Họ và tên: Vũ Thành An

Mã Sinh Viên: 21002184

Lớp: K66 Kĩ Thuật Điện Tử Tin Học

**Cấu hình và chạy SparkSQL truy vấn cơ sở dữ liệu với DOCKER**

**1. Thiết lập Docker**

-Thực hiện tải xuống Docker Desktop cho hệ máy Windows.

-Tạo một thư mục có tên “Final”. Chuyển tập dữ liệu “data.csv” vào thư mục.

- Mở thư mục trên ứng dụng Visual Studio Code.

-Tạo một file với tên Dockerfile không có hậu tố.

-Thêm vào file đó dòng sau:

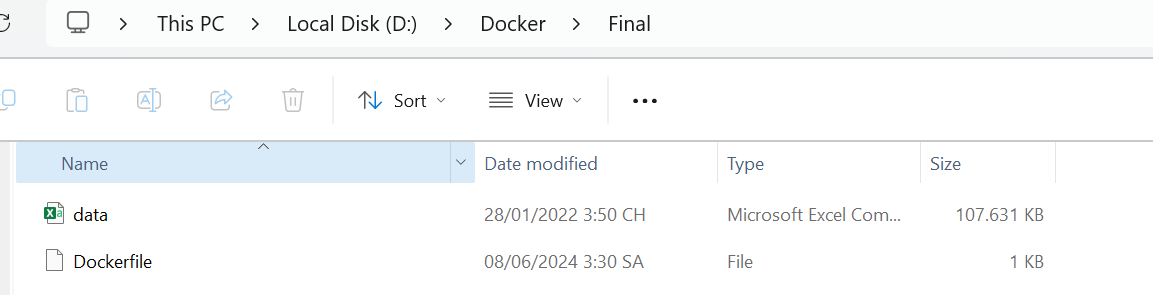
#Lựa chọn images gốc cho container

**FROM ubuntu**

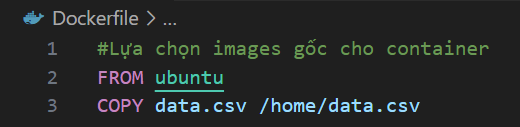
**COPY data.csv /home/data.csv**

- Sau đó tại thư mục “Final”, mở terminal và tạo một images có tên “lab-final” bằng lệnh: "**docker build -t lab-final .**"

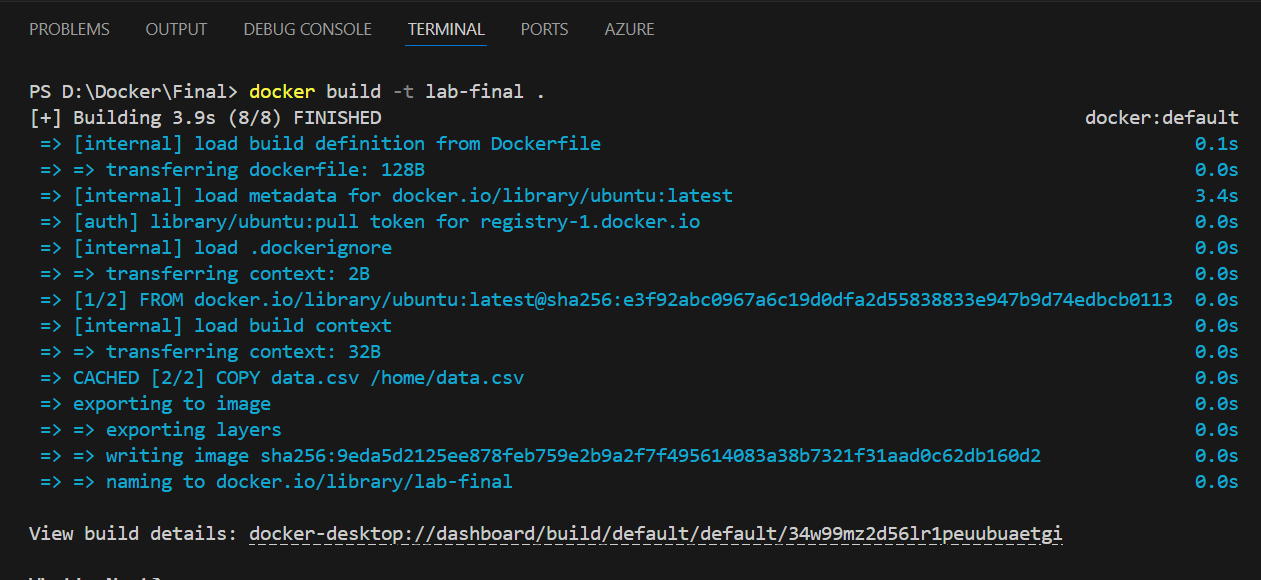
- Sau đó chạy images đó để tạo container bằng lệnh: "**docker run -it lab-final bash**" .



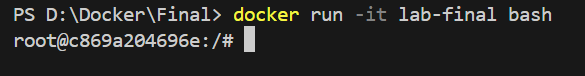
**Hình 1. Thư mục lưu trữ**



**Hình 2: Cấu hình Dockerfile**



**Hình 3: Build images**



**Hình 4: Chạy và tương tác với container thông qua cmd**

**2. Cài đặt Spark**

- Để chạy Spark trên container vừa tạo ta cần cài đặt một số công cụ cần thiết cho việc sử dụng và chạy. Sử dụng Dockerfile tạo ở phần 1 và nhập các lệnh sau:

**RUN apt-get update**

**RUN apt-get -y install openjdk-8-jdk**

**RUN apt-get -y install python3**

**RUN apt-get -y install wget**

**RUN apt-get -y install tar**

**RUN apt-get -y install curl**

**RUN apt-get -y install python3-pip**

**RUN apt-get -y install net-tools**

**RUN apt-get -y install vim**

**RUN wget h**[**ttps://dlcdn.apache.org/spark/spark-3.5.1/spark-3.5.1-bin-h**](ttps://dlcdn.apache.org/spark/spark-3.5.1/spark-3.5.1-bin-h)**adoop3.tgz**

**RUN tar xvf spark-3.5.1-bin-hadoop3.tgz**

**RUN mv spark-3.5.1-bin-hadoop3 /opt/spark**

**RUN rm spark-3.5.1-bin-hadoop3.tgz**

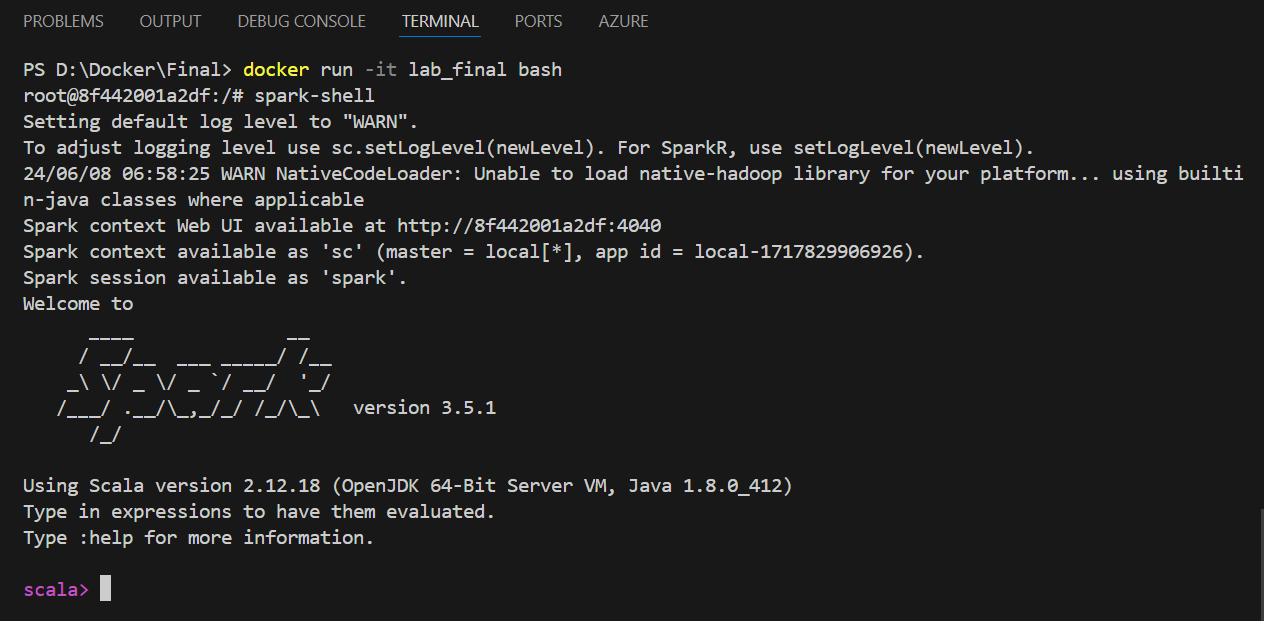
- Sau khi đã lựa chọn các công cụ cần thiết, giờ ta cần cấu hình đường dẫn chạy cho Spark vào trong Dockerfile:

**ENV SPARK\_HOME=/opt/spark**

**ENV PATH=$SPARK\_HOME/bin:$SPARK\_HOME/sbin:$PATH**

-Sau đó, **build** lại images và chạy container.

-Cuối cùng, nhập lệnh "**spark-shell**" để xác minh Spark đã chạy.

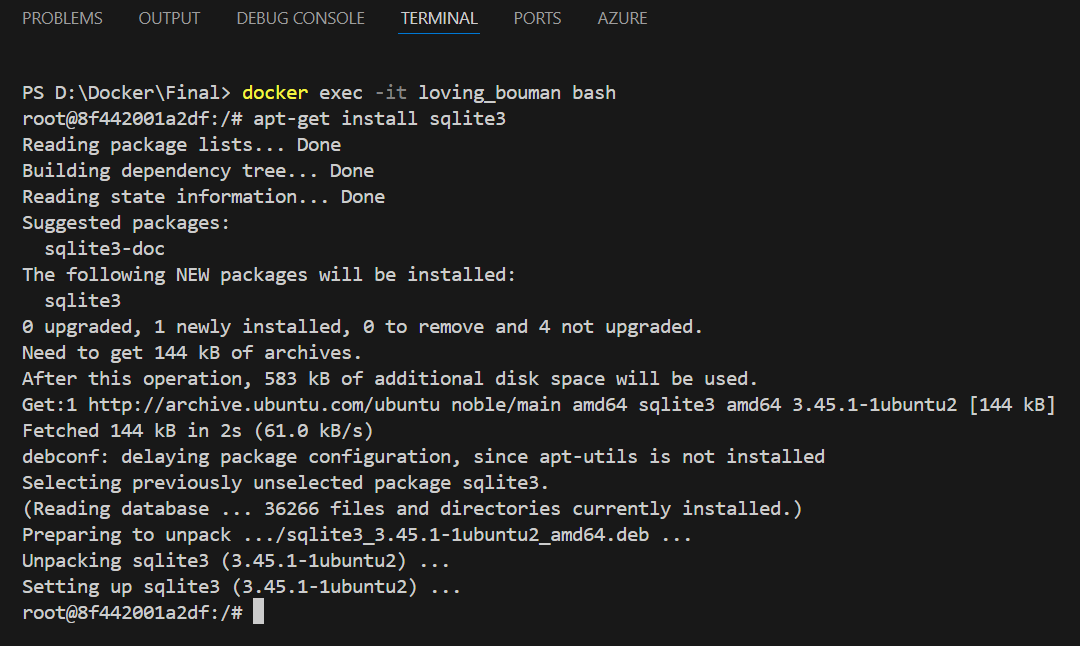
****

**Hình 5: Xác minh Spark đã chạy**

**3. Thiết lập Database**

**-** Mở terminal mới, nhập **“docker exec -it {tên container} bash”** để truy cập vào container.

-Chạy lệnh **apt-get install sqlite3** để tải về SQLite.



**Hình 6: Tải xuống SQLite**

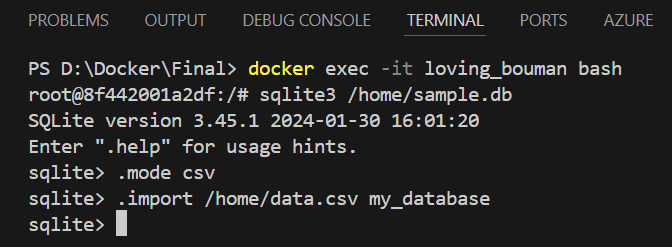
- Ở phần 1 ta đã copy một file có tên “data.csv” vào thư mục **/home** với tên **/data.csv**. Bây giờ, ta sẽ tạo và chạy cơ sở dữ liệu bằng lệnh sau: “**sqlite3 /home/sample.db”**

-Giao diện lúc này sẽ hiển thị đang tương tác với database, lúc này thực hiện 2 lệnh:

**.mode csv**

**.import /home/data.csv my\_database**

- Quá trình này import file “data.csv” gồm khoảng1 triệu dòng vào bảng “my\_database”.



**Hình 7: Tạo database và nhập dữ liệu vào bảng**

**4. Các thành phần cần thiết**

- Ở phần này chúng ta sẽ thêm 2 thành phần cần thiết đó là PySpark và JDBC cho SQLite để giúp kết nối và thực hiện truy vấn từ Spark đến cơ sở dữ liệu.

- Mở terminal mới, chạy lệnh sau để kéo file về máy:

**wget https://repo1.maven.org/maven2/org/xerial/sqlite-jdbc/3.34.0/sqlite-jdbc-3.34.0.jar**

**-** Nhập **“docker exec -it {tên container} bash”** để truy cập vào container rồi chạy 2 lệnh sau để cài đặt:

**pip3 install pyspark**

**wget https://repo1.maven.org/maven2/org/xerial/sqlite-jdbc/3.34.0/sqlite-jdbc-3.34.0.jar**

**5. Cấu hình**

- Để Spark có thể kết nối đến cơ sở dữ liệu ta cần di chuyển thư viện JDBC vào thư mục jars của Spark, đây là thư mục giúp Spark tự động phát hiện và chạy các thư viện phụ trợ nếu cần thiết. Nhập lệnh:

**mv sqlite-jdbc-3.34.0.jar /opt/spark/jars**

-Sau khi di chuyển hoàn tất, ta có thể thử truy cập và tương tác với cơ sở dữ liệu bằng PySpark.

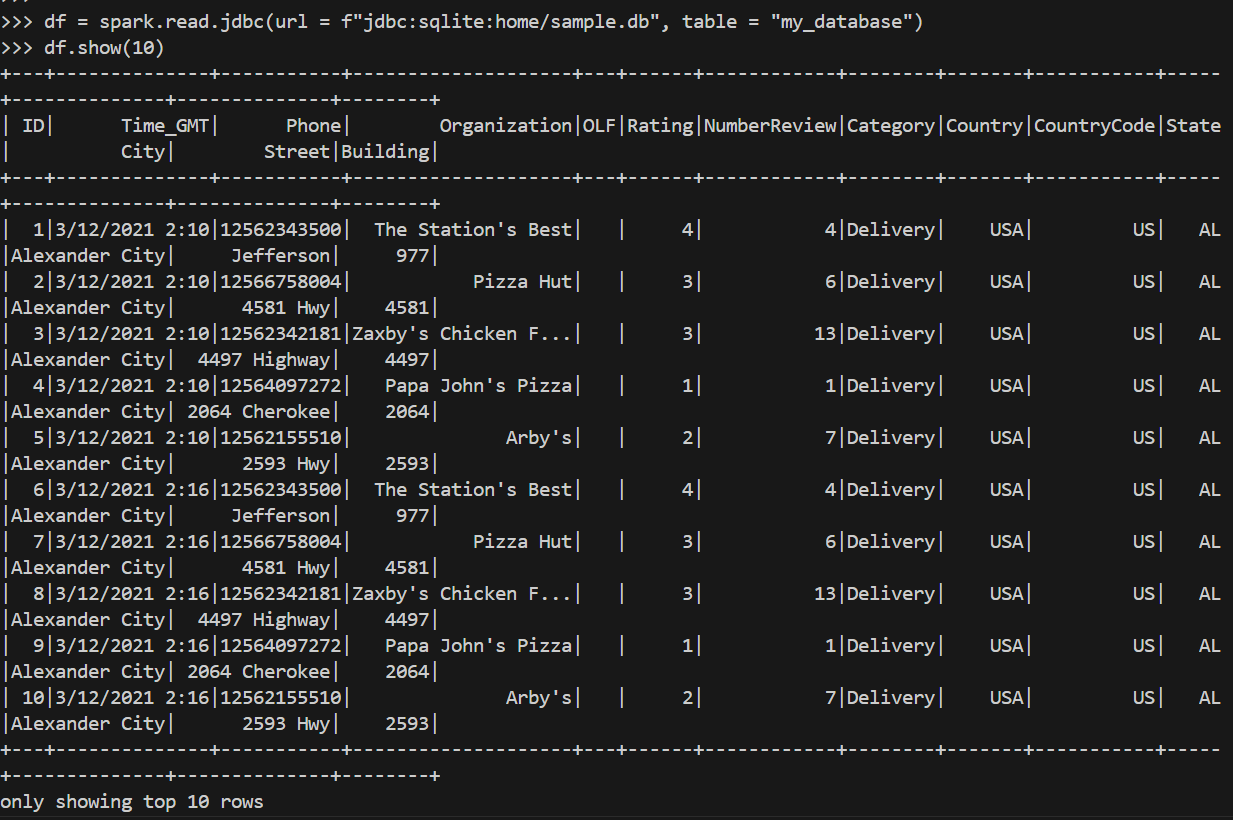
-Tại terminal, sau khi truy cập vào container, nhập "**pyspark**" để khởi chạy, lúc này một phiên làm việc của Spark thông qua giao diện dòng lệnh sẽ bắt đầu, nó cung cấp một môi trường tương tác với Spark bằng Python.

- Ví dụ ta nhập lệnh sau :

**df = spark.read.jdbc(url = f"jdbc:sqlite:home/sample.db", table = "my\_database")**

**df.show(10)**

- Kết quả thử nghiệm kết nối đến cơ sở dữ liệu “sample.db”, bảng “my\_database” và hiển thị 10 dòng đầu tiên.



**Hình 8: Kết nối và tương tác Spark với database**

**6. Truy vấn dữ liệu**

**6.1. Truy vấn dữ liệu đơn giản**

-Đầu tiên, ta tạo một file có tên “get\_session\_data.py” để chạy session và khởi động trình truy vấn SQL.

- Mở terminal mới, truy cập và container, nhập **“cd home”** để truy cập vào thư mục **/home**, rồi nhập lệnh "**vi get\_session\_data.py**" để tạo file mới. Nhập vào đoạn code bên dưới:



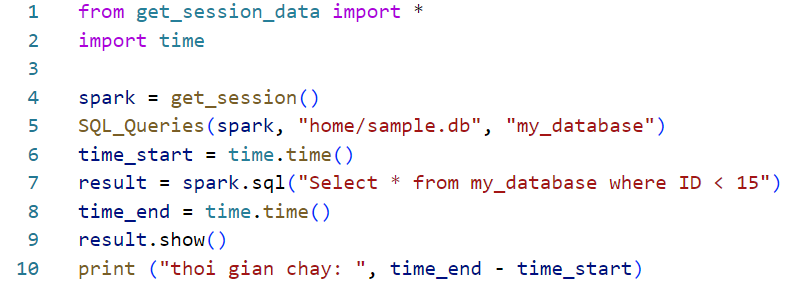
**Hình 9: File get\_session\_data.py**

-Hàm **get\_session()** cho phép khởi tạo các thiết lập và tài nguyên cần thiết cho việc thao tác với Spark

-Hàm **SQL\_Queries()** cho phép ta khởi tạo một trình truy vấn SQL với cơ sở dữ liệu đươc chọn dưới dạng tạm thời và sẽ không làm ảnh hưởng đến cơ sở dữ liệu thực của bạn

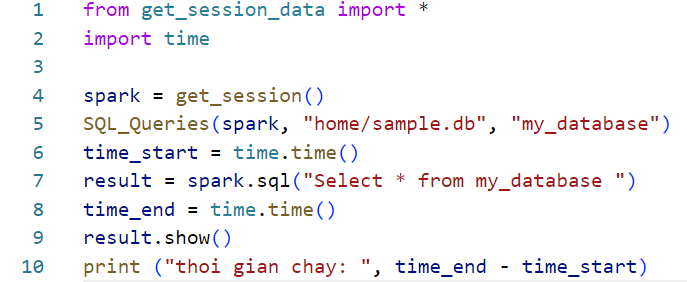
-Sau khi đã tạo file cấu hình xong, giờ ta sẽ tạo file để chạy truy vấn đồng thời kiểm tra thời gian chạy của nó.

-Tạo 2 file tên “test1.py” và “test2.py”tương tự như trên để truy vấn cơ sở dữ liệu và nhập vào đoạn code bên dưới.



**Hình 9: File test1.py**

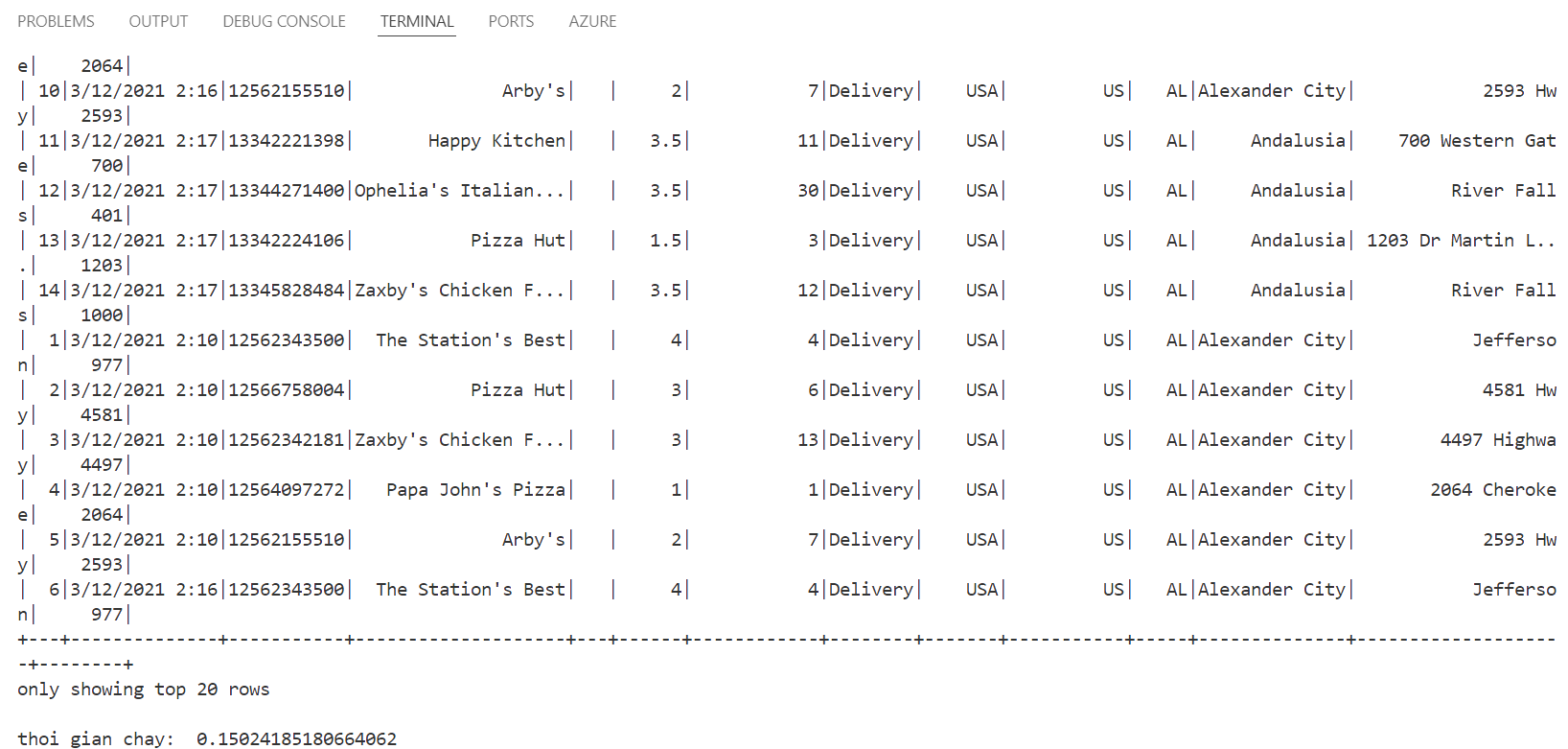
Hàm đo thời gian sẽ chỉ tính trong phạm vi câu truy vấn. Sự khác biệt 2 file là điều kiện **where ID < 10** để xem sự ảnh hưởng của **WHERE** đến truy vấn.



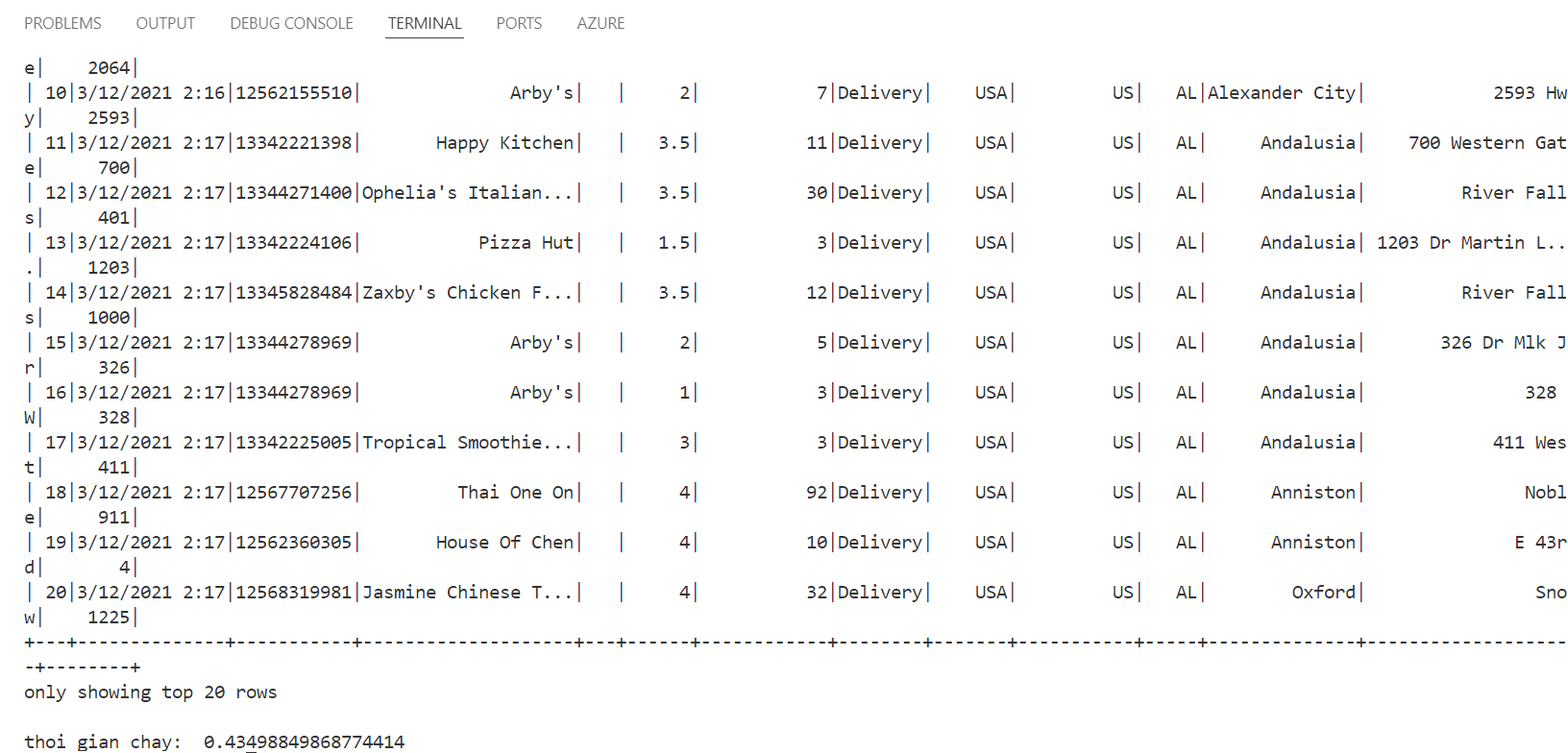
**Hình 10: File test2.py**

- Mở terminal, truy cập vào container, nhập **“python3 home/test1.py”** để chạy file “test1.py”. Thực hiện tương tự với file “test2.py”

- Kết quả chạy 2 file như sau: với điều kiện where, file “test1.py” mất khoảng **0.15s** để thực hiện trong khi “test2.py” không có where mất khoảng **0.435s**



**Hình 11: Thời gian chạy file test1.py**

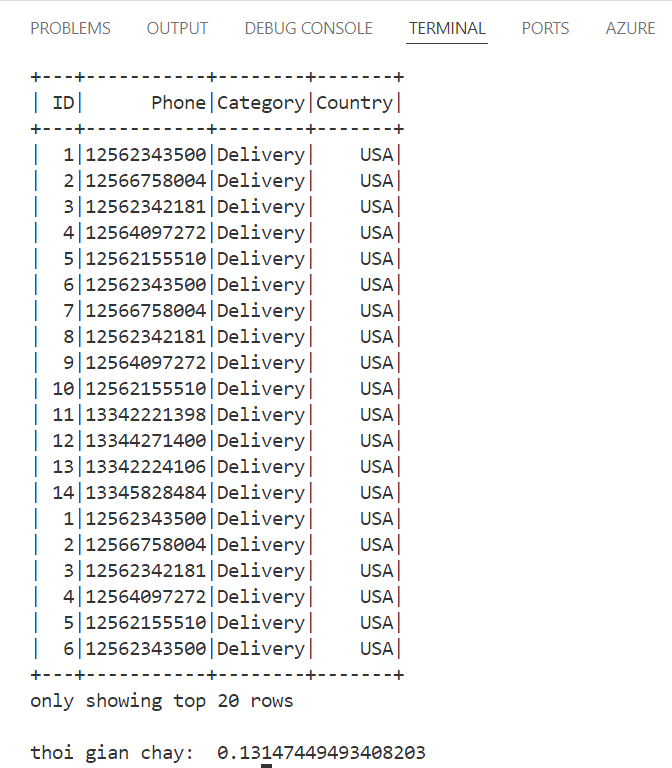


**Hình 12: Thời gian chạy file test2.py**

-Thử nghiệm với một vài truy vấn khác, tại file “test3.py” dưới đây, tôi thử nghiệm quá trình truy vấn với việc chỉ chọn 4 cột dữ liệu ID, Phone, Category, Country và vẫn với điều kiện ID < 15. Kết quả cho thấy, thời gian chạy chỉ hết khoảng **0.13s**

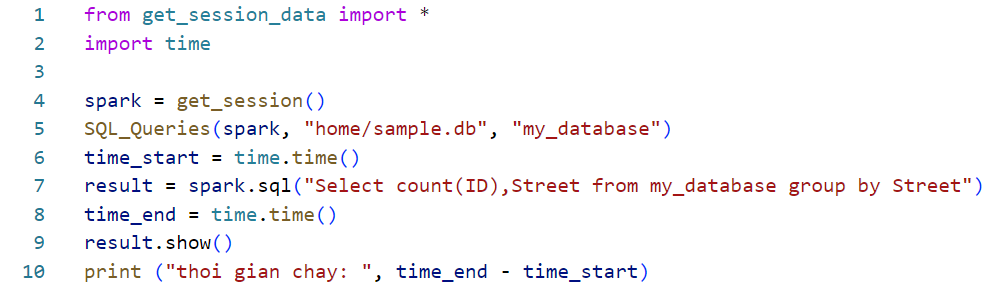


**Hình 13: File test3.py**

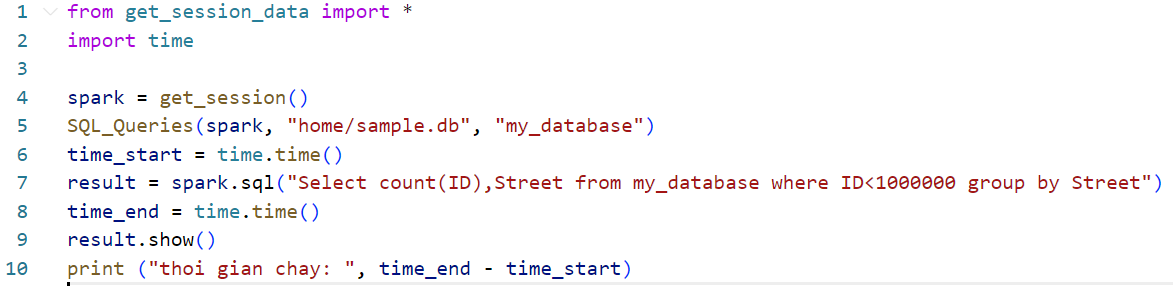


**Hình 14: Thời gian chạy file test3.py**

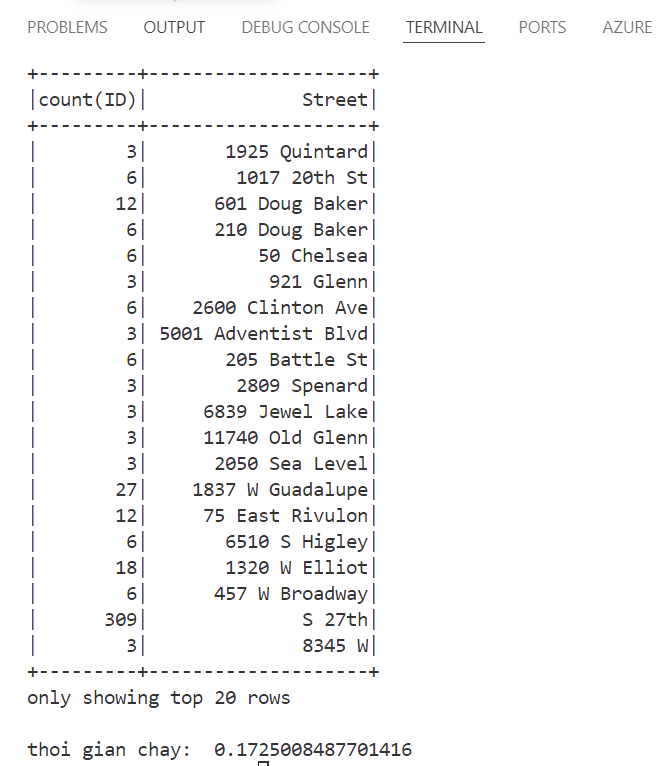
- Thử với một truy vấn liên quan đến **GROUP BY**, tôi truy vấn đếm số lượng người đến từ các quốc gia khác nhau từ cơ sở dữ liệu file “test4.py”và“test5.py”với sự khác biệt nằm ở **where ID < 1000001**



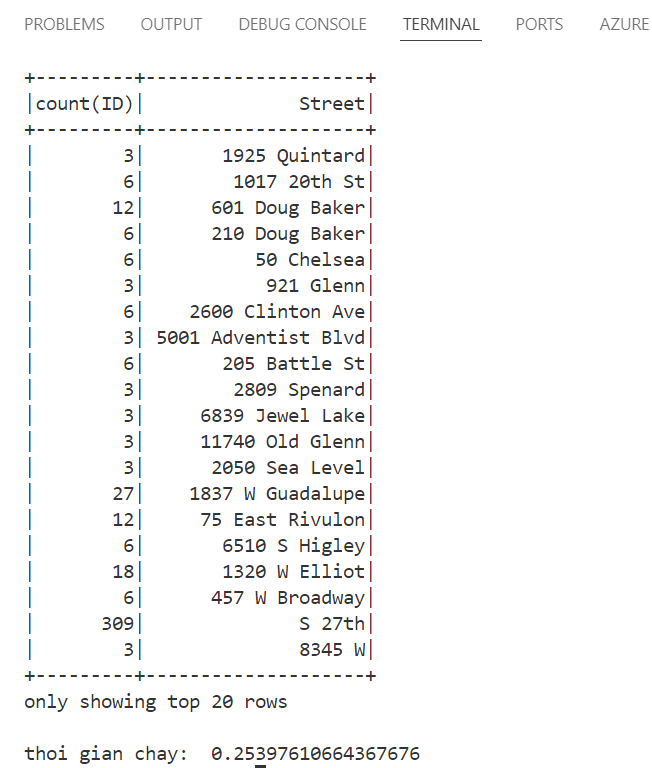
**Hình 15: File test4.py**



**Hình 15: File test5.py**



**Hình 16: Thời gian chạy file test4.py**



**Hình 17: Thời gian chạy file test5.py**

**6.2. Kết luận**

Có thể thấy, lệnh **WHERE** ảnh hưởng tương đối đến tốc độ truy vấn tới database:

Sử dụng chỉ mục:

* Khi mệnh đề **WHERE** sử dụng các cột có chỉ mục, truy vấn sẽ nhanh hơn vì cơ sở dữ liệu có thể nhanh chóng tìm thấy các hàng cần thiết.
* Ngược lại, nếu điều kiện **WHERE** không sử dụng các cột có chỉ mục, cơ sở dữ liệu sẽ phải thực hiện quét toàn bộ bảng (full table scan), làm chậm quá trình truy vấn.

Độ phức tạp của điều kiện:

* Điều kiện đơn giản và rõ ràng (ví dụ: WHERE age = 30) thường sẽ nhanh hơn so với điều kiện phức tạp (ví dụ: WHERE age > 30 AND (salary < 50000 OR department = 'HR')).
* Điều kiện sử dụng các phép toán phức tạp hoặc hàm cũng sẽ làm chậm truy vấn.

Kích thước của bảng:

* Với các bảng lớn, ảnh hưởng của mệnh đề **WHERE** càng quan trọng. Một mệnh đề **ss** được tối ưu hóa sẽ giảm đáng kể thời gian truy vấn trên các bảng lớn.
* Trên các bảng nhỏ, sự khác biệt có thể không quá rõ ràng.