 PERCENT DO NOTIFY;

The screenshot shows the Snowflake SQL editor with the query: `CREATE RESOURCE MONITOR "RM_USER_ALERT" WITH CREDIT_QUOTA = 100 NOTIFY_USERS = ('SNOWFLAKE97098') TRIGGERS ON 90 PERCENT DO SUSPEND ON 95 PERCENT DO SUSPEND_IMMEDIATE ON 70 PERCENT DO NOTIFY ON 80 PERCENT DO NOTIFY;`. The results pane displays a single row with the status "Resource monitor RM_USER_ALERT successfully created." and a query duration of 80ms.

| status |
|--|
| Resource monitor RM_USER_ALERT successfully created. |

5. To view the list of users who were given access to email alerts of resource monitors, use below SQL command below.

Query: SHOW RESOURCE MONITORS;

The screenshot shows the Snowflake SQL editor with the query: `SHOW RESOURCE MONITORS;`. The results pane displays a table with two rows of resource monitor information.

| | name | credit_quota | used_credits | remaining_credits | level | frequency | start_time |
|---|---------------|--------------|--------------|-------------------|-----------|-----------|-----------------|
| 1 | RM_DEMO | 100.00 | 0.00 | 100.00 | WAREHOUSE | MONTHLY | 2024-05-31 17:0 |
| 2 | RM_USER_ALERT | 100.00 | 0.00 | 100.00 | null | MONTHLY | 2024-05-31 17:0 |

6. The users with Account Admin access by default have access to email alerts and they are not displayed under notify_users.

Query acceleration service

1. Create a demo warehouse.

```
Query: CREATE WAREHOUSE "DEMO_WH"  
WITH WAREHOUSE_SIZE = 'SMALL'  
AUTO_SUSPEND = 600  
AUTO_RESUME = TRUE  
MIN_CLUSTER_COUNT = 1  
MAX_CLUSTER_COUNT = 2  
SCALING_POLICY = 'STANDARD'  
COMMENT = ' ' +
```

The screenshot shows the Snowflake SQL Editor interface. The query editor at the top contains the SQL code to create a warehouse named 'DEMO_WH'. Below the editor, the 'Results' tab is active, displaying a single row with the status 'Warehouse DEMO_WH successfully created.' To the right, the 'Query Details' panel shows the query duration as 102ms.

| status |
|---|
| Warehouse DEMO_WH successfully created. |

Query Details
Query duration: 102ms

2. Now go to Warehouses and edit the Demo_WH warehouse.

The screenshot shows the 'Edit Warehouse' dialog box for the 'DEMO_WH' warehouse. The dialog has several settings: 'Auto suspend' is checked with a suspend time of 10 minutes; 'Multi-cluster Warehouse' is checked with 1 min cluster and 2 max clusters; 'Query Acceleration' is checked with a scale factor of x8. The 'Save Warehouse' button is at the bottom right.

Edit Warehouse
DEMO_WH as ACCOUNTADMIN

- ☒ **Auto suspend**
Automatically suspends the warehouse if it is inactive for the specified period of time.
Suspend After: 10 min(s) of inactivity
- ☒ **Multi-cluster Warehouse**
Scale compute resources as query concurrency needs change. [Learn more](#)
Min Clusters: 1, Max Clusters: 2, Scaling Policy: Standard
- ☒ **Query Acceleration**
Accelerate outlier queries with additional flexible compute resources. [Learn more](#)
Scale Factor: x8

Cancel Save Warehouse

3. Go back to the worksheet. Run the below query under Compute_WH warehouse.

Query: USE SCHEMA snowflake_sample_data.tpcds_sf10tcl;

```
select store.s_store_id, item.i_item_id, sum(ss_sales_price) ss_sales_price  
from store_sales  
  ,item  
  ,time_dim, store  
where ss_sold_time_sk = time_dim.t_time_sk  
  and ss_item_sk = item.i_item_sk  
  and ss_store_sk = s_store_sk
```

```

and time_dim.t_hour = 8
and time_dim.t_minute >= 30
and store.s_store_name = 'ese'
group by store.s_store_id, item.i_item_id;

```

4. Here you can see it took 4.22 seconds.

The screenshot shows a Snowflake query interface. The query is executed against the 'COMPUTE_WH (X-Small)' warehouse. The results table shows 7 rows of data. The 'Query Details' panel on the right indicates a query duration of 4m 22s and 19.3M rows.

| | S_STORE_ID | I_ITEM_ID | SS_SALES_PRICE |
|---|-------------------|------------------|----------------|
| 1 | AAAAAAAAEMDAAAAA | AAAAAAAAAFOIEAAA | 78.48 |
| 2 | AAAAAAAAAKHCAAAAA | AAAAAAAAAJDJBAAA | 10.65 |
| 3 | AAAAAAAAAQAIAAAA | AAAAAAAAALOLBAAA | 828.40 |
| 4 | AAAAAAAAAGDFAAAA | AAAAAAAAAPEPNAAA | 121.43 |
| 5 | AAAAAAAAAPEAAAAA | AAAAAAAAAPOEBAAA | 214.91 |
| 6 | AAAAAAAAELEAAAAA | AAAAAAAAALKBGAAA | 149.75 |
| 7 | AAAAAACODAAAAA | AAAAAAMDONCAAA | 186.93 |

5. Next run the same query under Demo_WH warehouse.

6. Now check the time here. It took only 83ms

The screenshot shows the same Snowflake query interface, but the query is now executed against the 'Demo_WH' warehouse. The results table is identical to the previous one. The 'Query Details' panel on the right indicates a query duration of 83ms and 19.3M rows.

| | S_STORE_ID | I_ITEM_ID | SS_SALES_PRICE |
|---|-------------------|------------------|----------------|
| 1 | AAAAAAAAEMDAAAAA | AAAAAAAAAFOIEAAA | 78.48 |
| 2 | AAAAAAAAAKHCAAAAA | AAAAAAAAAJDJBAAA | 10.65 |
| 3 | AAAAAAAAAQAIAAAA | AAAAAAAAALOLBAAA | 828.40 |
| 4 | AAAAAAAAAGDFAAAA | AAAAAAAAAPEPNAAA | 121.43 |
| 5 | AAAAAAAAAPEAAAAA | AAAAAAAAAPOEBAAA | 214.91 |
| 6 | AAAAAAAAELEAAAAA | AAAAAAAAALKBGAAA | 149.75 |
| 7 | AAAAAACODAAAAA | AAAAAAMDONCAAA | 186.93 |

Materialized Views

1. Materialized views in Snowflake are precomputed result sets stored physically in the database.
2. They are helpful for improving query performance by reducing the need to recompute complex queries or aggregations repeatedly.

Query: CREATE OR REPLACE MATERIALIZED VIEW EmpView as
select GENDER, Sum(SALARY) as TotalSalary from EMPLOYEE_DETAILS Group by GENDER;

"SAMPLE_SNOW"."Snowflake" Settings Code Versions

```
1 CREATE OR REPLACE MATERIALIZED VIEW EmpView as
2 select GENDER, Sum(SALARY) as TotalSalary from EMPLOYEE_DETAILS Group by GENDER;
3
```

Results Chart

| status | |
|--------|---|
| 1 | Materialized view EMPVIEW successfully created. |

Query Details

Query duration 70

3. Now check the view data.

Query: Select * from EmpView;

"SAMPLE_SNOW"."Snowflake" Settings Code Versions

```
1 Select * from EmpView;
```

Results Chart

| | GENDER | TOTALSALARY |
|---|--------|-------------|
| 1 | male | 410000 |
| 2 | female | 425000 |

Query Details

Query duration 3

Rows

Non Materialized Views

1. Below is an example of a non-materialized view created on top of the Employee_Details table selecting only the details(fields) which are required for an Employee_View.

Query: Create or Replace view Employee_View as select ID, Name, Age, Salary, Gender, Email_Id from Employee_Details

"SAMPLE_SNOW"."Snowflake" Updated 2 se

```
1 Create or Replace view Employee_View as select ID, Name, Age, Salary, Gender, Email_Id from EMPLOYEE_DETAILS
```

Results Chart

| status | |
|--------|--|
| | View EMPLOYEE_VIEW successfully created. |

Query Details

Query duration

2. Check the view data.

The screenshot shows a Snowflake query editor with the following query:

```
1 | Select * from Employee_View
```

The results are displayed in a table with 6 rows and 7 columns:

| | ID | NAME | AGE | SALARY | GENDER | EMAIL_ID |
|---|----|-----------|-----|--------|--------|---------------------|
| 1 | 6 | rebanta | 26 | 160000 | male | rebanta@gmail.com |
| 2 | 1 | sagar | 24 | 120000 | male | sagar@gmail.com |
| 3 | 2 | ratan | 22 | 130000 | male | ratan@gmail.com |
| 4 | 3 | sakshi | 23 | 125000 | female | sakshi@gmail.com |
| 5 | 4 | sanskriti | 24 | 145000 | female | sanskriti@gmail.com |
| 6 | 5 | roshi | 25 | 155000 | female | roshi@gmail.com |

Query Details:

- Query duration: 188ms
- Rows: 6
- Query ID: 01b510f1-0001-2586-0...

3. SHOW VIEWS command in Snowflake lists the views, including secure views, for which you have access privileges.

Query: SHOW VIEWS

The screenshot shows a Snowflake query editor with the following query:

```
1 | SHOW VIEWS
```

The results are displayed in a table with 3 rows and 7 columns:

| | created_on | name | reserved | database_name | schema_name | owner |
|---|-------------------------------|---------------|----------|---------------|-------------|--------------|
| 1 | 2024-06-17 01:48:38.333 -0700 | EMPLOYEE_VIEW | | SAMPLE_SNOW | Snowflake | ACCOUNTADMIN |
| 2 | 2024-06-17 01:45:15.425 -0700 | EMPVIEW | | SAMPLE_SNOW | Snowflake | ACCOUNTADMIN |
| 3 | 2024-06-16 22:52:05.741 -0700 | ORG_PATIENTS | | SAMPLE_SNOW | Snowflake | ACCOUNTADMIN |

Query Details:

- Query duration: 39ms
- Rows: 3
- Query ID: 01b510f1-0001-2586-0...

SELECT Commands

1. Use the following query to bring limited data in the Select statement.

Query: SELECT * from Employee_Details Limit 3

2. This query will display only the first 3 rows.

The screenshot shows a Snowflake query editor with the following query:

```
1 | SELECT * from Employee_Details Limit 3
```

The results are displayed in a table with 3 rows and 8 columns:

| | ID | NAME | AGE | SALARY | GENDER | CONTACT_NUMBER | EMAIL_ID |
|---|----|--------|-----|---------|--------|----------------|------------------|
| 1 | 1 | sagar | 24 | 120,000 | male | 89,942,345 | sagar@gmail.com |
| 2 | 2 | ratan | 22 | 130,000 | male | 80,052,345 | ratan@gmail.com |
| 3 | 3 | sakshi | 23 | 125,000 | female | 72,983,456 | sakshi@gmail.com |

3. Use the following query to display the usage of the last 10 days.

Query: SELECT * FROM TABLE
(INFORMATION_SCHEMA.DATABASE_STORAGE_USAGE_HISTORY
(DATEADD('days', -10, CURRENT_DATE()), CURRENT_DATE()))

"SAMPLE_SNOW"."Snowflake" Settings Code Versions

```

1 SELECT * FROM TABLE (INFORMATION_SCHEMA.DATABASE_STORAGE_USAGE_HISTORY
2 (DATEADD('days', -10, CURRENT_DATE()), CURRENT_DATE()))
3

```

Results Chart

| | USAGE_DATE | DATABASE_NAME | AVERAGE_DATABASE_BYTES | AVERAGE_FAILSAFE_BYTES |
|---|------------|---------------|------------------------|------------------------|
| 1 | 2024-06-14 | MY_AWESOME_DB | 1022 | 0 |
| 2 | 2024-06-15 | MY_AWESOME_DB | 1021 | 0 |
| 3 | 2024-06-16 | MY_AWESOME_DB | 1021 | 0 |
| 4 | 2024-06-17 | MY_AWESOME_DB | 1023 | 0 |
| 5 | 2024-06-14 | SAMPLE_SNOW | 18524 | 415 |
| 6 | 2024-06-15 | SAMPLE_SNOW | 5329 | 13609 |
| 7 | 2024-06-16 | SAMPLE_SNOW | 5603 | 13821 |
| 8 | 2024-06-17 | SAMPLE_SNOW | 12278 | 13823 |

Query Details

Query duration 360ms

Rows 8

Query ID 01b510f4-0001-2586-0...

USAGE_DATE

2024-06-14 2024-06-17

4. Now select the SNOWFLAKE_SAMPLE_DATA database and TPC_SF1 schema as shown below.

SNOWFLAKE_SAMPLE_DATA.TPCH_SF1 Settings

Databases

- SAMPLE_SNOW
- IIICS
- Sample_Snow
- SNOWFLAKE
- SNOWFLAKE_SAMPLE_DATA**
- TEST

Schemas

- INFORMATION_SCHEMA
- TPCDS_SF100TCL
- TPCDS_SF10TCL
- TPCH_SF1**
- TPCH_SF10
- TPCH_SF100

5. To check variables, run the following queries in sequence.

First Query: SELECT * FROM snowflake_sample_data.tpch_sf1.region
JOIN snowflake_sample_data.tpch_sf1.nation
ON r_regionkey = n_regionkey;

```

1 SELECT * FROM snowflake_sample_data.tpch_sf1.region
2 JOIN snowflake_sample_data.tpch_sf1.nation
3 ON r_regionkey = n_regionkey;

```

Results Chart

| | R_REGIONKEY | R_NAME | R_COMMENT |
|----|-------------|-------------|--|
| 1 | 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular |
| 2 | 1 | AMERICA | hs use ironic, even requests. s |
| 3 | 1 | AMERICA | hs use ironic, even requests. s |
| 4 | 1 | AMERICA | hs use ironic, even requests. s |
| 5 | 4 | MIDDLE EAST | uickly special accounts cajole carefully blithely close requests. carefully final asymp |
| 6 | 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular |
| 7 | 3 | EUROPE | ly final courts cajole furiously final excuse |
| 8 | 3 | EUROPE | ly final courts cajole furiously final excuse |
| 9 | 2 | ASIA | ges. thinly even pinto beans ca |
| 10 | 2 | ASIA | ges. thinly even pinto beans ca |

Second Query: `select * from table(result_scan(last_query_id()));`

SNOWFLAKE_SAMPLE_DATA.TPCH_SF1 Settings

```
1 | select * from table(result_scan(last_query_id()));
```

Results Chart

| | R_REGIONKEY | R_NAME | R_COMMENT | ... |
|--|-------------|-------------|--|-----|
| | 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular | |
| | 1 | AMERICA | hs use ironic, even requests. s | |
| | 1 | AMERICA | hs use ironic, even requests. s | |
| | 1 | AMERICA | hs use ironic, even requests. s | |
| | 4 | MIDDLE EAST | uickly special accounts cajole carefully blithely close requests. carefully final asymp | |
| | 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular | |
| | 3 | EUROPE | ly final courts cajole furiously final excuse | |
| | 3 | EUROPE | ly final courts cajole furiously final excuse | |

Third Query: `SELECT * FROM snowflake_sample_data.tpch_sf1.region
JOIN snowflake_sample_data.tpch_sf1.nation
ON r_regionkey = n_regionkey;`

```
1 | SELECT * FROM snowflake_sample_data.tpch_sf1.region  
2 | JOIN snowflake_sample_data.tpch_sf1.nation  
3 | ON r_regionkey = n_regionkey;
```

Results Chart

| | R_REGIONKEY | R_NAME | R_COMMENT | ... |
|----|-------------|-------------|--|-----|
| 1 | 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular | |
| 2 | 1 | AMERICA | hs use ironic, even requests. s | |
| 3 | 1 | AMERICA | hs use ironic, even requests. s | |
| 4 | 1 | AMERICA | hs use ironic, even requests. s | |
| 5 | 4 | MIDDLE EAST | uickly special accounts cajole carefully blithely close requests. carefully final asymp | |
| 6 | 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular | |
| 7 | 3 | EUROPE | ly final courts cajole furiously final excuse | |
| 8 | 3 | EUROPE | ly final courts cajole furiously final excuse | |
| 9 | 2 | ASIA | ges. thinly even pinto beans ca | |
| 10 | 2 | ASIA | ges. thinly even pinto beans ca | |

Forth Query: `SET q1 = LAST_QUERY_ID();`

SNOWFLAKE_SAMPLE_DATA.TPCH_SF1

```
1 | SET q1 = LAST_QUERY_ID();
```

Results Chart

| | status |
|---|----------------------------------|
| 1 | Statement executed successfully. |

Fifth Query: select \$q1;

SNOWFLAKE_SAMPLE_DATA.TPCH_SF1 ▾

select \$q1;

Results Chart

\$Q1

01abdd21-0000-f149-0001-3d26000301de

Sixth Query: SELECT * FROM TABLE(result_scan(\$q1)) ;

SNOWFLAKE_SAMPLE_DATA.TPCH_SF1 ▾

SELECT * FROM TABLE(result_scan(\$q1));

Results Chart

| R_REGIONKEY | R_NAME | R_COMMENT |
|-------------|-------------|--|
| 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular |
| 1 | AMERICA | hs use ironic, even requests. s |
| 1 | AMERICA | hs use ironic, even requests. s |
| 1 | AMERICA | hs use ironic, even requests. s |
| 4 | MIDDLE EAST | uickly special accounts cajole carefully blithely close requests. carefully final asymp |
| 0 | AFRICA | lar deposits. blithely final packages cajole. regular waters are final requests. regular |
| 3 | EUROPE | ly final courts cajole furiously final excuse |

Seventh Query: SHOW VARIABLES;

"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾ Code Versions 🔍

1 SHOW VARIABLES;

Results Chart

| | session_id | created_on | updated_on | name | value |
|---|-------------|-------------------------------|-------------------------------|------|---------------------------|
| 1 | 21081505821 | 2024-06-17 01:53:50.146 -0700 | 2024-06-17 01:53:50.170 -0700 | Q1 | 01b510f4-0001-2586-0000-0 |

Query Details ...

Query duration 30ms

6. Use the following command to see all the columns in the table.

Query: SHOW COLUMNS in table Employee_Details

"SAMPLE_SNOW":Snowflake Settings

```
SHOW COLUMNS in table Employee_Details
```

Results Chart

| table_name | schema_name | column_name | data_type |
|------------------|-------------|----------------|---|
| EMPLOYEE_DETAILS | Snowflake | ID | { "type": "FIXED", "precision": 38, "scale": 0, "nullable": true } |
| EMPLOYEE_DETAILS | Snowflake | NAME | { "type": "TEXT", "length": 30, "byteLength": 120, "nullable": true } |
| EMPLOYEE_DETAILS | Snowflake | AGE | { "type": "FIXED", "precision": 38, "scale": 0, "nullable": true } |
| EMPLOYEE_DETAILS | Snowflake | SALARY | { "type": "FIXED", "precision": 38, "scale": 0, "nullable": true } |
| EMPLOYEE_DETAILS | Snowflake | GENDER | { "type": "TEXT", "length": 30, "byteLength": 120, "nullable": true } |
| EMPLOYEE_DETAILS | Snowflake | CONTACT_NUMBER | { "type": "FIXED", "precision": 38, "scale": 0, "nullable": true } |
| EMPLOYEE_DETAILS | Snowflake | EMAIL_ID | { "type": "TEXT", "length": 30, "byteLength": 120, "nullable": true } |

7. Use the following command to show all the parameters provided by Snowflake.

Query: SHOW PARAMETERS;

"SAMPLE_SNOW":Snowflake Settings

```
1 SHOW PARAMETERS;
```

Results Chart

| | key | value | default |
|---|---|-------|---------|
| 1 | ABORT_DETACHED_QUERY | false | false |
| 2 | AUTOCOMMIT | true | true |
| 3 | AUTOCOMMIT_API_SUPPORTED | true | true |
| 4 | BINARY_INPUT_FORMAT | HEX | HEX |
| 5 | BINARY_OUTPUT_FORMAT | HEX | HEX |
| 6 | CLIENT_ENABLE_CONSERVATIVE_MEMORY_USAGE | true | true |
| 7 | CLIENT_ENABLE_DEFAULT_OVERWRITE_IN_PUT | false | false |
| 8 | CLIENT_ENABLE_LOG_INFO_STATEMENT_PARAMETERS | false | false |
| 9 | CLIENT_MEMORY_LIMIT | 1536 | 1536 |
| 0 | CLIENT_METADATA_REQUEST_USE_CONNECTION_CTX | false | false |
| 1 | CLIENT_METADATA_USE_SESSION_DATABASE | false | false |
| 2 | CLIENT_PREFETCH_THREADS | 4 | 4 |

8. Snowflake's search optimization service enhances query performance by optimizing search operations, including search predicates in queries.

Query: SELECT * FROM EMPLOYEE_DETAILS WHERE CONTAINS(GENDER, 'fe');

"SAMPLE_SNOW":Snowflake Settings Code Versions

```
1 SELECT * FROM EMPLOYEE_DETAILS WHERE CONTAINS(GENDER, 'fe');
```

Results Chart

| | ID | NAME | AGE | SALARY | GENDER | CONTACT_NUMBER | EMAIL_ID |
|---|----|-----------|-----|--------|--------|----------------|---------------------|
| 1 | 3 | sakshi | 23 | 125000 | female | 72983456 | sakshi@gmail.com |
| 2 | 4 | sanskriti | 24 | 145000 | female | 78453478 | sanskriti@gmail.com |
| 3 | 5 | roshi | 25 | 155000 | female | 93287456 | roshi@gmail.com |

Query Details ...

Query duration 140ms

Rows 3

Query ID 01b510f7-0001-2587-0...

Persisted Query Results

1. Persisted query results in Snowflake allow storing the results of a query for reuse or sharing purposes.
2. This feature can be useful for creating reports or sharing data without rerunning the query.

Query: CREATE OR REPLACE TABLE persisted_result AS

SELECT * FROM EMPLOYEE_DETAILS

WHERE GENDER = 'male';

3. Check the query.

The screenshot shows the Snowflake query editor interface. At the top, there are tabs for "SAMPLE_SNOW:" "Snowflake" and "Settings". On the right, there is a "Code Versions" link. The query editor contains the following SQL code:

```
1 CREATE OR REPLACE TABLE persisted_result AS
2 SELECT * FROM EMPLOYEE_DETAILS
3 WHERE GENDER = 'male';
```

Below the query editor, there is a "Results" tab and a "Chart" tab. The "Results" tab is active, showing a table with one row and one column:

| status |
|--|
| 1 Table PERSISTED_RESULT successfully created. |

On the right side of the "Results" tab, there is a "Query Details" panel. It shows "Query duration" as 5' and a progress bar.

4. Query the persisted result set.

Query: SELECT * FROM persisted_result;

The screenshot shows the Snowflake query editor interface. At the top, there are tabs for "SAMPLE_SNOW:" "Snowflake" and "Settings". On the right, there is a "Code Versions" link. The query editor contains the following SQL code:

```
1 SELECT * FROM persisted_result;
```

Below the query editor, there is a "Results" tab and a "Chart" tab. The "Results" tab is active, showing a table with 8 columns and 3 rows:

| | ID | NAME | AGE | SALARY | GENDER | CONTACT_NUMBER | EMAIL_ID |
|---|----|---------|-----|--------|--------|----------------|-------------------|
| 1 | 6 | rebanta | 26 | 160000 | male | 83465873 | rebanta@gmail.com |
| 2 | 1 | sagar | 24 | 120000 | male | 89942345 | sagar@gmail.com |
| 3 | 2 | ratan | 22 | 130000 | male | 80052345 | ratan@gmail.com |

On the right side of the "Results" tab, there is a "Query Details" panel. It shows "Query duration" as 262ms, "Rows" as 3, and "Query ID" as 01b510fa-0001-2586-Q...