DATA 226- DATAWAREHOUSE

Homework 4

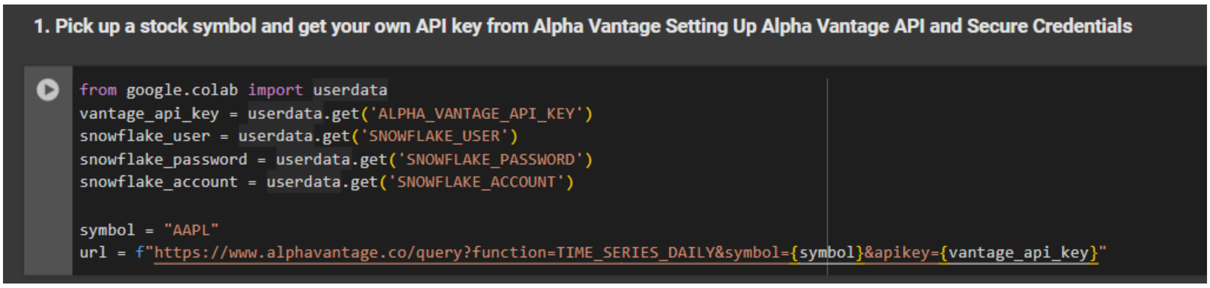
Name : Vimalanandhan Sivanandham

SJSU ID: 017596436

1. (+1) Pick up a stock symbol and get your own API key from Alpha Vantage

<u>CODE:</u>

```
from google.colab import userdata
vantage_api_key = userdata.get('ALPHA_VANTAGE_API_KEY')
snowflake_user = userdata.get('SNOWFLAKE_USER')
snowflake_password = userdata.get('SNOWFLAKE_PASSWORD')
snowflake_account = userdata.get('SNOWFLAKE_ACCOUNT')


symbol = "AAPL"
url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}"
```



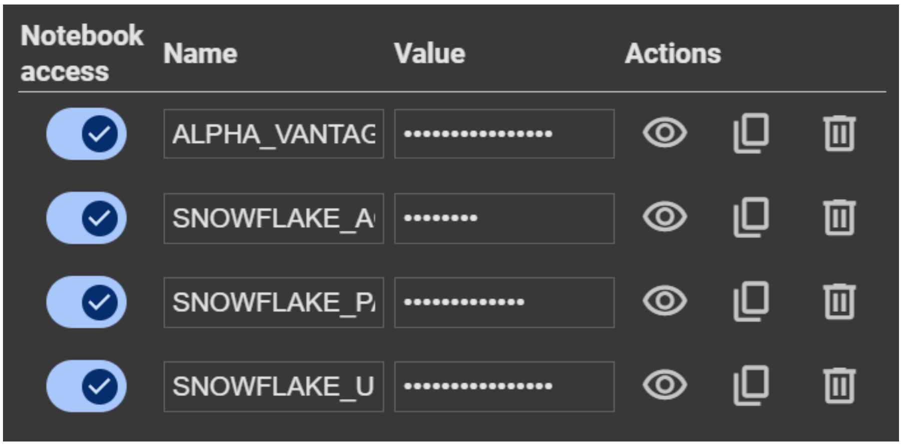
1. Pick up a stock symbol and get your own API key from Alpha Vantage Setting Up Alpha Vantage API and Secure Credentials

```
from google.colab import userdata
vantage_api_key = userdata.get('ALPHA_VANTAGE_API_KEY')
snowflake_user = userdata.get('SNOWFLAKE_USER')
snowflake_password = userdata.get('SNOWFLAKE_PASSWORD')
snowflake_account = userdata.get('SNOWFLAKE_ACCOUNT')

symbol = "AAPL"
url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}"
```

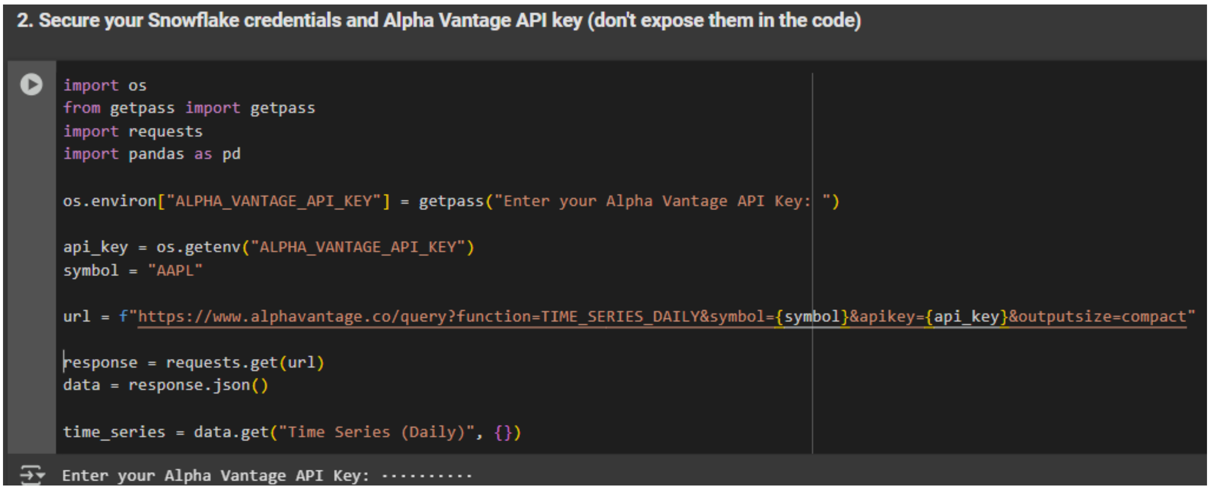
2. (+1) Secure your Snowflake credentials and Alpha Vantage API key (don't expose them in the code)

<u>CODE:</u>

```
import os
from getpass import getpass
import requests
import pandas as pd

os.environ["ALPHA_VANTAGE_API_KEY"] = getpass("Enter your Alpha Vantage API Key: ")

api_key = os.getenv("ALPHA_VANTAGE_API_KEY")
symbol = "AAPL"

url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={api_key}&outputsize=compact"

response = requests.get(url)
data = response.json()

time_series = data.get("Time Series (Daily)", {})
```



Notebook access	Name	Value	Actions
✓	ALPHA_VANTAG	••••••••••••••••	👁 ⧉ 🗑
✓	SNOWFLAKE_A	•••••••••	👁 ⧉ 🗑
✓	SNOWFLAKE_P	••••••••••••••	👁 ⧉ 🗑
✓	SNOWFLAKE_U	••••••••••••••••	👁 ⧉ 🗑



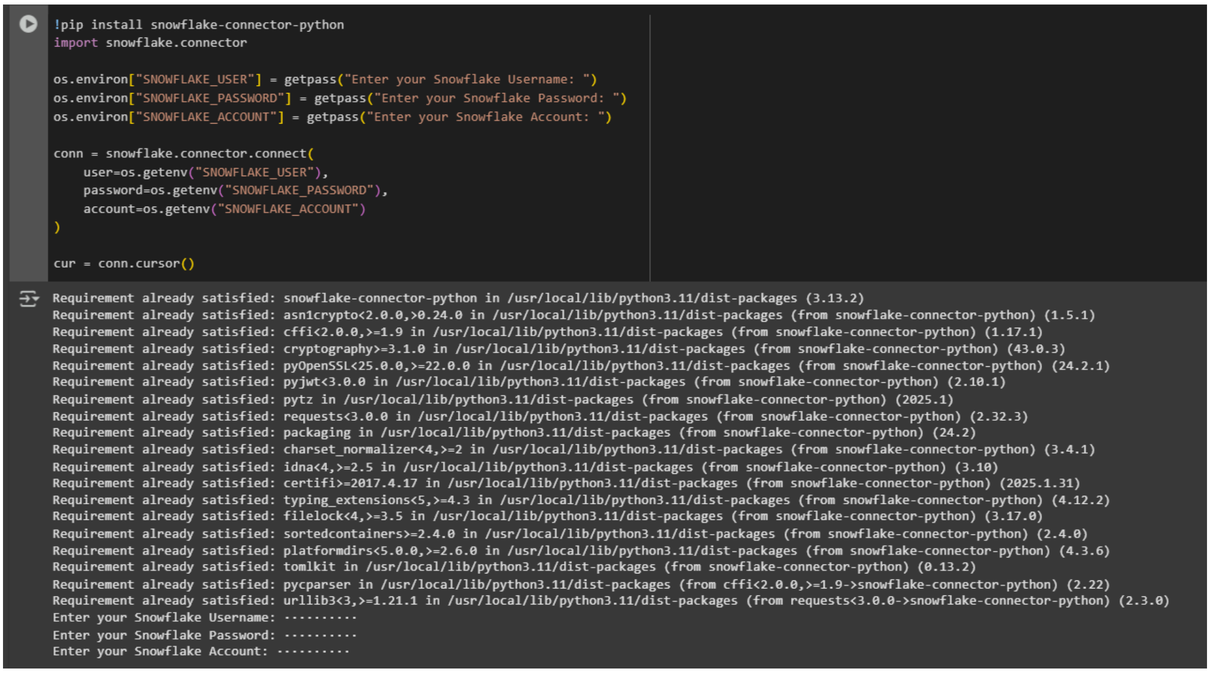
2. Secure your Snowflake credentials and Alpha Vantage API key (don't expose them in the code)

```
import os
from getpass import getpass
import requests
import pandas as pd

os.environ["ALPHA_VANTAGE_API_KEY"] = getpass("Enter your Alpha Vantage API Key: ")

api_key = os.getenv("ALPHA_VANTAGE_API_KEY")
symbol = "AAPL"

url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={api_key}&outputsize=compact"

response = requests.get(url)
data = response.json()

time_series = data.get("Time Series (Daily)", {})
```

```
Enter your Alpha Vantage API Key: ··········
```

<u>CODE:</u>

```
!pip install snowflake-connector-python

import snowflake.connector


os.environ["SNOWFLAKE_USER"] = getpass("Enter your Snowflake Username: ")

os.environ["SNOWFLAKE_PASSWORD"] = getpass("Enter your Snowflake Password: ")

os.environ["SNOWFLAKE_ACCOUNT"] = getpass("Enter your Snowflake Account: ")


conn = snowflake.connector.connect(

  user=os.getenv("SNOWFLAKE_USER"),

  password=os.getenv("SNOWFLAKE_PASSWORD"),

  account=os.getenv("SNOWFLAKE_ACCOUNT")

)


cur = conn.cursor()
```



```
!pip install snowflake-connector-python
import snowflake.connector

os.environ["SNOWFLAKE_USER"] = getpass("Enter your Snowflake Username: ")
os.environ["SNOWFLAKE_PASSWORD"] = getpass("Enter your Snowflake Password: ")
os.environ["SNOWFLAKE_ACCOUNT"] = getpass("Enter your Snowflake Account: ")

conn = snowflake.connector.connect(
    user=os.getenv("SNOWFLAKE_USER"),
    password=os.getenv("SNOWFLAKE_PASSWORD"),
    account=os.getenv("SNOWFLAKE_ACCOUNT")
)

cur = conn.cursor()
```

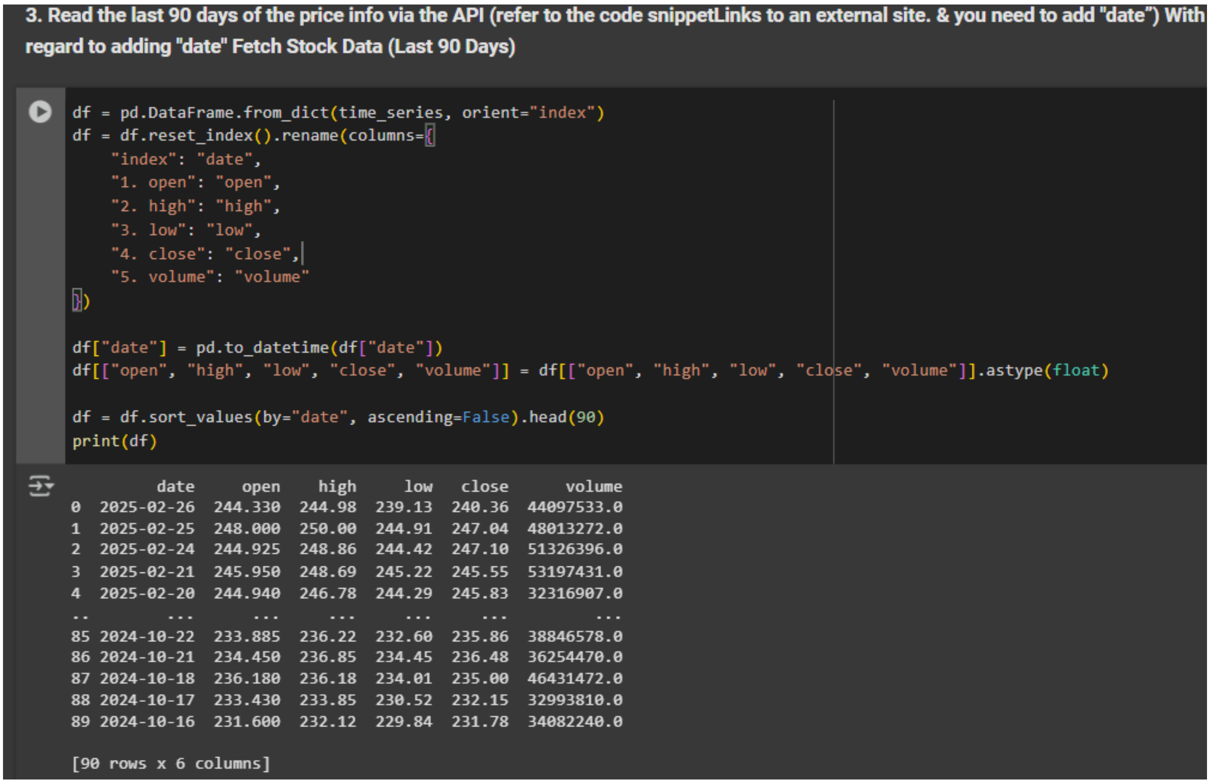
```
Requirement already satisfied: snowflake-connector-python in /usr/local/lib/python3.11/dist-packages (3.13.2)
Requirement already satisfied: asn1crypto<2.0.0,>0.24.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (1.5.1)
Requirement already satisfied: cffi<2.0.0,>=1.9 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (1.17.1)
Requirement already satisfied: cryptography>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (43.0.3)
Requirement already satisfied: pyOpenSSL<25.0.0,>=22.0.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (24.2.1)
Requirement already satisfied: pyjwt<3.0.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (2.10.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (2025.1)
Requirement already satisfied: requests<3.0.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (2.32.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (24.2)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (2025.1.31)
Requirement already satisfied: typing_extensions<5,>=4.3 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (4.12.2)
Requirement already satisfied: filelock<4,>=3.5 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (3.17.0)
Requirement already satisfied: sortedcontainers>=2.4.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (2.4.0)
Requirement already satisfied: platformdirs<5.0.0,>=2.6.0 in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (4.3.6)
Requirement already satisfied: tomlkit in /usr/local/lib/python3.11/dist-packages (from snowflake-connector-python) (0.13.2)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi<2.0.0,>=1.9->snowflake-connector-python) (2.22)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0->snowflake-connector-python) (2.3.0)
Enter your Snowflake Username: ··········
Enter your Snowflake Password: ··········
Enter your Snowflake Account: ··········
```

3. **(+2) Read the last 90 days of the price info via the API (refer to the code snippetLinks to an external site. & you need to add "date")**
    1. **With regard to adding "date", please look at the next slide**

CODE:

```
df = pd.DataFrame.from_dict(time_series, orient="index")

df = df.reset_index().rename(columns={

  "index": "date",

  "1. open": "open",

  "2. high": "high",

  "3. low": "low",

  "4. close": "close",

  "5. volume": "volume"

})


df["date"] = pd.to_datetime(df["date"])

df[["open", "high", "low", "close", "volume"]] = df[["open", "high", "low", "close", "volume"]].astype(float)


df = df.sort_values(by="date", ascending=False).head(90)
```

print(df)



3. Read the last 90 days of the price info via the API (refer to the code snippetLinks to an external site. & you need to add "date") With regard to adding "date" Fetch Stock Data (Last 90 Days)

```
df = pd.DataFrame.from_dict(time_series, orient="index")
df = df.reset_index().rename(columns={
    "index": "date",
    "1. open": "open",
    "2. high": "high",
    "3. low": "low",
    "4. close": "close",
    "5. volume": "volume"
})

df["date"] = pd.to_datetime(df["date"])
df[["open", "high", "low", "close", "volume"]] = df[["open", "high", "low", "close", "volume"]].astype(float)

df = df.sort_values(by="date", ascending=False).head(90)
print(df)
```

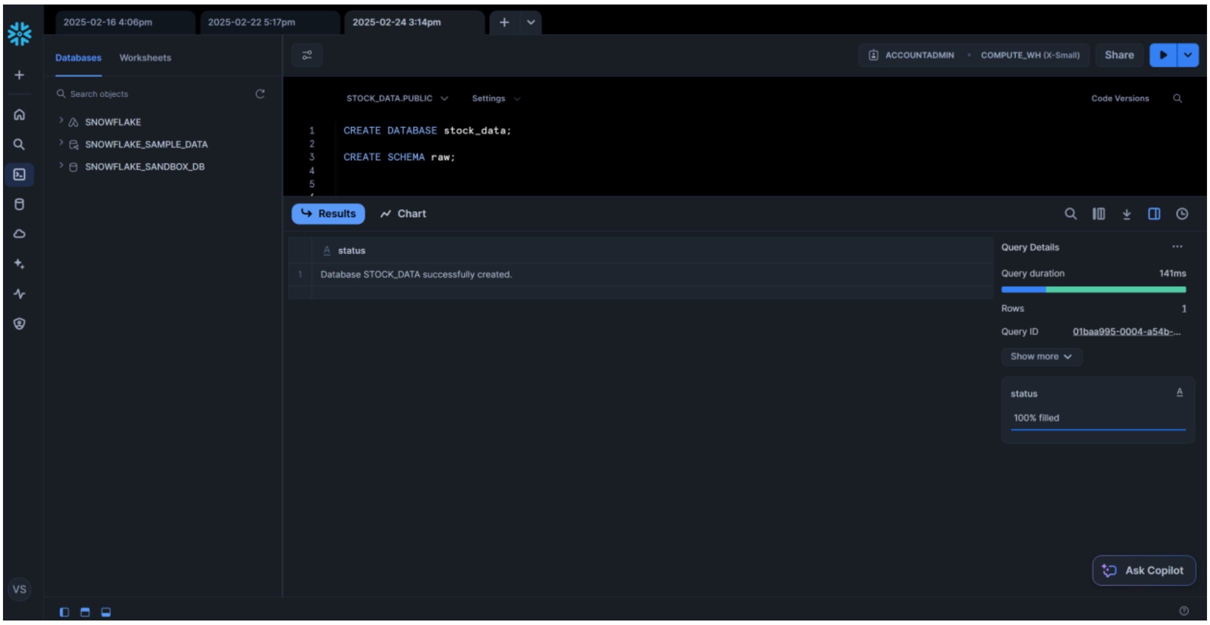
```
          date     open    high     low   close      volume
0   2025-02-26  244.330  244.98  239.13  240.36  44097533.0
1   2025-02-25  248.000  250.00  244.91  247.04  48013272.0
2   2025-02-24  244.925  248.86  244.42  247.10  51326396.0
3   2025-02-21  245.950  248.69  245.22  245.55  53197431.0
4   2025-02-20  244.940  246.78  244.29  245.83  32316907.0
..         ...      ...     ...     ...     ...         ...
85  2024-10-22  233.885  236.22  232.60  235.86  38846578.0
86  2024-10-21  234.450  236.85  234.45  236.48  36254470.0
87  2024-10-18  236.180  236.18  234.01  235.00  46431472.0
88  2024-10-17  233.430  233.85  230.52  232.15  32993810.0
89  2024-10-16  231.600  232.12  229.84  231.78  34082240.0

[90 rows x 6 columns]
```

4. **(+1) Create a table under "raw" schema if it doesn't exist to capture the info from the API**

    1. **symbol, date, open, close, high, low, volume: symbol and date should be primary keys**

<u>CODE:</u>

CREATE DATABASE stock_data;



```
CREATE DATABASE stock_data;

CREATE SCHEMA raw;
```

	status
1	Database STOCK_DATA successfully created.

Query duration 141ms

Rows 1

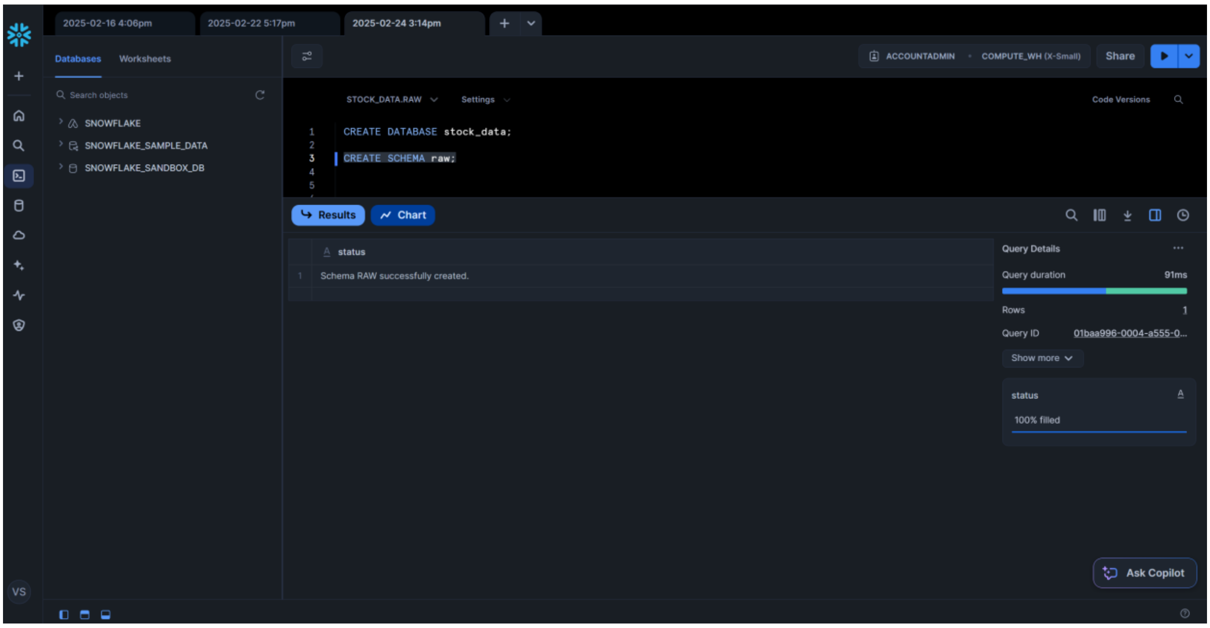
Query ID 01baa995-0004-a54b-...

status

100% filled

CODE:

CREATE SCHEMA raw;



```
CREATE DATABASE stock_data;

CREATE SCHEMA raw;
```

	status
1	Schema RAW successfully created.

Query duration 91ms

Rows 1

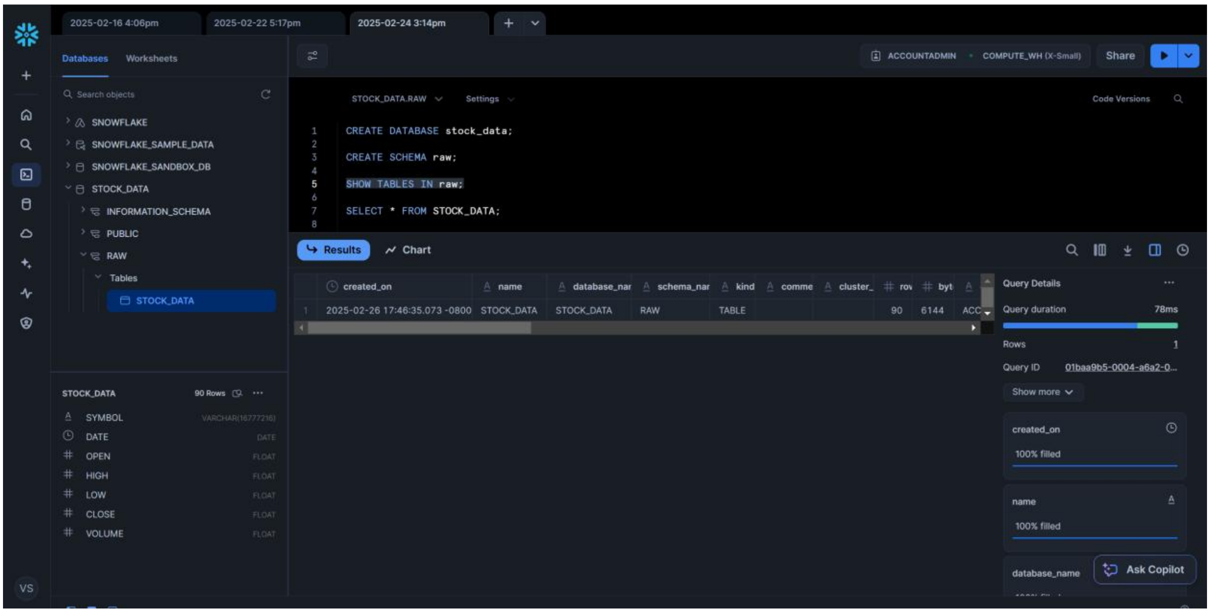
Query ID 01baa996-0004-a555-0...

status

100% filled

CODE:

SHOW TABLES IN raw;



```
CREATE DATABASE stock_data;

CREATE SCHEMA raw;

SHOW TABLES IN raw;

SELECT * FROM STOCK_DATA;
```

	created_on	name	database_nar	schema_nar	kind	comme	cluster_	rov	byt	ACC
1	2025-02-26 17:46:35.073 -0800	STOCK_DATA	STOCK_DATA	RAW	TABLE			90	6144	ACC

Query Details

Query duration 78ms

Rows 1

Query ID 01baa9b5-0004-a6a2-0...

created_on 100% filled

name 100% filled

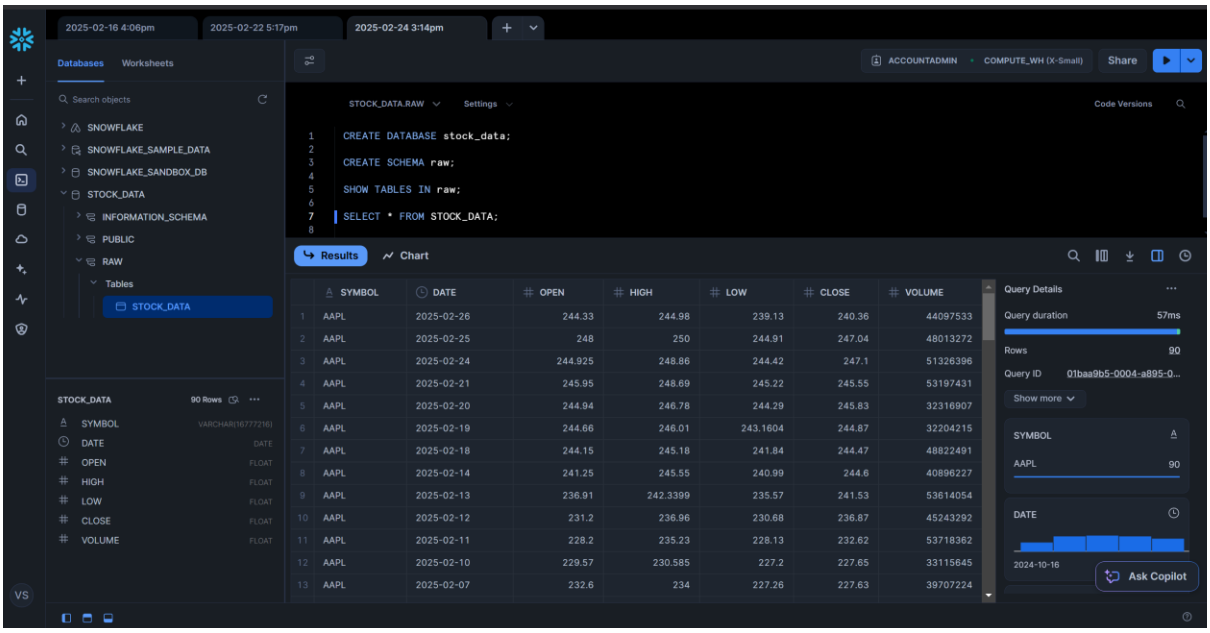
database_name

STOCK_DATA 90 Rows

SYMBOL	VARCHAR(16777216)
DATE	DATE
OPEN	FLOAT
HIGH	FLOAT
LOW	FLOAT
CLOSE	FLOAT
VOLUME	FLOAT

CODE:

SELECT * FROM STOCK_DATA;



```
CREATE DATABASE stock_data;

CREATE SCHEMA raw;

SHOW TABLES IN raw;

SELECT * FROM STOCK_DATA;
```

	SYMBOL	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME
1	AAPL	2025-02-26	244.33	244.98	239.13	240.36	44097533
2	AAPL	2025-02-25	248	250	244.91	247.04	48013272
3	AAPL	2025-02-24	244.925	248.86	244.42	247.1	51326396
4	AAPL	2025-02-21	245.95	248.69	245.22	245.55	53197431
5	AAPL	2025-02-20	244.94	246.78	244.29	245.83	32316907
6	AAPL	2025-02-19	244.66	246.01	243.1604	244.87	32204215
7	AAPL	2025-02-18	244.15	245.18	241.84	244.47	48822491
8	AAPL	2025-02-14	241.25	245.55	240.99	244.6	40896227
9	AAPL	2025-02-13	236.91	242.3399	235.57	241.53	53614054
10	AAPL	2025-02-12	231.2	236.96	230.68	236.87	45243292
11	AAPL	2025-02-11	228.2	235.23	228.13	232.62	53718362
12	AAPL	2025-02-10	229.57	230.585	227.2	227.65	33115645
13	AAPL	2025-02-07	232.6	234	227.26	227.63	39707224

Query Details

Query duration 57ms

Rows 90

Query ID 01baa9b5-0004-a895-0...

SYMBOL

AAPL 90

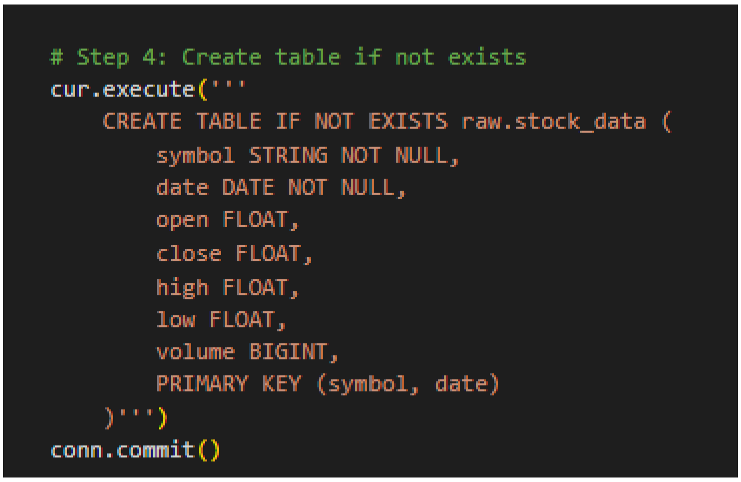
DATE

STOCK_DATA 90 Rows

SYMBOL	VARCHAR(16777216)
DATE	DATE
OPEN	FLOAT
HIGH	FLOAT
LOW	FLOAT
CLOSE	FLOAT
VOLUME	FLOAT

CODE:

```
 cur.execute('''
      CREATE TABLE IF NOT EXISTS raw.stock_data (
        symbol STRING NOT NULL,
        date DATE NOT NULL,
        open FLOAT,

        close FLOAT,

        high FLOAT,

        low FLOAT,

        volume BIGINT,

        PRIMARY KEY (symbol, date)

    )''')

conn.commit()
```

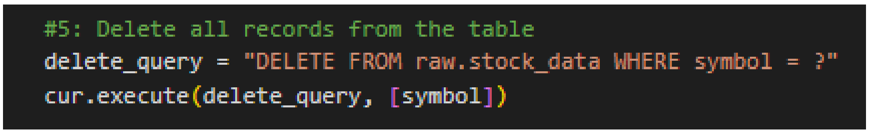


```
# Step 4: Create table if not exists
cur.execute('''
    CREATE TABLE IF NOT EXISTS raw.stock_data (
        symbol STRING NOT NULL,
        date DATE NOT NULL,
        open FLOAT,
        close FLOAT,
        high FLOAT,
        low FLOAT,
        volume BIGINT,
        PRIMARY KEY (symbol, date)
    )''')
conn.commit()
```

5. (+1) Delete all records from the table

CODE:

```
#5: Delete all records from the table
    delete_query = "DELETE FROM raw.stock_data WHERE symbol = ?"
    cur.execute(delete_query, [symbol])
```



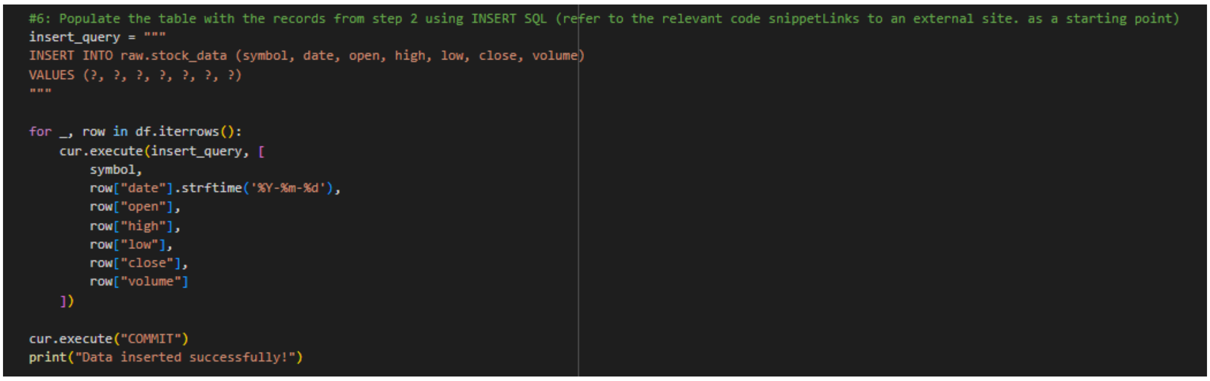
```
#5: Delete all records from the table
delete_query = "DELETE FROM raw.stock_data WHERE symbol = ?"
cur.execute(delete_query, [symbol])
```

6. (+1) Populate the table with the records from step 2 using INSERT SQL (refer to [the relevant code snippetLinks to an external site.]() as a starting point)

CODE:

```
_#6: Populate the table with the records from step 2 using INSERT SQL (refer to the relevant code snippetLinks to an external site. as a starting point)
    insert_query = """
```

```
INSERT INTO raw.stock_data (symbol, date, open, high, low, close, volume)
VALUES (?, ?, ?, ?, ?, ?, ?)
"""
for _, row in df.iterrows():
  cur.execute(insert_query, [
    symbol,
    row["date"].strftime('%Y-%m-%d'),
    row["open"],
    row["high"],
    row["low"],
    row["close"],
    row["volume"]
  ])
cur.execute("COMMIT")
print("Data inserted successfully!")
```



```
#6: Populate the table with the records from step 2 using INSERT SQL (refer to the relevant code snippetLinks to an external site. as a starting point)
insert_query = """
INSERT INTO raw.stock_data (symbol, date, open, high, low, close, volume)
VALUES (?, ?, ?, ?, ?, ?, ?)
"""

for _, row in df.iterrows():
    cur.execute(insert_query, [
        symbol,
        row["date"].strftime('%Y-%m-%d'),
        row["open"],
        row["high"],
        row["low"],
        row["close"],
        row["volume"]
    ])

cur.execute("COMMIT")
print("Data inserted successfully!")
```

7. (+4) Steps 4 and 6 need to be done together

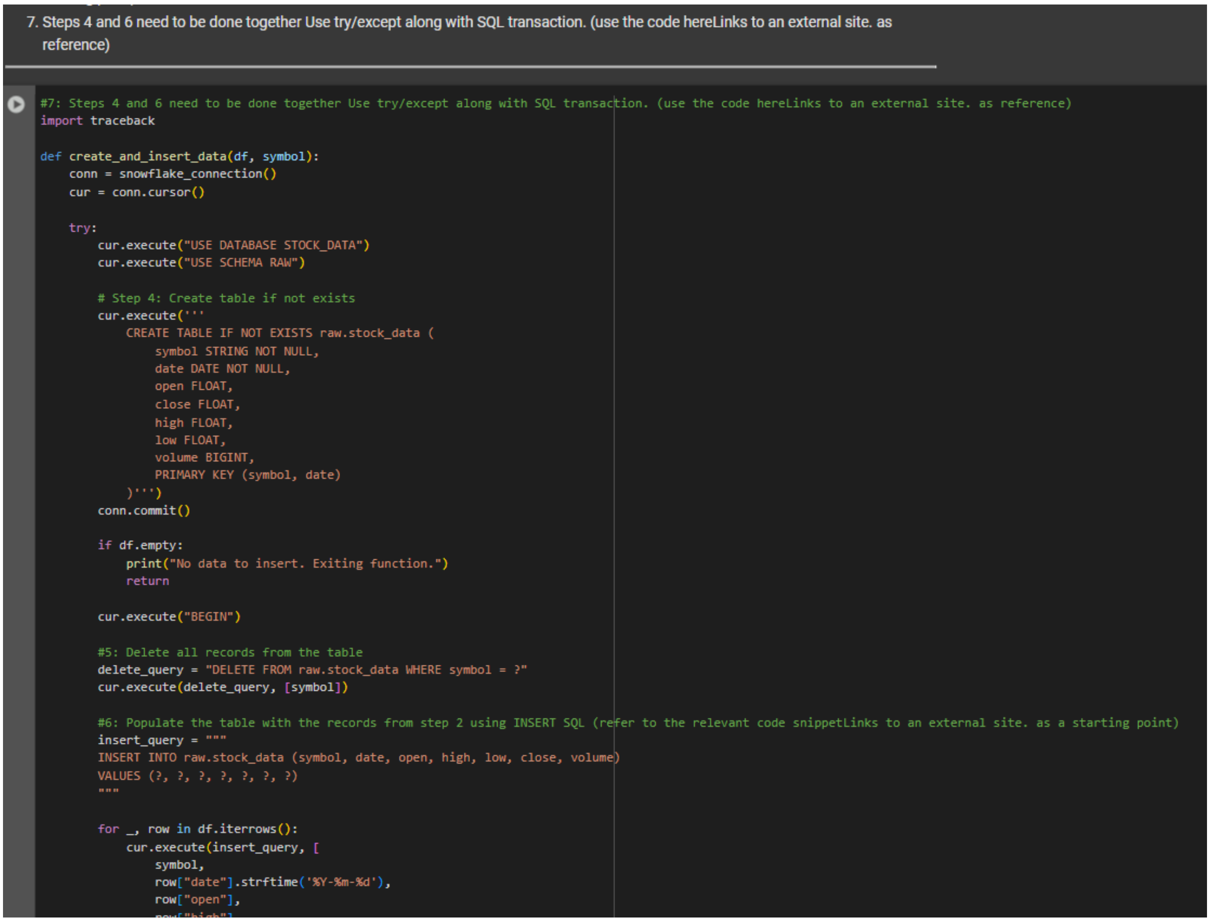
    1. Use try/except along with SQL transaction. (use the code hereLinks to an external site. as reference)

CODE:

#7: Steps 4 and 6 need to be done together Use try/except along with SQL transaction. (use the code hereLinks to an external site. as reference)

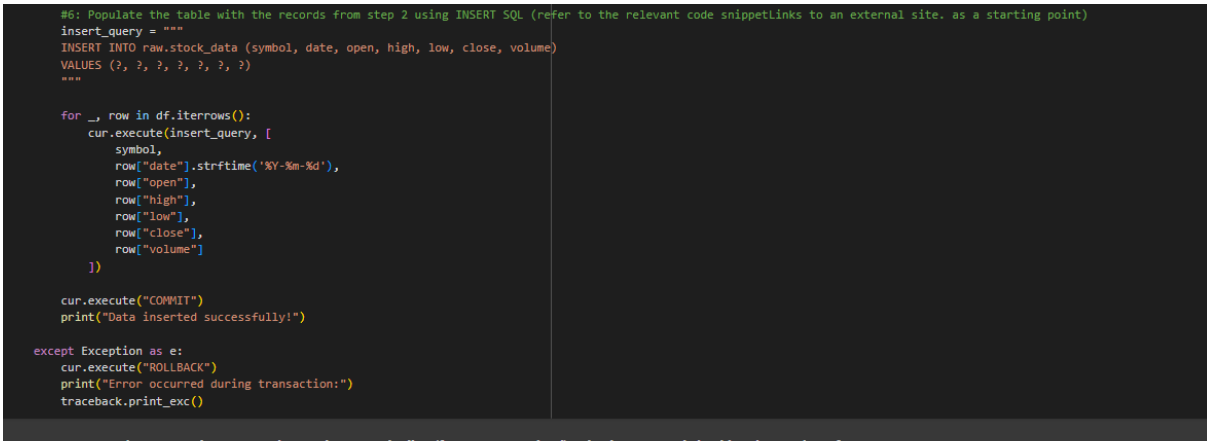
```
import traceback


def create_and_insert_data(df, symbol):

  conn = snowflake_connection()

  cur = conn.cursor()
```

```
    try:
        cur.execute("USE DATABASE STOCK_DATA")
        cur.execute("USE SCHEMA RAW")

        # Step 4: Create table if not exists
        cur.execute('''
            CREATE TABLE IF NOT EXISTS raw.stock_data (
                symbol STRING NOT NULL,
                date DATE NOT NULL,
                open FLOAT,
                close FLOAT,
                high FLOAT,
                low FLOAT,
                volume BIGINT,
                PRIMARY KEY (symbol, date)
            )''')
        conn.commit()

        if df.empty:
            print("No data to insert. Exiting function.")
            return

        cur.execute("BEGIN")

        #5: Delete all records from the table
        delete_query = "DELETE FROM raw.stock_data WHERE symbol = ?"
        cur.execute(delete_query, [symbol])

        #6: Populate the table with the records from step 2 using INSERT SQL (refer to the relevant code snippetLinks to an external site. as a starting point)
        insert_query = """
        INSERT INTO raw.stock_data (symbol, date, open, high, low, close, volume)
        VALUES (?, ?, ?, ?, ?, ?, ?)
        """

        for _, row in df.iterrows():
            cur.execute(insert_query, [
                symbol,
                row["date"].strftime('%Y-%m-%d'),
                row["open"],
                row["high"],
                row["low"],
                row["close"],
                row["volume"]
            ])

        cur.execute("COMMIT")
        print("Data inserted successfully!")

    except Exception as e:
        cur.execute("ROLLBACK")
        print("Error occurred during transaction:")
        traceback.print_exc()
```



7. Steps 4 and 6 need to be done together Use try/except along with SQL transaction. (use the code hereLinks to an external site. as reference)

```
#7: Steps 4 and 6 need to be done together Use try/except along with SQL transaction. (use the code hereLinks to an external site. as reference)
import traceback

def create_and_insert_data(df, symbol):
    conn = snowflake_connection()
    cur = conn.cursor()

    try:
        cur.execute("USE DATABASE STOCK_DATA")
        cur.execute("USE SCHEMA RAW")

        # Step 4: Create table if not exists
        cur.execute('''
            CREATE TABLE IF NOT EXISTS raw.stock_data (
                symbol STRING NOT NULL,
                date DATE NOT NULL,
                open FLOAT,
                close FLOAT,
                high FLOAT,
                low FLOAT,
                volume BIGINT,
                PRIMARY KEY (symbol, date)
            )''')
        conn.commit()

        if df.empty:
            print("No data to insert. Exiting function.")
            return

        cur.execute("BEGIN")

        #5: Delete all records from the table
        delete_query = "DELETE FROM raw.stock_data WHERE symbol = ?"
        cur.execute(delete_query, [symbol])

        #6: Populate the table with the records from step 2 using INSERT SQL (refer to the relevant code snippetLinks to an external site. as a starting point)
        insert_query = """
        INSERT INTO raw.stock_data (symbol, date, open, high, low, close, volume)
        VALUES (?, ?, ?, ?, ?, ?, ?)
        """

        for _, row in df.iterrows():
            cur.execute(insert_query, [
                symbol,
                row["date"].strftime('%Y-%m-%d'),
                row["open"],
```

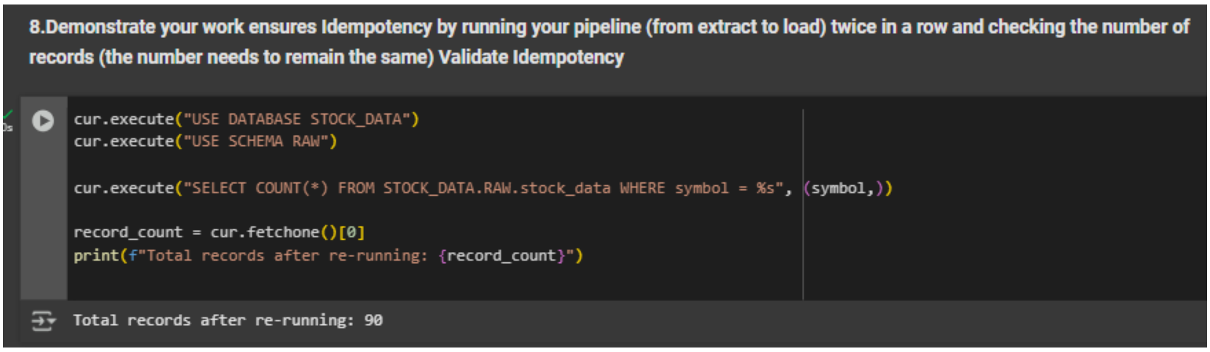


```
        #6: Populate the table with the records from step 2 using INSERT SQL (refer to the relevant code snippetLinks to an external site. as a starting point)
        insert_query = """
        INSERT INTO raw.stock_data (symbol, date, open, high, low, close, volume)
        VALUES (?, ?, ?, ?, ?, ?, ?)
        """

        for _, row in df.iterrows():
            cur.execute(insert_query, [
                symbol,
                row["date"].strftime('%Y-%m-%d'),
                row["open"],
                row["high"],
                row["low"],
                row["close"],
                row["volume"]
            ])

        cur.execute("COMMIT")
        print("Data inserted successfully!")

    except Exception as e:
        cur.execute("ROLLBACK")
        print("Error occurred during transaction:")
        traceback.print_exc()
```

8. **(+1) Demonstrate your work ensures Idempotency by running your pipeline (from extract to load) twice in a row and checking the number of records (the number needs to remain the same)**

CODE:

```
cur.execute("USE DATABASE STOCK_DATA")

cur.execute("USE SCHEMA RAW")


cur.execute("SELECT COUNT(*) FROM STOCK_DATA.RAW.stock_data WHERE
symbol = %s", (symbol,))


record_count = cur.fetchone()[0]

print(f"Total records after re-running: {record_count}")
```



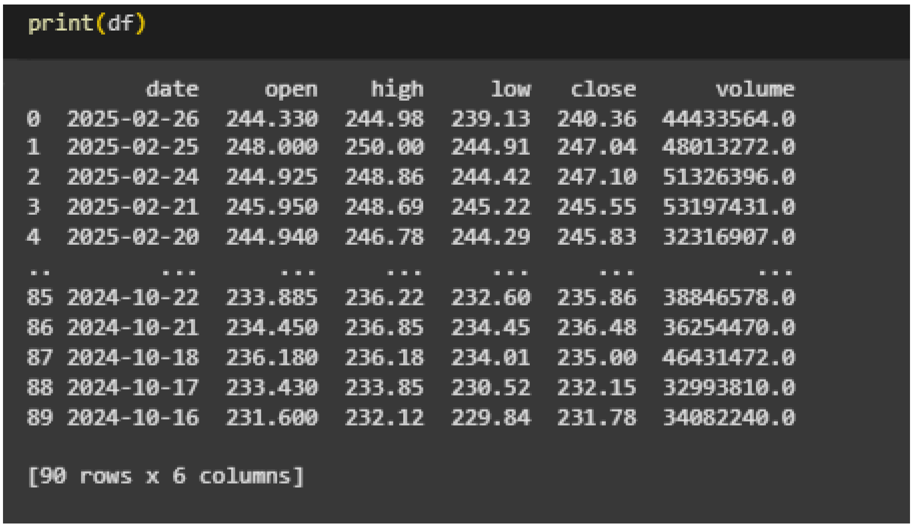
8.Demonstrate your work ensures Idempotency by running your pipeline (from extract to load) twice in a row and checking the number of records (the number needs to remain the same) Validate Idempotency

```
cur.execute("USE DATABASE STOCK_DATA")
cur.execute("USE SCHEMA RAW")

cur.execute("SELECT COUNT(*) FROM STOCK_DATA.RAW.stock_data WHERE symbol = %s", (symbol,))

record_count = cur.fetchone()[0]
print(f"Total records after re-running: {record_count}")
```

```
Total records after re-running: 90
```

<u>CODE:</u>

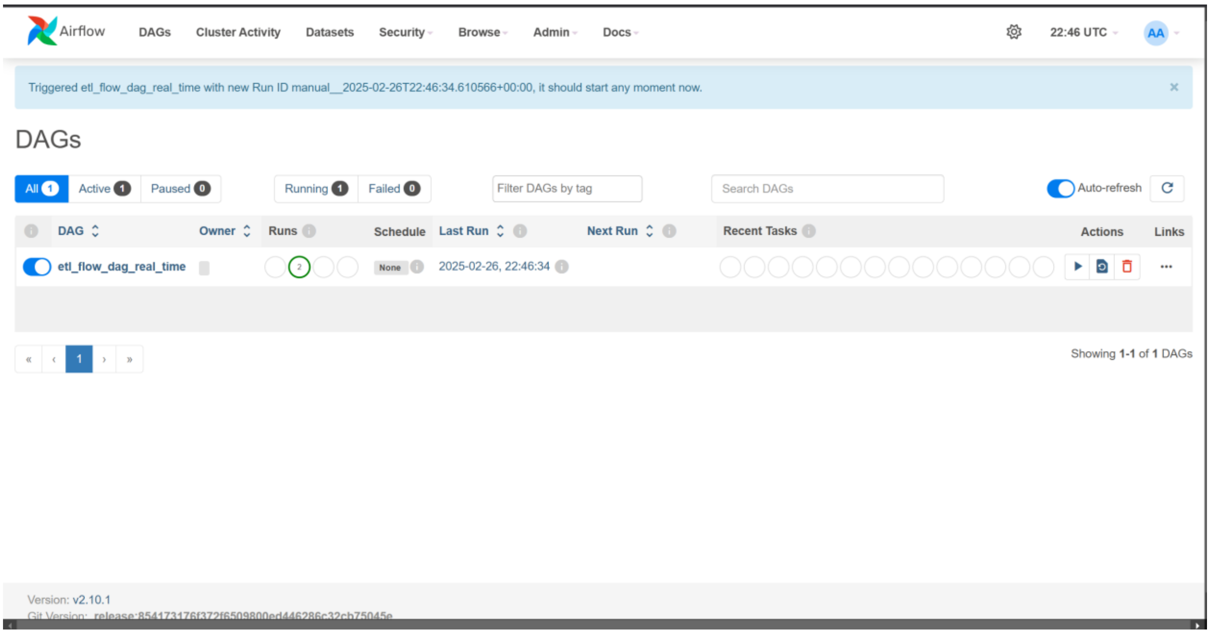
Print(df)



```
print(df)
```

```
          date    open    high     low   close      volume
0   2025-02-26  244.330  244.98  239.13  240.36  44433564.0
1   2025-02-25  248.000  250.00  244.91  247.04  48013272.0
2   2025-02-24  244.925  248.86  244.42  247.10  51326396.0
3   2025-02-21  245.950  248.69  245.22  245.55  53197431.0
4   2025-02-20  244.940  246.78  244.29  245.83  32316907.0
..         ...      ...     ...     ...     ...         ...
85  2024-10-22  233.885  236.22  232.60  235.86  38846578.0
86  2024-10-21  234.450  236.85  234.45  236.48  36254470.0
87  2024-10-18  236.180  236.18  234.01  235.00  46431472.0
88  2024-10-17  233.430  233.85  230.52  232.15  32993810.0
89  2024-10-16  231.600  232.12  229.84  231.78  34082240.0

[90 rows x 6 columns]
```

9. (+2) Follow today's demo and capture Docker Desktop screen showing Airflow



Triggered etl_flow_dag_real_time with new Run ID manual__2025-02-26T22:46:34.610566+00:00, it should start any moment now.

DAGs

All 1 | Active 1 | Paused 0 | Running 1 | Failed 0 | Filter DAGs by tag | Search DAGs | Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
etl_flow_dag_real_time		2	None	2025-02-26, 22:46:34				

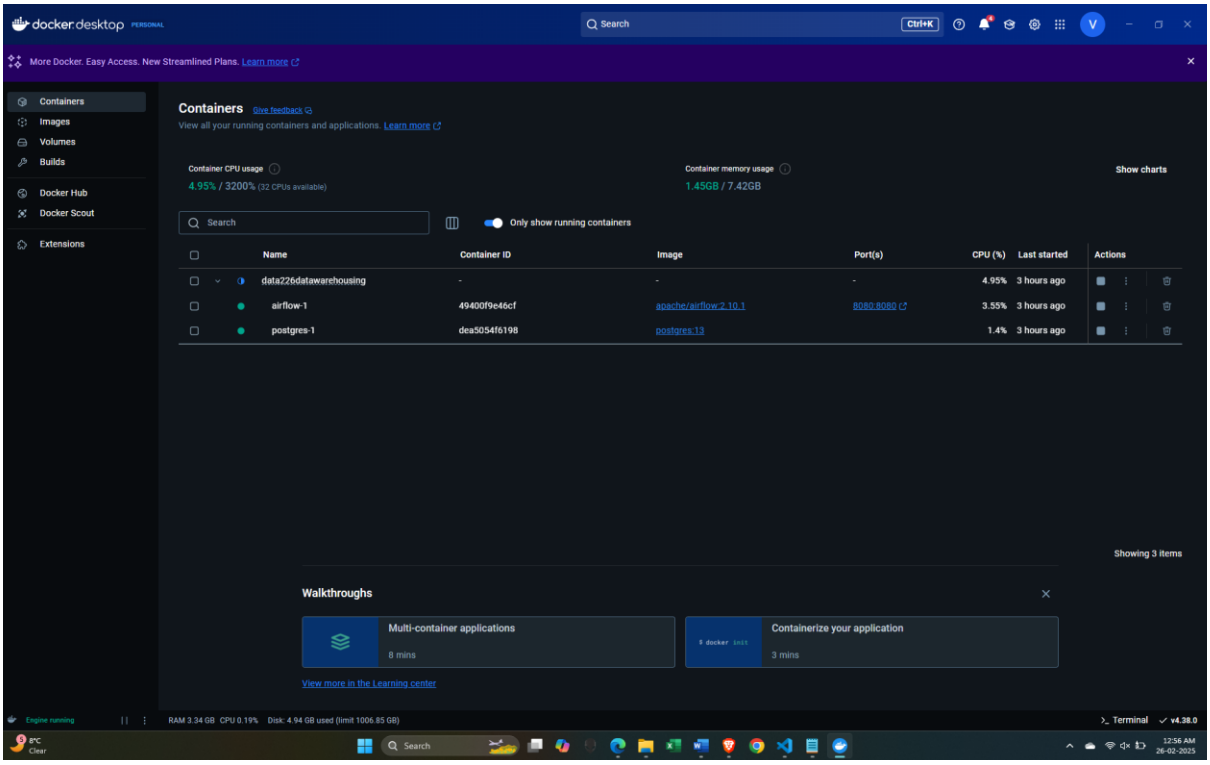
« ‹ 1 › »

Showing 1-1 of 1 DAGs

Version: v2.10.1
Git Version: .release:854173176f372f6509800ed446286c32cb75045e

(+1) Overall formatting

For step 9, here is a screenshot:



More Docker. Easy Access. New Streamlined Plans. Learn more

Containers Give feedback

View all your running containers and applications. Learn more

Container CPU usage
4.95% / 3200% (32 CPUs available)

Container memory usage
1.45GB / 7.42GB

Show charts

Search | Only show running containers

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
□	data226datawarehousing	-	-	-	4.95%	3 hours ago	
□	airflow-1	49400f9e46cf	apache/airflow:2.10.1	8080:8080	3.55%	3 hours ago	
□	postgres-1	dea5054f6198	postgres:13		1.4%	3 hours ago	

Showing 3 items

Walkthroughs

Multi-container applications
8 mins

Containerize your application
3 mins

View more in the Learning center

Engine running | RAM 3.34 GB CPU 0.19% Disk: 4.94 GB used (limit 1006.85 GB) | Terminal | v4.38.0

8°C Clear | Search | 12:56 AM 26-02-2025