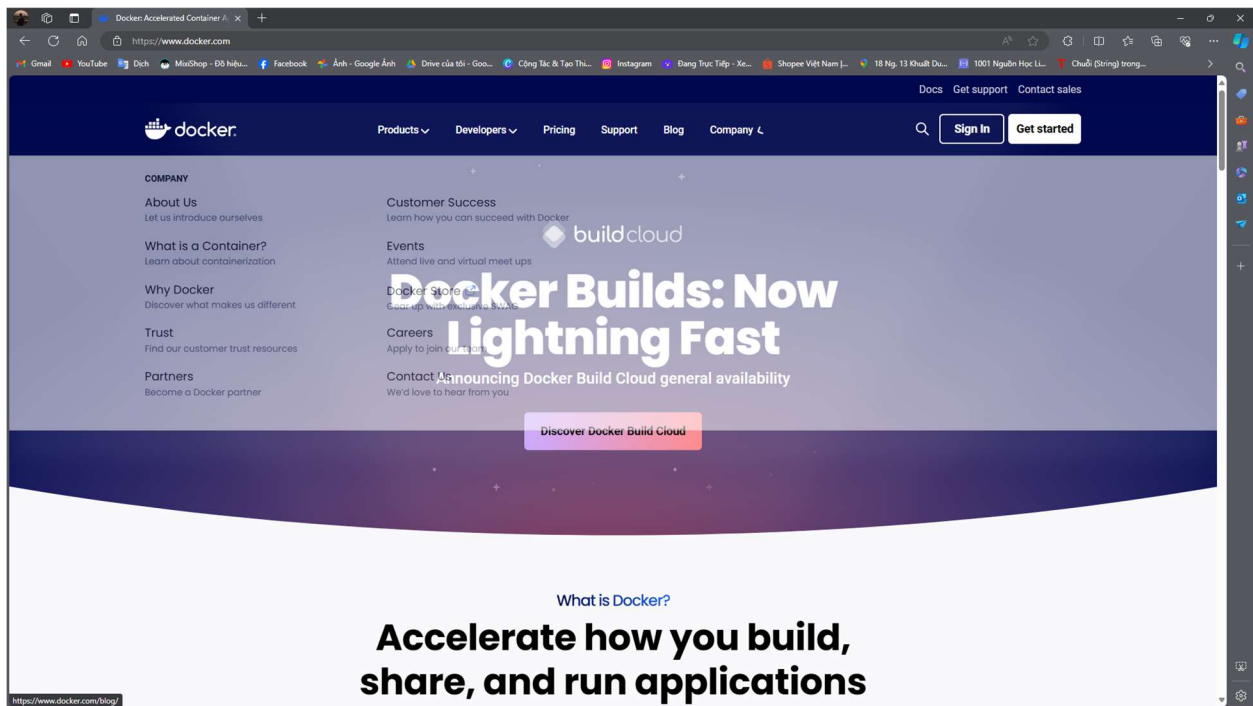


Mục tiêu: Mục tiêu của dự án này là cho phép sinh viên thiết lập và chạy SparkSQL trong vùng chứa Docker. Sinh viên sẽ cài đặt các phần phụ thuộc cần thiết trong vùng chứa để thực hiện các truy vấn SQL trên cơ sở dữ liệu được lưu trữ trong cùng một vùng chứa.

Cài đặt cơ bản cho dự án

Cài đặt Docker desktop: Docker Desktop là một công cụ mạnh mẽ để container hóa, cho phép các nhà phát triển xây dựng, chia sẻ và chạy các ứng dụng bằng cách sử dụng container. Dưới đây là một số điểm chính về Docker Desktop:



Docker Desktop là gì?

Docker Desktop là một ứng dụng máy tính để bàn gốc được thiết kế cho cả người dùng Windows và macOS.

Nó cung cấp một cách dễ dàng để chạy, xây dựng, gỡ lỗi và kiểm tra các ứng dụng trong vùng chứa Docker.

Các tính năng và lợi ích:

Thiết lập nhanh: Trình cài đặt đơn của Docker Desktop sẽ thiết lập mọi thứ bạn cần để bắt đầu làm việc với các vùng chứa trong vài giây.

Bảo trì: Được bảo trì thường xuyên với các bản sửa lỗi và cập nhật bảo mật.

Bảo mật: Bảo mật ngay từ lần tải xuống đầu tiên, với tính năng giám sát và quản lý bản vá nhất quán.

Khả năng mở rộng: Được thiết kế để cùng bạn phát triển, cho dù bạn là một công ty khởi nghiệp nhỏ hay một tổ chức lớn.

Sẵn sàng cho doanh nghiệp: Được 70% công ty Fortune 100 tin tưởng để phát triển container.

Cấp độ đăng ký:

Ưu điểm: Lý tưởng cho các nhà phát triển cá nhân muốn có các công cụ năng suất.

Nhóm: Thích hợp cho các nhóm nhỏ hơn yêu cầu các tính năng cộng tác và năng suất.

Kinh doanh: Cung cấp quản lý tập trung và khả năng bảo mật nâng cao.

Tiện ích mở rộng:

Docker Desktop Extensions mở rộng khả năng của nó, tích hợp liền mạch các công cụ quan trọng.

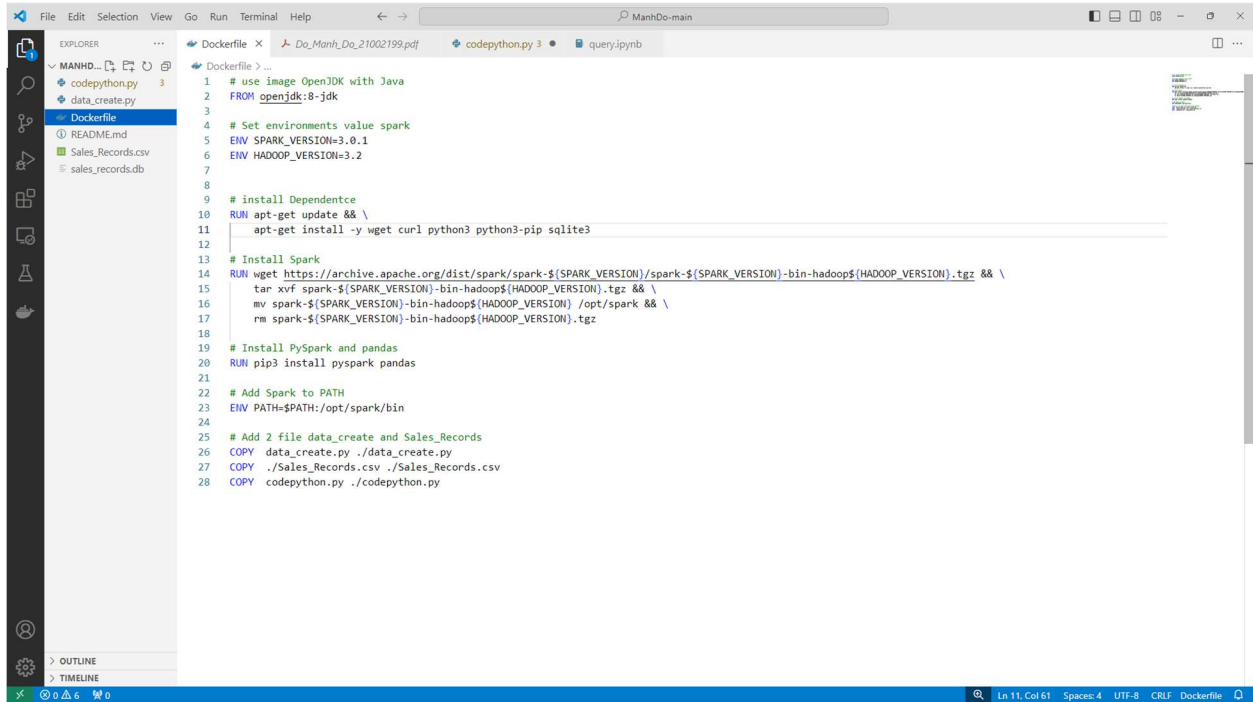
Ta có thể tải về trên trang web

<https://www.docker.com/products/docker-desktop/> tùy theo hệ điều hành máy sử dụng.

Docker setup

Bước 1: Ta bắt đầu với việc tạo một Dockerfile:

Ta sẽ sử dụng những câu lệnh sau cho “Dockerfile” của mình:



```
1 # use image OpenJDK with Java
2 FROM openjdk:8-jdk
3
4 # Set environments value spark
5 ENV SPARK_VERSION=3.0.1
6 ENV HADOOP_VERSION=3.2
7
8
9 # install Dependence
10 RUN apt-get update && \
11     apt-get install -y wget curl python3 python3-pip sqlite3
12
13 # Install Spark
14 RUN wget https://archive.apache.org/dist/spark/spark-${SPARK_VERSION}/spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
15     tar xvf spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
16     mv spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION} /opt/spark && \
17     rm spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz
18
19 # Install PySpark and pandas
20 RUN pip3 install pyspark pandas
21
22 # Add Spark to PATH
23 ENV PATH=$PATH:/opt/spark/bin
24
25 # Add 2 file data_create and Sales_Records
26 COPY data_create.py ./data_create.py
27 COPY ./Sales_Records.csv ./Sales_Records.csv
28 COPY codepython.py ./codepython.py
```

Với dự án này, ta sẽ sử dụng một Docker image từ Docker hub là OpenJDK với Java, sau đó, tạo các biến môi trường cho spark và cài đặt spark, pyspark, pandas và bên cạnh đó còn cài đặt thêm python3, sqlite3 để phục vụ cho những công việc của dự án. Cuối cùng là ta sẽ sao chép các file “data_create.py” để tạo một cơ sở dữ liệu, “Sales_Records.csv” là file database, “codepython.py” được sử dụng để chạy các câu lệnh cần thiết cho dự án sau khi chạy xong Image (điều này giúp ta không cần chạy từng dòng lệnh trên terminal của Image)

Bước 2: Ta sẽ xây dựng Docker Image

Ta xây dựng và đặt tên cho Docker Image là “exam” bằng câu lệnh sau:

docker build -t exam .

Tiếp đó, ta sẽ chạy Image đó bằng câu lệnh:

docker run -it exam

Sau khi chạy xong 2 câu lệnh thì Docker Desktop sẽ hiển thị image và container như sau:

Container:

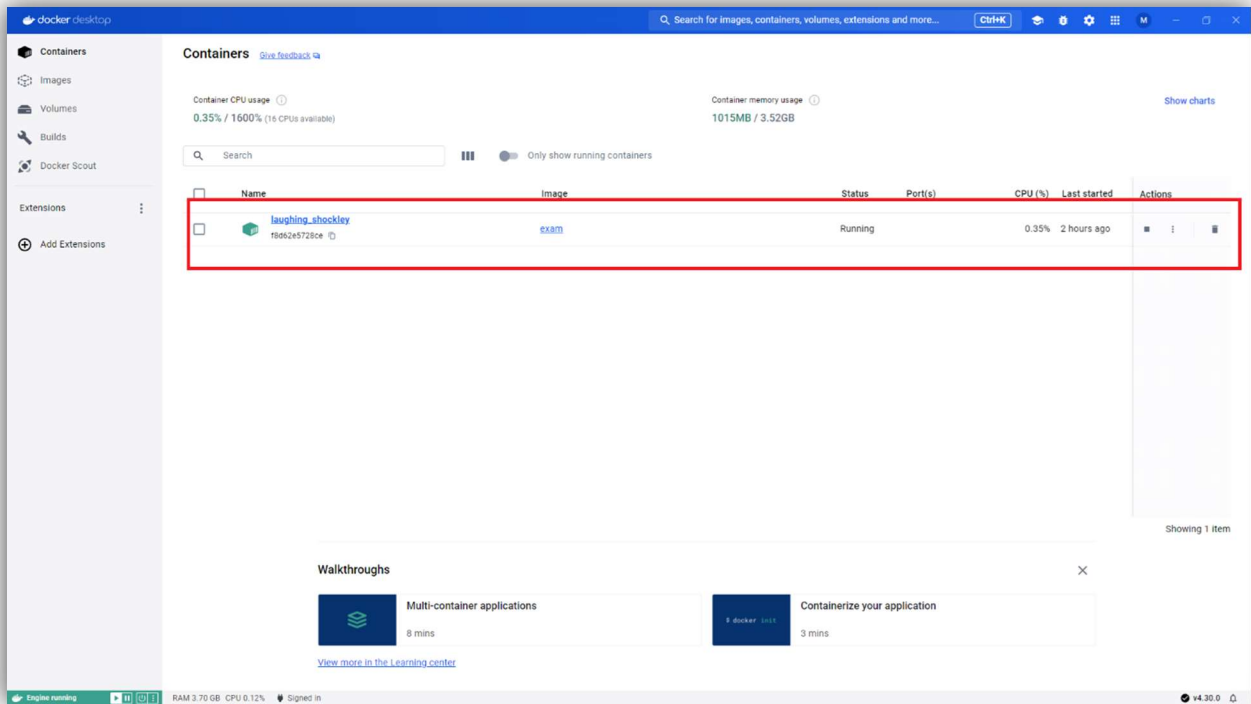
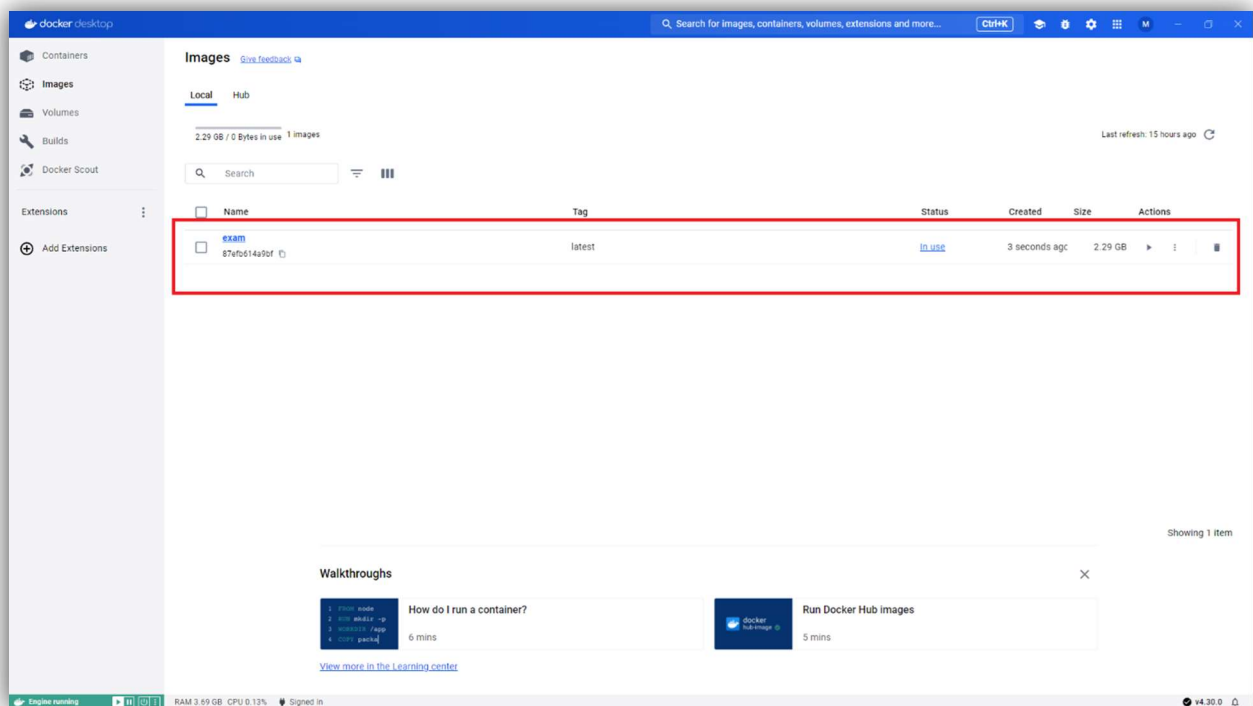


Image:



Spark Installation

Như đã trình bày ở phần tạo “Dockerfile”

```
# Set environments value spark
ENV SPARK_VERSION=3.0.1
ENV HADOOP_VERSION=3.2

# Install Spark
RUN wget https://archive.apache.org/dist/spark/spark-${SPARK_VERSION}/spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
    tar xvf spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
    mv spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION} /opt/spark && \
    rm spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz
```

Với những câu lệnh wget để truy cập vào trang web và cài đặt spark

Database Setup

Ở phần này ta sẽ thu thập 1 tập dữ liệu với 1 triệu giá trị ở trên mạng, và sử dụng một file python data_create.py để tạo ra một cơ sở dữ liệu SQLite bằng sqlite3.

Cùng với tập dataset “Sales_Records.csv” ta đã tạo ra được một cơ sở dữ liệu với

```
data_create.py > ...
1  import sqlite3
2  import random
3  import string
4  import pandas as pd
5
6  data = pd.read_csv('Sales_Records.csv')
7  # Connect to SQLite database (creates it if it doesn't exist)
8  conn = sqlite3.connect('sales_records.db')
9  cursor = conn.cursor()
10
11 # Create a sample table
12 cursor.execute('''
13 CREATE TABLE IF NOT EXISTS sales (
14     Country VARCHAR(255) NOT NULL,
15     Price FLOAT NOT NULL,
16     Cost FLOAT NOT NULL,
17     Revenue FLOAT NOT NULL,
18     TotalCost FLOAT NOT NULL
19 );
20 ''')
21
22 # Insert data from CSV into the database
23 data.to_sql('sales', conn, if_exists='append', index=False)
24
25 # Commit and close the connection
26 conn.commit()
27 conn.close()
28
```

các cột “Country, Price, Cost, Revenue, TotalCost” cùng với 1 triệu hàng giá trị.

Đầu tiên để thực thi code, ta sử dụng pandas để đọc dữ liệu từ file “Sales_Records.csv” mà ta đã chuẩn bị. Sau đó sử dụng sqlite3 để tạo ra một database có tên là “sales_records.db” gồm có 5 cột “Country, Price, Cost, Revenue, TotalCost” bao gồm có 1000001 dòng.

Dependencies

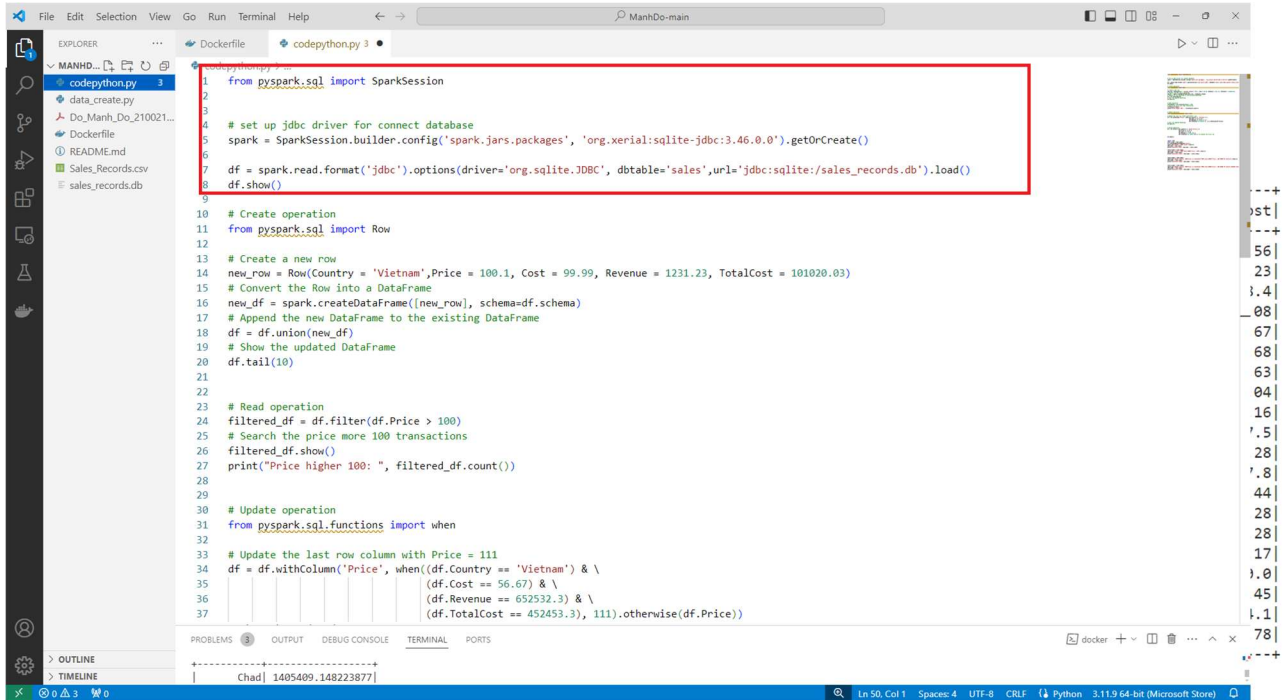
Cũng như đã đề cập ở phần tạo “Dockerfile”

Ta đã khai báo các phụ thuộc cần thiết cho Spark

```
# install Dependence
RUN apt-get update && \
    apt-get install -y wget curl python3 python3-pip sqlite3
```

Configuration

Bước đầu tiên của phần này, là ta sẽ kết nối với database:



```
1 from pyspark.sql import SparkSession
2
3
4 # set up jdbc driver for connect database
5 spark = SparkSession.builder.config('spark.jars.packages', 'org.xerial:sqlite-jdbc:3.46.0.0').getOrCreate()
6
7 df = spark.read.format('jdbc').options(driver='org.sqlite.JDBC', dbtable='sales', url='jdbc:sqlite:/sales_records.db').load()
8 df.show()
9
10 # Create operation
11 from pyspark.sql import Row
12
13 # Create a new row
14 new_row = Row(Country = 'Vietnam', Price = 100.1, Cost = 99.99, Revenue = 1231.23, TotalCost = 101020.03)
15 # Convert the Row into a DataFrame
16 new_df = spark.createDataFrame([new_row], schema=df.schema)
17 # Append the new DataFrame to the existing DataFrame
18 df = df.union(new_df)
19 # Show the updated DataFrame
20 df.tail(10)
21
22
23 # Read operation
24 filtered_df = df.filter(df.Price > 100)
25 # Search the price more 100 transactions
26 filtered_df.show()
27 print("Price higher 100: ", filtered_df.count())
28
29
30 # Update operation
31 from pyspark.sql.functions import when
32
33 # Update the last row column with Price = 111
34 df = df.withColumn("Price", when((df.Country == 'Vietnam') & \
35                                (df.Cost == 56.67) & \
36                                (df.Revenue == 652532.3) & \
37                                (df.TotalCost == 452453.3), 111).otherwise(df.Price))
```

Để kết nối với database SQLite, dự án sử dụng một chuẩn API để tương tác với cơ sở dữ liệu có tên là JDBC driver. Cụ thể dự án sử dụng sqlite jdbc driver 3.46.0.0

Running Queries

Thực hiện các thao tác CRUD

Để thực hiện các thao tác CRUD hiệu quả, dự án sử dụng DataFrame của pyspark vì DataFrame đã được xây dựng để hoạt động hiệu quả với dataset lớn hay Big Data.

*Create:

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Traceback (most recent call last):

File "/usr/local/lib/python3.9/dist-packages/pyspark/sql/types.py", line 2187, in verify

verify_value(obj)

File "/usr/local/lib/python3.9/dist-packages/pyspark/sql/types.py", line 2160, in verify_struct

verifier(v)

File "/usr/local/lib/python3.9/dist-packages/pyspark/sql/types.py", line 2187, in verify

verify_value(obj)

File "/usr/local/lib/python3.9/dist-packages/pyspark/sql/types.py", line 2181, in verify_default

verify_acceptable_types(obj)

File "/usr/local/lib/python3.9/dist-packages/pyspark/sql/types.py", line 2006, in verify_acceptable_types

raise PySparkTypeError(

pyspark.errors.exceptions.base.PySparkTypeError: [CANNOT_ACCEPT_OBJECT_IN_TYPE] `FloatType()` can not accept object `100.01` in type `str`.

```
>>> new_row = Row(Contry='VietNam', Price=100.01, Cost=99.99, Revenue=1231.23, TotalCost=101020.03)
```

```
>>> new_df = spark.createDataFrame([new_row], schema=df.schema)
```

```
>>> df = df.union(new_df)
```

```
>>> df.tail(10)
```

```
[Row(Country='Netherlands', Price=9.329999923706055, Cost=6.920000076293945, Revenue=52201.3515625, TotalCost=38717.3984375), Row(Country='Iran', Price=81.7300033569336, Cost=56.66999816894531, Revenue=347434.21875, TotalCost=240904.171875), Row(Country='Armenia', Price=81.7300033569336, Cost=56.66999816894531, Revenue=610359.625, TotalCost=423211.5625), Row(Country='Germany', Price=651.2100219726562, Cost=524.9600219726562, Revenue=5658363.5, TotalCost=4561377.5), Row(Country='Senegal', Price=255.27999877929688, Cost=159.4199981689453, Revenue=864633.375, TotalCost=539955.5625), Row(Country='Panama', Price=651.2100219726562, Cost=524.9600219726562, Revenue=2649122.25, TotalCost=2135537.25), Row(Country='Norway', Price=651.2100219726562, Cost=524.9600219726562, Revenue=3429271.75, TotalCost=2764439.25), Row(Country='Montenegro', Price=47.45000076293945, Cost=31.790000915527344, Revenue=405744.9375, TotalCost=271836.28125), Row(Country='Nicaragua', Price=421.8900146484375, Cost=364.69000244140625, Revenue=3172191.0, TotalCost=2742104.0), Row(Country='VietNam', Price=100.01000213623047, Cost=99.98999786376953, Revenue=1231.22998046875, TotalCost=101020.03125)]
```

```
>>> filtered_df = df.filter(df.Transaction_Count > 2000)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/usr/local/lib/python3.9/dist-packages/pyspark/sql/dataframe.py", line 3127, in __getattr__

raise AttributeError(

0 6 0

Ln 7, Col 1 Spaces: 4 UTF-8

*Read:

```
>>> filtered_df = df.filter(df.Price > 100)
>>> filtered_df.show()
+-----+-----+-----+-----+
|          Country| Price|   Cost|  Revenue|TotalCost|
+-----+-----+-----+-----+
|          Morocco| 109.28|  35.84| 503890.1|165258.23|
| Papua New Guinea| 421.89| 364.69| 151880.4| 131288.4|
|          Djibouti| 109.28|  35.84| 61415.36| 20142.08|
|           Ghana| 651.21| 524.96| 583484.2|470364.16|
|          Tanzania| 437.2| 263.33|3396169.5|2045547.5|
|          Algeria| 437.2| 263.33|4227287.0|2546137.8|
|          Singapore| 152.58|  97.44|1171204.1|747949.44|
| Papua New Guinea| 109.28|  35.84|993573.75|325857.28|
|          Zimbabwe| 651.21| 524.96|6266594.0|5051690.0|
|          Ethiopia| 437.2| 263.33| 289426.4|174324.45|
|           France| 437.2| 263.33|2517397.5|1516254.1|
|           Haiti| 651.21| 524.96|1336282.9|1077217.9|
|          Nicaragua| 668.27| 502.54|5206491.5|3915289.2|
|      Turkmenistan| 154.06|  90.93|1027580.2| 606503.1|
|      United Kingdom| 437.2| 263.33| 453813.6|273336.53|
| Dominican Republic| 255.28| 159.42| 69946.72| 43681.08|
|           China| 651.21| 524.96|3771157.0|3040043.2|
|           Uganda| 437.2| 263.33|2636753.2|1588143.2|
|           Kuwait| 668.27| 502.54| 979683.8| 736723.6|
| United Arab Emirates| 651.21| 524.96|6253569.5|5041191.0|
+-----+-----+-----+-----+
only showing top 20 rows
```

*Update:

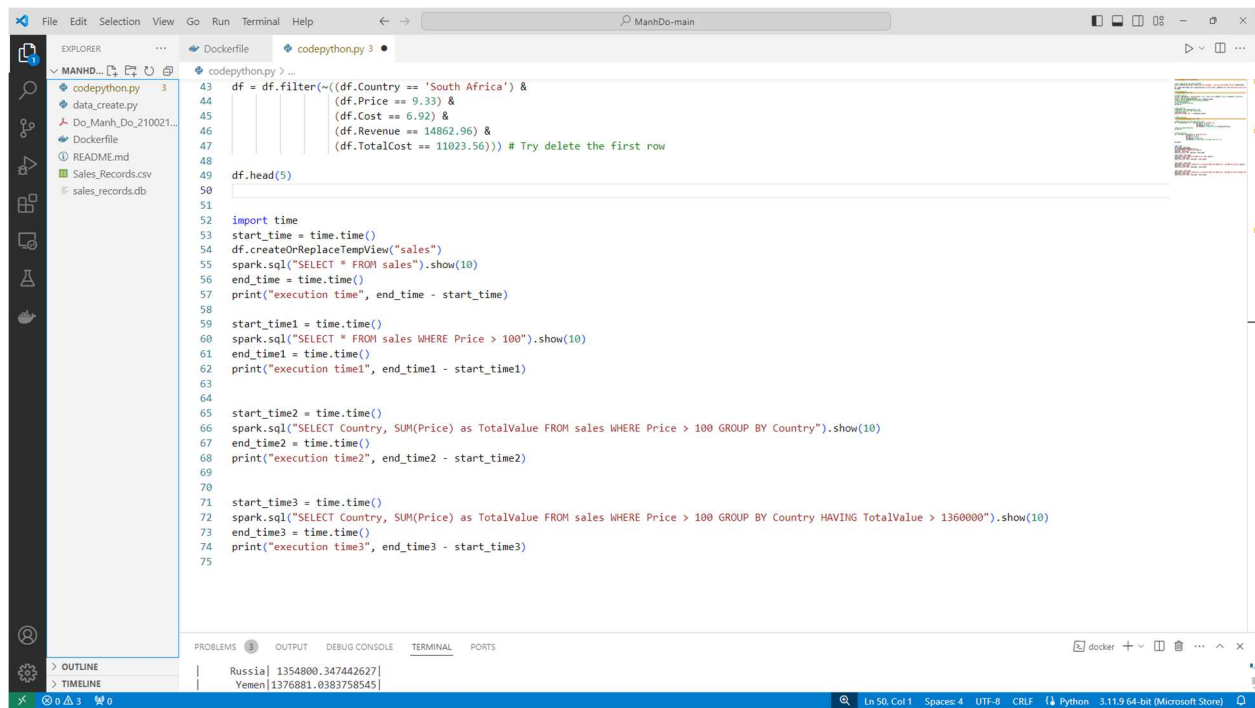
```
>>> df = df.withColumn('Price', when((df.Country == 'Vietnam') & \
...                               (df.Cost == 56.67) & \
...                               (df.Revenue == 652532.3) & \
...                               (df.TotalCost == 452453.3), 111).otherwise(df.Price))
```

*Delete:

```
>>> df = df.filter(~((df.Country == 'Morocco') &
...                  (df.Price == 109.28) &
...                  (df.Cost == 35.84) &
...                  (df.Revenue == 503890.1) &
...                  (df.TotalCost == 11023.56)))
```

So sánh tốc độ thực thi câu lệnh truy vấn:

Ở phần này ta sẽ sử dụng thư viện time trong python để tính thời gian thực thi câu lệnh truy vấn.



```
codepython.py
43 df = df.filter(~((df.Country == 'South Africa') &
44 (df.Price == 9.33) &
45 (df.Cost == 6.92) &
46 (df.Revenue == 14862.96) &
47 (df.TotalCost == 11023.56))) # Try delete the first row
48
49 df.head(5)
50
51
52 import time
53 start_time = time.time()
54 df.createOrReplaceTempView("sales")
55 spark.sql("SELECT * FROM sales").show(10)
56 end_time = time.time()
57 print("execution time", end_time - start_time)
58
59 start_time1 = time.time()
60 spark.sql("SELECT * FROM sales WHERE Price > 100").show(10)
61 end_time1 = time.time()
62 print("execution time1", end_time1 - start_time1)
63
64
65 start_time2 = time.time()
66 spark.sql("SELECT Country, SUM(Price) as TotalValue FROM sales WHERE Price > 100 GROUP BY Country").show(10)
67 end_time2 = time.time()
68 print("execution time2", end_time2 - start_time2)
69
70
71 start_time3 = time.time()
72 spark.sql("SELECT Country, SUM(Price) as TotalValue FROM sales WHERE Price > 100 GROUP BY Country HAVING TotalValue > 1360000").show(10)
73 end_time3 = time.time()
74 print("execution time3", end_time3 - start_time3)
75
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

OUTLINE
TIMELINE

Russia	1354800.347442627
Yemen	1376881.0383758545

Ln 50, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store)

The screenshot shows a VS Code terminal window with the following content:

```
Price higher 100: 749585
+-----+-----+-----+-----+
| Country | Price | Cost | Revenue | TotalCost |
+-----+-----+-----+-----+
| South Africa | 9.33 | 6.92 | 14862.69 | 11023.56 |
| Morocco | 109.28 | 35.84 | 503890.1 | 165258.23 |
| Papua New Guinea | 421.89 | 364.69 | 151880.4 | 131288.4 |
| Djibouti | 109.28 | 35.84 | 61415.36 | 20142.08 |
| Slovakia | 47.45 | 31.79 | 188518.84 | 126301.67 |
| Sri Lanka | 9.33 | 6.92 | 12866.07 | 9542.68 |
| Seychelles | 47.45 | 31.79 | 28327.65 | 18978.63 |
| Tanzania | 47.45 | 31.79 | 70036.2 | 46922.04 |
| Ghana | 651.21 | 524.96 | 583484.2 | 470364.16 |
| Tanzania | 437.2 | 263.33 | 3396169.5 | 2045547.5 |
+-----+-----+-----+-----+
only showing top 10 rows

execution time 0.4169647693634033
+-----+-----+-----+-----+
| Country | Price | Cost | Revenue | TotalCost |
+-----+-----+-----+-----+
| Morocco | 109.28 | 35.84 | 503890.1 | 165258.23 |
| Papua New Guinea | 421.89 | 364.69 | 151880.4 | 131288.4 |
| Djibouti | 109.28 | 35.84 | 61415.36 | 20142.08 |
| Ghana | 651.21 | 524.96 | 583484.2 | 470364.16 |
| Tanzania | 437.2 | 263.33 | 3396169.5 | 2045547.5 |
| Algeria | 437.2 | 263.33 | 4227287.0 | 2546137.8 |
| Singapore | 152.58 | 97.44 | 1171204.1 | 747949.44 |
| Papua New Guinea | 109.28 | 35.84 | 993573.75 | 325857.28 |
| Zimbabwe | 651.21 | 524.96 | 626594.0 | 5051690.0 |
| Ethiopia | 437.2 | 263.33 | 289426.4 | 174324.45 |
+-----+-----+-----+-----+
only showing top 10 rows

execution time1 0.3521144390106201
+-----+-----+
| Country | TotalValue |
+-----+-----+
| Chad | 1405409.148223877 |
| Russia | 1354800.347442627 |
| Yemen | 1376881.0383758545 |
| Senegal | 1370885.9773712158 |
| Sweden | 1364464.3274383545 |
| Kiribati | 1363623.497654053 |
| Philippines | 1368445.4976501465 |
+-----+-----+
only showing top 10 rows

execution time2 2.863360643386841
+-----+-----+
| Country | TotalValue |
+-----+-----+
| Chad | 1405409.148223877 |
| Yemen | 1376881.0383758545 |
| Senegal | 1370885.9773712158 |
| Sweden | 1364464.3274383545 |
| Kiribati | 1363623.497654053 |
| Philippines | 1368445.4976501465 |
| Eritrea | 1365752.9179382324 |
| Djibouti | 1397834.4687194824 |
| Singapore | 1424357.4690093994 |
| Malaysia | 1380113.7681274414 |
+-----+-----+
only showing top 10 rows

execution time3 2.1196095943450928
root@ac6bb1868dc6:/#
```

Trong terminal kết quả của những câu lệnh truy vấn ta dễ thấy:

- Thời gian khi không có Where: execution time = 0.4169647693634033(s).
- Thời gian khi có Where: execution time1 = 0.3521144390106201(s).
Nhanh hơn một chút so với khi không có Where
- Thời gian khi có Group By: execution time2 = 2.863360643386841(s).
Dễ thấy thời gian đã lâu hơn bởi vì cần phải tính tổng

- Thời gian khi có HAVING: execution time3 2.1196095943450928(s).
Nhanh hơn so với Group By