SNOWFLAKE ASSIGNMENT 2

TASK 1: Scale Virtual Warehouses and test performance with large datasets using Snowpark.

Objective:

We'll scale Virtual Warehouses (VWs) in Snowflake and test query performance when processing large datasets via Snowpark.

This helps you:

- Measure compute scalability
- Compare Small vs. Medium vs. Large warehouses
- Identify bottlenecks in data transformations

Step-by-Step: Scaling and Testing Performance

1. Create or Identify a Virtual Warehouse

In Snowflake UI (or SQL worksheet):

```
-- Create three warehouses for performance comparison

    CREATE WAREHOUSE WH_SMALL  WITH WAREHOUSE_SIZE = 'SMALL'
    AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;

    CREATE WAREHOUSE WH_MEDIUM WITH WAREHOUSE_SIZE = 'MEDIUM'
    AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;

    CREATE WAREHOUSE WH_LARGE  WITH WAREHOUSE_SIZE = 'LARGE'
    AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;
```

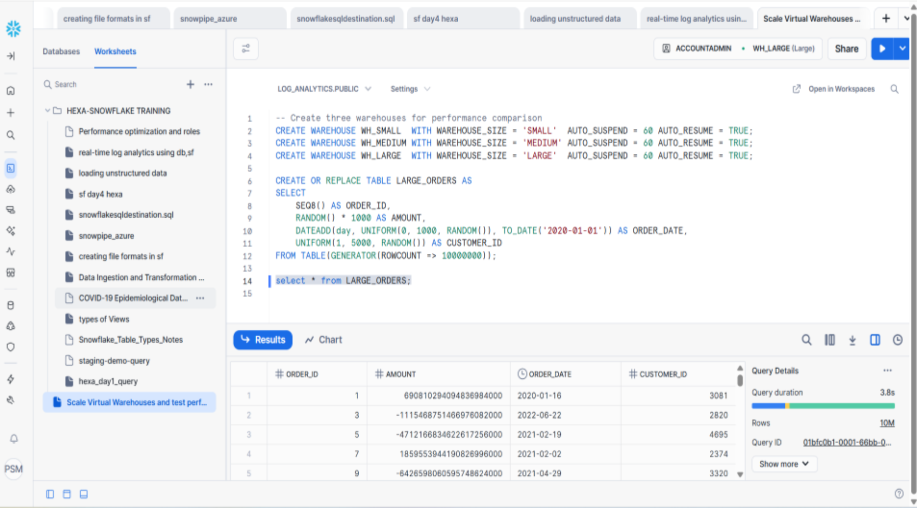
Each warehouse size adds more compute clusters (SMALL → 2 nodes, MEDIUM → 4 nodes, etc.)

2. Prepare a Large Dataset

Option 1 — Use internal table:

```
CREATE OR REPLACE TABLE LARGE_ORDERS AS

SELECT SEQ8() AS ORDER_ID,

    RANDOM() * 1000 AS AMOUNT,

    TO_DATE('2020-01-01') + (RANDOM() * 1000) AS ORDER_DATE,

    UNIFORM(1, 5000, RANDOM()) AS CUSTOMER_ID

FROM TABLE(GENERATOR(ROWCOUNT => 10000000));  -- 10M records
```

Option 2 — Load from Azure Blob or ADLS external stage if you're doing it with Databricks + Snowpark.


creating file formats in sf
snowpipe_azure
snowflakesqldestination.sql
sf day4 hexa
loading unstructured data
real-time log analytics usin...
Scale Virtual Warehouses ...
+
Databases
Worksheets
Search
HEXA-SNOWFLAKE TRAINING
Performance optimization and roles
real-time log analytics using db,sf
loading unstructured data
sf day4 hexa
snowflakesqldestination.sql
snowpipe_azure
creating file formats in sf
Data Ingestion and Transformation ...
COVID-19 Epidemiological Dat...
types of Views
Snowflake_Table_Types_Notes
staging-demo-query
hexa_day1_query
Scale Virtual Warehouses and test perf...
PSM
ACCOUNTADMIN
WH_LARGE (Large)
Share
LOG_ANALYTICS.PUBLIC
Settings
Open in Workspaces
Results
Chart

```sql
-- Create three warehouses for performance comparison
CREATE WAREHOUSE WH_SMALL  WITH WAREHOUSE_SIZE = 'SMALL'  AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;
CREATE WAREHOUSE WH_MEDIUM WITH WAREHOUSE_SIZE = 'MEDIUM' AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;
CREATE WAREHOUSE WH_LARGE  WITH WAREHOUSE_SIZE = 'LARGE'  AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;

CREATE OR REPLACE TABLE LARGE_ORDERS AS
SELECT
    SEQ8() AS ORDER_ID,
    RANDOM() * 1000 AS AMOUNT,
    DATEADD(day, UNIFORM(0, 1000, RANDOM()), TO_DATE('2020-01-01')) AS ORDER_DATE,
    UNIFORM(1, 5000, RANDOM()) AS CUSTOMER_ID
FROM TABLE(GENERATOR(ROWCOUNT => 10000000));

select * from LARGE_ORDERS;
```

	ORDER_ID	AMOUNT	ORDER_DATE	CUSTOMER_ID
1	1	6908102940948369840000	2020-01-16	3081
2	3	-1115468751466976082000	2022-06-22	2820
3	5	-4712166834622617256000	2021-02-19	4695
4	7	1859553944190826996000	2021-02-02	2374
5	9	-6426598060595748624000	2021-04-29	3320

Query Details

Query duration 3.8s

Rows 10M

Query ID 01bfc0b1-0001-66bb-0...

Show more

3. Connect via Snowpark (Python)

```
from snowflake.snowpark import Session

from snowflake.snowpark.functions import col, avg

import time

sfOptions = {

    "account": "sqishot-fa68768",

    "user": "poojashree",

    "password": "Poojashree@307",

    "warehouse": "WH_SMALL",   # Start with SMALL

    "database": "LOG_ANALYTICS",

    "schema": "PUBLIC"

}

session = Session.builder.configs(sfOptions).create()
```

4. Run Transformations and Measure Execution Time

Example: Aggregate transformation

```
df = session.table("LARGE_ORDERS")

start = time.time()
```

result = df.group_by(col("CUSTOMER_ID")).agg(avg(col("AMOUNT")).alias("AVG_AMOUNT")).collect()

end = time.time()

print(f"Execution Time: {end - start:.2f} seconds, Warehouse: WH_SMALL")

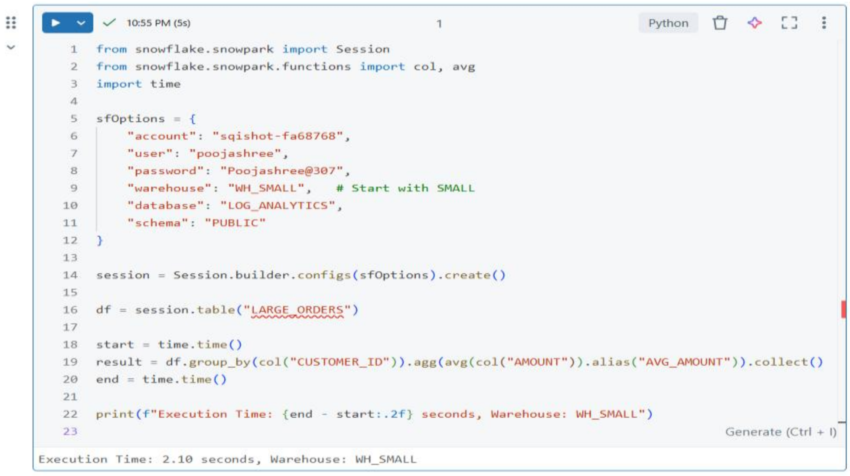
Now repeat the same code, changing only:

sfOptions["warehouse"] = "WH_MEDIUM"

sfOptions["warehouse"] = "WH_LARGE"

You can record all timings in a list or CSV for comparison.

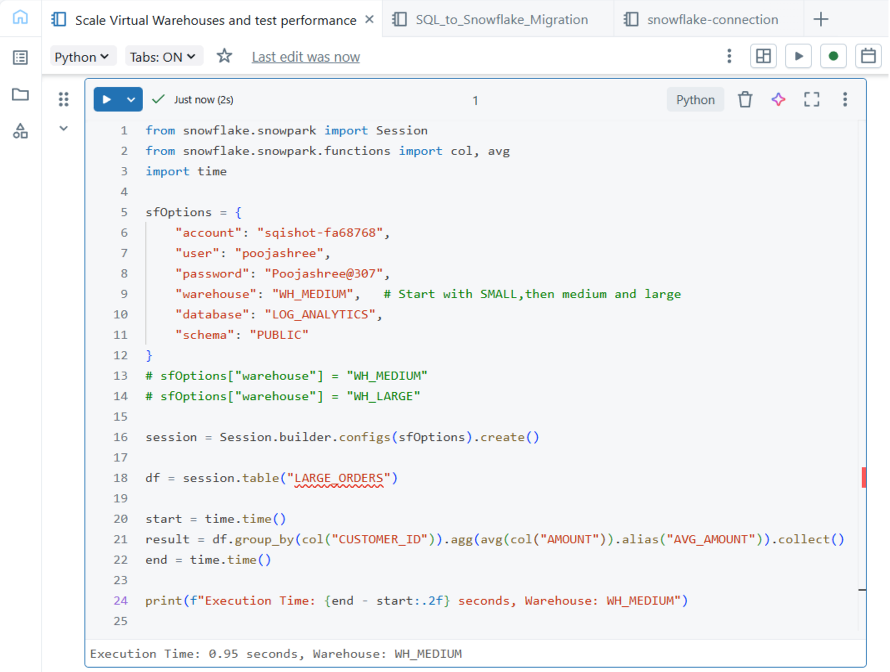
SMALL:



```python
from snowflake.snowpark import Session
from snowflake.snowpark.functions import col, avg
import time

sfOptions = {
    "account": "sqishot-fa68768",
    "user": "poojashree",
    "password": "Poojashree@307",
    "warehouse": "WH_SMALL",    # Start with SMALL
    "database": "LOG_ANALYTICS",
    "schema": "PUBLIC"
}

session = Session.builder.configs(sfOptions).create()

df = session.table("LARGE_ORDERS")

start = time.time()
result = df.group_by(col("CUSTOMER_ID")).agg(avg(col("AMOUNT")).alias("AVG_AMOUNT")).collect()
end = time.time()

print(f"Execution Time: {end - start:.2f} seconds, Warehouse: WH_SMALL")
```

```
Execution Time: 2.10 seconds, Warehouse: WH_SMALL
```

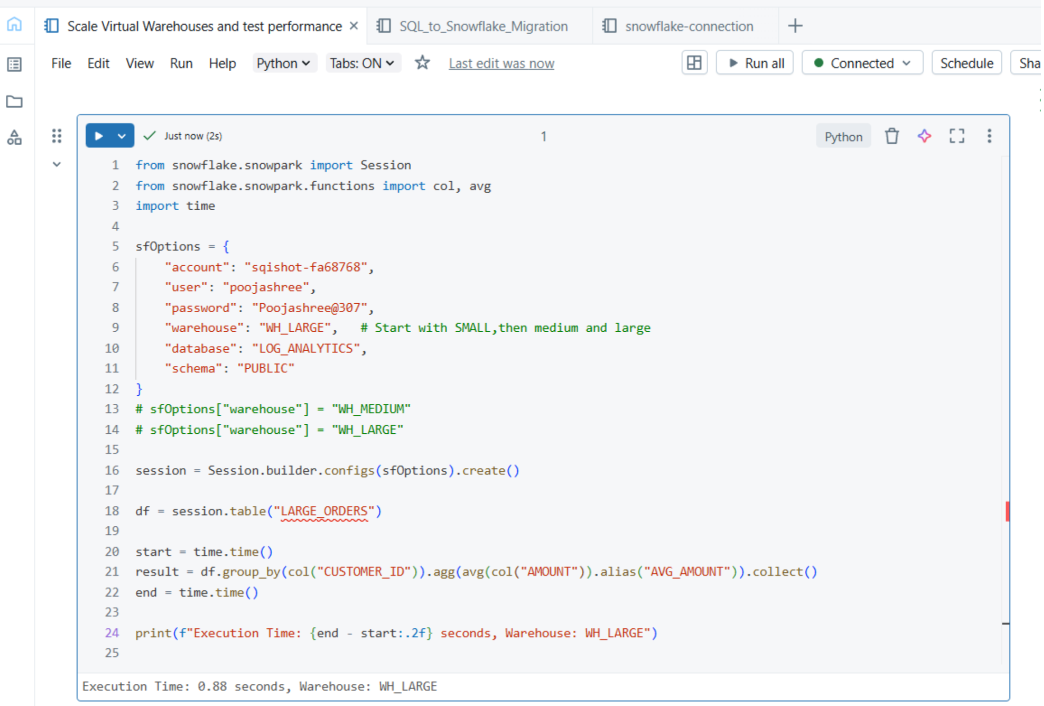
MEDIUM:


Scale Virtual Warehouses and test performance ×
SQL_to_Snowflake_Migration
snowflake-connection
+
Python
Tabs: ON
Last edit was now
Just now (2s)
1
Python

```python
from snowflake.snowpark import Session
from snowflake.snowpark.functions import col, avg
import time

sfOptions = {
    "account": "sqishot-fa68768",
    "user": "poojashree",
    "password": "Poojashree@307",
    "warehouse": "WH_MEDIUM",    # Start with SMALL,then medium and large
    "database": "LOG_ANALYTICS",
    "schema": "PUBLIC"
}
# sfOptions["warehouse"] = "WH_MEDIUM"
# sfOptions["warehouse"] = "WH_LARGE"

session = Session.builder.configs(sfOptions).create()

df = session.table("LARGE_ORDERS")

start = time.time()
result = df.group_by(col("CUSTOMER_ID")).agg(avg(col("AMOUNT")).alias("AVG_AMOUNT")).collect()
end = time.time()

print(f"Execution Time: {end - start:.2f} seconds, Warehouse: WH_MEDIUM")
```

```
Execution Time: 0.95 seconds, Warehouse: WH_MEDIUM
```

LARGE:



Scale Virtual Warehouses and test performance
SQL_to_Snowflake_Migration
snowflake-connection
File Edit View Run Help Python Tabs: ON Last edit was now
Run all
Connected
Schedule
Sha
Just now (2s)
1
Python

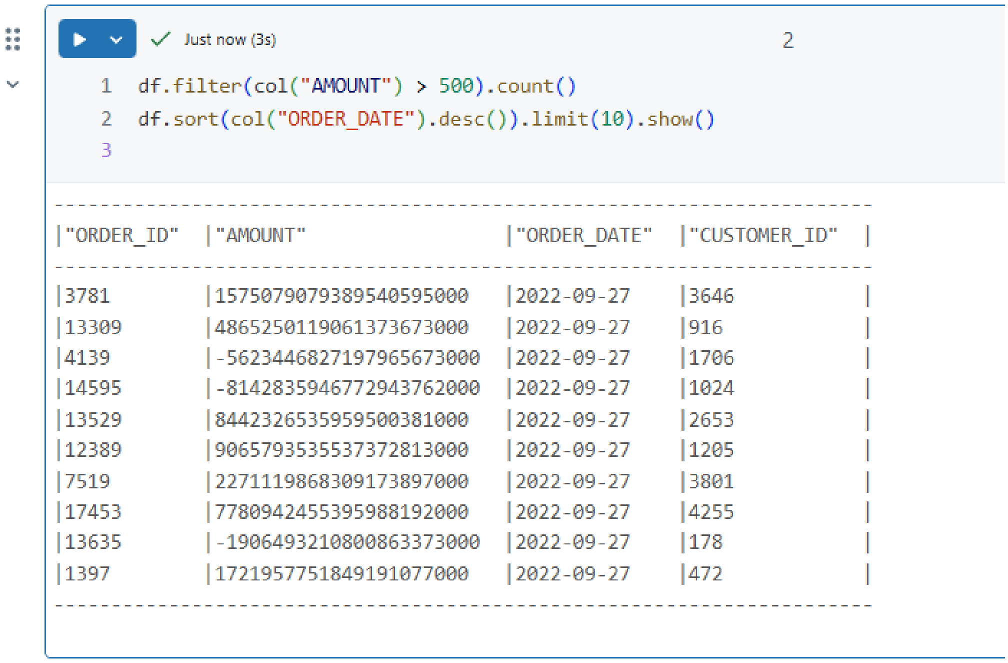
```python
from snowflake.snowpark import Session
from snowflake.snowpark.functions import col, avg
import time

sfOptions = {
    "account": "sqishot-fa68768",
    "user": "poojashree",
    "password": "Poojashree@307",
    "warehouse": "WH_LARGE",    # Start with SMALL,then medium and large
    "database": "LOG_ANALYTICS",
    "schema": "PUBLIC"
}
# sfOptions["warehouse"] = "WH_MEDIUM"
# sfOptions["warehouse"] = "WH_LARGE"

session = Session.builder.configs(sfOptions).create()

df = session.table("LARGE_ORDERS")

start = time.time()
result = df.group_by(col("CUSTOMER_ID")).agg(avg(col("AMOUNT")).alias("AVG_AMOUNT")).collect()
end = time.time()

print(f"Execution Time: {end - start:.2f} seconds, Warehouse: WH_LARGE")

```

```
Execution Time: 0.88 seconds, Warehouse: WH_LARGE
```

5. (Optional) Measure DataFrame Operations

You can also test:

```
df.filter(col("AMOUNT") > 500).count()
df.sort(col("ORDER_DATE").desc()).limit(10).show()
```



```python
df.filter(col("AMOUNT") > 500).count()
df.sort(col("ORDER_DATE").desc()).limit(10).show()

```

```
-----------------------------------------------------------------------
|"ORDER_ID"  |"AMOUNT"                 |"ORDER_DATE"  |"CUSTOMER_ID"  |
-----------------------------------------------------------------------
|3781        |1575079079389540595000   |2022-09-27    |3646           |
|13309       |4865250119061373673000   |2022-09-27    |916            |
|4139        |-5623446827197965673000  |2022-09-27    |1706           |
|14595       |-8142835946772943762000  |2022-09-27    |1024           |
|13529       |8442326535959500381000   |2022-09-27    |2653           |
|12389       |9065793535537372813000   |2022-09-27    |1205           |
|7519        |2271119868309173897000   |2022-09-27    |3801           |
|17453       |7780942455395988192000   |2022-09-27    |4255           |
|13635       |-1906493210800863373000  |2022-09-27    |178            |
|1397        |1721957751849191077000   |2022-09-27    |472            |
-----------------------------------------------------------------------
```

Record time for each operation at each warehouse scale.

6. Analyze Performance

Example result table:

Warehouse	Row Count	Query Type	Time (s)	Credits Used	Notes
SMALL	10M	Group By	2.10	1 credit/hr	CPU bound
MEDIUM	10M	Group By	0.95	2 credits/hr	Balanced
LARGE	10M	Group By	0.88	4 credits/hr	Fast but costly

Observation:
Performance improves with larger warehouses, but cost also increases. Ideal choice depends on workload criticality and SLA requirements.

7. Auto-Scaling & Multi-Cluster (Optional)

If you expect variable load:

```
ALTER WAREHOUSE WH_MEDIUM SET
  MIN_CLUSTER_COUNT = 1
  MAX_CLUSTER_COUNT = 3
  SCALING_POLICY = 'ECONOMY';
```

Snowflake will auto-scale horizontally when concurrent queries increase — very useful in production pipelines.

8. Clean Up Resources

After testing:

```
DROP WAREHOUSE IF EXISTS WH_SMALL;
DROP WAREHOUSE IF EXISTS WH_MEDIUM;
DROP WAREHOUSE IF EXISTS WH_LARGE;
```

Snowflake charges per-second usage, so cleanup avoids extra credits.