Stock Data Load using Python and Snowflake

Shweta Ajay Shinde

Masters in Data Analytics, San Jose State University

Data 226: Data warehousing

Instructor: Keeyong Han

24th Sept 2024

Stock Data Load

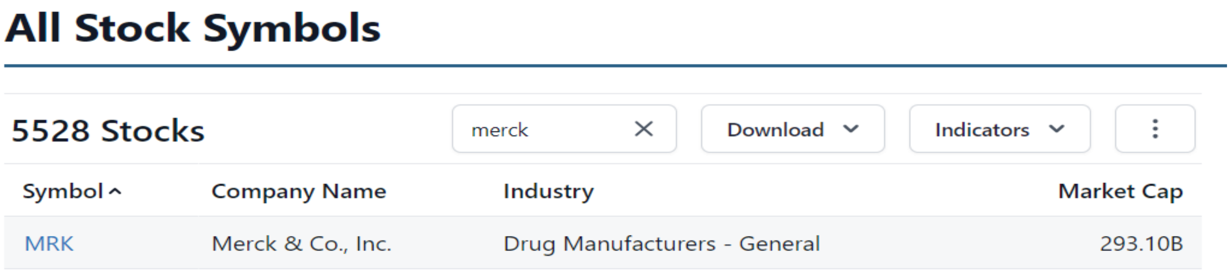
Stock Data Analysis

Please refer to the attached Google Colab notebook side by side for more detailed code and analysis.

Homework4.ipynb

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

Stock Symbol: **MRK** **:-** Merck(Pharmaceutical Company)



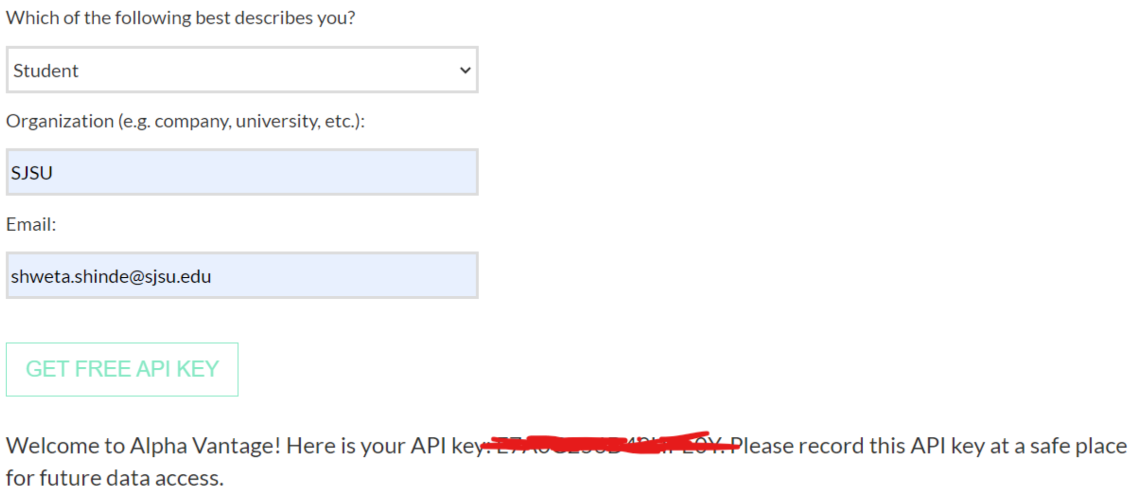
All Stock Symbols

5528 Stocks

merck

Symbol	Company Name	Industry	Market Cap
MRK	Merck & Co., Inc.	Drug Manufacturers - General	293.10B

API key from Alpha Vantage.



Which of the following best describes you?

Student

Organization (e.g. company, university, etc.):

SJSU

Email:

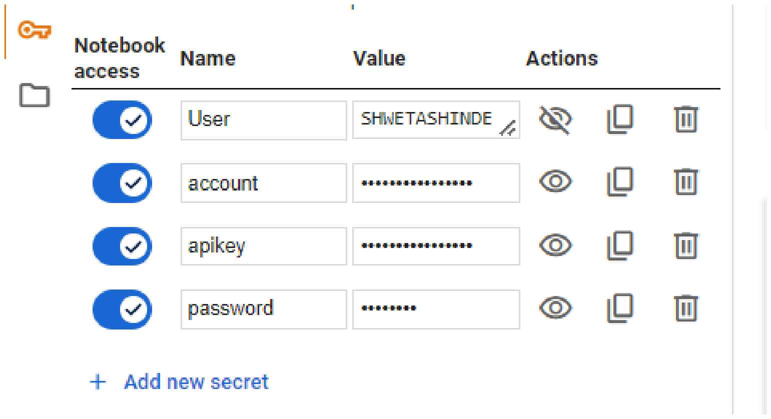
shweta.shinde@sjsu.edu

Welcome to Alpha Vantage! Here is your API key: [redacted] Please record this API key at a safe place for future data access.

Stock Data Load

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

Keys Encrypted:-



Notebook access	Name	Value	Actions
✓	User	SHWETASHINDE	
✓	account	•••••••••••••••	
✓	apikey	•••••••••••••••	
✓	password	••••••••	

3.(+2) Read the last 90 days of the price info via the API (refer to the relevant code snippetLinks to an external site. and you need to add "date").

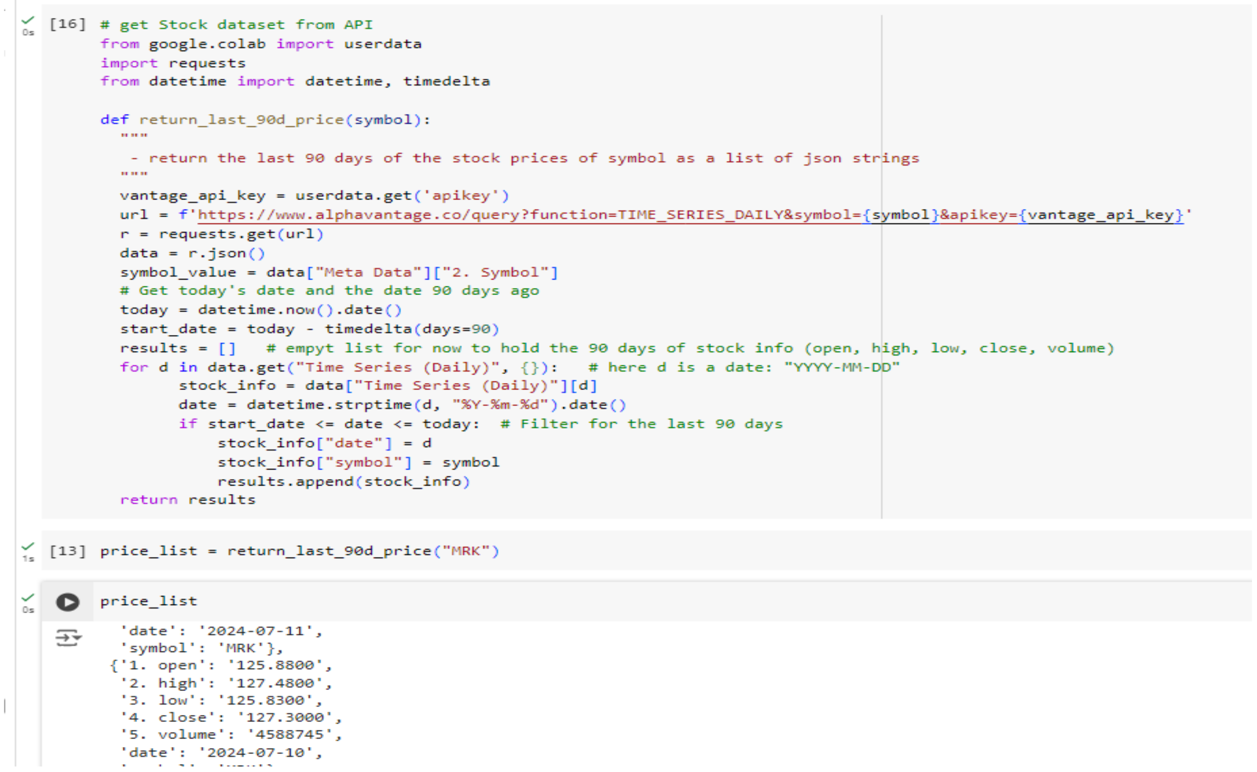
Python Code

```
# Get Stock dataset from API
from google.colab import userdata
import requests
from datetime import datetime, timedelta

def return_last_90d_price(symbol):
    """
     - return the last 90 days of the stock prices of symbol as a list of json strings
    """
    vantage_api_key = userdata.get('apikey')
    url =
  f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symb
  ol}&apikey={vantage_api_key}'
    r = requests.get(url)
    data = r.json()
    symbol_value = data["Meta Data"]["2. Symbol"]
    # Get today's date and the date 90 days ago
    today = datetime.now().date()
    start_date = today - timedelta(days=90)
    results = []   # empyt list for now to hold the 90 days of stock info (open, high, low,
  close, volume)
```

Stock Data Load

```
    for d in data.get("Time Series (Daily)", {}):   # here d is a date: "YYYY-MM-DD"
        stock_info = data["Time Series (Daily)"][d]
         date = datetime.strptime(d, "%Y-%m-%d").date()
        if start_date <= date <= today:  # Filter for the last 90 days
            stock_info["date"] = d
            stock_info["symbol"] = symbol
            results.append(stock_info)
    return results
price_list = return_last_90d_price("MRK")
price_list
[{'1. open': '113.6200',
  ‘2. high': '113.9900',
  '3. low': '112.9400',
  '4. close': '113.0800',
  '5. volume': '11125840',
  'date': '2024-09-26',
  'symbol': 'MRK'},
```

Screenshot:-



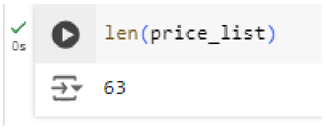
```
[16] # get Stock dataset from API
     from google.colab import userdata
     import requests
     from datetime import datetime, timedelta

     def return_last_90d_price(symbol):
       """
        - return the last 90 days of the stock prices of symbol as a list of json strings
       """
       vantage_api_key = userdata.get('apikey')
       url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}'
       r = requests.get(url)
       data = r.json()
       symbol_value = data["Meta Data"]["2. Symbol"]
       # Get today's date and the date 90 days ago
       today = datetime.now().date()
       start_date = today - timedelta(days=90)
       results = []   # empyt list for now to hold the 90 days of stock info (open, high, low, close, volume)
       for d in data.get("Time Series (Daily)", {}):   # here d is a date: "YYYY-MM-DD"
             stock_info = data["Time Series (Daily)"][d]
             date = datetime.strptime(d, "%Y-%m-%d").date()
             if start_date <= date <= today:  # Filter for the last 90 days
                 stock_info["date"] = d
                 stock_info["symbol"] = symbol
                 results.append(stock_info)
       return results
```

```
[13] price_list = return_last_90d_price("MRK")
```

```
price_list
```

```
  'date': '2024-07-11',
  'symbol': 'MRK'},
 {'1. open': '125.8800',
  '2. high': '127.4800',
  '3. low': '125.8300',
  '4. close': '127.3000',
  '5. volume': '4588745',
  'date': '2024-07-10',
```

Number of records:-



```
len(price_list)
```

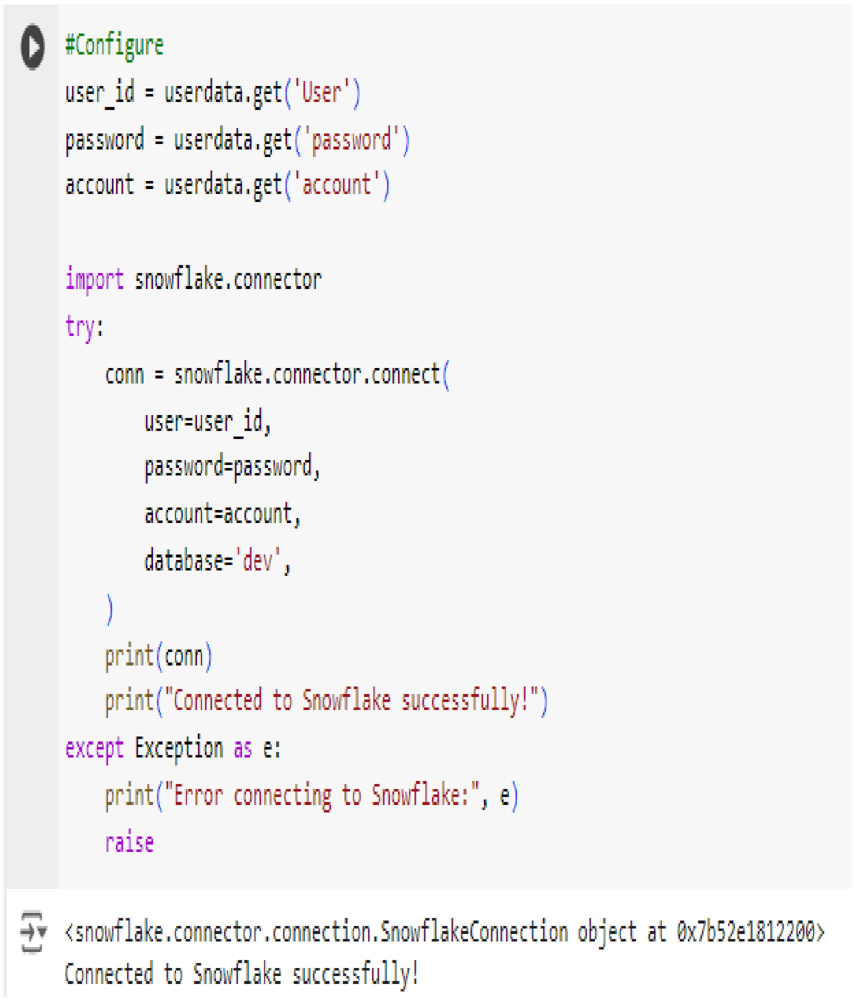
```
63
```

Stock Data Load

4(+1) Create or replace a table with a primary key under raw_data schema to capture the info from the API.

1. It should have **date**, open, high, low, close, volume and symbol fields

   First, we need to connect to DB from python



```
#Configure
user_id = userdata.get('User')
password = userdata.get('password')
account = userdata.get('account')

import snowflake.connector
try:
    conn = snowflake.connector.connect(
        user=user_id,
        password=password,
        account=account,
        database='dev',
    )
    print(conn)
    print("Connected to Snowflake successfully!")
except Exception as e:
    print("Error connecting to Snowflake:", e)
    raise
```

```
<snowflake.connector.connection.SnowflakeConnection object at 0x7b52e1812200>
Connected to Snowflake successfully!
```

Python Code:

```
#Step 2
#Configure
user_id = userdata.get('User')
password = userdata.get('password')
account = userdata.get('account')

import snowflake.connector

try:
   conn =
snowflake.connector.connect(
       user=user_id,
       password=password,
       account=account,
       database='dev',
   )
   print(conn)
   print("Connected to Snowflake
successfully!")
except Exception as e:
   print("Error connecting to
Snowflake:", e)
   raise
```

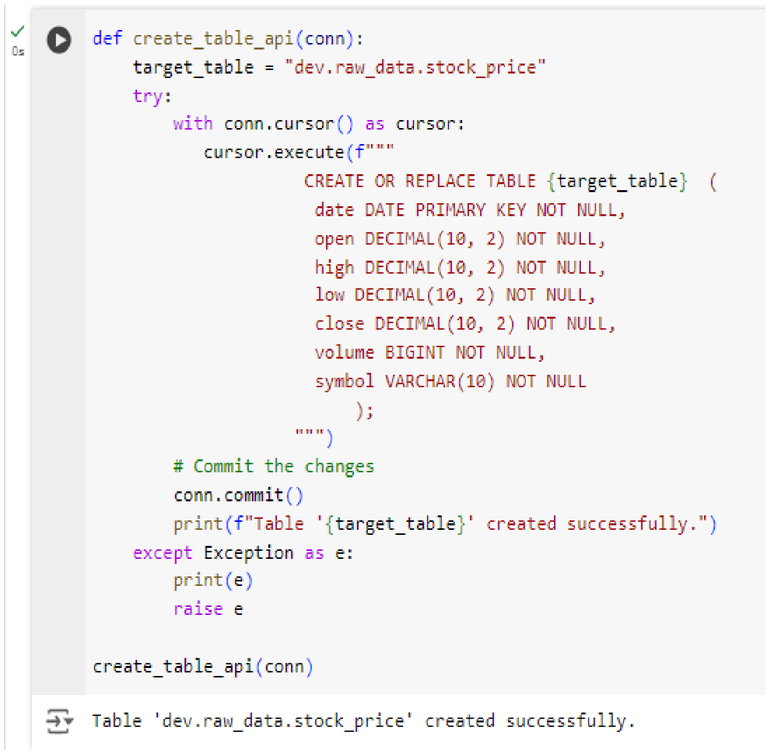
Stock Data Load

Python query to create table in snowflake

```
def create_table_api(conn):
   target_table = "dev.raw_data.stock_price"
   try:
      with conn.cursor() as cursor:
        cursor.execute(f"""
             CREATE OR REPLACE TABLE {target_table}  (
                date DATE  PRIMARY KEY NOT NULL,
                open DECIMAL(10, 2) NOT NULL,
                high DECIMAL(10, 2) NOT NULL,
                low DECIMAL(10, 2) NOT NULL,
                close DECIMAL(10, 2) NOT NULL,
                volume BIGINT NOT NULL,
                symbol VARCHAR(10) NOT NULL
                   );
              """)
      # Commit the changes
      conn.commit()
      print(f"Table '{target_table}' created successfully.")
   except Exception as e:
      print(e)
      raise e

create_table_api(conn)
```

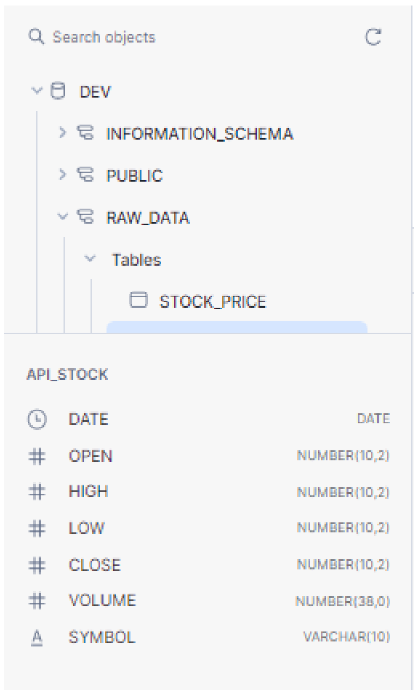
Stock Data Load

Screenshot : **Table Created in snowflake:**



```
def create_table_api(conn):
    target_table = "dev.raw_data.stock_price"
    try:
        with conn.cursor() as cursor:
           cursor.execute(f"""
                     CREATE OR REPLACE TABLE {target_table}  (
                      date DATE PRIMARY KEY NOT NULL,
                      open DECIMAL(10, 2) NOT NULL,
                      high DECIMAL(10, 2) NOT NULL,
                      low DECIMAL(10, 2) NOT NULL,
                      close DECIMAL(10, 2) NOT NULL,
                      volume BIGINT NOT NULL,
                      symbol VARCHAR(10) NOT NULL
                          );
                    """)
        # Commit the changes
        conn.commit()
        print(f"Table '{target_table}' created successfully.")
    except Exception as e:
        print(e)
        raise e

create_table_api(conn)
```

```
Table 'dev.raw_data.stock_price' created successfully.
```



API_STOCK

DATE	DATE
OPEN	NUMBER(10,2)
HIGH	NUMBER(10,2)
LOW	NUMBER(10,2)
CLOSE	NUMBER(10,2)
VOLUME	NUMBER(38,0)
SYMBOL	VARCHAR(10)

5.(+2) Populate the table with the records from step 3 using INSERT SQL (refer to the relevant code snippetLinks to an external site. as a starting point)

Python Code:

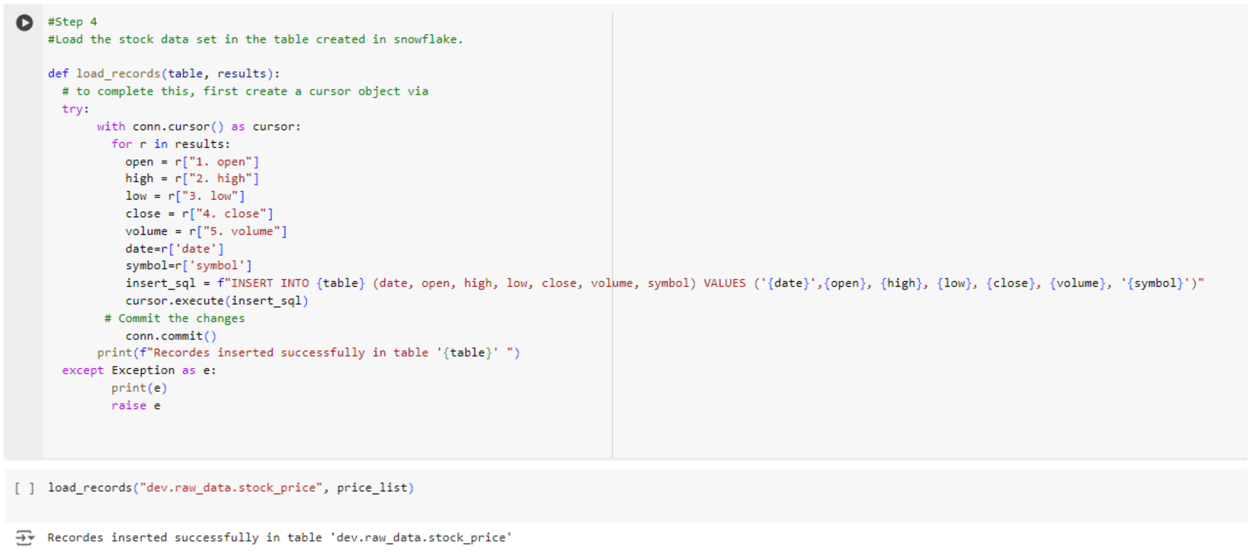
```
def load_records(table, results):
    target_table = "dev.raw_data.stock_price"
        try:
          with conn.cursor() as cursor:
            for r in results:
                open = r["1. open"]
                 high = r["2. high"]
                 low = r["3. low"]
                 close = r["4. close"]
                  volume = r["5. volume"]
                  date=r['date']
                   symbol=r['symbol']
                   insert_sql = f"INSERT INTO {table} (date, open, high, low,
close, volume, symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume},
'{symbol}')"
                   cursor.execute(insert_sql)
                   conn.commit()
              print(f"Recordes inserted successfully in table '{target_table}' ")
        except Exception as e:
        print(e)
```

Stock Data Load

raise e

load_records(dev.raw_data.stock_price,price_list)

Screenshot:

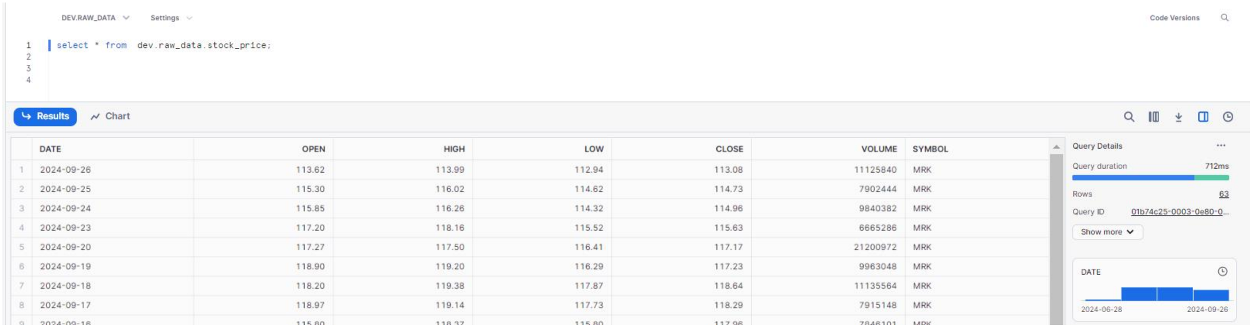


```
#Step 4
#Load the stock data set in the table created in snowflake.

def load_records(table, results):
  # to complete this, first create a cursor object via
  try:
       with conn.cursor() as cursor:
         for r in results:
           open = r["1. open"]
           high = r["2. high"]
           low = r["3. low"]
           close = r["4. close"]
           volume = r["5. volume"]
           date=r['date']
           symbol=r['symbol']
           insert_sql = f"INSERT INTO {table} (date, open, high, low, close, volume, symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume}, '{symbol}')"
           cursor.execute(insert_sql)
        # Commit the changes
           conn.commit()
       print(f"Recordes inserted successfully in table '{table}' ")
  except Exception as e:
         print(e)
         raise e
```

```
[ ] load_records("dev.raw_data.stock_price", price_list)
```

```
Recordes inserted successfully in table 'dev.raw_data.stock_price'
```

Snowflake Screenshot



```
select * from dev.raw_data.stock_price;
```

	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME	SYMBOL
1	2024-09-26	113.62	113.99	112.94	113.08	11125840	MRK
2	2024-09-25	115.30	116.02	114.62	114.73	7902444	MRK
3	2024-09-24	115.85	116.26	114.32	114.96	9840382	MRK
4	2024-09-23	117.20	118.16	115.52	115.63	6665286	MRK
5	2024-09-20	117.27	117.50	116.41	117.17	21200972	MRK
6	2024-09-19	118.90	119.20	116.29	117.23	9963048	MRK
7	2024-09-18	118.20	119.38	117.87	118.64	11135564	MRK
8	2024-09-17	118.97	119.14	117.73	118.29	7915148	MRK

6.(+4) Steps 4 and 5 need to be done together

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

For this we can use 2 methods

1. **Incremental Load:** An incremental load transfers only new or changed data from a source to a target database. This method saves time and resources by updating only what's necessary. It's commonly used for regular updates to keep the target database current.
   # Load data in existing table.
   #Step1:- Create or replace table and load the data in Staging table.
   #Step2:- Using the staging table insert the records if not exist in target table or update if any data changed for those records.
   Python code:-

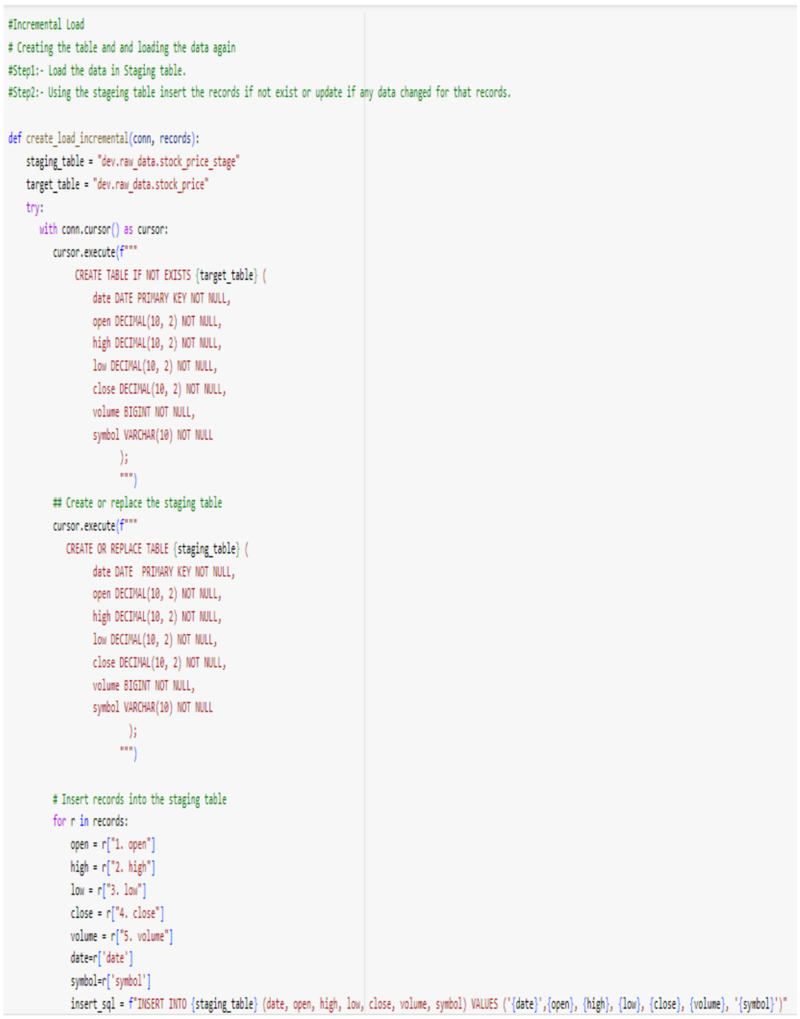
Stock Data Load

```
def create_load_incremental(conn, records):
    staging_table = "dev.raw_data.stock_price_stage"
    target_table = "dev.raw_data.stock_price"
    try:
        with conn.cursor() as cursor:
            cursor.execute(f"""
                CREATE TABLE IF NOT EXISTS {target_table} (
                    date DATE PRIMARY KEY NOT NULL,
                    open DECIMAL(10, 2) NOT NULL,
                    high DECIMAL(10, 2) NOT NULL,
                    low DECIMAL(10, 2) NOT NULL,
                    close DECIMAL(10, 2) NOT NULL,
                    volume BIGINT NOT NULL,
                    symbol VARCHAR(10) NOT NULL
                        );
                        """)
            ## Create or replace the staging table
            cursor.execute(f"""
              CREATE OR REPLACE TABLE {staging_table} (
                    date DATE  PRIMARY KEY NOT NULL,
                    open DECIMAL(10, 2) NOT NULL,
                    high DECIMAL(10, 2) NOT NULL,
                    low DECIMAL(10, 2) NOT NULL,
                    close DECIMAL(10, 2) NOT NULL,
                    volume BIGINT NOT NULL,
                    symbol VARCHAR(10) NOT NULL
                         );
                        """)
            # Insert records into the staging table
            for r in records:
                open = r["1. open"]
                high = r["2. high"]
                low = r["3. low"]
                close = r["4. close"]
                volume = r["5. volume"]
                date=r['date']
                symbol=r['symbol']
                insert_sql = f"INSERT INTO {staging_table} (date, open, high, low, close, 
volume, symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume}, 
'{symbol}')"
                cursor.execute(insert_sql) # Execute within the with block

        conn.commit()
```

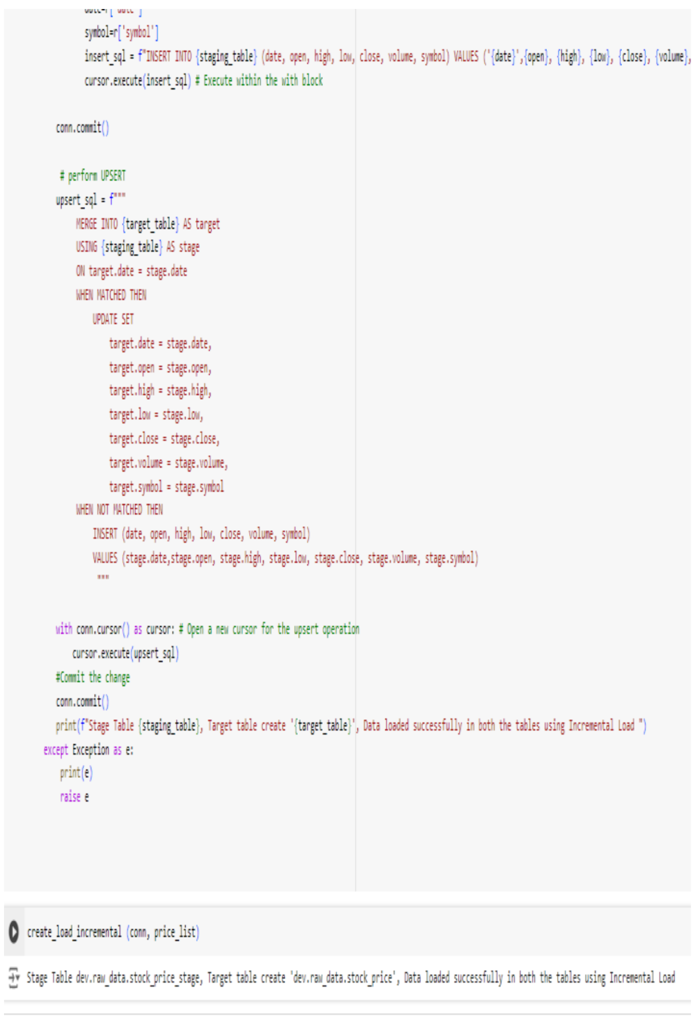
```
         # Perform UPSERT
        upsert_sql = f"""
            MERGE INTO {target_table} AS target
            USING {staging_table} AS stage
            ON target.date = stage.date
            WHEN MATCHED THEN
                UPDATE SET
                    target.date = stage.date,
                    target.open = stage.open,
                    target.high = stage.high,
                    target.low = stage.low,
                    target.close = stage.close,
                    target.volume = stage.volume,
                    target.symbol = stage.symbol
            WHEN NOT MATCHED THEN
                INSERT (date, open, high, low, close, volume, symbol)
                VALUES (stage.date,stage.open, stage.high, stage.low, stage.close,
stage.volume, stage.symbol)
                 """
        with conn.cursor() as cursor: # Open a new cursor for the upsert operation
            cursor.execute(upsert_sql)
        #Commit the change
        conn.commit()
        print(f"Stage Table {staging_table}, Target table create '{target_table}', Data loaded
successfully in both the tables using Incremental Load ")
    except Exception as e:
         print(e)
         raise e


create_load_incremental (conn, price_list)
```

Stock Data Load

Python code



```
#Incremental Load
# Creating the table and and loading the data again
#Step1:- Load the data in Staging table.
#Step2:- Using the stageing table insert the records if not exist or update if any data changed for that records.

def create_load_incremental(conn, records):
    staging_table = "dev.raw_data.stock_price_stage"
    target_table = "dev.raw_data.stock_price"
    try:
      with conn.cursor() as cursor:
        cursor.execute(f"""
            CREATE TABLE IF NOT EXISTS {target_table} (
                date DATE PRIMARY KEY NOT NULL,
                open DECIMAL(10, 2) NOT NULL,
                high DECIMAL(10, 2) NOT NULL,
                low DECIMAL(10, 2) NOT NULL,
                close DECIMAL(10, 2) NOT NULL,
                volume BIGINT NOT NULL,
                symbol VARCHAR(10) NOT NULL
                    );
                    """)
        ## Create or replace the staging table
        cursor.execute(f"""
          CREATE OR REPLACE TABLE {staging_table} (
                date DATE  PRIMARY KEY NOT NULL,
                open DECIMAL(10, 2) NOT NULL,
                high DECIMAL(10, 2) NOT NULL,
                low DECIMAL(10, 2) NOT NULL,
                close DECIMAL(10, 2) NOT NULL,
                volume BIGINT NOT NULL,
                symbol VARCHAR(10) NOT NULL
                     );
                    """)


        # Insert records into the staging table
        for r in records:
          open = r["1. open"]
          high = r["2. high"]
          low = r["3. low"]
          close = r["4. close"]
          volume = r["5. volume"]
          date=r['date']
          symbol=r['symbol']
          insert_sql = f"INSERT INTO {staging_table} (date, open, high, low, close, volume, symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume}, '{symbol}')"
```



```
          symbol=r['symbol']
          insert_sql = f"INSERT INTO {staging_table} (date, open, high, low, close, volume, symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume},
```

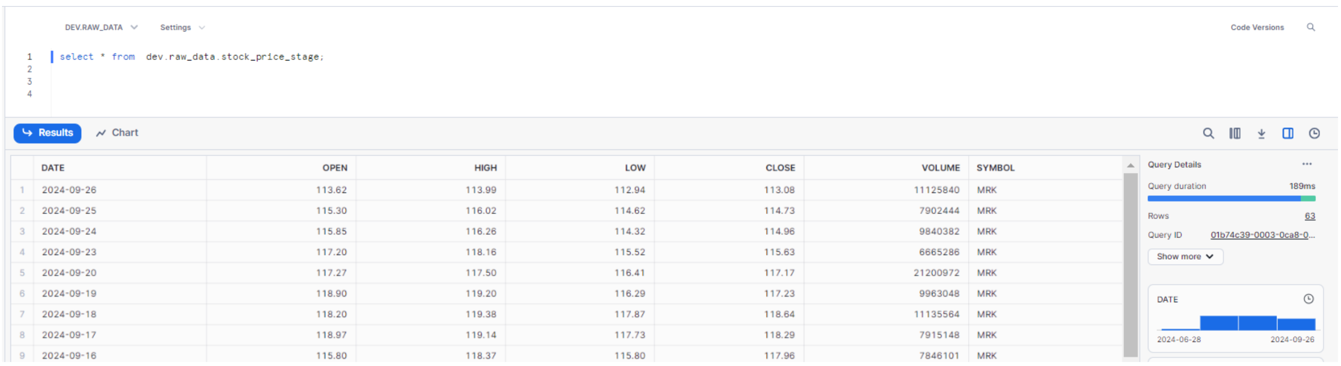
```
          cursor.execute(insert_sql) # Execute within the with block

      conn.commit()

       # perform UPSERT
      upsert_sql = f"""
         MERGE INTO {target_table} AS target
         USING {staging_table} AS stage
         ON target.date = stage.date
         WHEN MATCHED THEN
            UPDATE SET
               target.date = stage.date,
               target.open = stage.open,
               target.high = stage.high,
               target.low = stage.low,
               target.close = stage.close,
               target.volume = stage.volume,
               target.symbol = stage.symbol
         WHEN NOT MATCHED THEN
            INSERT (date, open, high, low, close, volume, symbol)
            VALUES (stage.date,stage.open, stage.high, stage.low, stage.close, stage.volume, stage.symbol)
             """


      with conn.cursor() as cursor: # Open a new cursor for the upsert operation
         cursor.execute(upsert_sql)
      #Commit the change
      conn.commit()
      print(f"Stage Table {staging_table}, Target table create '{target_table}', Data loaded successfully in both the tables using Incremental Load ")
    except Exception as e:
       print(e)
       raise e
```

```
create_load_incremental (conn, price_list)
```

```
Stage Table dev.raw_data.stock_price_stage, Target table create 'dev.raw_data.stock_price', Data loaded successfully in both the tables using Incremental Load
```

Snowflake Screenshot:-

Stage table



```
select * from  dev.raw_data.stock_price_stage;
```

DATE	OPEN	HIGH	LOW	CLOSE	VOLUME	SYMBOL
2024-09-26	113.62	113.99	112.94	113.08	11125840	MRK
2024-09-25	115.30	116.02	114.62	114.73	7902444	MRK
2024-09-24	115.85	116.26	114.32	114.96	9840382	MRK
2024-09-23	117.20	118.16	115.52	115.63	6665286	MRK
2024-09-20	117.27	117.50	116.41	117.17	21200972	MRK
2024-09-19	118.90	119.20	116.29	117.23	9963048	MRK
2024-09-18	118.20	119.38	117.87	118.64	11135564	MRK
2024-09-17	118.97	119.14	117.73	118.29	7915148	MRK
2024-09-16	115.80	118.37	115.80	117.96	7846101	MRK

Query Details

Query duration	189ms
Rows	63
Query ID	01b74c39-0003-0ca8-0...

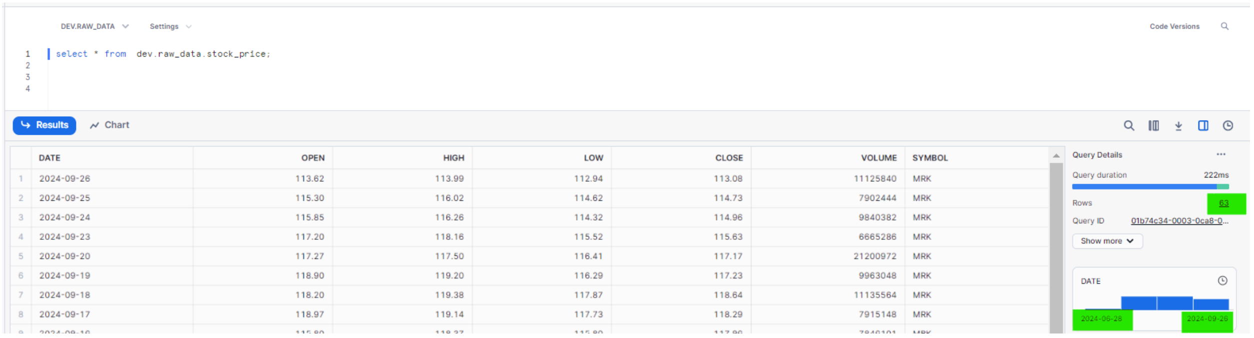
Show more

DATE

2024-06-28 2024-09-26

Stock Data Load

Target Table



```
select * from  dev.raw_data.stock_price;
```

	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME	SYMBOL
1	2024-09-26	113.62	113.99	112.94	113.08	11125840	MRK
2	2024-09-25	115.30	116.02	114.62	114.73	7902444	MRK
3	2024-09-24	115.85	116.26	114.32	114.96	9840382	MRK
4	2024-09-23	117.20	118.16	115.52	115.63	6665286	MRK
5	2024-09-20	117.27	117.50	116.41	117.17	21200972	MRK
6	2024-09-19	118.90	119.20	116.29	117.23	9963048	MRK
7	2024-09-18	118.20	119.38	117.87	118.64	11135564	MRK
8	2024-09-17	118.97	119.14	117.73	118.29	7915148	MRK

Query Details
Query duration 222ms
Rows 63
Query ID 01b74c34-0003-0ca8-0...
Show more
DATE
2024-06-28 2024-09-26

2. **Full Load:** A full load is when all the data from a source is moved to a target database, replacing any old data. This process makes sure that the target has the latest information .

#Step 1: We will create or replace table every time we run the load.
#Step 2: Once the table is created, we will insert the records again from initial.

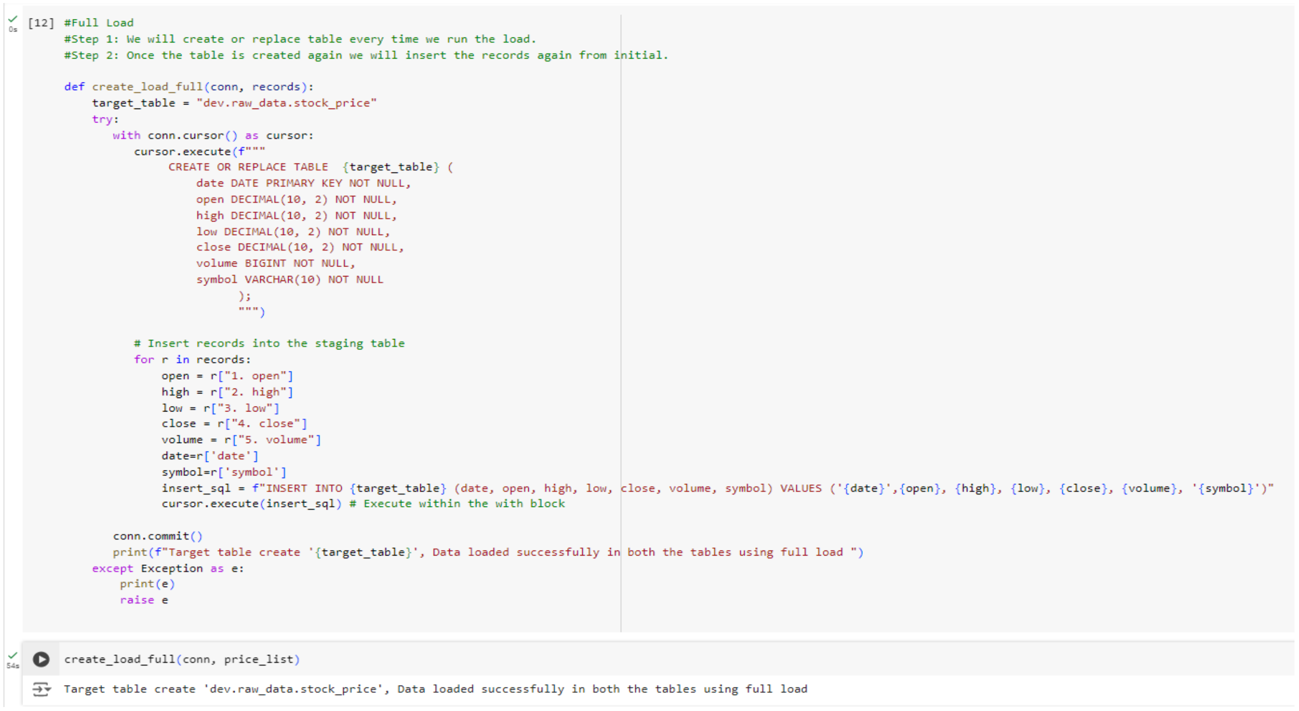
Python Code

```
def create_load_full(conn, records):

      target_table = "dev.raw_data.stock_price"
      try:
        with conn.cursor() as cursor:
          cursor.execute(f"""
              CREATE OR REPLACE TABLE  {target_table} (
                 date DATE PRIMARY KEY NOT NULL,
                 open DECIMAL(10, 2) NOT NULL,
                 high DECIMAL(10, 2) NOT NULL,
                 low DECIMAL(10, 2) NOT NULL,
                 close DECIMAL(10, 2) NOT NULL,
                 volume BIGINT NOT NULL,
                 symbol VARCHAR(10) NOT NULL
                      );
                      """)
          # Insert records into the staging table
          for r in records:
             open = r["1. open"]
             high = r["2. high"]
             low = r["3. low"]
             close = r["4. close"]
             volume = r["5. volume"]
             date=r['date']
             symbol=r['symbol']
             insert_sql = f"INSERT INTO {target_table} (date, open, high, low, close, volume,
    symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume}, '{symbol}')"
             cursor.execute(insert_sql) # Execute within the with block
        conn.commit()
```

Stock Data Load

```
        print(f"Target table create '{target_table}', Data loaded successfully in both the tables
using full load ")
    except Exception as e:
        print(e)
        raise e

    create_load_full(conn, price_list)
```

Screenshot:

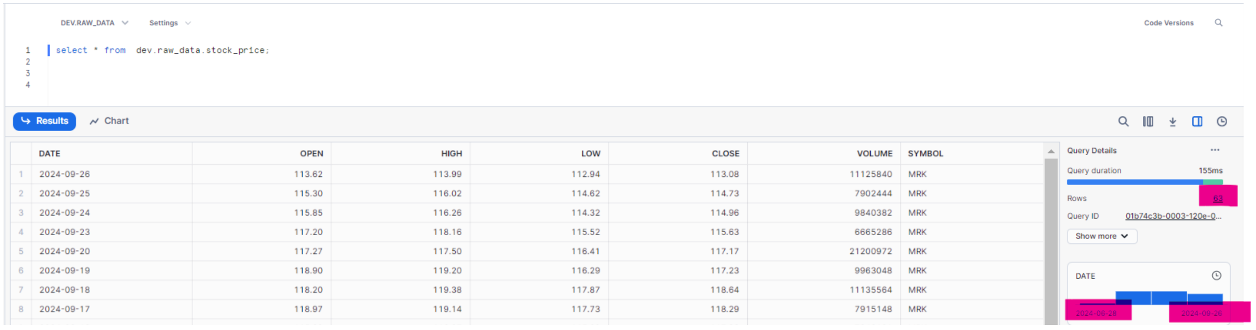


```
[12] #Full Load
     #Step 1: We will create or replace table every time we run the load.
     #Step 2: Once the table is created again we will insert the records again from initial.

     def create_load_full(conn, records):
         target_table = "dev.raw_data.stock_price"
         try:
            with conn.cursor() as cursor:
               cursor.execute(f"""
                    CREATE OR REPLACE TABLE  {target_table} (
                        date DATE PRIMARY KEY NOT NULL,
                        open DECIMAL(10, 2) NOT NULL,
                        high DECIMAL(10, 2) NOT NULL,
                        low DECIMAL(10, 2) NOT NULL,
                        close DECIMAL(10, 2) NOT NULL,
                        volume BIGINT NOT NULL,
                        symbol VARCHAR(10) NOT NULL
                              );
                              """)

               # Insert records into the staging table
               for r in records:
                   open = r["1. open"]
                   high = r["2. high"]
                   low = r["3. low"]
                   close = r["4. close"]
                   volume = r["5. volume"]
                   date=r['date']
                   symbol=r['symbol']
                   insert_sql = f"INSERT INTO {target_table} (date, open, high, low, close, volume, symbol) VALUES ('{date}',{open}, {high}, {low}, {close}, {volume}, '{symbol}')"
                   cursor.execute(insert_sql) # Execute within the with block

            conn.commit()
            print(f"Target table create '{target_table}', Data loaded successfully in both the tables using full load ")
         except Exception as e:
             print(e)
             raise e
```

```
create_load_full(conn, price_list)
```

```
Target table create 'dev.raw_data.stock_price', Data loaded successfully in both the tables using full load
```

Snowflake screenshot



```
select * from  dev.raw_data.stock_price;
```

	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME	SYMBOL
1	2024-09-26	113.62	113.99	112.94	113.08	11125840	MRK
2	2024-09-25	115.30	116.02	114.62	114.73	7902444	MRK
3	2024-09-24	115.85	116.26	114.32	114.96	9840382	MRK
4	2024-09-23	117.20	118.16	115.52	115.63	6665286	MRK
5	2024-09-20	117.27	117.50	116.41	117.17	21200972	MRK
6	2024-09-19	118.90	119.20	116.29	117.23	9963048	MRK
7	2024-09-18	118.20	119.38	117.87	118.64	11135564	MRK
8	2024-09-17	118.97	119.14	117.73	118.29	7915148	MRK

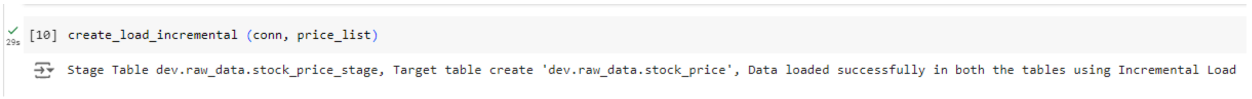
Query Details
Query duration 155ms
Rows 63
Query ID 01b74c3b-0003-120e-0...
Show more
DATE
2024-06-28 2024-09-26

Stock Data Load

7.(+1) Demonstrate your work ensures Idempotency by running it twice in a row and checking the number of records.

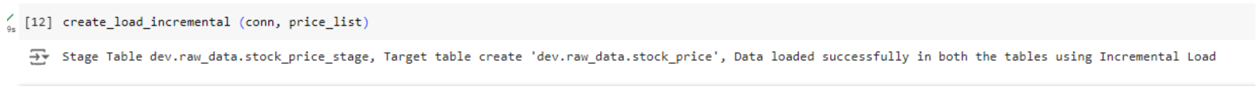
Idempotency means that when you perform an action multiple times, the result will be the same as if you only did it once. In case of data, even if we run the load multiple time it should not create duplicate records. If it is a full load, it should be deleted and loaded again. In case of incremental load, it should not be inserted unless it's a new record or and update in the existing one .

For Incremental Load(CREATE TABLE IF NOT EXIST):-



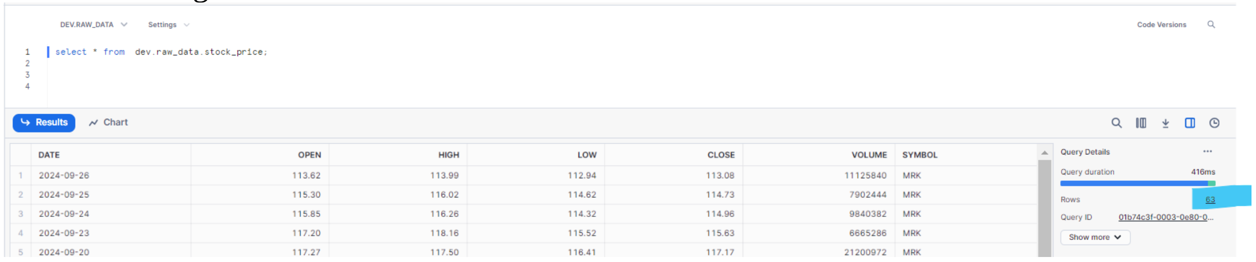
```
29s [10] create_load_incremental (conn, price_list)

Stage Table dev.raw_data.stock_price_stage, Target table create 'dev.raw_data.stock_price', Data loaded successfully in both the tables using Incremental Load
```

After running the code multiple times in a row.



```
9s [12] create_load_incremental (conn, price_list)

Stage Table dev.raw_data.stock_price_stage, Target table create 'dev.raw_data.stock_price', Data loaded successfully in both the tables using Incremental Load
```

Snowflake Target:

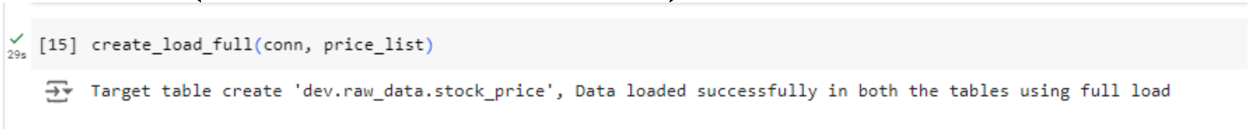


```
select * from dev.raw_data.stock_price;
```

	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME	SYMBOL
1	2024-09-26	113.62	113.99	112.94	113.08	11125840	MRK
2	2024-09-25	115.30	116.02	114.62	114.73	7902444	MRK
3	2024-09-24	115.85	116.26	114.32	114.96	9840382	MRK
4	2024-09-23	117.20	118.16	115.52	115.63	6665286	MRK
5	2024-09-20	117.27	117.50	116.41	117.17	21200972	MRK

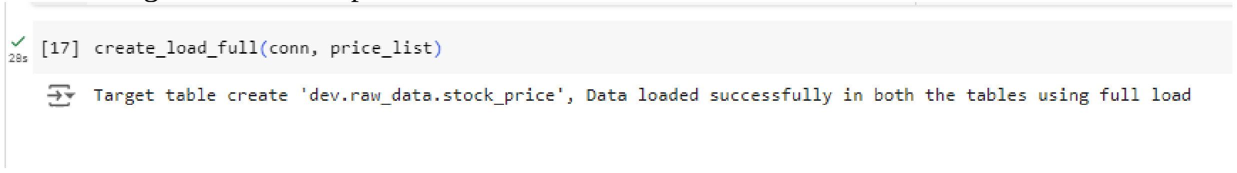
Query Details
Query duration 416ms
Rows 63
Query ID 01b74c3f-0003-0e80-0...
Show more

For Full Load(CREATE OR REPLACE TABLE):-



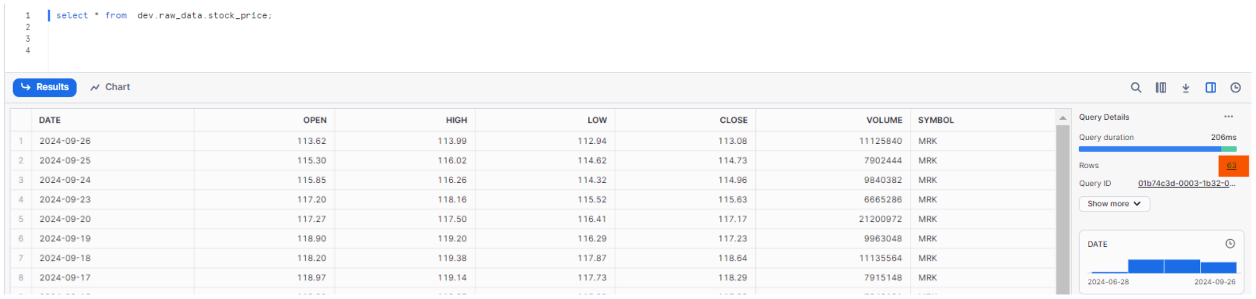
```
29s [15] create_load_full(conn, price_list)

Target table create 'dev.raw_data.stock_price', Data loaded successfully in both the tables using full load
```

After running the code multiple times in a row.



```
28s [17] create_load_full(conn, price_list)

Target table create 'dev.raw_data.stock_price', Data loaded successfully in both the tables using full load
```

Snowflake table:



```
select * from dev.raw_data.stock_price;
```

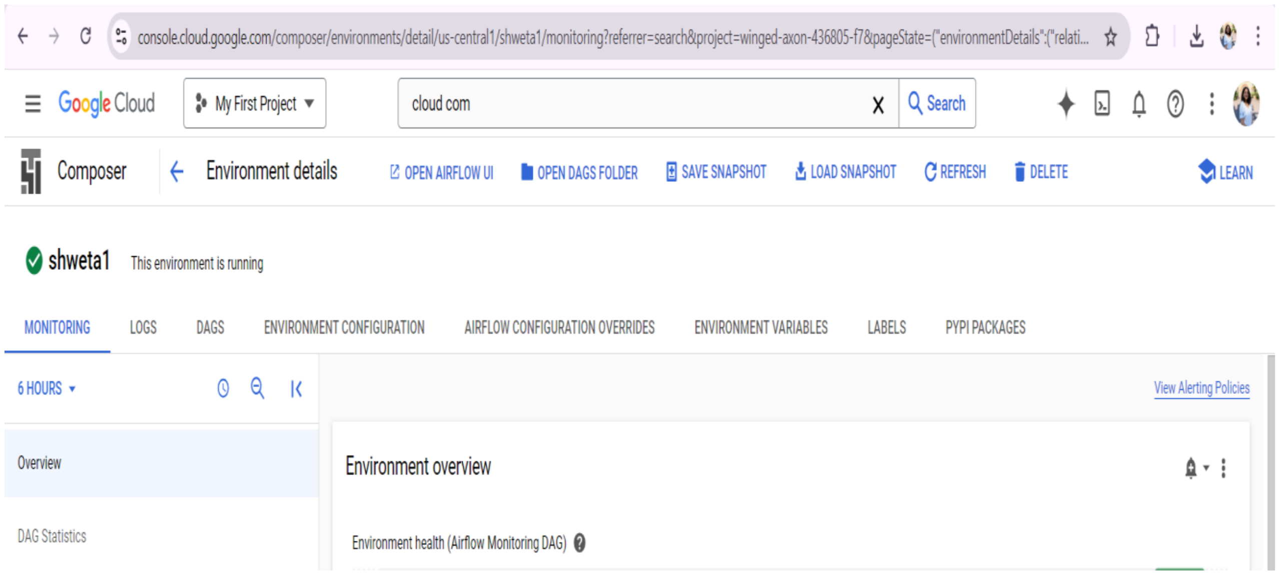
	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME	SYMBOL
1	2024-09-26	113.62	113.99	112.94	113.08	11125840	MRK
2	2024-09-25	115.30	116.02	114.62	114.73	7902444	MRK
3	2024-09-24	115.85	116.26	114.32	114.96	9840382	MRK
4	2024-09-23	117.20	118.16	115.52	115.63	6665286	MRK
5	2024-09-20	117.27	117.50	116.41	117.17	21200972	MRK
6	2024-09-19	118.90	119.20	116.29	117.23	9963048	MRK
7	2024-09-18	118.20	119.38	117.87	118.64	11135564	MRK
8	2024-09-17	118.97	119.14	117.73	118.29	7915148	MRK

Query Details
Query duration 206ms
Rows 63
Query ID 01b74c3d-0003-1b32-0...
Show more

Stock Data Load

8.(+1) Follow today's demo (you can find relevant slides from today's lecture notes too) and capture your Cloud Composer Environment screen.

Composer Environment Screenshot:-



console.cloud.google.com/composer/environments/detail/us-central1/shweta1/monitoring?referrer=search&project=winged-axon-436805-f7&pageState=("environmentDetails":("relati...

shweta1 This environment is running