Context

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

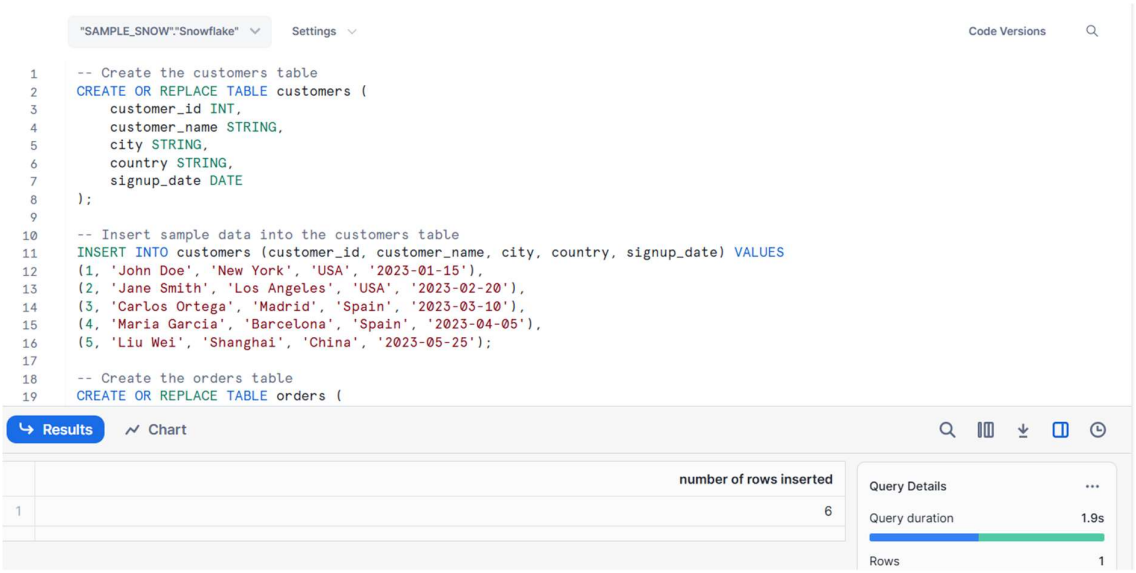
```
CREATE OR REPLACE TABLE customers (
   customer_id INT,
   customer_name STRING,
   city STRING,
   country STRING,
   signup_date DATE
);

-- 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);
```

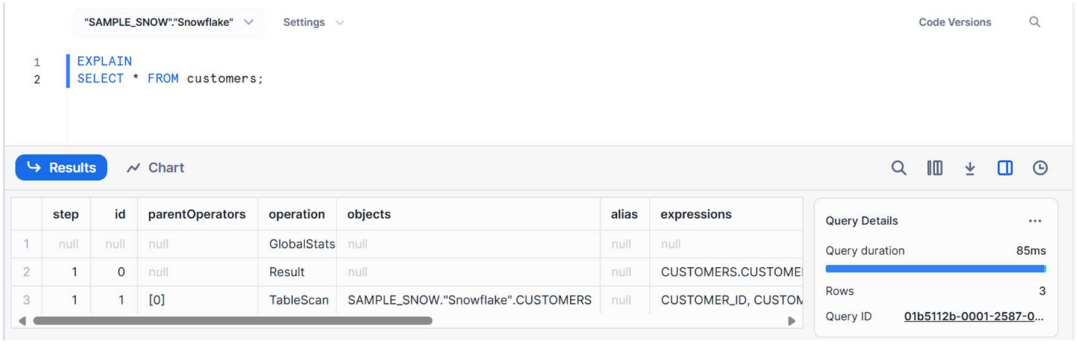
4. Check the queries.



5. Simple Select Query.

Query: EXPLAIN

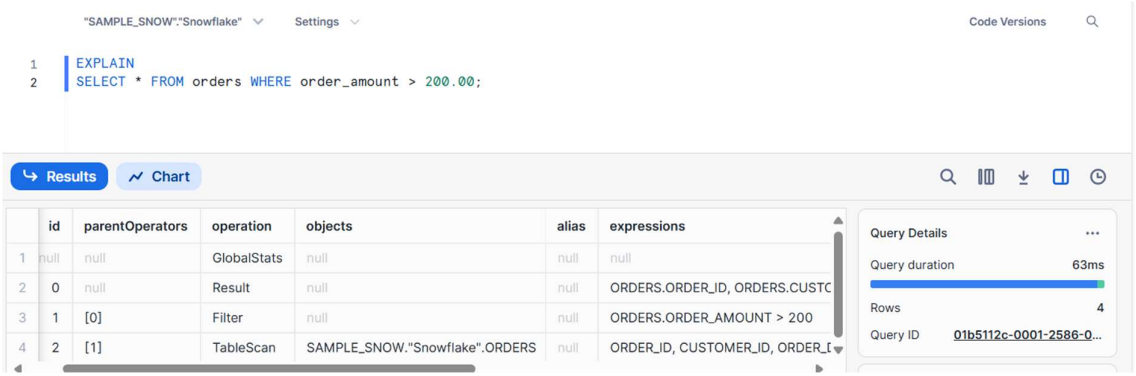
SELECT * FROM customers;



6. Query with Filter Conditions.

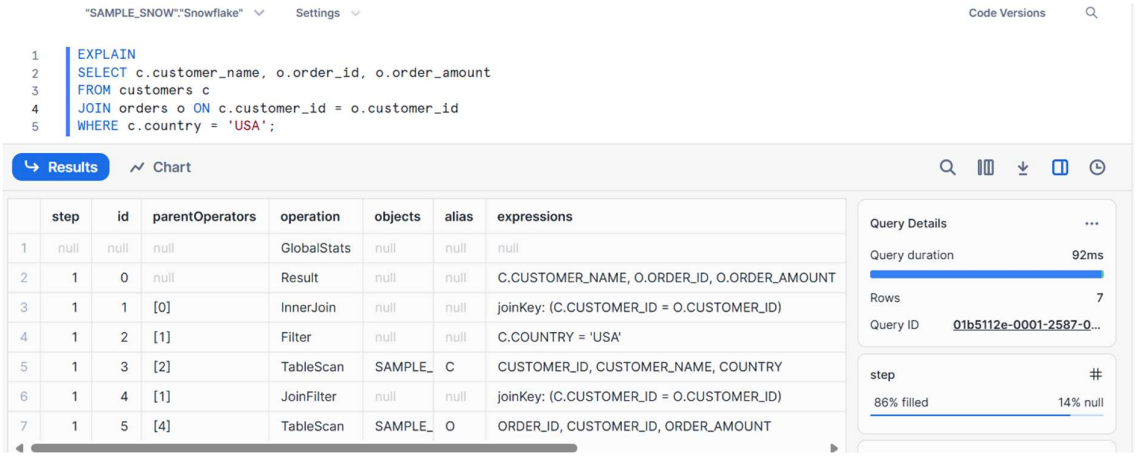
Query: EXPLAIN

SELECT * FROM orders WHERE order_amount > 200.00;



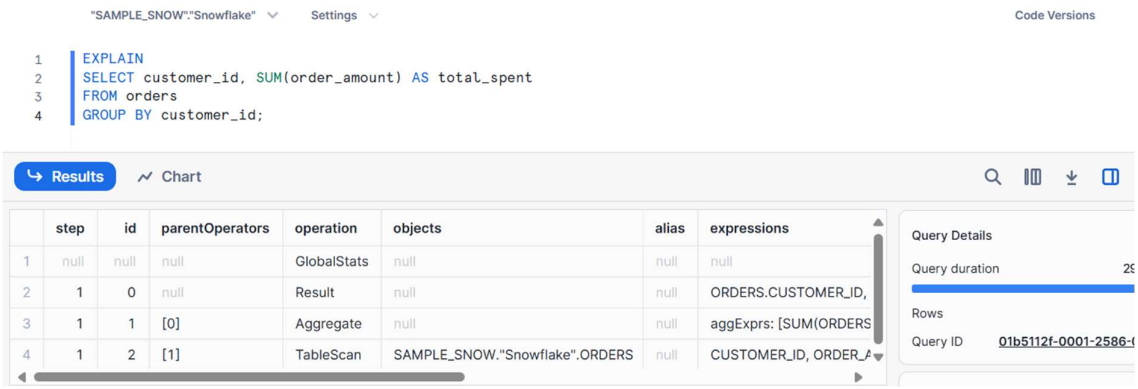
7. The TABLE SCAN reads all rows, and the FILTER reduces the number of rows to those that match the condition (order_amount > 200.00).
8. Join Query.

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';



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;



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

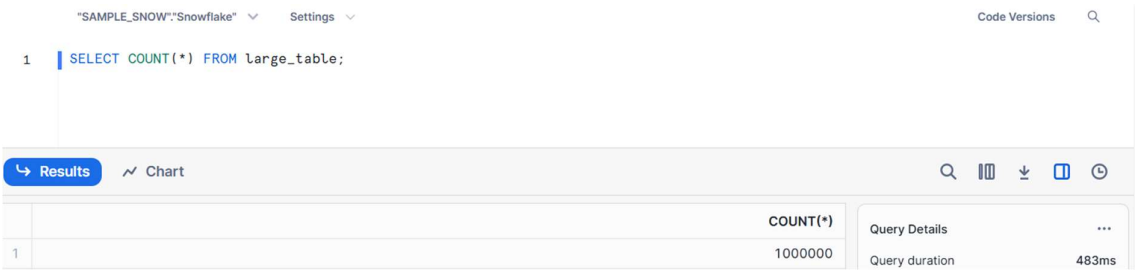
Data Spilling

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));

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

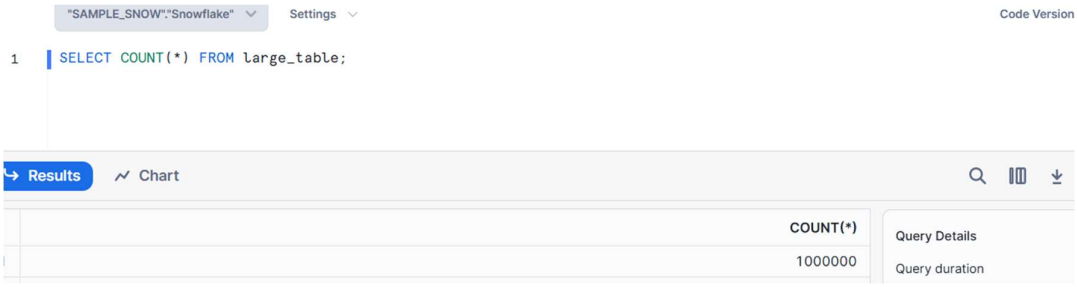
Query: SELECT COUNT(*) FROM large_table;



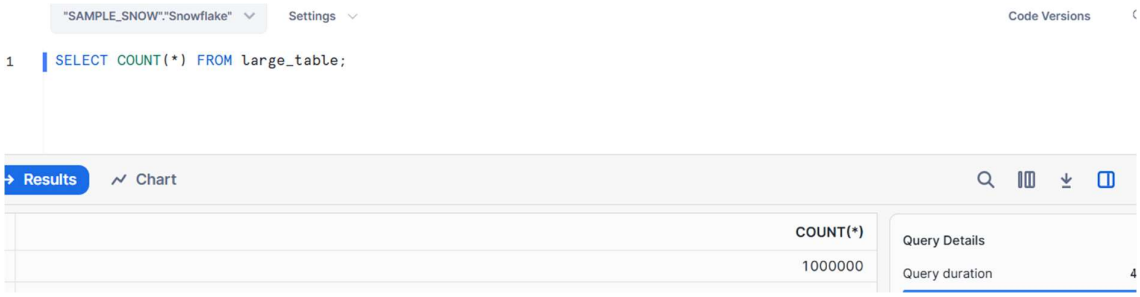
Use of the Data Cache

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;



4. Execute the same query again (result in cache demonstration).



5. Utilize the warehouse cache for faster access.

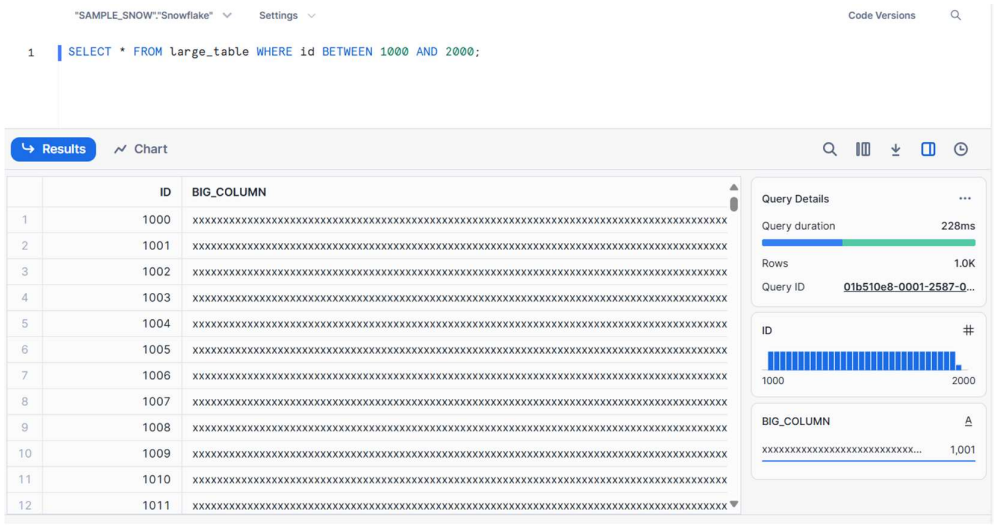
Query: SELECT * FROM large_table WHERE id = 1000;

Micro-Partition Pruning

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.

Query: SELECT * FROM large_table WHERE id BETWEEN 1000 AND 2000;

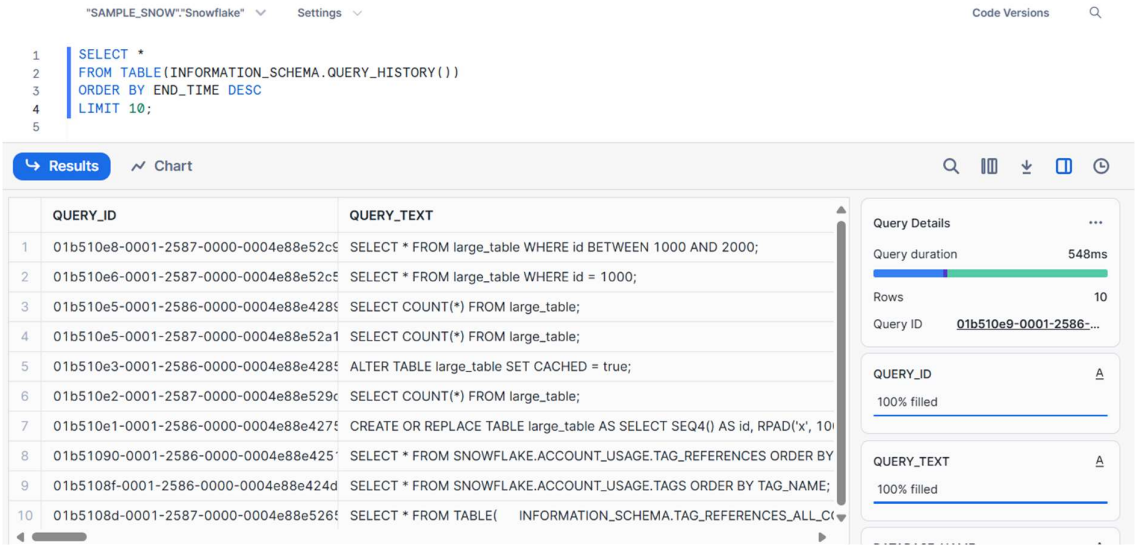
4. In this example, Snowflake will only scan the micro-partitions containing data relevant to the id range specified (1000 to 2000), ignoring other micro-partitions.



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.

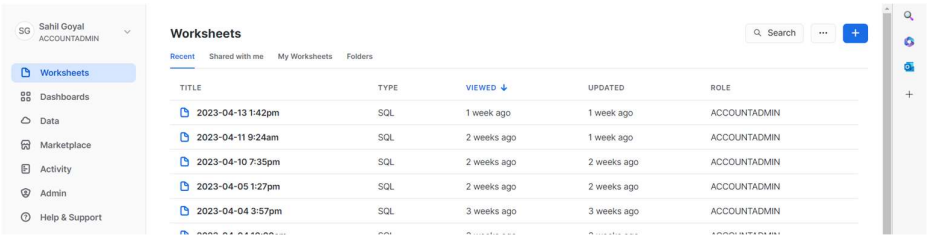
Query: SELECT * FROM TABLE(INFORMATION_SCHEMA.QUERY_HISTORY())
ORDER BY END_TIME DESC
LIMIT 10;



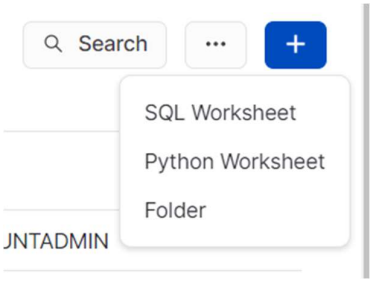
4. This query retrieves the 10 most recent queries executed in the Snowflake account, showing details such as start time, end time, duration, and status.

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.

```
1  CREATE WAREHOUSE "TEST_WH"
2  WITH WAREHOUSE_SIZE = 'SMALL'
3  AUTO_SUSPEND = 600
4  AUTO_RESUME = TRUE
5  MIN_CLUSTER_COUNT = 1
6  MAX_CLUSTER_COUNT = 2
7  SCALING_POLICY = 'STANDARD'
8  COMMENT = ' '
```

Results | Chart

	status
1	Warehouse TEST_WH 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
```

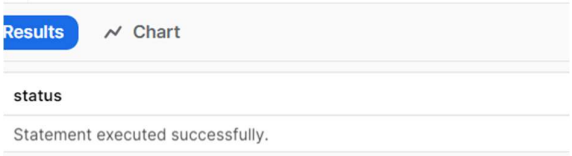
Results | Chart

	status
1	Statement executed successfully.

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

Query: ALTER WAREHOUSE "TEST_WH" RESUME If SUSPENDED

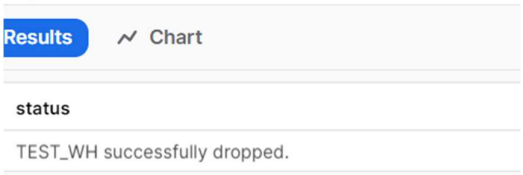
```
ALTER WAREHOUSE "TEST_WH" RESUME If SUSPENDED
```



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

Query: DROP WAREHOUSE "TEST_WH"

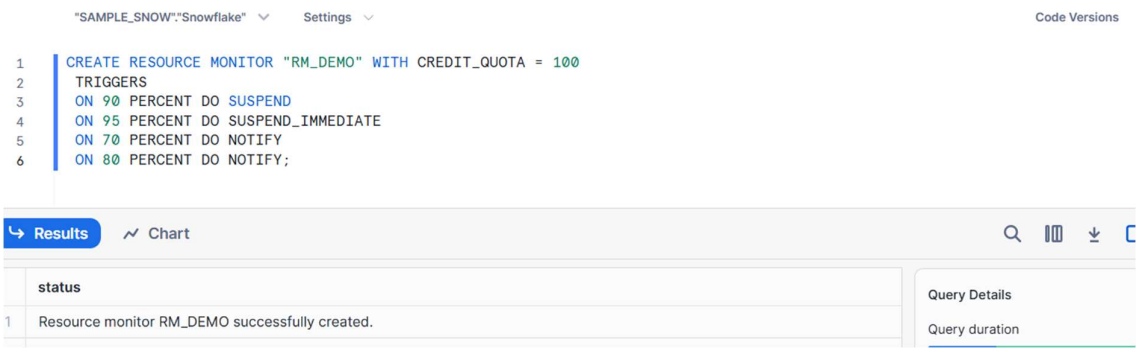
```
DROP WAREHOUSE "TEST_WH"
```



Resource Monitors

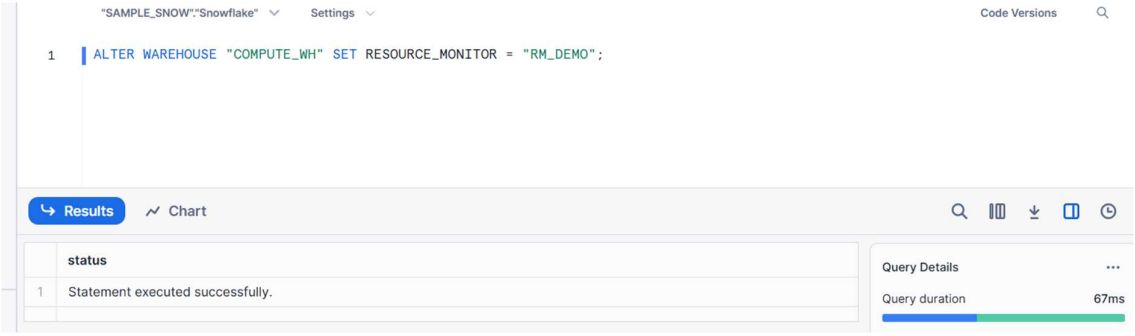
1. Resource Monitors can also be created using the CREATE RESOURCE MONITOR command.
2. The below image shows an example of a resource monitor with a default schedule.

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";

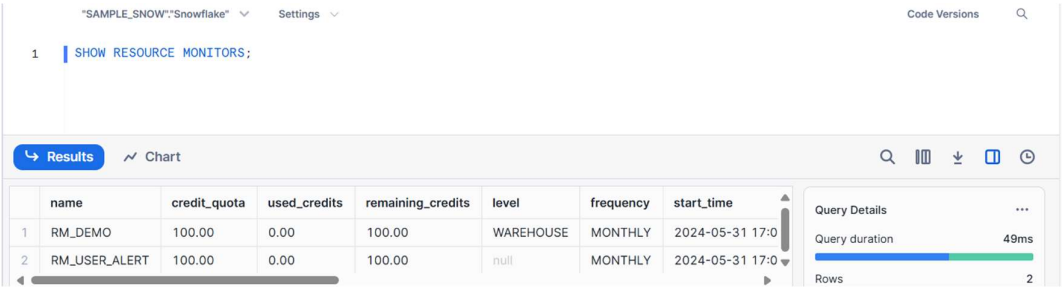


4. Email Notifications for Non-Admin users cannot be enabled directly from the Web interface. It can only enabled through SQL statements as shown below.

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;

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;

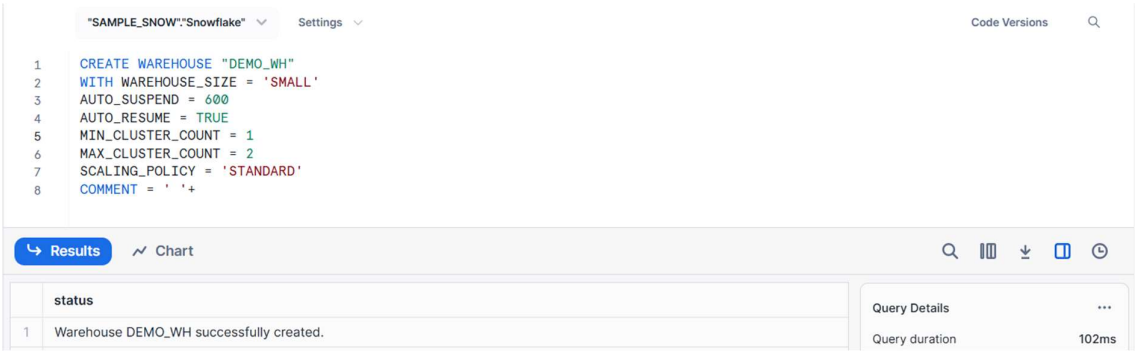


6. The users with Account Admin access by default have access to email alerts and they are not displayed under notify_users.

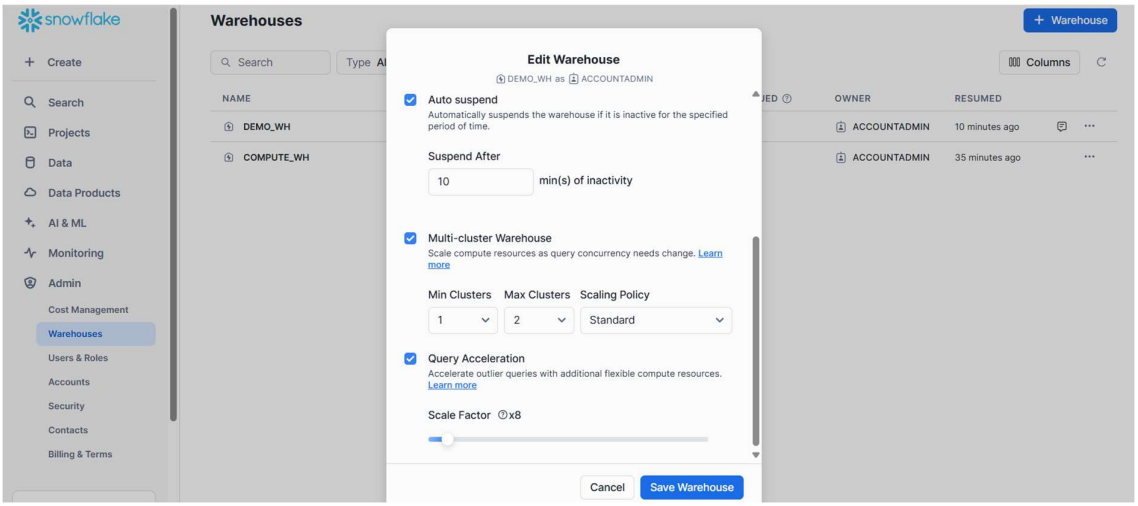
Query acceleration service

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 = ' '+



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



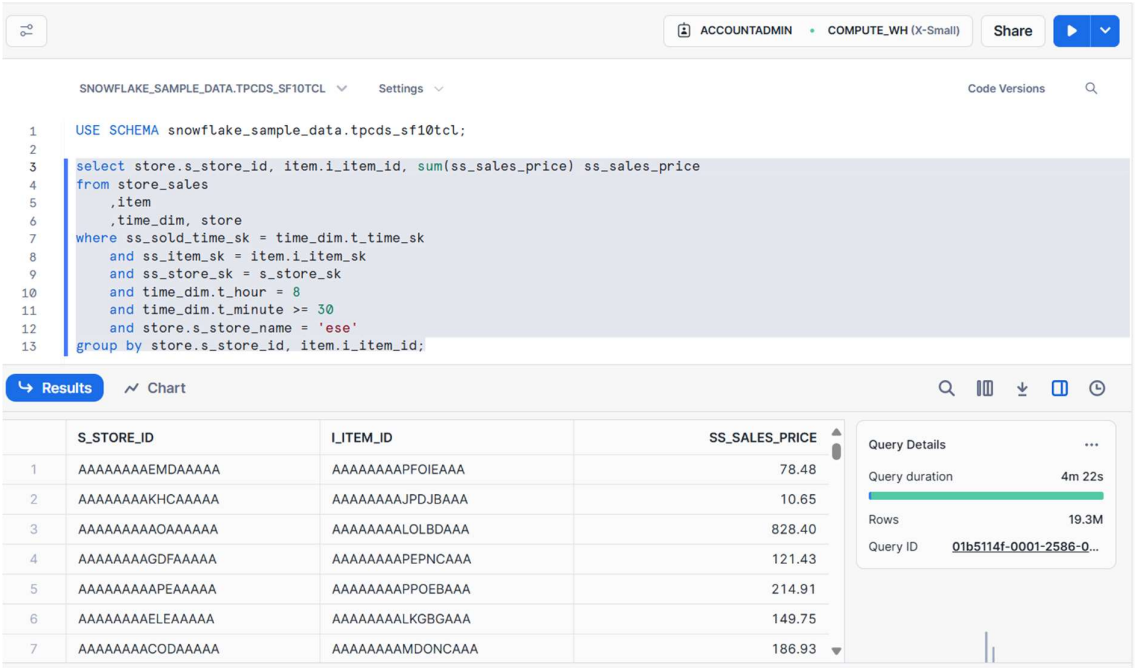
3. Go back to the worksheet. Run the below query under Compute_WH warehouse.

Query: USE SCHEMA snowflake_sample_data.tpcds_sf10tcl;

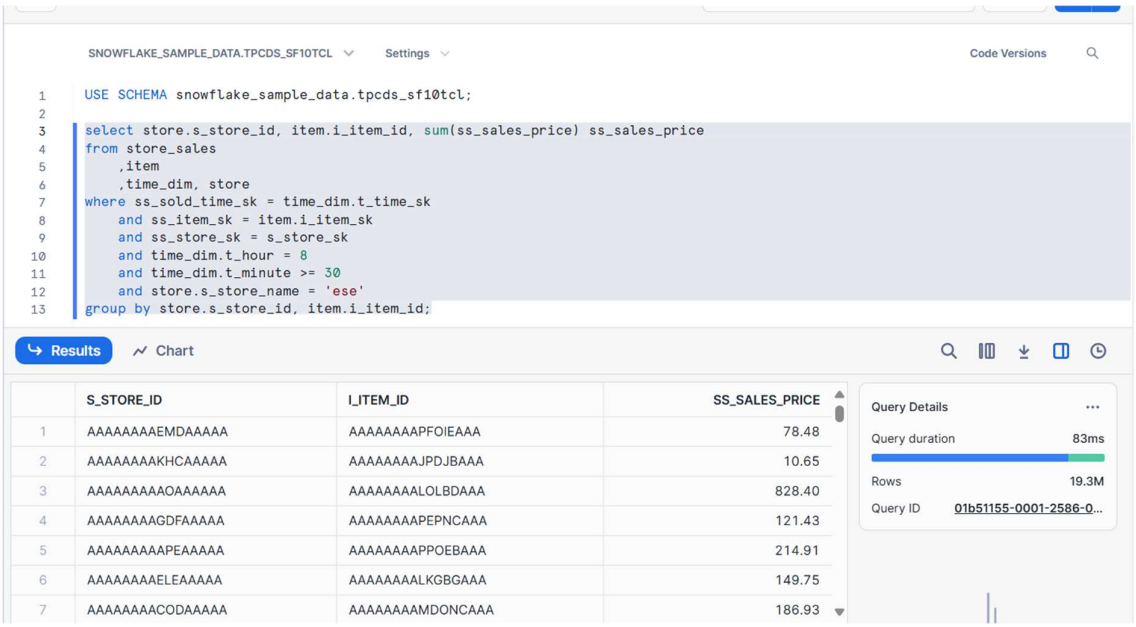
```
select store.s_store_id, item.i_item_id, sum(ss_sales_price) ss_sales_price
from store_sales
   ,item
   ,time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
   and ss_item_sk = item.i_item_sk
   and ss_store_sk = s_store_sk
```

```
  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.



5. Next run the same query under Demo_WH warehouse.
6. Now check the time here. It took only 83ms



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;

3. Now check the view data.

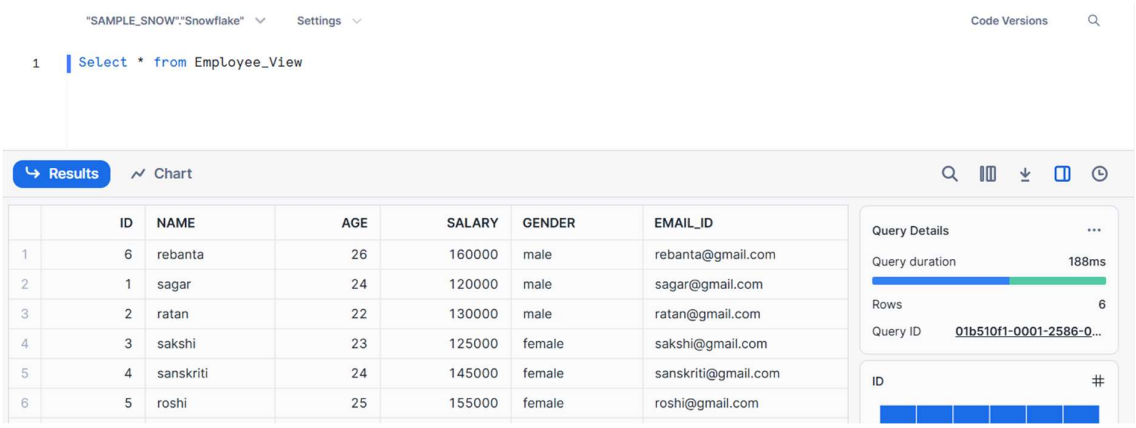
Query: Select * from EmpView;

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.

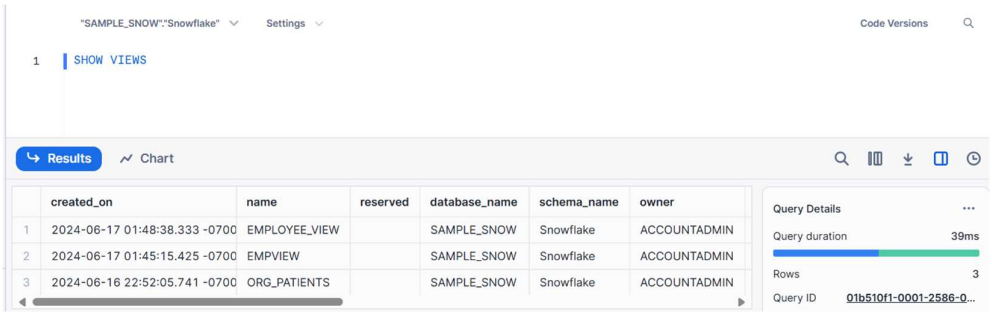
Query: Create or Replace view Employee_View as select ID, Name, Age, Salary, Gender, Email_Id from Employee_Details

2. Check the view data.



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

Query: SHOW VIEWS

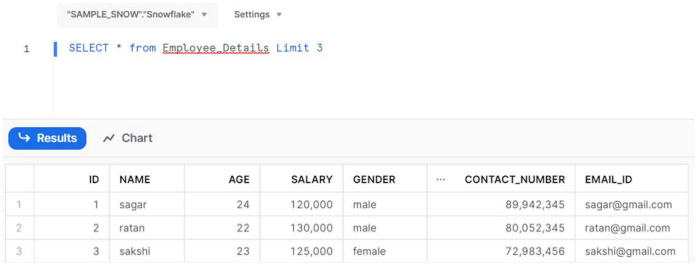


SELECT Commands

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

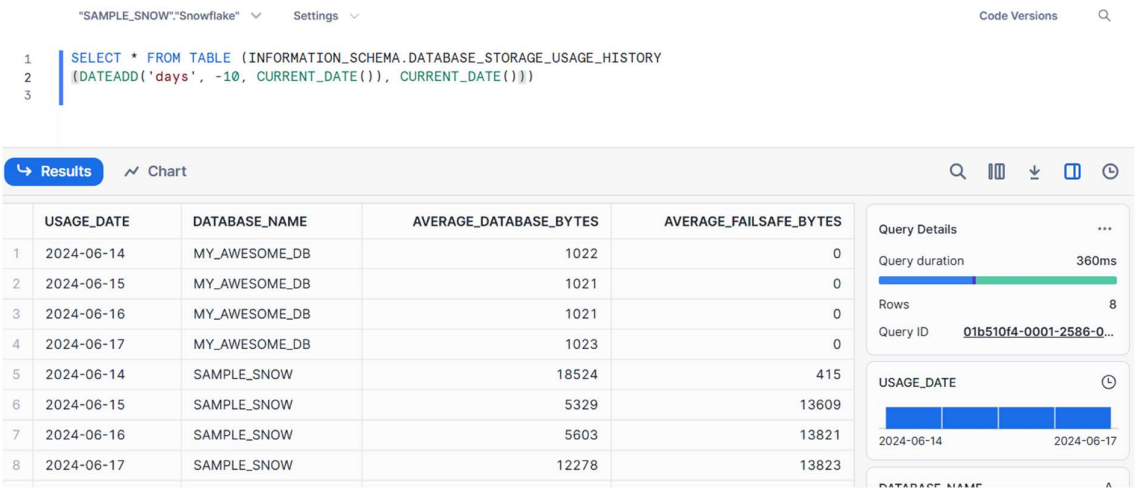
Query: SELECT * from Employee_Details Limit 3

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

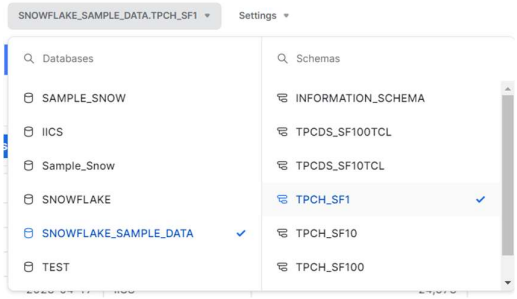


3. Use the following query to display the usage of the last 10 days.

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

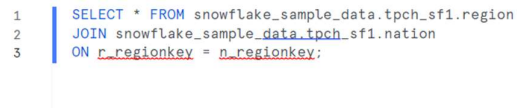


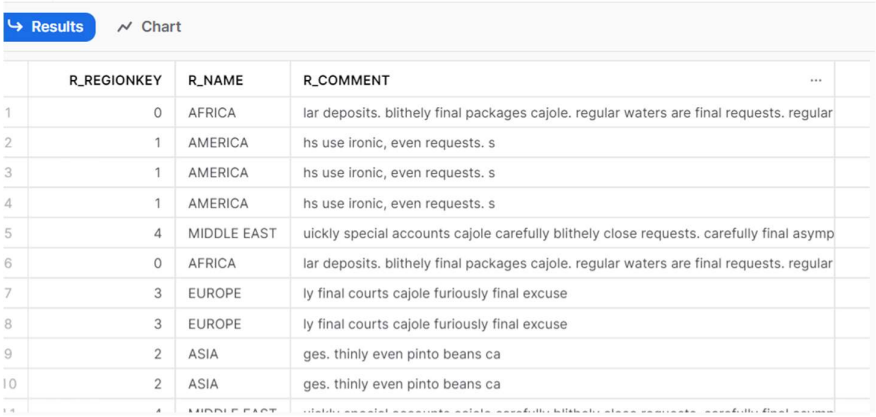
4. Now select the SNOWFLAKE_SAMPLE_DATA database and TPCH_SF1 schema as shown below.



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;





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

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

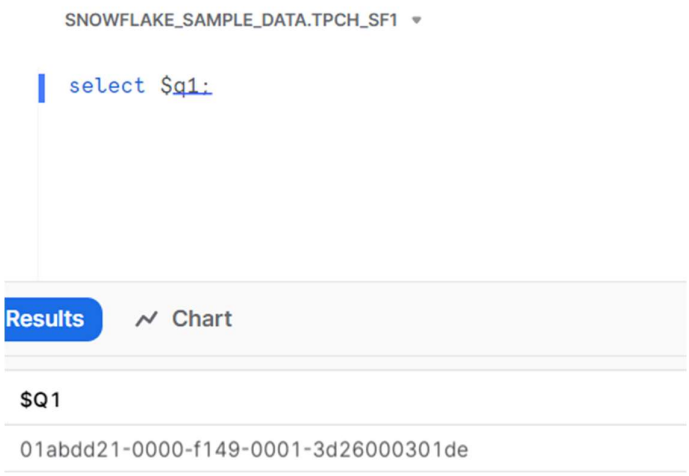
Forth Query: SET q1 = LAST_QUERY_ID();

```
SNOWFLAKE_SAMPLE_DATA.TPCH_SF1
1  SET q1 = LAST_QUERY_ID();
```

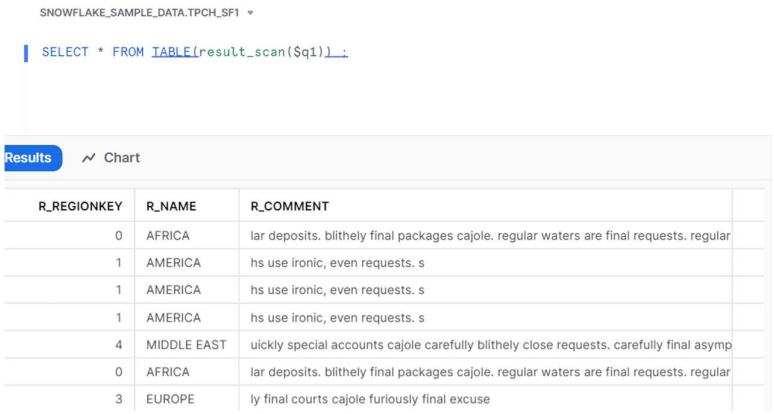
Results | Chart

	status
1	Statement executed successfully.

Fifth Query: select $q1;

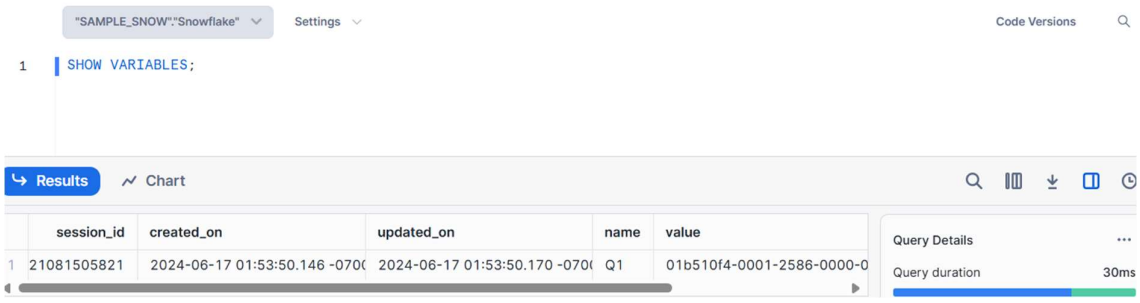


Sixth Query: SELECT * FROM TABLE(result_scan($q1)) ;



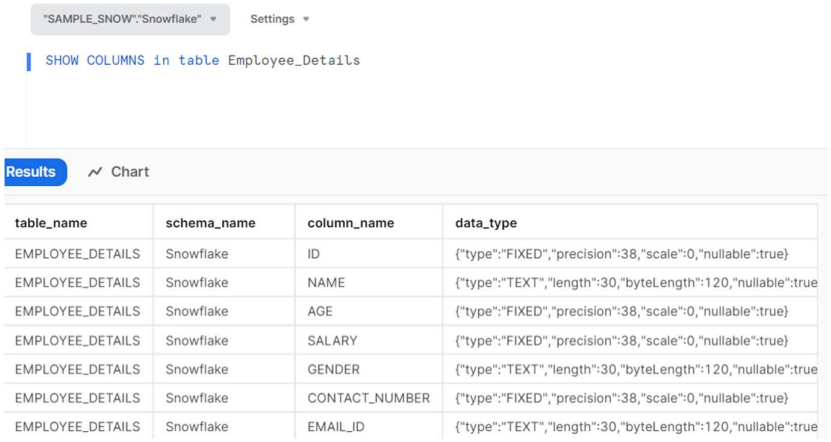
R_REGIONKEY	R_NAME	R_COMMENT
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regular
1	AMERICA	hs use ironic, even requests. s
1	AMERICA	hs use ironic, even requests. s
1	AMERICA	hs use ironic, even requests. s
4	MIDDLE EAST	uickly special accounts cajole carefully 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;



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

Query: SHOW COLUMNS in table Employee_Details



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

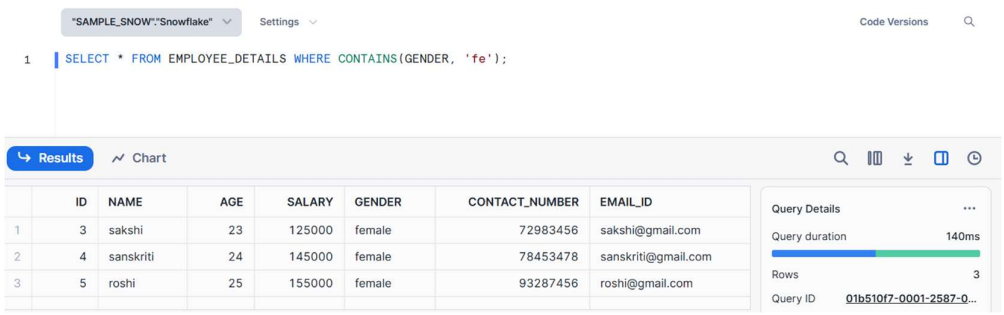
7. Use the following command to show all the parameters provided by Snowflake.

Query: SHOW PARAMETERS;

	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
10	CLIENT_METADATA_REQUEST_USE_CONNECTION_CTX	false	false
11	CLIENT_METADATA_USE_SESSION_DATABASE	false	false
12	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');



	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

Persisted Query Results

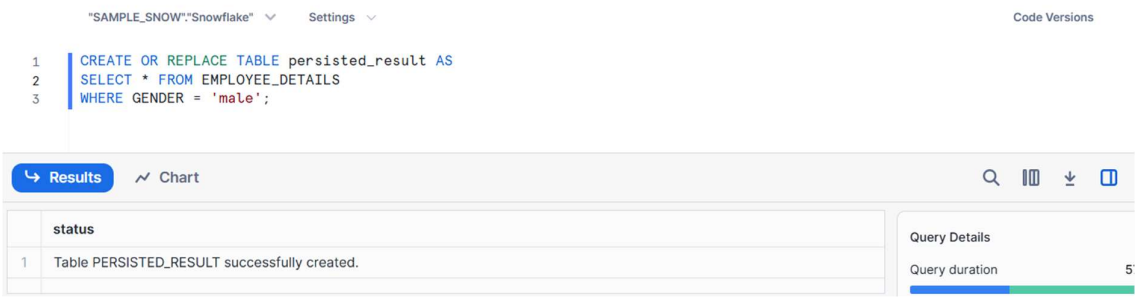
1. Persisted query results in Snowflake allow storing the results of a query for reuse or sharing purposes.
2. This feature can be useful for creating reports or sharing data without rerunning the query.

Query: CREATE OR REPLACE TABLE persisted_result AS

SELECT * FROM EMPLOYEE_DETAILS

WHERE GENDER = 'male';

3. Check the query.



4. Query the persisted result set.

Query: SELECT * FROM persisted_result;