

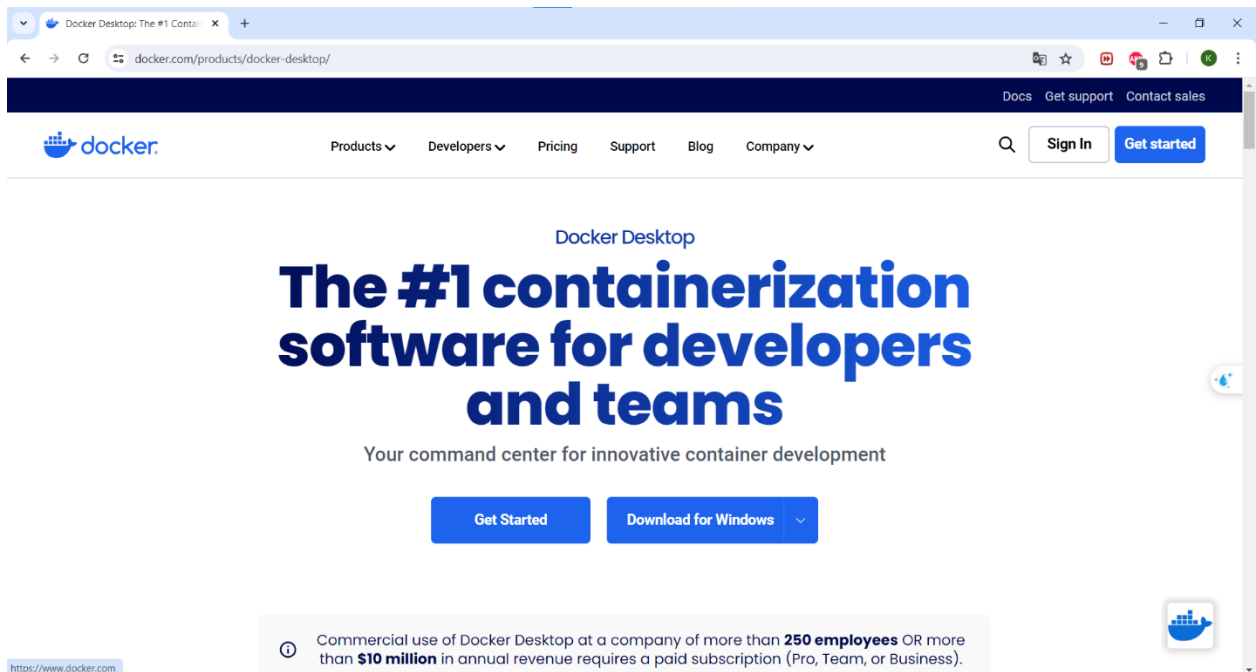
SparkSQL in Docker Project

Mục tiêu

Mục tiêu của project này là cài đặt và chạy SparkSQL trong một Docker container. Container này sẽ lưu trữ một cơ sở dữ liệu (cụ thể là SQLite database) và cho phép thực hiện các truy vấn SQL sử dụng SparkSQL.

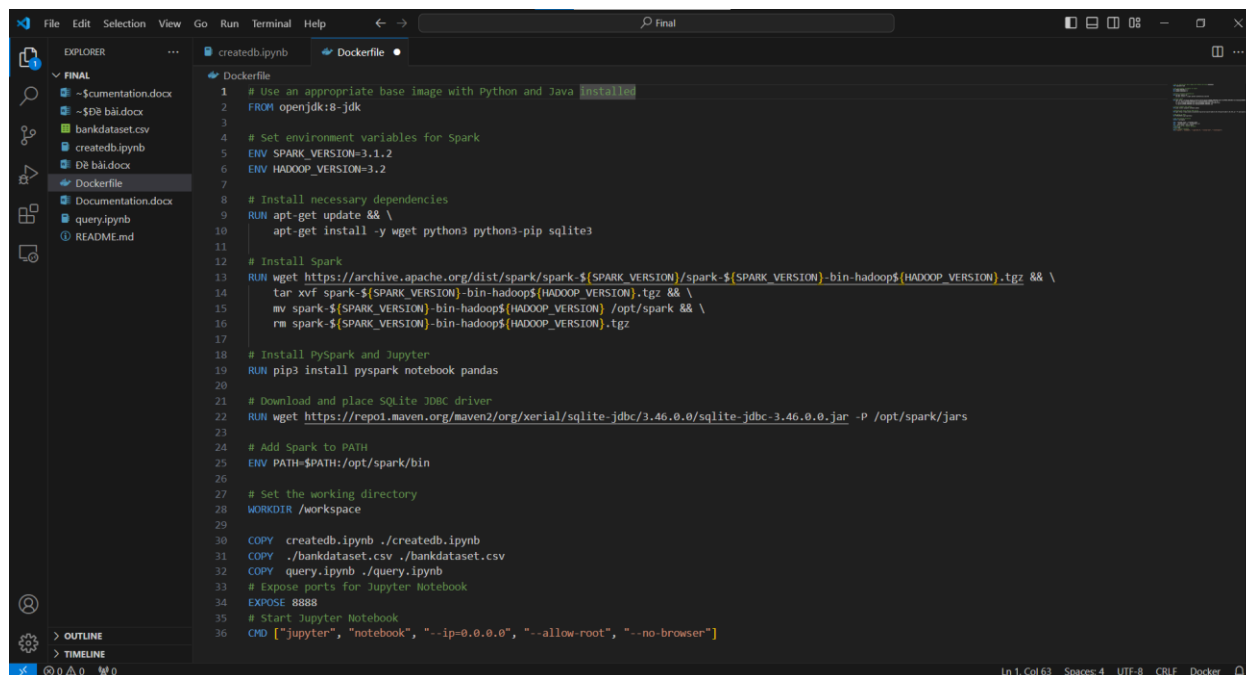
Điều kiện tiên quyết

Trước khi bắt đầu project, hãy đảm bảo rằng máy tính đủ bộ nhớ (tối thiểu 30GB) và đã cài đặt Docker. Để Docker đơn giản, dự án sử dụng Docker Desktop với một giao diện dễ nhìn và dễ thực hiện. Ta có thể tải về trên trang web <https://www.docker.com/products/docker-desktop/> tùy theo hệ điều hành máy sử dụng.



Cài đặt

Bước 1: Tạo một Dockerfile với nội dung như sau:



```
1 # Use an appropriate base image with Python and Java installed
2 FROM openjdk:8-jdk
3
4 # Set environment variables for spark
5 ENV SPARK_VERSION=3.1.2
6 ENV HADOOP_VERSION=3.2
7
8 # Install necessary dependencies
9 RUN apt-get update && \
10     apt-get install -y wget python3 python3-pip sqlite3
11
12 # Install Spark
13 RUN wget https://archive.apache.org/dist/spark/spark-${SPARK_VERSION}/spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
14     tar xvf spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz && \
15     mv spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION} /opt/spark && \
16     rm spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz
17
18 # Install Pyspark and Jupyter
19 RUN pip3 install pyspark notebook pandas
20
21 # Download and place SQLite JDBC driver
22 RUN wget https://repo1.maven.org/maven2/org/xerial/sqlite-jdbc/3.46.0.0/sqlite-jdbc-3.46.0.0.jar -P /opt/spark/jars
23
24 # Add Spark to PATH
25 ENV PATH=$PATH:/opt/spark/bin
26
27 # Set the working directory
28 WORKDIR /workspace
29
30 COPY createdb.ipynb ./createdb.ipynb
31 COPY ./bankdataset.csv ./bankdataset.csv
32 COPY query.ipynb ./query.ipynb
33
34 # Expose ports for Jupyter Notebook
35 EXPOSE 8888
36
37 # Start Jupyter Notebook
38 CMD ["jupyter", "notebook", "--ip=0.0.0.0", "--allow-root", "--no-browser"]
```

Ở đây, dự án sử dụng một Docker image base từ [Docker hub](#) là OpenJDK, một image base phổ biến cho ứng dụng Java, sau đó cài đặt Python, Spark và Hadoop vào trong đó. Mục đích chung là chuẩn bị cho Pyspark, do đó ta có thể thay đổi thứ tự cài đặt ví dụ như sử dụng image base cho Python sau đó cài đặt Java, Spark, Hadoop. Bên cạnh đó dự án cài đặt cơ sở dữ liệu SQLite và công cụ Pip để tải các gói thư viện trong Python.

Tiếp theo, dự án cài đặt các Jupiter Notebook để thực hiện các câu lệnh dễ nhìn hơn. Cuối cùng đặt working directory, copy các file cần thiết và đặt cổng ra cho Jupyter Notebook là 8888. Như vậy ta đã chuẩn bị các cài đặt cơ bản cho một Container.

Bước 2: Xây dựng Docker Image

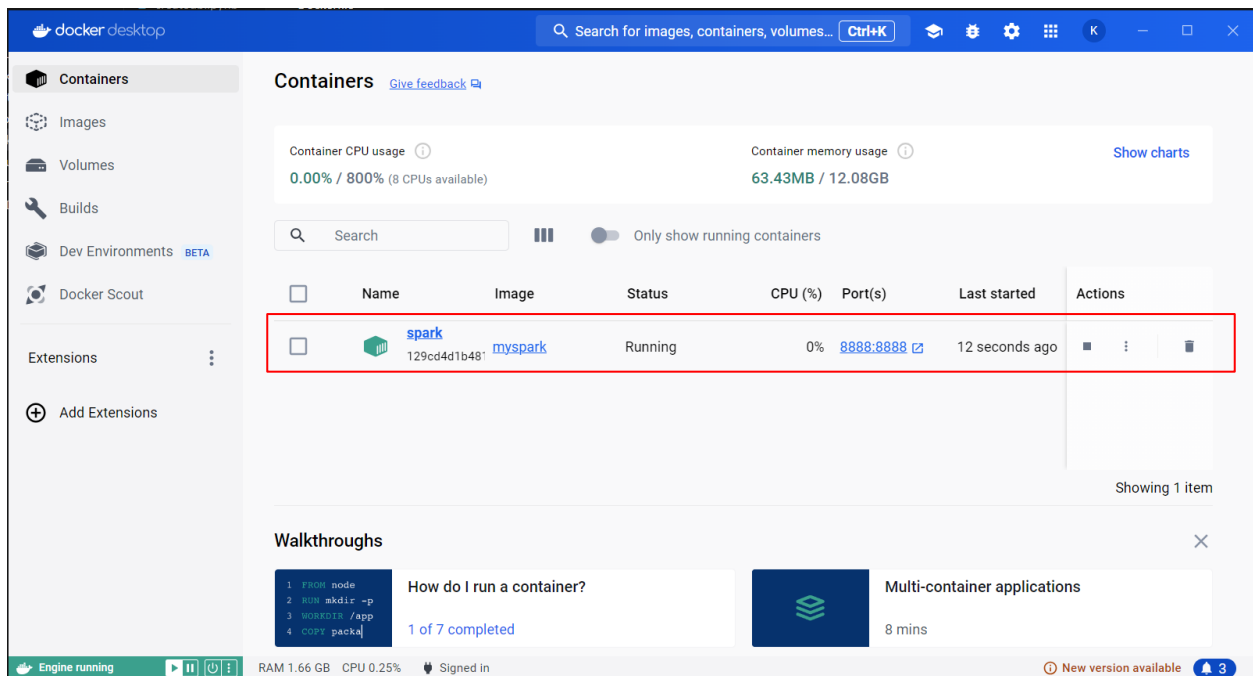
Để xây dựng Docker image ta sử dụng câu lệnh sau và đặt tên là “myspark”

`docker build . -t myspark`

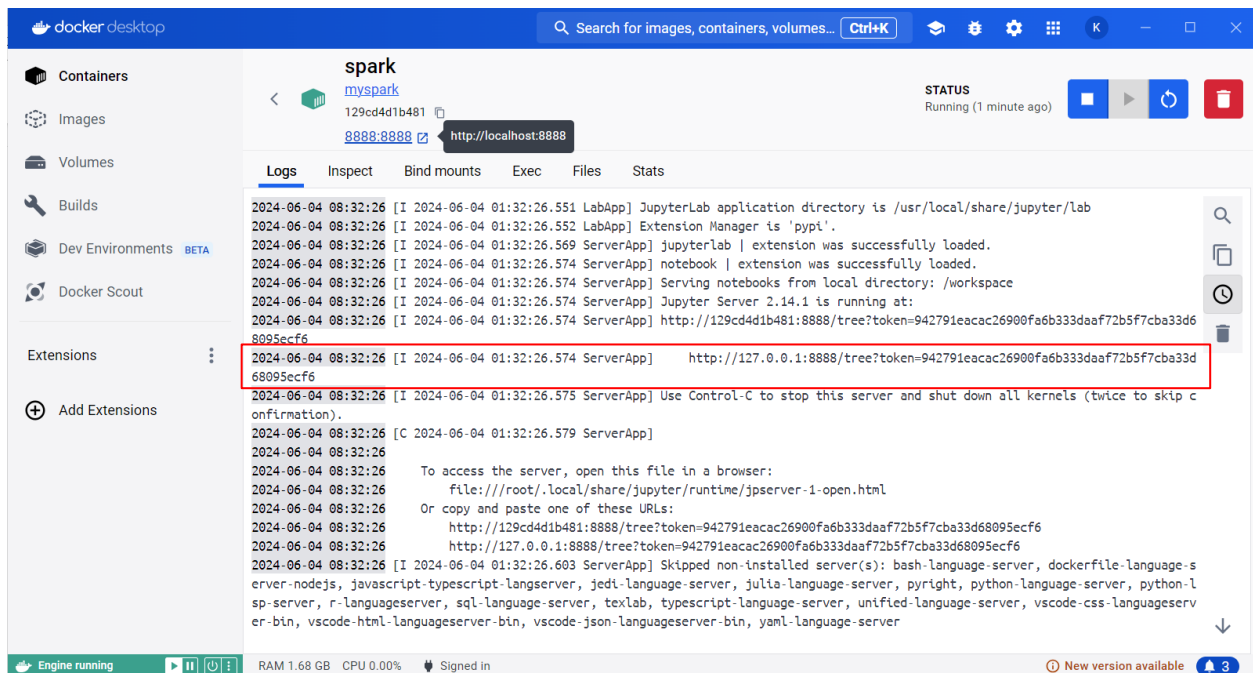
Sau đó, ta chạy Image đó với cổng 8888 và tên là “spark”.

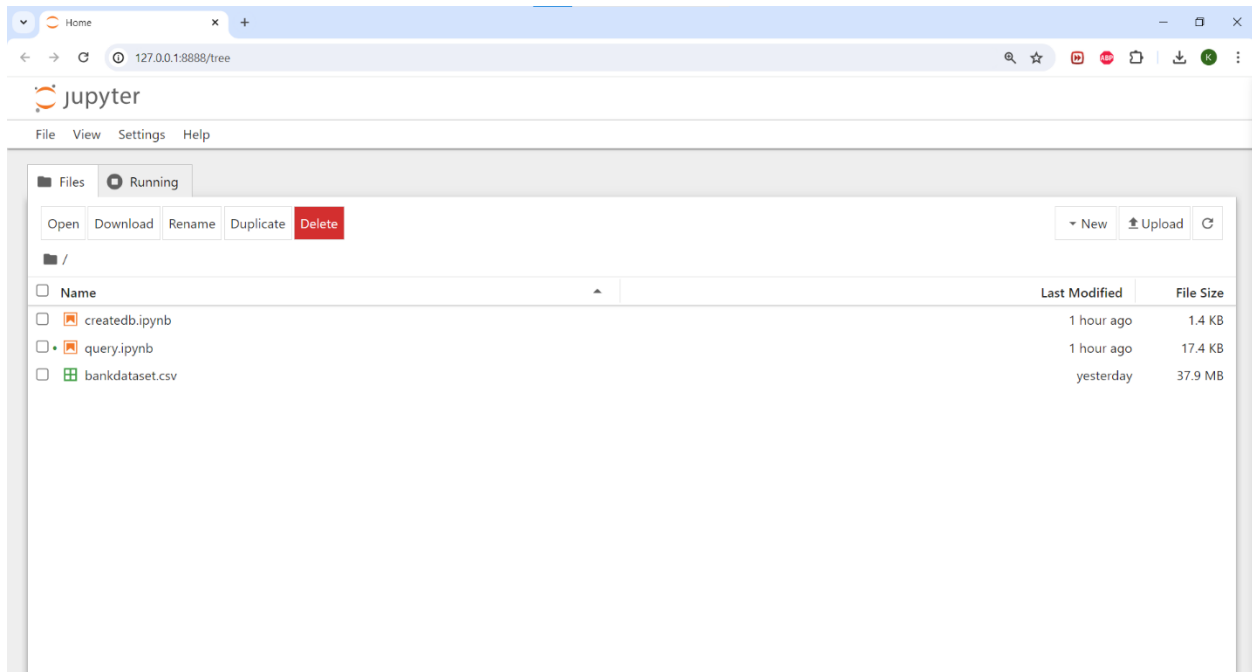
`docker run -p 8888:8888 --name spark -d myspark`

Khi đó trong Docker Desktop sẽ xuất hiện một Image như sau:



Chọn vào Image đó, ta sẽ tìm được một đường link truy cập vào Jupyter Notebook.



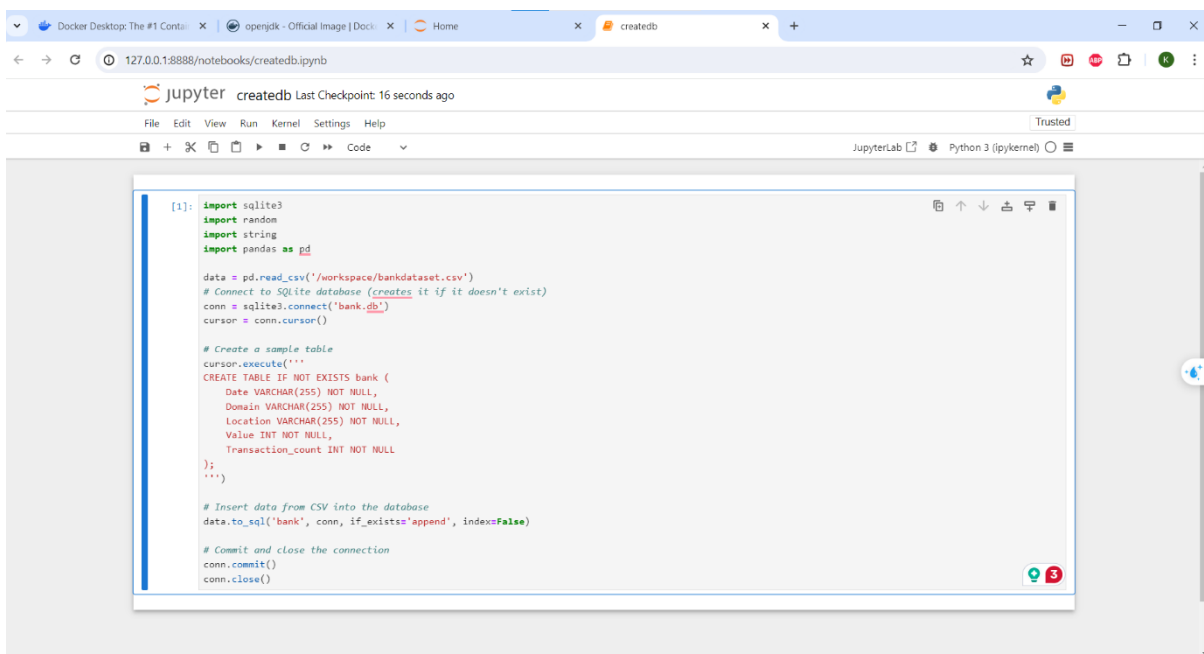


Như vậy bước cài đặt đã thành công và trong các bước sau ta sẽ thực hiện một số các câu lệnh truy vấn trong Pyspark.

Thực thi

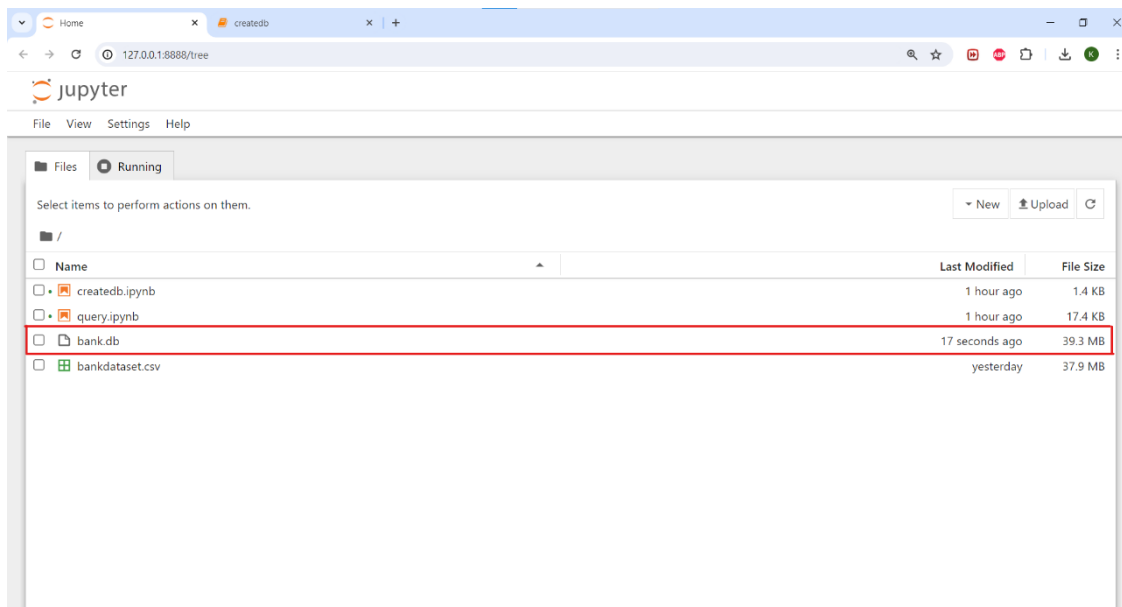
Bước 1: Tạo cơ sở dữ liệu

Ta sẽ tạo một cơ sở dữ liệu ở đây. Dự án đã chuẩn bị một file createdb.ipynb giúp ta tạo ra một cơ sở dữ liệu SQLite bằng sqlite3.



Đầu tiên ta tạo một database có tên là bank. Sau đó tạo một Table cũng tên là bank bao gồm có 5 cột: Date, Domain, Location, Value và Transaction_count. Ta insert vào bảng một file bankdataset.csv bao gồm 1004480 dòng. Đây là một bộ dataset thống kê tổng số giao dịch và tổng số tiền của các giao dịch tại các thành phố ở Ấn Độ, với mỗi dịch vụ khác nhau (như nhà hàng, đầu tư, bán lẻ, ...) mỗi ngày trong năm 2022.

Kiểm tra chắc chắn xem file “bank.db” đã được tạo ra chưa. Sau đó qua bước tiếp theo.



Bước 2: Kết nối với database

```
[ ]: from pyspark.sql import SparkSession

spark = SparkSession.builder.config('spark.jars.packages', 'org.xerial:sqlite-jdbc:3.46.0.0').getOrCreate()

[2]: df = spark.read.format('jdbc').options(driver='org.sqlite.JDBC', dbtable='bank', url='jdbc:sqlite:/workspace/bank.db').load()

[3]: df.show()
```

Date	Domain	Location	Value	Transaction_count
1/1/2022	RESTAURANT	Bhuj	365554	1932
1/1/2022	INVESTMENTS	Ludhiana	847444	1721
1/1/2022	RETAIL	Goa	786941	1573
1/1/2022	INTERNATIONAL	Mathura	368610	2049
1/1/2022	RESTAURANT	Madurai	615681	1519
1/1/2022	INTERNATIONAL	Daman	1191092	1813
1/1/2022	INTERNATIONAL	Buxar	968883	2098
1/1/2022	PUBLIC	Trichy	1030297	606
1/1/2022	RESTAURANT	Kullu	688655	1463
1/1/2022	MEDICAL	Hyderabad	1174302	1463
1/1/2022	PUBLIC	Lucknow	912902	1035
1/1/2022	INVESTMENTS	Bikaner	436534	1093
1/1/2022	PUBLIC	Amritsar	849803	2013
1/1/2022	INVESTMENTS	Mathura	1180043	2068
1/1/2022	RETAIL	Doda	1003497	1654
1/1/2022	MEDICAL	Jaipur	616724	1686
1/1/2022	PUBLIC	Kannur	933938	1362
1/1/2022	RETAIL	Lunglei	521003	2304
1/1/2022	INVESTMENTS	Bombay	849779	1202
1/1/2022	INVESTMENTS	Kannur	770080	2431

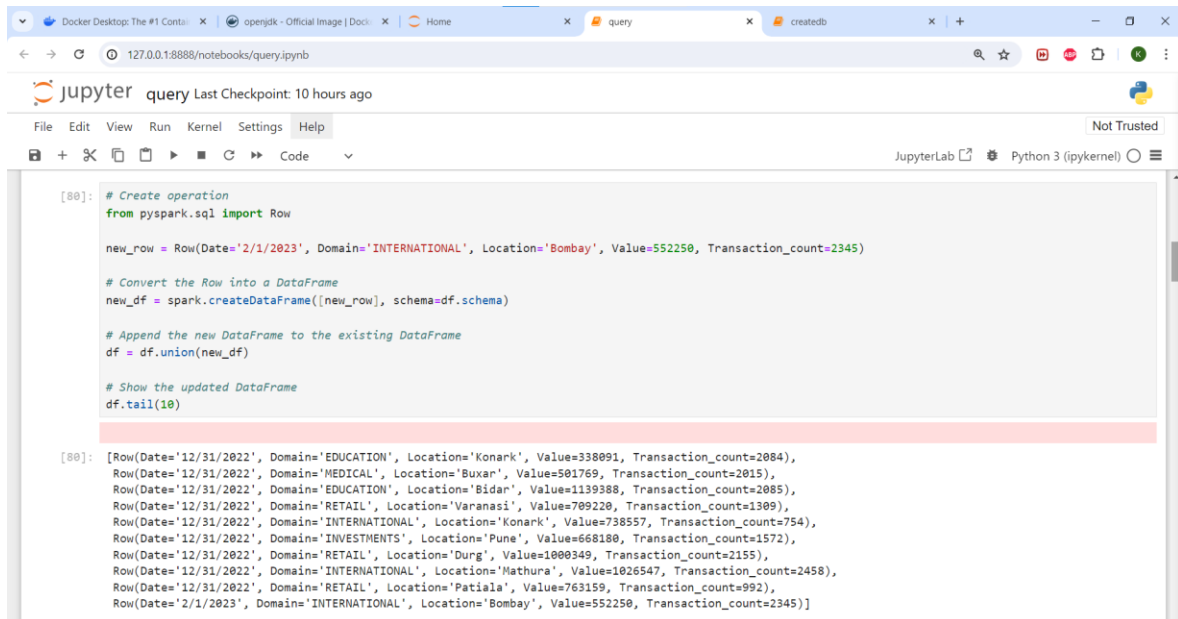
only showing top 20 rows

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.

Bước 3: 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.

Với thao tác Create:



```
[80]: # Create operation
from pyspark.sql import Row

new_row = Row(Date='2/1/2023', Domain='INTERNATIONAL', Location='Bombay', Value=552250, Transaction_count=2345)

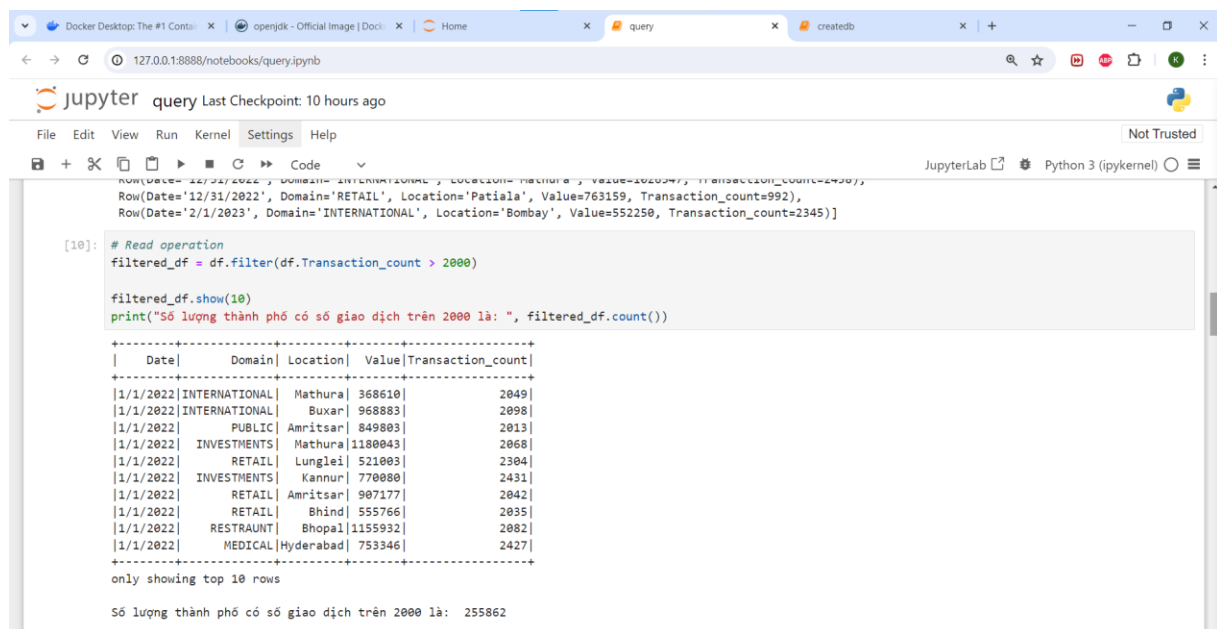
# Convert the Row into a DataFrame
new_df = spark.createDataFrame([new_row], schema=df.schema)

# Append the new DataFrame to the existing DataFrame
df = df.union(new_df)

# Show the updated DataFrame
df.tail(10)

[80]: [Row(Date='12/31/2022', Domain='EDUCATION', Location='Konark', Value=338091, Transaction_count=2084),
Row(Date='12/31/2022', Domain='MEDICAL', Location='Buxar', Value=501769, Transaction_count=2015),
Row(Date='12/31/2022', Domain='EDUCATION', Location='Bidar', Value=1139388, Transaction_count=2085),
Row(Date='12/31/2022', Domain='RETAIL', Location='Varanasi', Value=709220, Transaction_count=1309),
Row(Date='12/31/2022', Domain='INTERNATIONAL', Location='Konark', Value=738557, Transaction_count=754),
Row(Date='12/31/2022', Domain='INVESTMENTS', Location='Pune', Value=668180, Transaction_count=1572),
Row(Date='12/31/2022', Domain='RETAIL', Location='Durg', Value=1000349, Transaction_count=2155),
Row(Date='12/31/2022', Domain='INTERNATIONAL', Location='Mathura', Value=1026547, Transaction_count=2458),
Row(Date='12/31/2022', Domain='RETAIL', Location='Patiala', Value=763159, Transaction_count=992),
Row(Date='2/1/2023', Domain='INTERNATIONAL', Location='Bombay', Value=552250, Transaction_count=2345)]
```

Với thao tác Read:



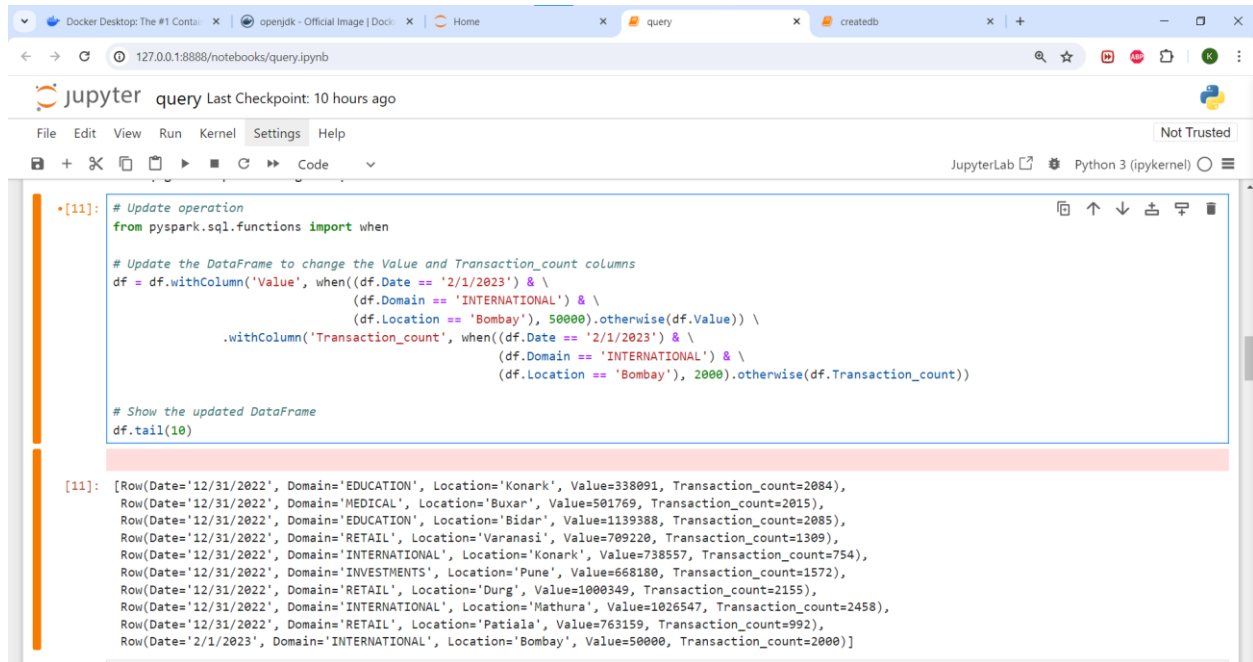
```
[10]: # Read operation
filtered_df = df.filter(df.Transaction_count > 2000)

filtered_df.show(10)
print("Số lượng thành phố có số giao dịch trên 2000 là: ", filtered_df.count())

+-----+-----+-----+-----+-----+
| Date | Domain | Location | Value | Transaction_count |
+-----+-----+-----+-----+-----+
| 1/1/2022 | INTERNATIONAL | Mathura | 368610 | 2049 |
| 1/1/2022 | INTERNATIONAL | Buxar | 968883 | 2098 |
| 1/1/2022 | PUBLIC | Amritsar | 849803 | 2013 |
| 1/1/2022 | INVESTMENTS | Mathura | 1180043 | 2068 |
| 1/1/2022 | RETAIL | Lunglei | 521003 | 2304 |
| 1/1/2022 | INVESTMENTS | Kannur | 770000 | 2431 |
| 1/1/2022 | RETAIL | Amritsar | 907177 | 2042 |
| 1/1/2022 | RETAIL | Bhind | 555766 | 2035 |
| 1/1/2022 | RESTRAUNT | Bhopal | 1155932 | 2082 |
| 1/1/2022 | MEDICAL | Hyderabad | 753346 | 2427 |
+-----+-----+-----+-----+-----+
only showing top 10 rows

Số lượng thành phố có số giao dịch trên 2000 là: 255862
```

Với thao tác Update:



The screenshot shows a JupyterLab window with a browser tab at 127.0.0.1:8888/notebooks/query.ipynb. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The main area displays a code cell with the following Python code:

```
[11]: # Update operation
from pyspark.sql.functions import when

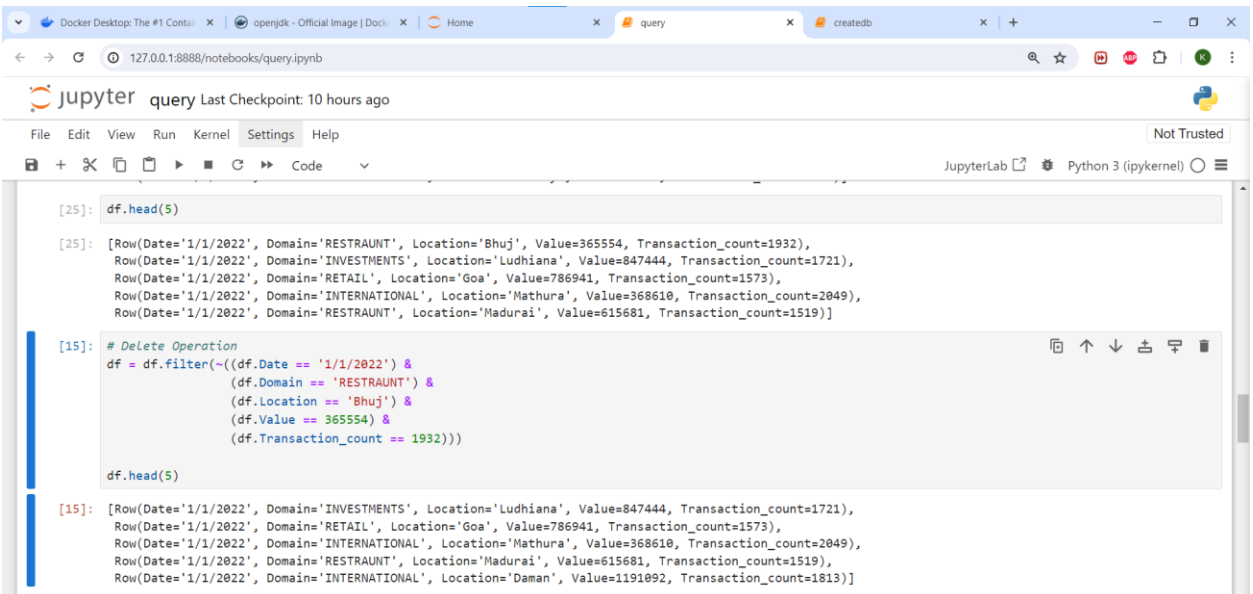
# Update the DataFrame to change the Value and Transaction_count columns
df = df.withColumn('Value', when((df.Date == '2/1/2023') & \
    (df.Domain == 'INTERNATIONAL') & \
    (df.Location == 'Bombay'), 50000).otherwise(df.Value)) \
    .withColumn('Transaction_count', when((df.Date == '2/1/2023') & \
    (df.Domain == 'INTERNATIONAL') & \
    (df.Location == 'Bombay'), 2000).otherwise(df.Transaction_count))

# Show the updated DataFrame
df.tail(10)
```

The output of the code cell shows the last 10 rows of the DataFrame:

```
[11]: [Row(Date='12/31/2022', Domain='EDUCATION', Location='Konark', Value=338091, Transaction_count=2084),
Row(Date='12/31/2022', Domain='MEDICAL', Location='Buxar', Value=501769, Transaction_count=2015),
Row(Date='12/31/2022', Domain='EDUCATION', Location='Bidar', Value=1139388, Transaction_count=2085),
Row(Date='12/31/2022', Domain='RETAIL', Location='Varanasi', Value=709220, Transaction_count=1309),
Row(Date='12/31/2022', Domain='INTERNATIONAL', Location='Konark', Value=738557, Transaction_count=754),
Row(Date='12/31/2022', Domain='INVESTMENTS', Location='Pune', Value=668180, Transaction_count=1572),
Row(Date='12/31/2022', Domain='RETAIL', Location='Durg', Value=1000349, Transaction_count=2155),
Row(Date='12/31/2022', Domain='INTERNATIONAL', Location='Mathura', Value=1026547, Transaction_count=2458),
Row(Date='12/31/2022', Domain='RETAIL', Location='Patiala', Value=763159, Transaction_count=992),
Row(Date='2/1/2023', Domain='INTERNATIONAL', Location='Bombay', Value=50000, Transaction_count=2000)]
```

Với thao tác Delete:



The screenshot shows a JupyterLab window with a browser tab at 127.0.0.1:8888/notebooks/query.ipynb. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The main area displays a code cell with the following Python code:

```
[25]: df.head(5)
```

The output of the code cell shows the first 5 rows of the DataFrame:

```
[25]: [Row(Date='1/1/2022', Domain='RESTRAUNT', Location='Bhuj', Value=365554, Transaction_count=1932),
Row(Date='1/1/2022', Domain='INVESTMENTS', Location='Ludhiana', Value=847444, Transaction_count=1721),
Row(Date='1/1/2022', Domain='RETAIL', Location='Goa', Value=786941, Transaction_count=1573),
Row(Date='1/1/2022', Domain='INTERNATIONAL', Location='Mathura', Value=368610, Transaction_count=2049),
Row(Date='1/1/2022', Domain='RESTRAUNT', Location='Madurai', Value=615681, Transaction_count=1519)]
```

The code cell then shows a delete operation:

```
[15]: # Delete Operation
df = df.filter(~((df.Date == '1/1/2022') &
    (df.Domain == 'RESTRAUNT') &
    (df.Location == 'Bhuj') &
    (df.Value == 365554) &
    (df.Transaction_count == 1932)))

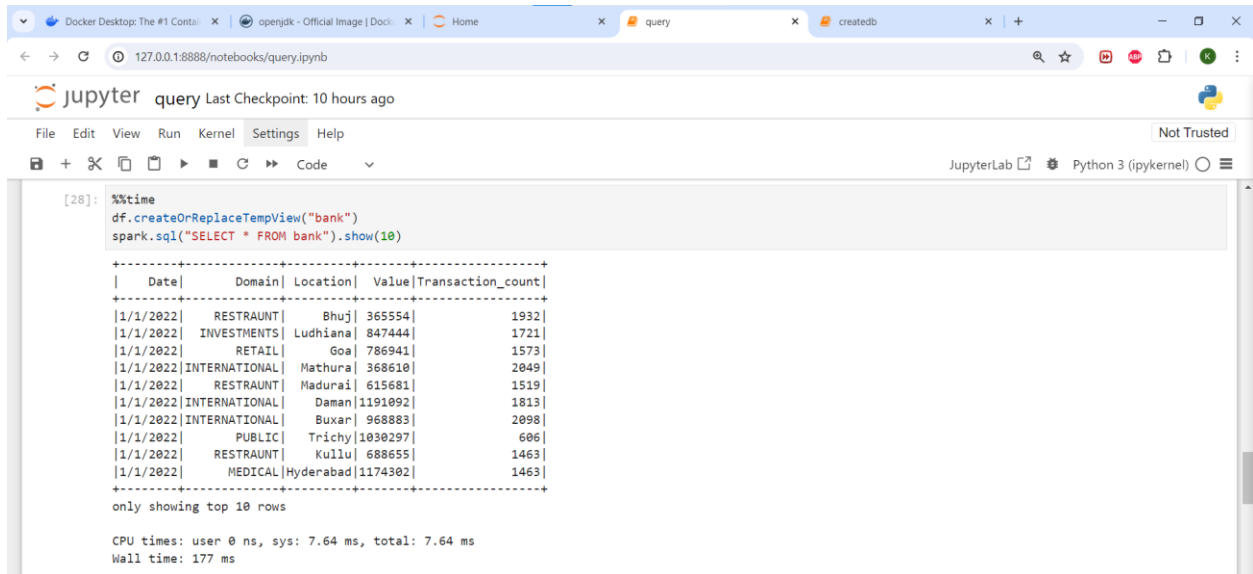
df.head(5)
```

The output of the code cell shows the first 5 rows of the DataFrame after the delete operation:

```
[15]: [Row(Date='1/1/2022', Domain='INVESTMENTS', Location='Ludhiana', Value=847444, Transaction_count=1721),
Row(Date='1/1/2022', Domain='RETAIL', Location='Goa', Value=786941, Transaction_count=1573),
Row(Date='1/1/2022', Domain='INTERNATIONAL', Location='Mathura', Value=368610, Transaction_count=2049),
Row(Date='1/1/2022', Domain='RESTRAUNT', Location='Madurai', Value=615681, Transaction_count=1519),
Row(Date='1/1/2022', Domain='INTERNATIONAL', Location='Daman', Value=1191092, Transaction_count=1813)]
```

Bước 4: So sánh thời gian chạy các câu lệnh truy vấn

Khi không có Where:



```
[28]: %%time
df.createOrReplaceTempView("bank")
spark.sql("SELECT * FROM bank").show(10)
```

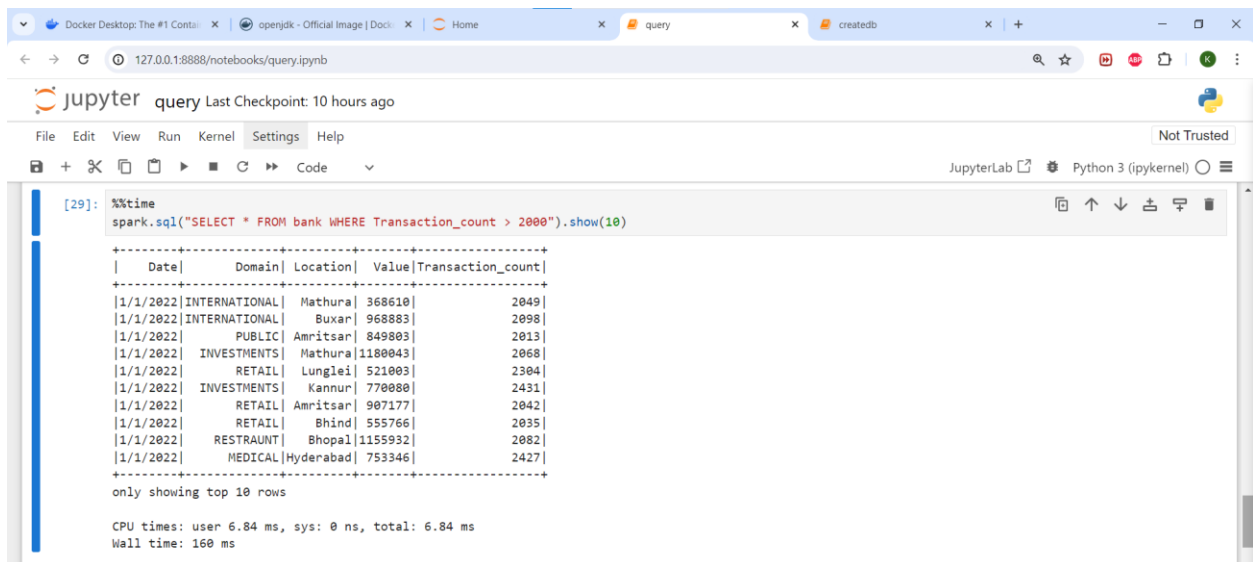
Date	Domain	Location	Value	Transaction_count
1/1/2022	RESTAUNT	Bhuj	365554	1932
1/1/2022	INVESTMENTS	Ludhiana	847444	1721
1/1/2022	RETAIL	Goa	786941	1573
1/1/2022	INTERNATIONAL	Mathura	368610	2049
1/1/2022	RESTAUNT	Madurai	615681	1519
1/1/2022	INTERNATIONAL	Daman	1191092	1813
1/1/2022	INTERNATIONAL	Buxar	968883	2098
1/1/2022	PUBLIC	Trichy	1030297	606
1/1/2022	RESTAUNT	Kullu	688655	1463
1/1/2022	MEDICAL	Hyderabad	1174302	1463

only showing top 10 rows

CPU times: user 0 ns, sys: 7.64 ms, total: 7.64 ms
Wall time: 177 ms

Thời gian chạy là 7.64 ms.

Khi có Where:



```
[29]: %%time
spark.sql("SELECT * FROM bank WHERE Transaction_count > 2000").show(10)
```

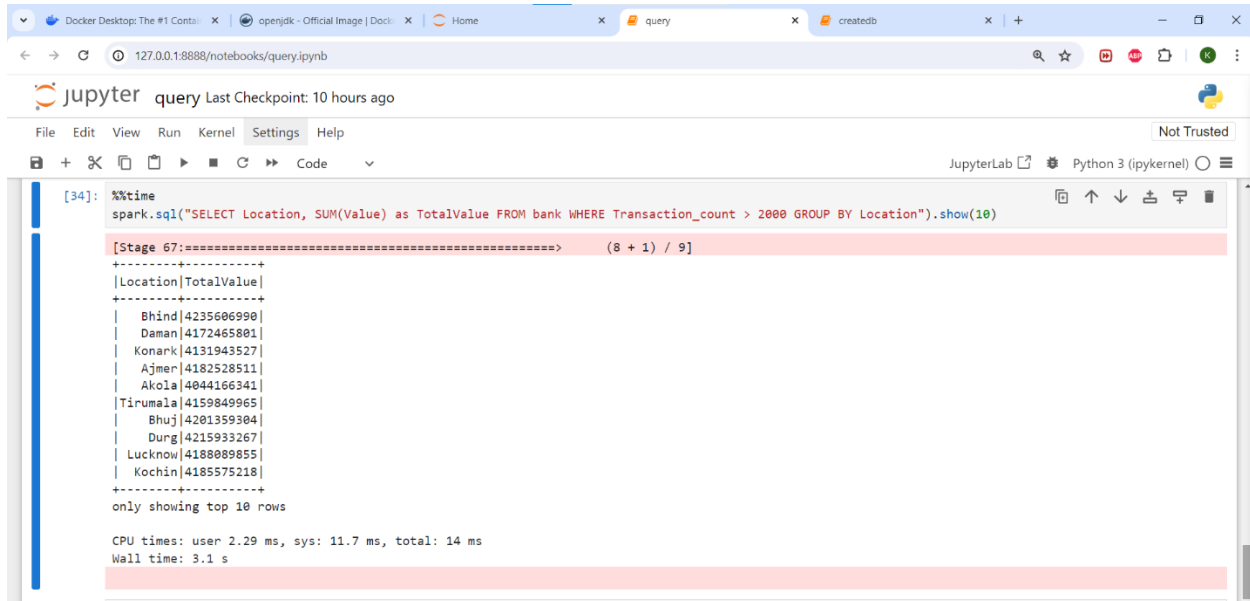
Date	Domain	Location	Value	Transaction_count
1/1/2022	INTERNATIONAL	Mathura	368610	2049
1/1/2022	INTERNATIONAL	Buxar	968883	2098
1/1/2022	PUBLIC	Amritsar	849803	2013
1/1/2022	INVESTMENTS	Mathura	1180043	2068
1/1/2022	RETAIL	Lunglei	521003	2304
1/1/2022	INVESTMENTS	Kannur	770080	2431
1/1/2022	RETAIL	Amritsar	907177	2042
1/1/2022	RETAIL	Bhind	555766	2035
1/1/2022	RESTAUNT	Bhopal	1155932	2082
1/1/2022	MEDICAL	Hyderabad	753346	2427

only showing top 10 rows

CPU times: user 6.84 ms, sys: 0 ns, total: 6.84 ms
Wall time: 160 ms

Thời gian chạy là 6.84 ms. Nhanh hơn một chút so với không có Where

Khi có Group By:



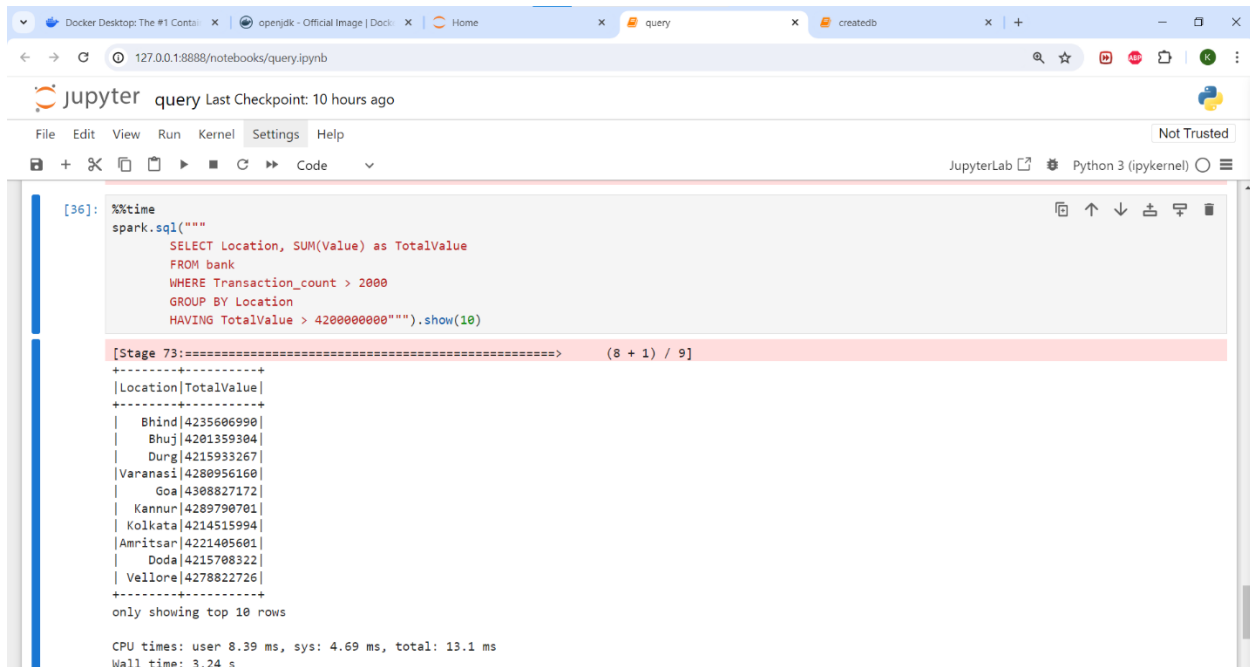
```
[34]: %%time
spark.sql("SELECT Location, SUM(Value) as TotalValue FROM bank WHERE Transaction_count > 2000 GROUP BY Location").show(10)

[Stage 67:=====] (8 + 1) / 9]
+-----+
|Location|TotalValue|
+-----+
|Bhind|4235606990|
|Daman|4172465801|
|Konark|4131943527|
|Ajmer|4182528511|
|Akoia|4044166341|
|Tirumala|4159849965|
|Bhuj|4201359304|
|Durg|4215933267|
|Lucknow|4188089855|
|Kochin|4185575218|
+-----+
only showing top 10 rows

CPU times: user 2.29 ms, sys: 11.7 ms, total: 14 ms
Wall time: 3.1 s
```

Thời gian chạy là 14 ms. Lâu hơn một chút do tính toán tổng của giá trị.

Khi có Having:



```
[36]: %%time
spark.sql("""
    SELECT Location, SUM(Value) as TotalValue
    FROM bank
    WHERE Transaction_count > 2000
    GROUP BY Location
    HAVING TotalValue > 4200000000""").show(10)

[Stage 73:=====] (8 + 1) / 9]
+-----+
|Location|TotalValue|
+-----+
|Bhind|4235606990|
|Bhuj|4201359304|
|Durg|4215933267|
|Varanasi|4280956160|
|Goa|4308827172|
|Kannur|4289790701|
|Kolkata|4214515994|
|Amritsar|4221405601|
|Doda|4215708322|
|Vellore|4278822726|
+-----+
only showing top 10 rows

CPU times: user 8.39 ms, sys: 4.69 ms, total: 13.1 ms
Wall time: 3.24 s
```

Thời gian chạy mất 13.1 ms. Nhanh hơn một chút so với khi không có Having.

Kết luận

Như vậy, documentation này đã cung cấp một hướng dẫn đầy đủ về việc cài đặt và sử dụng SparkSQL trong một Docker container, thực hiện được những thao tác CRUD cơ bản và so sánh thời gian thực hiện các câu truy vấn.

Contact

Lê Trung Kiên

Email: letrungkien3_t66@hus.edu.vn

Number Phone: 0353693404