Họ và tên: Nguyễn Xuân An

Mã Sinh Viên: 21002183

Lớp: K66 Kĩ Thuật Điện Tử Tin Học

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

**1. Docker Setup**

-Thực hiện tải xuống Docker Desktop cho hệ máy Window

-Tạo một thư mục có tên docker-build-run

-Tạo một File với tên Dockerfile không có hậu tố

-Sử dụng trình soạn thảo Notepad thêm vào File đó dòng sau:

#Lựa chọn images gốc cho container

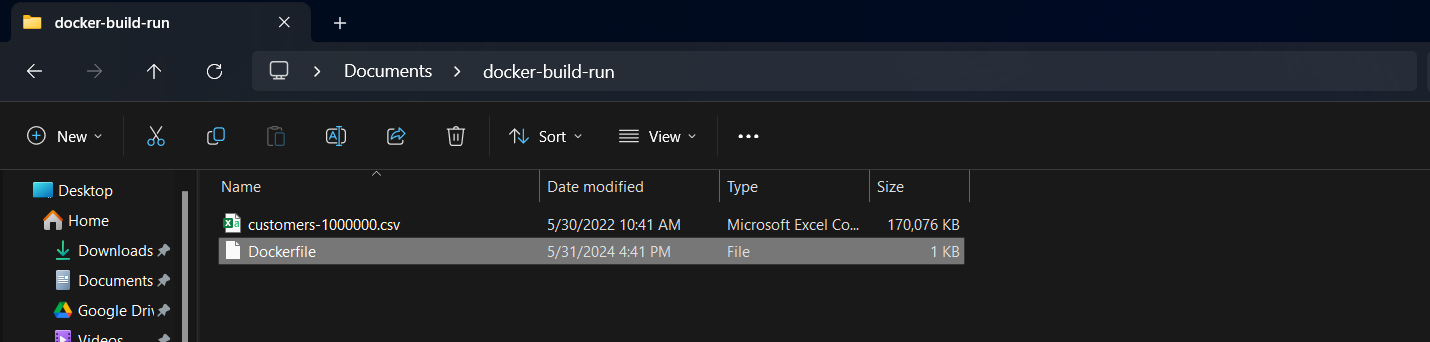
**FROM ubuntu:20.04**

**COPY customers-1000000.csv /home/data.csv**

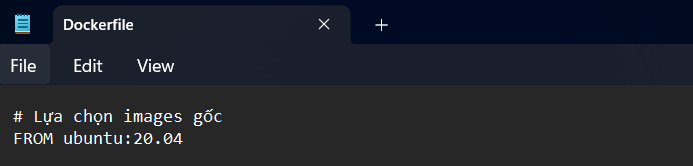
-Sau đó vào thư mục docker-build-run, chạy thư mục với cmd bằng cách chuột phải tại thư mục và chọn "**open in terminal**"

-Tạo một images bằng lệnh: "**docker build -t lab-final .**"

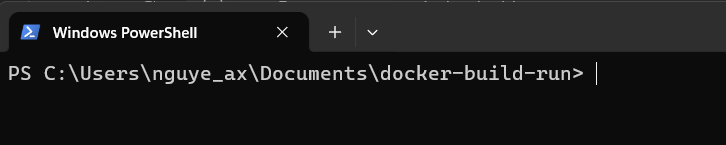
-Sau đó chạy images đó để chạy container bằng lệnh: "**docker run -it lab-final bash**" hoặc chạy trong giao diện của Docker Desktop và tương tác với container qua mục **Exec**



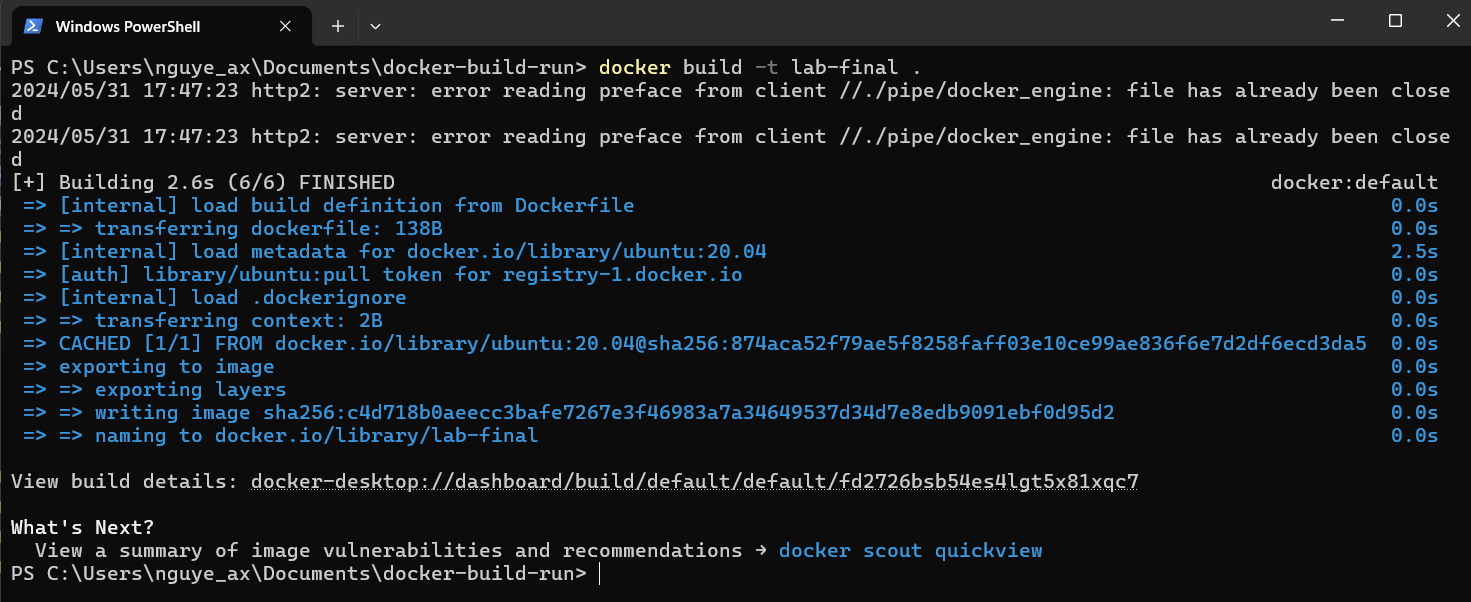
**Ảnh 1: Thư mục lưu trữ**



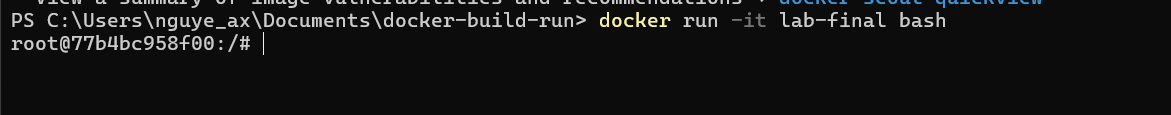
**Ảnh 2: Cấu hình Dockerfile**



**Ảnh 3: Chạy với giao diện cmd**



**Ảnh 4: Build images**



**Ảnh 5: Chạy và tương tác với container thông qua cmd**

**2. Spark Installation**

-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 thêm vào 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

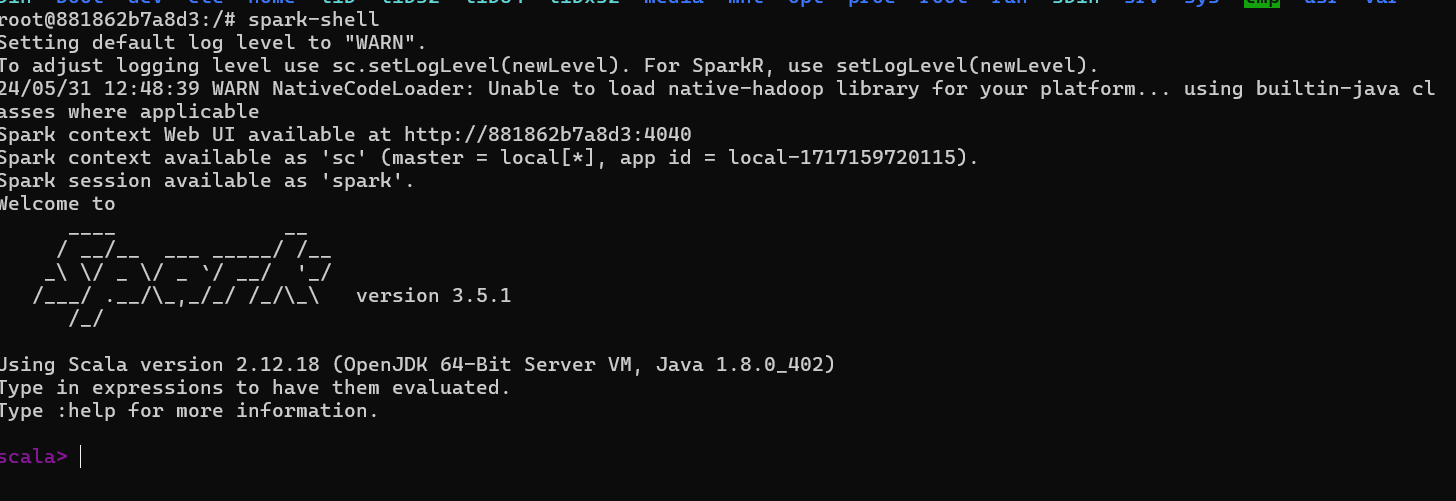
**ENV SPARK\_HOME=/opt/spark**

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

-Sau đó, **build** lại images với tên **lab-final1** và chạy container

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

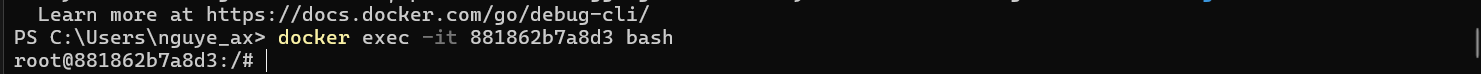
-Hiện tại, **Spark** đang chạy bằng ngôn ngữ Scala trên nền tảng OpenJDK 8. Quá trình khởi chạy đã thành công, giờ ta có thể sử dụng ngôn ngữ Scala để thực hiện các thao tác với dữ liệu

****

**Ảnh 6: Xác minh Spark đã chạy**

**3. Database Setup**

-Để truy cập vào container vừa mới build xong ta có thể sử dụng lệnh sau **docker exec -it <name\_container> bash**. Quá trình này sẽ cho phép ta tương tác với **Container** thông qua giao diện **cmd.** Lúc này, ta sẽ thao tác và cài đặt thêm một số công cụ cho **Container** này



-Đầu tiên chạy lệnh **apt-get install sqlite3** để tải về SQLite

-Tiếp theo, ở **bước 1** chúng ta đã copy một file có tên customers-1000000.csv vào thư mục **/home** với tên **/data.csv**

-Bây giờ tạo và chạy cơ sở dữ liệu bằng lệnh sau: **sqlite3 /home/sample.db**

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

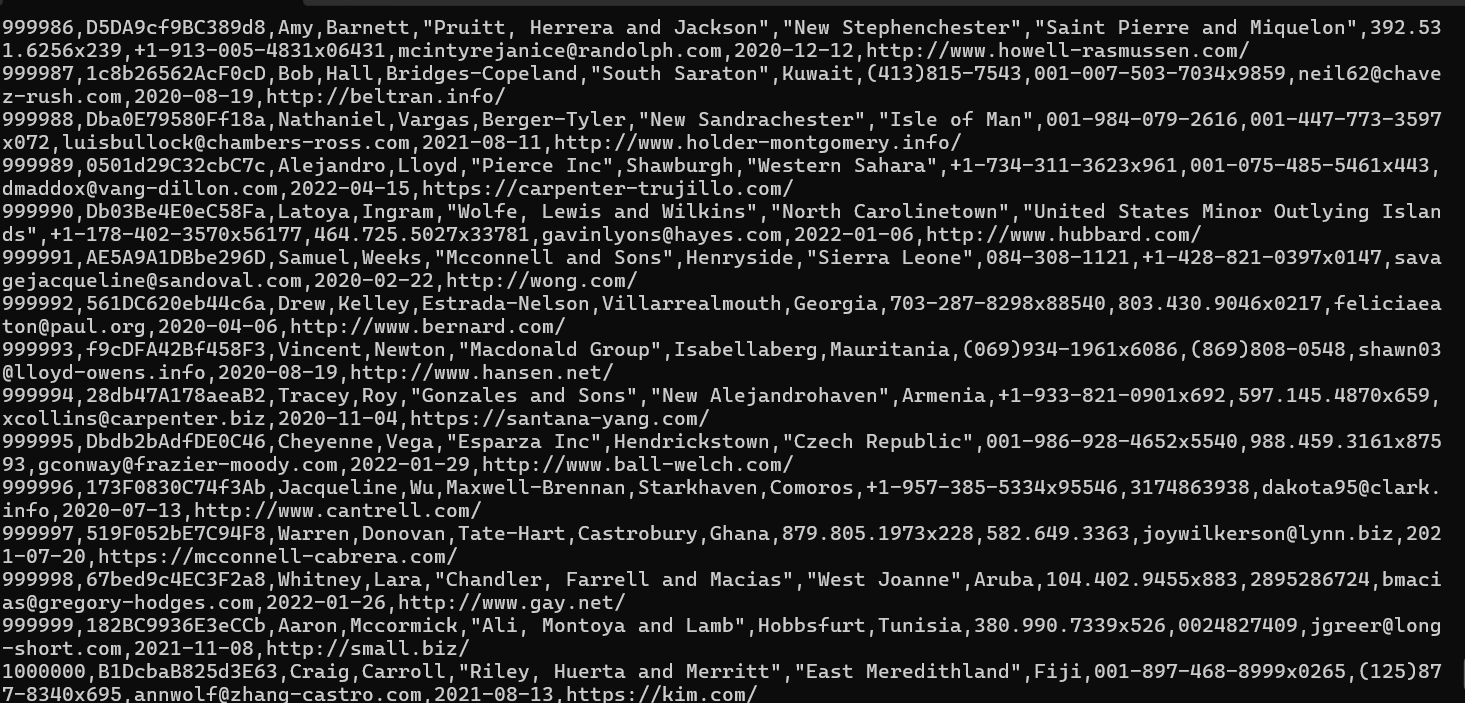
**.mode csv**

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

-Quá trình này import file data.csv gồm 1 triệu dòng vào my\_database

-**SELECT \* FROM my\_database;**

-Với câu truy vấn trên toàn bộ dữ liệu, dữ liệu cuối cùng chứa Index 1000000 đã phù hợp với nội dung yêu cầu



**4. Dependencies**

-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. Chạy 2 lệnh sau

**pip install pyspark**

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

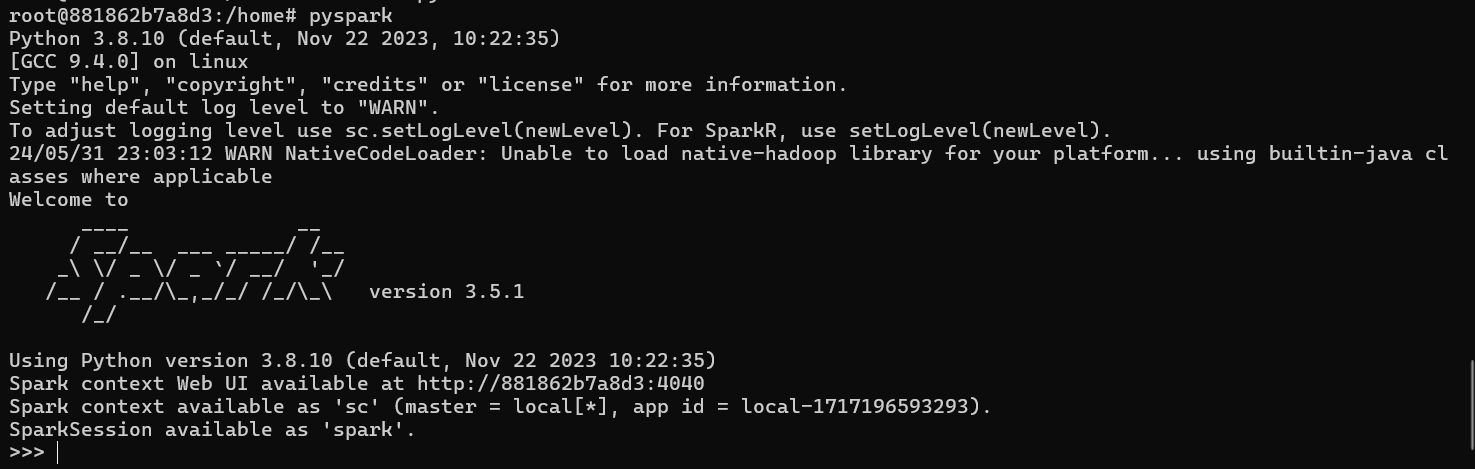
**5. Configuration**

-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 giao diện cmd, 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.

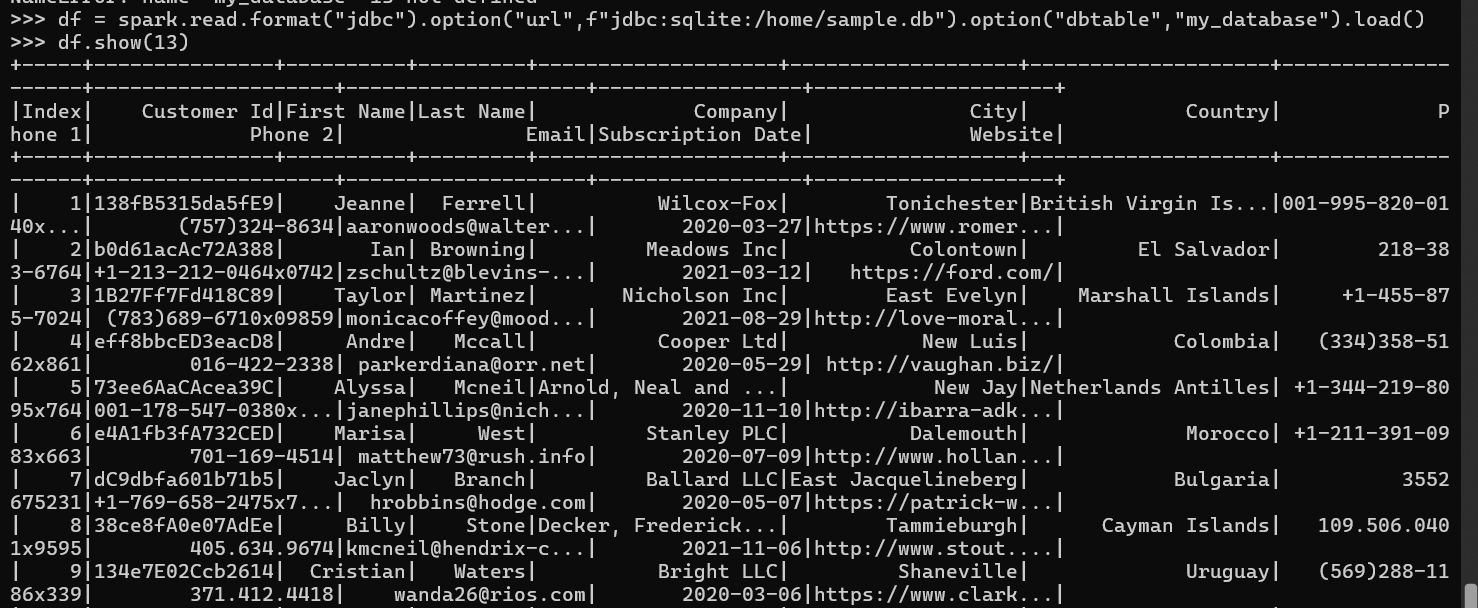


-Sau đó ta nhập lệnh sau

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

**df.show(13)**

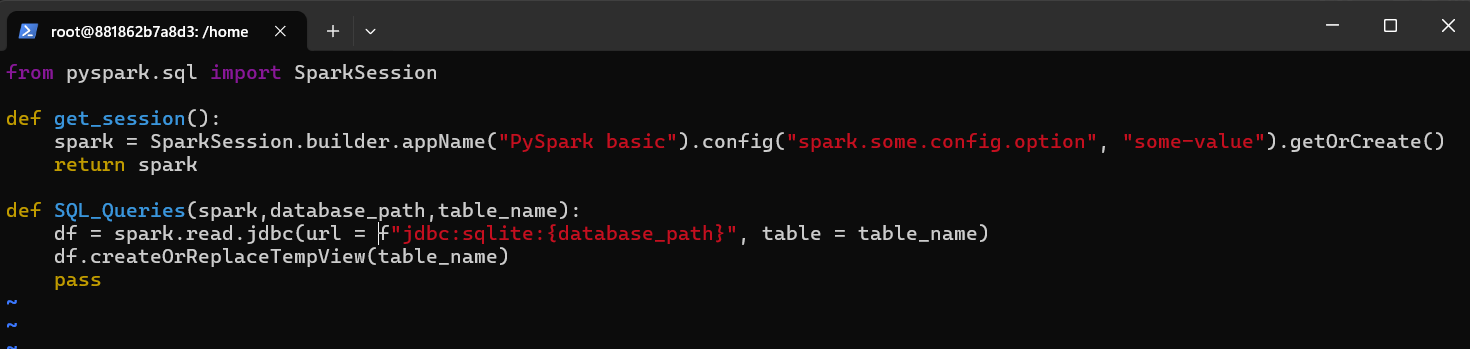
-Dưới đây là thử nghiệm kết nối đến cơ sở dữ liệu sample.db, bảng my\_database, hiển thị 13 dòng đầu tiên



**6. Running Queries**

-Đầu tiên, để có thể thực hiện các thao tác với cơ sở dữ liệu thông qua các file nội dung, ta cần cấu hình cho session của mỗi file. Tạo một file có tên get\_session\_data để chạy session và khởi động trình truy vấn SQL

-Nhập lệnh "**vi get\_session\_data.py**" rồi điền vào các nội dung sau



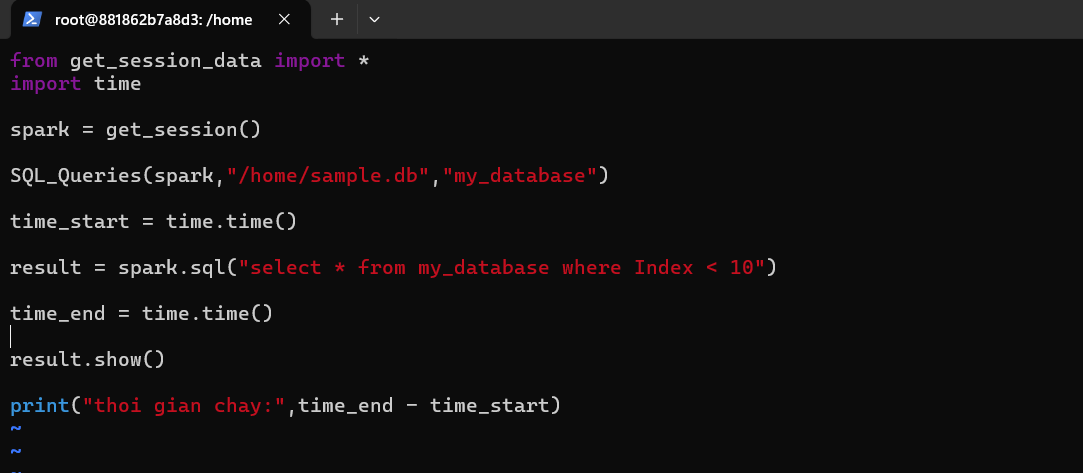
-Hàm **get\_session()** cho phép khởi tạo các thiết lập và tài nguyên cần thiết cho việc thao tác với Spark

-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

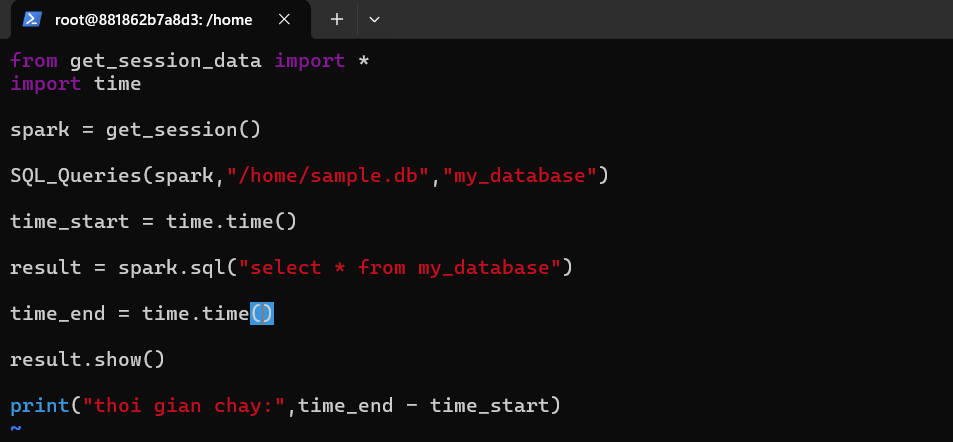
**6.1. Truy vấn cơ sở dữ liệu**

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

-Tạo 2 file tên **exam1.py** và **exam2.py** truy vấn cơ sở dữ liệu và điền vào nội dung sau. Hàm đo thời gian sẽ chỉ tính trong phạm vi câu truy vấn. Sự khác biệt 2 file là điều kiện **WHERE Index < 10** để xem sự ảnh hưởng của **WHERE** sẽ thế nào đến truy vấn

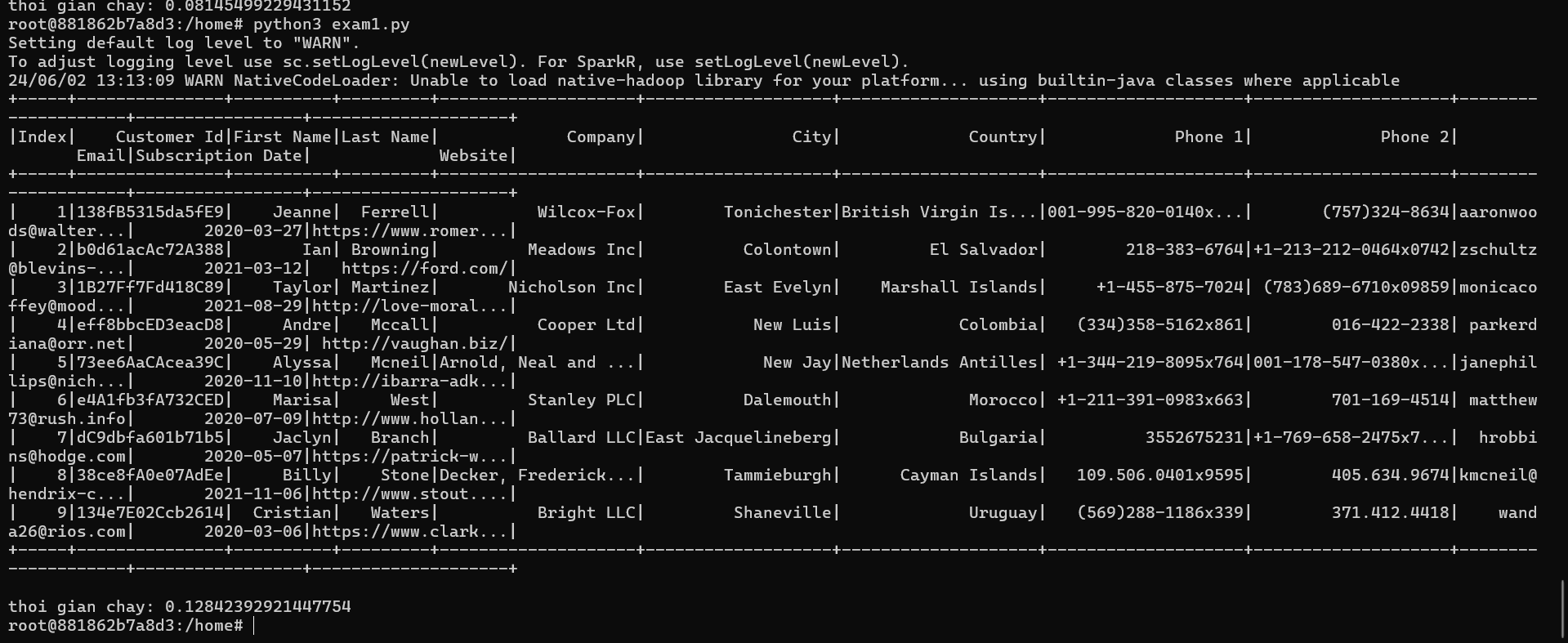


**Ảnh 1: nội dung exam1.py**

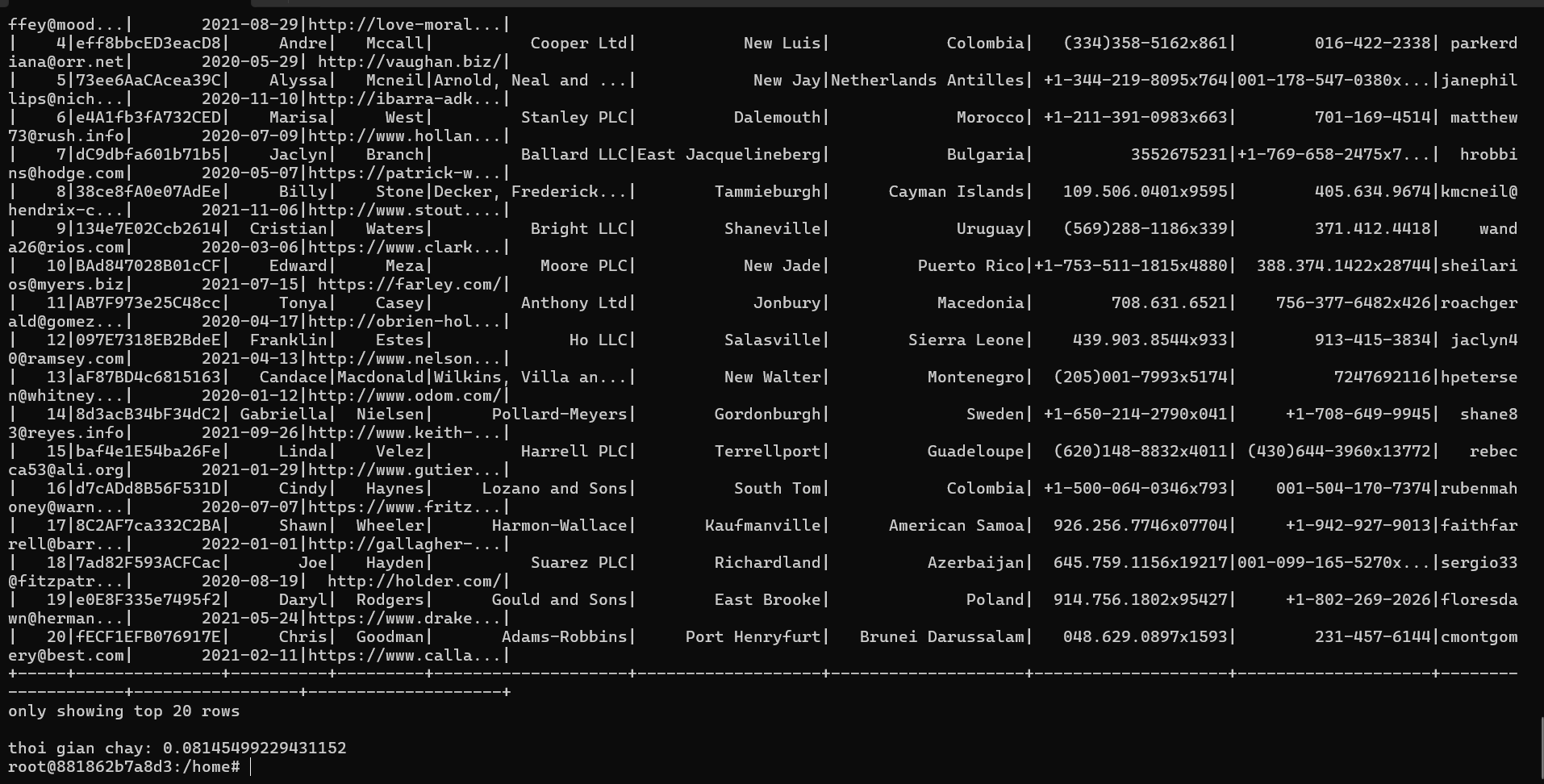


**Ảnh 2: Nội dung exam2.py**

-Kết quả với điều kiện **WHERE**, **exam1** mất khoảng **0.13s** để thực hiện trong khi **exam2.py** không có **WHERE** chỉ mất khoảng **0.08s**

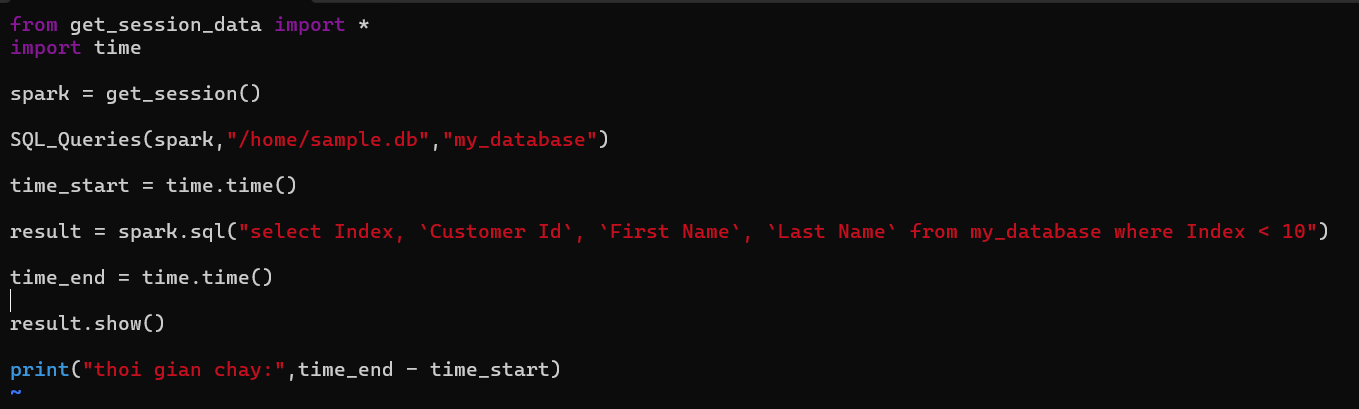


**Ảnh 3: Kết quả truy vấn exam1.py**

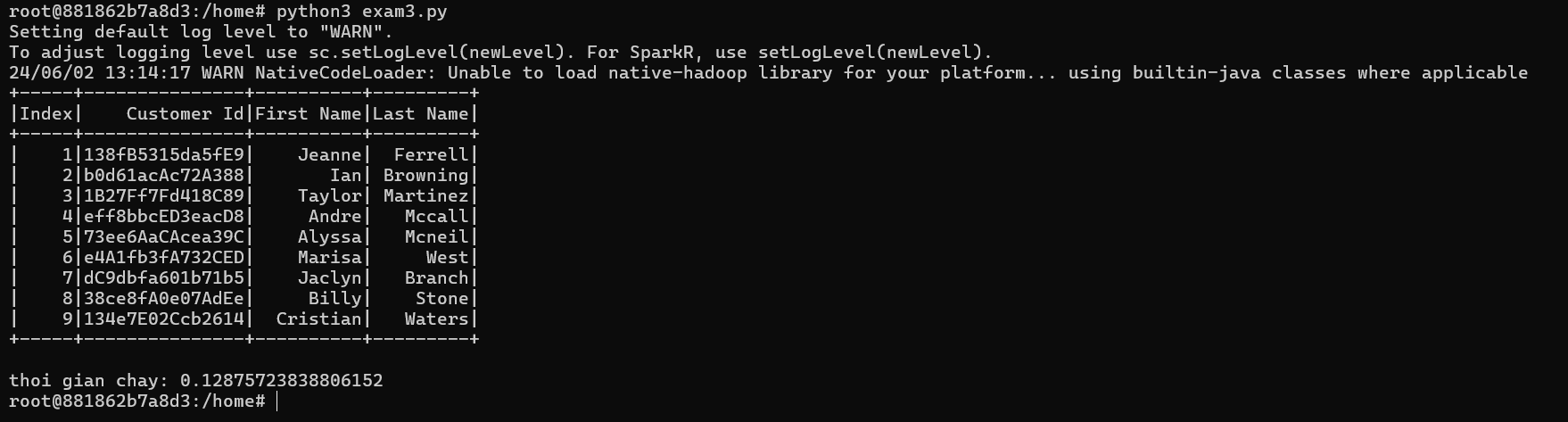


**Ảnh 4: Kết quả truy vấn exam2.py**

-Thử nghiệm với một vài truy vấn khác, tại file **exam3.py** dưới đây, tôi thử nghiệm quá trình truy vấn với việc chỉ chọn 4 cột dữ liệu Index, Customer Id, First Name, Last Name và vẫn với điều kiện Index < 10. Kết quả cho thấy, thời gian chạy chỉ hết khoảng **0.13s**



**Ảnh 5: Nội dung exam3.py**

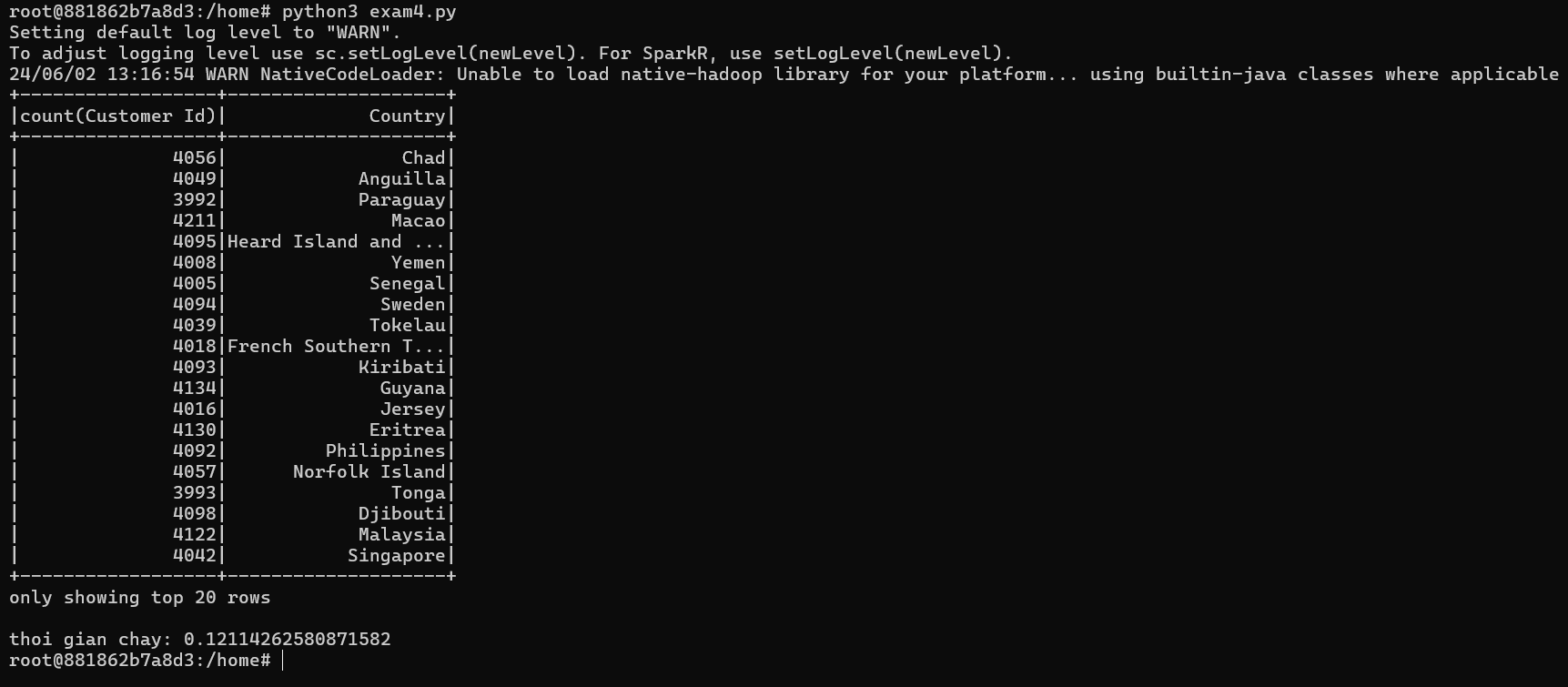


**Ảnh 6: Kết quả truy vấn exam3.py**

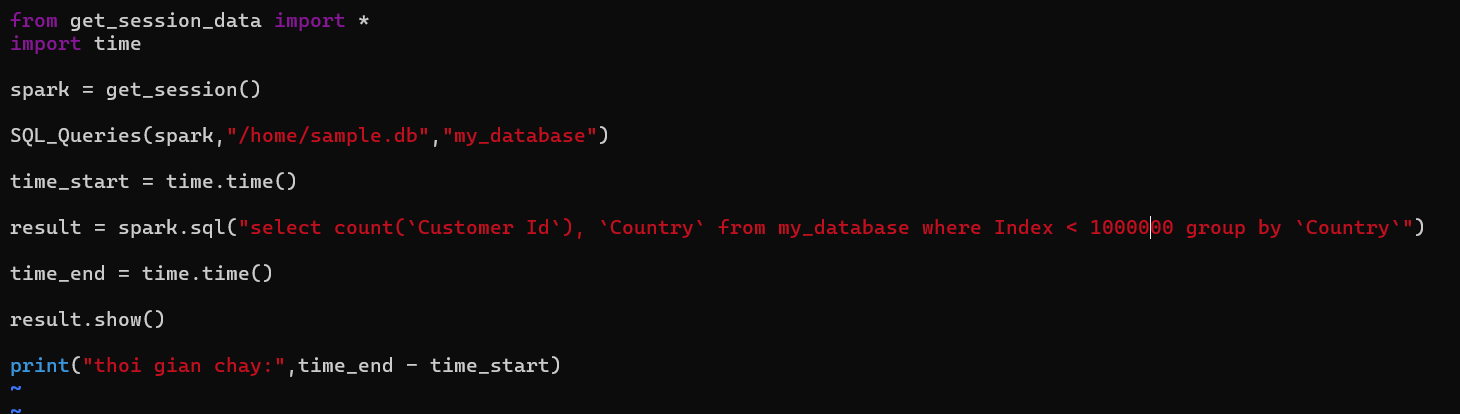
-Thử với một truy vấn liên quan đến **GROUP BY**, tôi truy vấn đếm số lượng người đến từ các quốc gia khác nhau từ cơ sở dữ liệu file **exam4.py** và **exam5.py** với sự khác biệt nằm ở **WHERE Index < 1000000**



**Ảnh 7: Nội dung exam4.py**



**Ảnh 8: Kết quả truy vấn exam4.py**



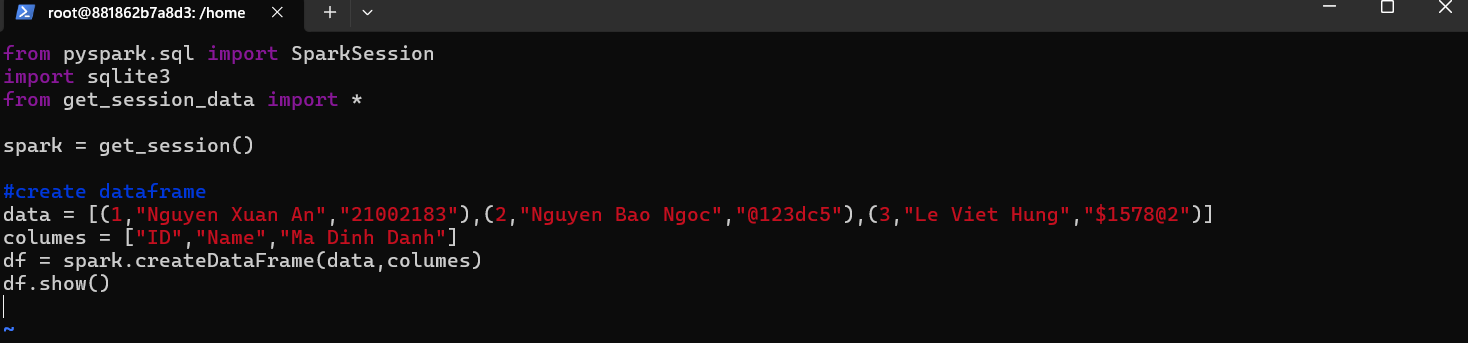
**Ảnh 9: Nội dung exam5.py**



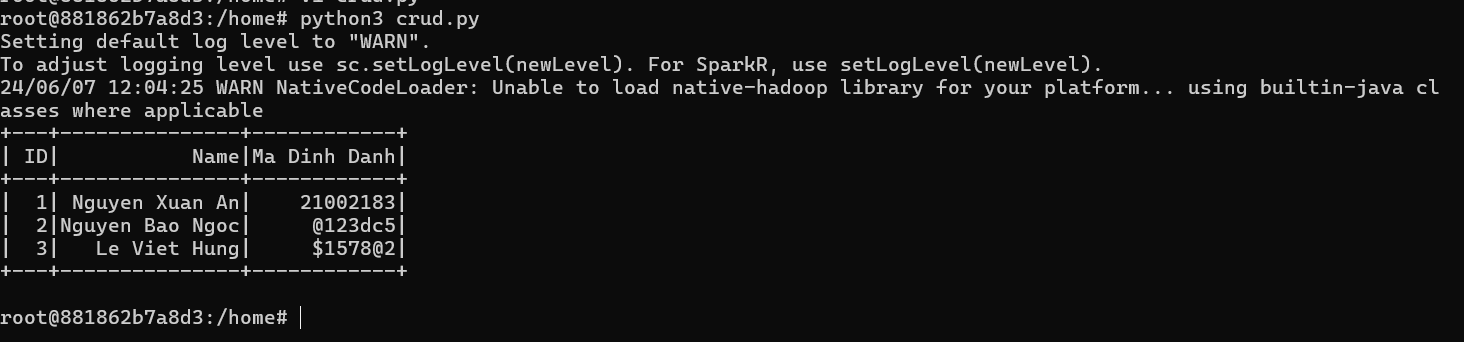
**Ảnh 10: Kết quả truy vấn exam5.py**

**6.2. CRUD**

-Tạo bảng chứa dữ liệu tạm thời với PySpark, chúng ta sử dụng cấu trúc lệnh như sau, kết quả thu được sẽ là

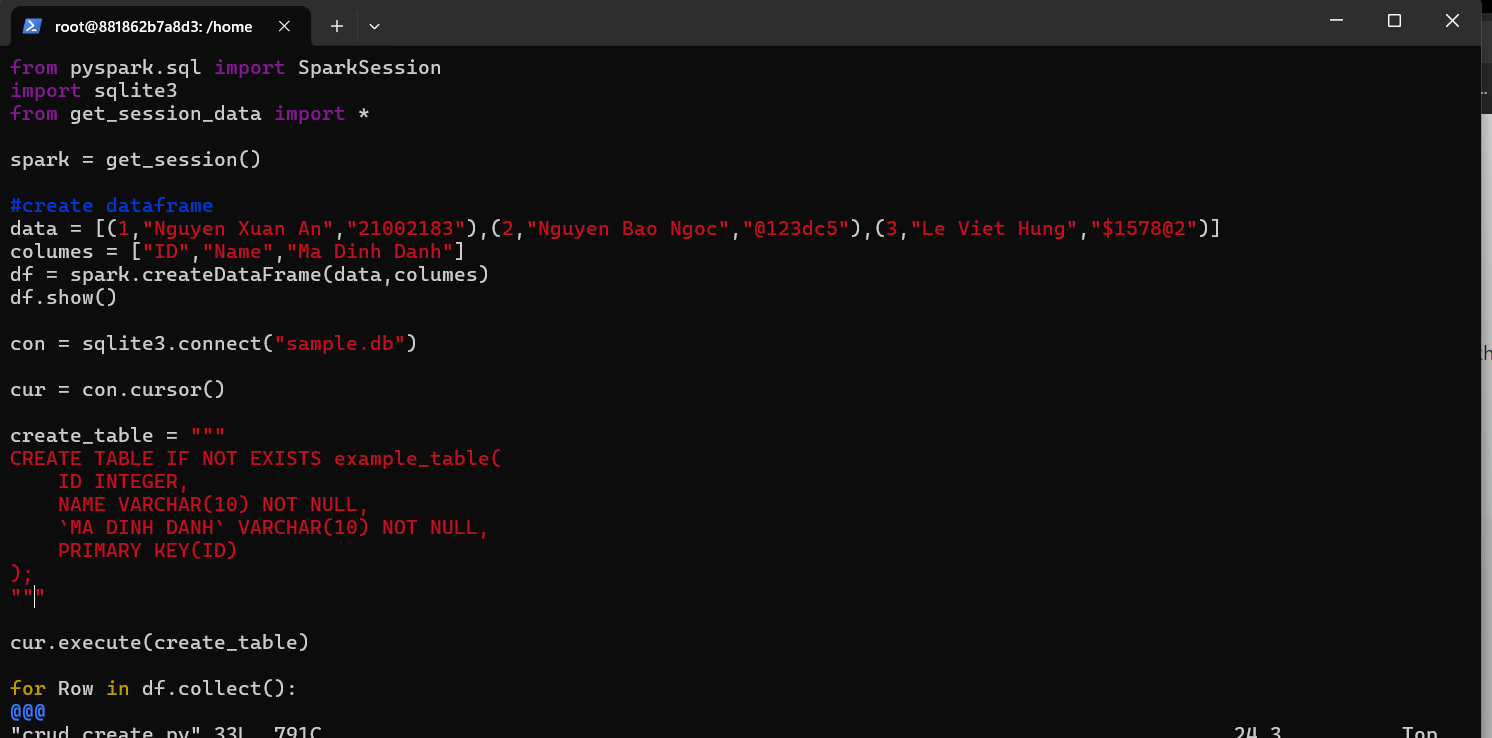


**Ảnh 11: Tạo dữ liệu**



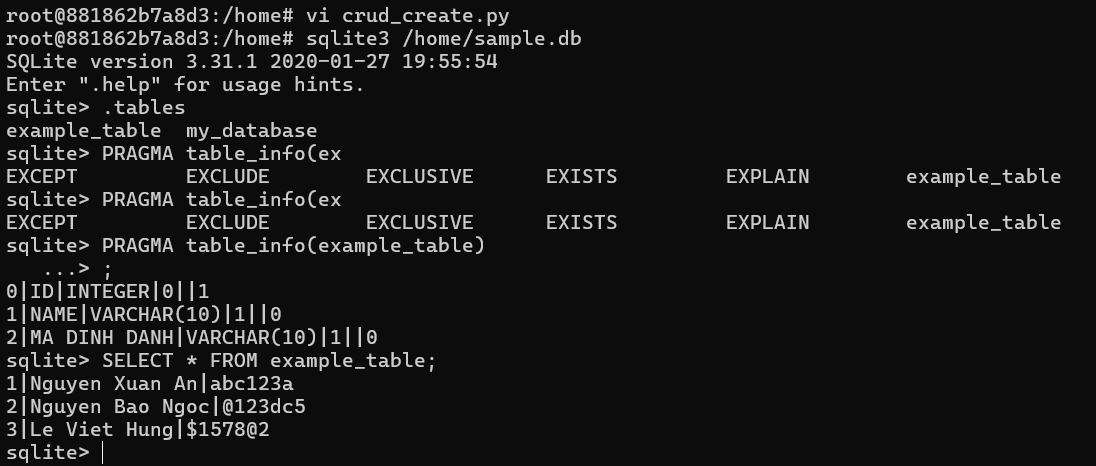
**Ảnh 12: Kết quả crud\_create.py**

-Đây là một bảng dữ liệu tạm thời, bây giờ để có thể lưu trữ bảng dữ liệu này trong cơ sở dữ liệu, ta cần thực hiện một số thao tác, thêm đoạn mã sau vào file code hiện tại. Thực hiện chạy file và kiểm tra bảng đã được tạo trong cơ sở dữ liệu **sample.db** chưa



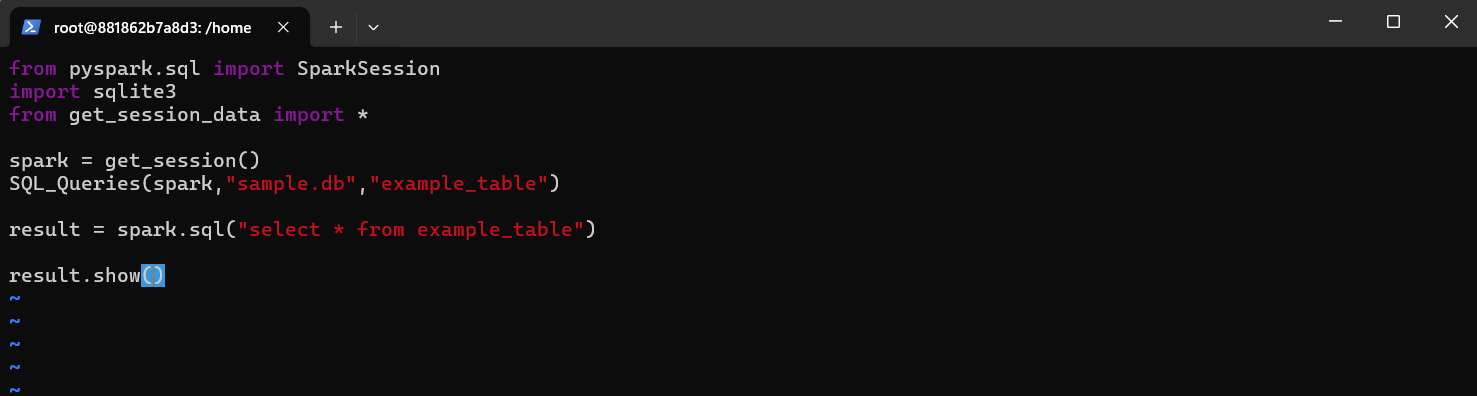
**Ảnh 13: Nội dung crud\_create.py**

-Nhập lênh **.tables** để hiển thị các bảng dữ liệu hiện có. Ta thấy được, bảng **example\_table** đã được tạo trong **sample.db**, sử dụng lệnh **PRAGMA** để kiểm tra thông tin về đặc tính của các cột dữ liệu

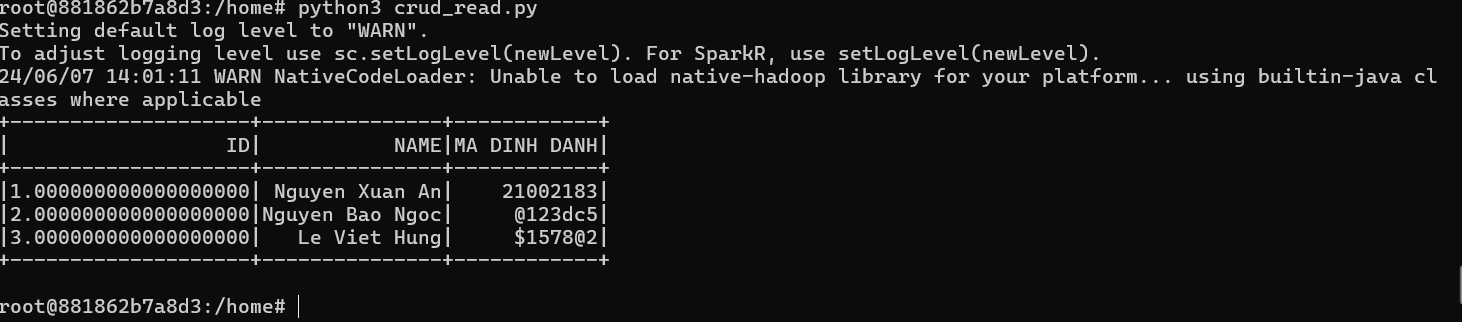


**Ảnh 14: Kiểm tra bảng đã được tạo**

-Ta thử đọc dữ liệu từ bảng vừa tạo

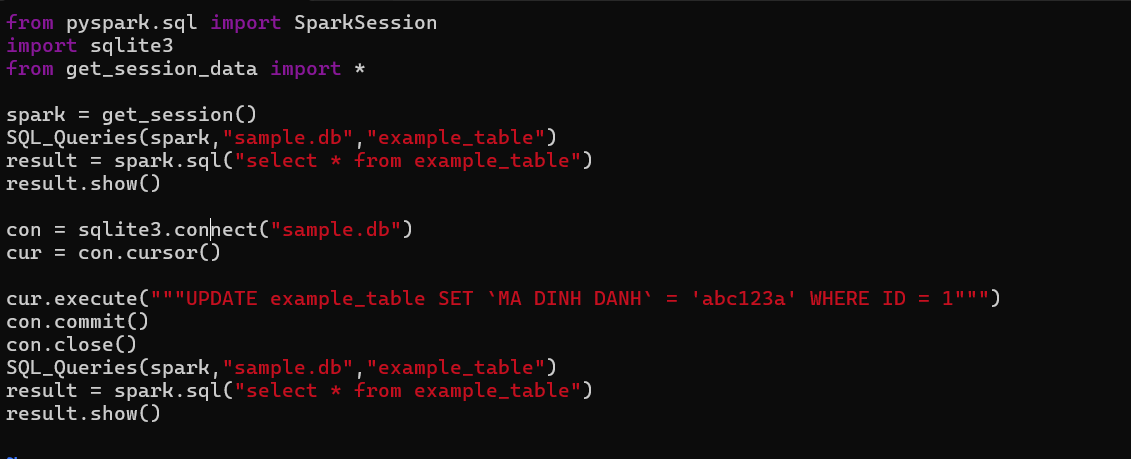


**Ảnh 15: Nội dung crud\_read.py**

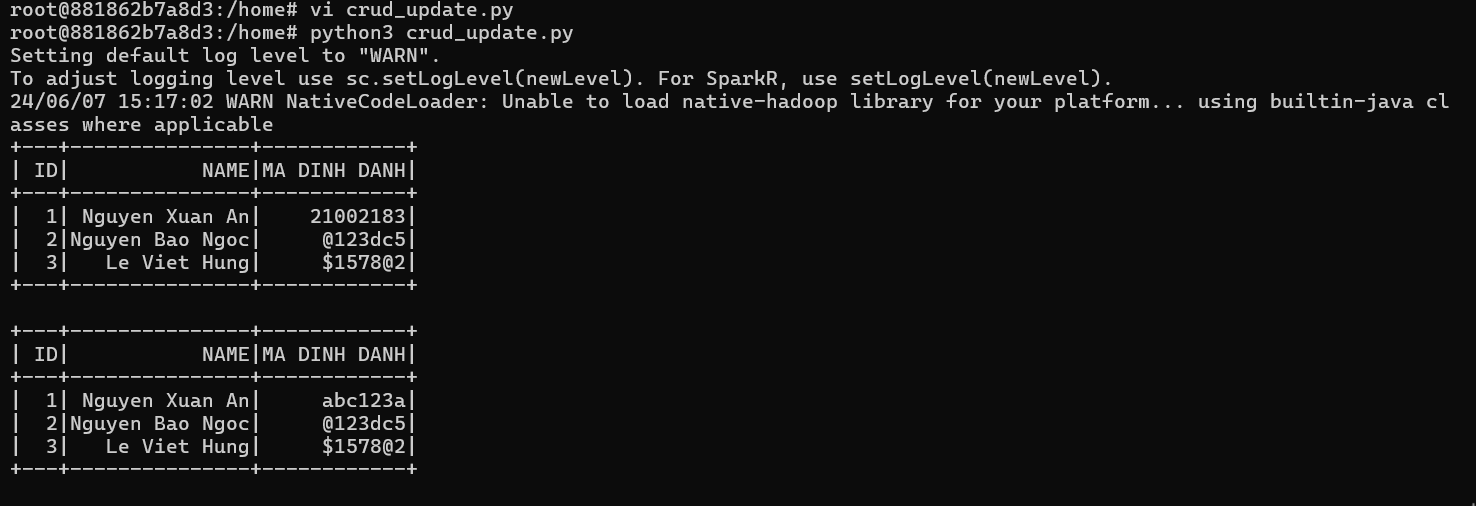


**Ảnh 16: Kết quả crud\_read.py**

-Bây giờ, ta thử thay đổi dữ liệu trong bảng, có thể thấy **`MA DINH DANH`** của **ID = 1** đã được thay đổi

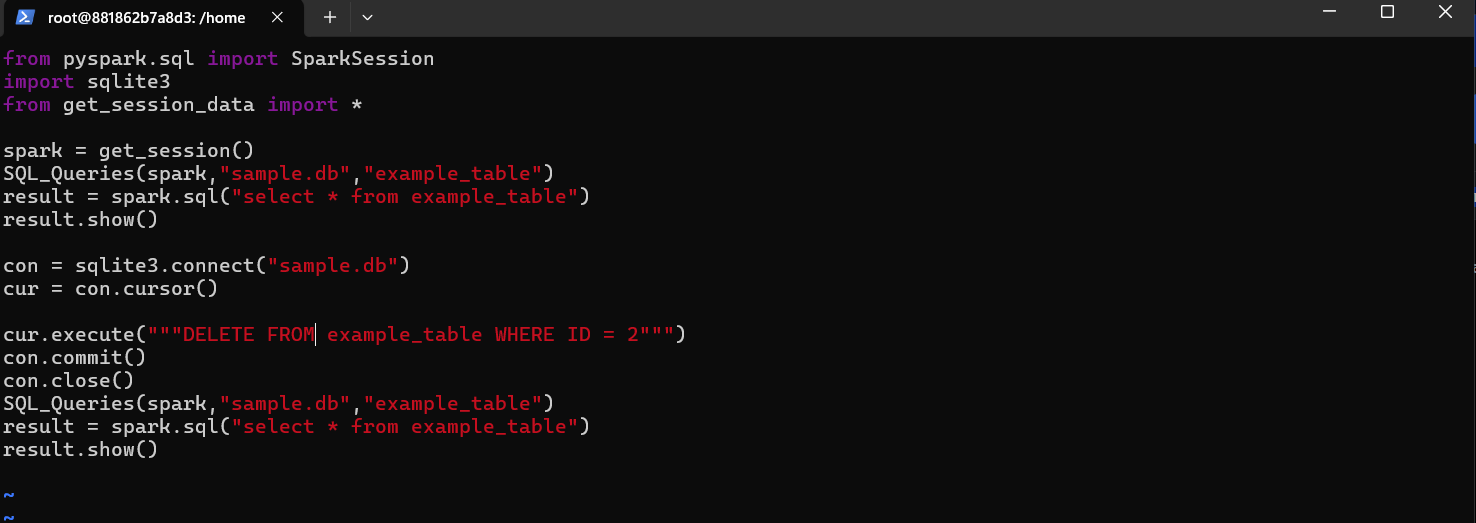


**Ảnh 17: Nội dung crud\_update.py**

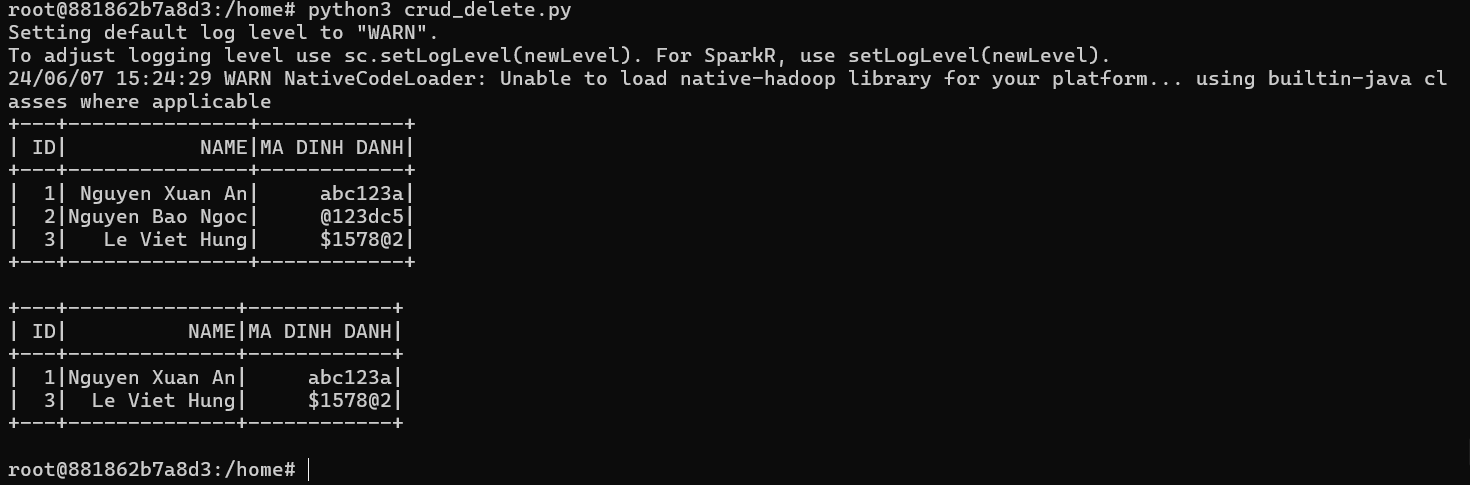


**Ảnh 18: Kết quả crud\_update.py**

-Cuối cùng, ta thử nghiệm xóa hàng **ID = 2** trong bảng



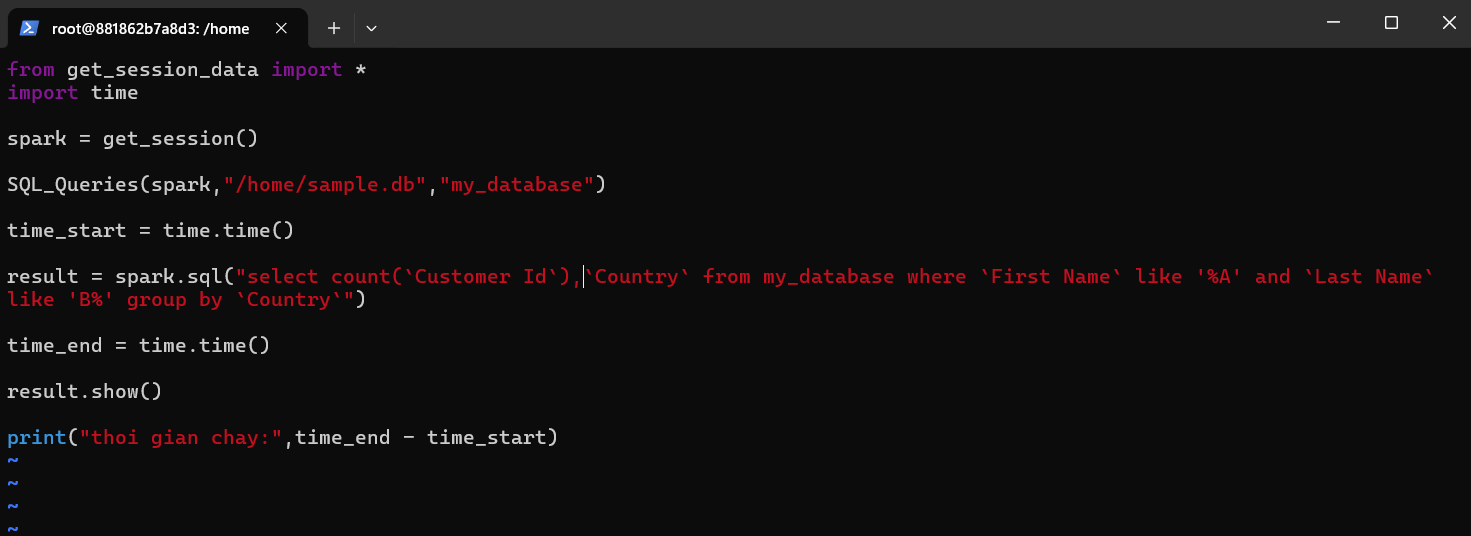
**Ảnh 19: Nội dung crud\_delete.py**



**Ảnh 20: Kết quả crud\_delete.py**

**6.3. Kiểm tra hiệu suất của WHERE với một số câu truy vấn**

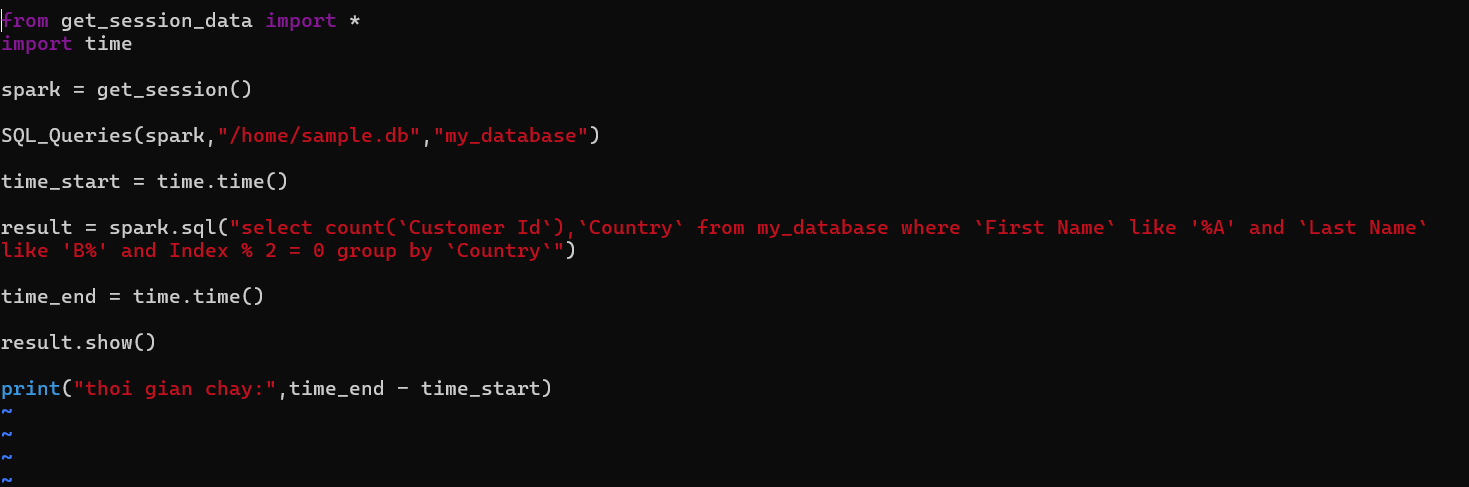
-Trong nội dung file **exam6.py** dưới đây ta cho đếm số "**Customer Id**" theo "**Country**" với các điều kiện "**First Name**" có kết thúc bằng kí tự '**a**' và "**Last Name**" có kết thúc bằng kí tự '**b**'. Cùng với đó, ta tạo một file **exam7.py** có nội dung gần tương tự khi thêm một điều kiện **WHERE** nữa là **Index** phải chia hết cho **2**. Tốc độ truy vấn của 2 file lần lượt là **0.173s** và **0.241s.** Điều này cho thấy điều kiện liên quan đến tính toán ảnh hưởng khá lớn tới hiệu suất truy vấn khi tốc độ truy vấn tăng tới **39%**

****

**Ảnh 21: Nội dung exam6.py**

****

**Ảnh 22: Kết quả exam6.py**

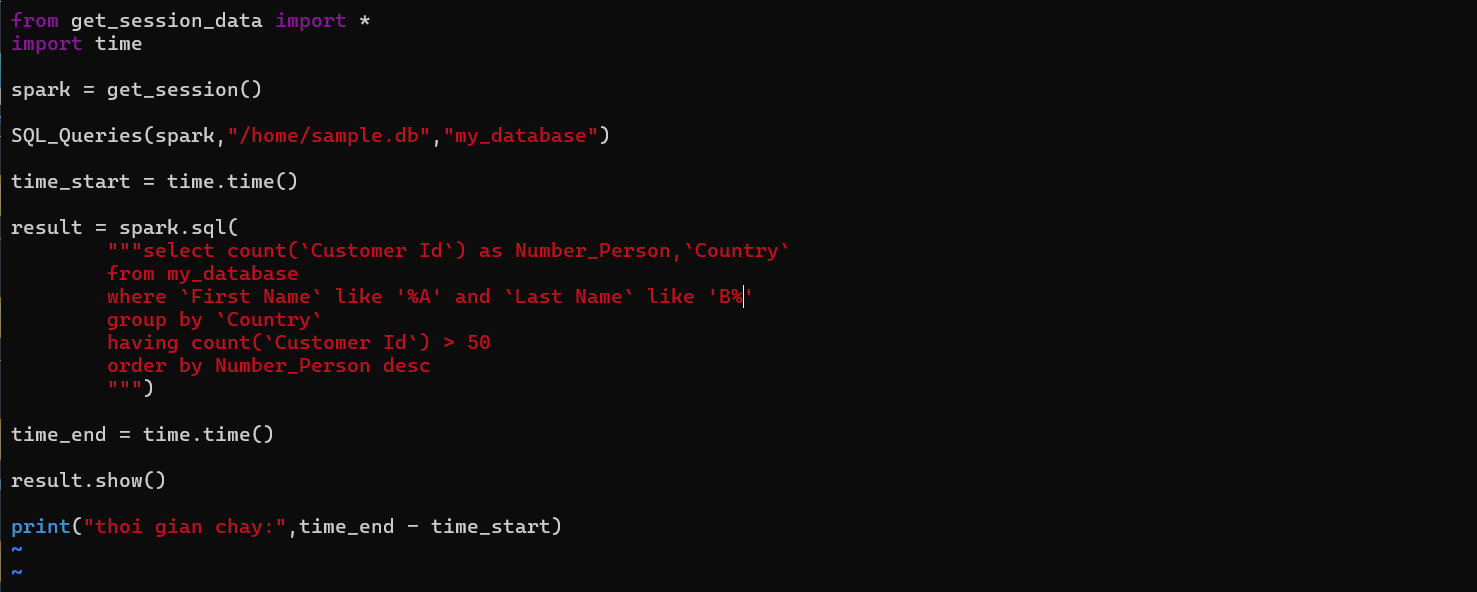
****

**Ảnh 23: Nội dung exam7.py**



**Ảnh 24: Kết quả exam7.py**

-Tiếp theo, ta kết hợp với một số điều kiện khác như **HAVING** và **ORDER BY** trong 2 file **exam8.py, exam9.py** thử so sánh tốc độ truy vấn với một số điều kiện khác biệt, cụ thể ta xây dựng điều kiện **WHERE** của 2 file có chút thay đổi, **exam8.py** sẽ dùng điều kiện liên quan đến kí tự, còn **exam9.py** sẽ dùng điều kiện liên quan đến tính toán

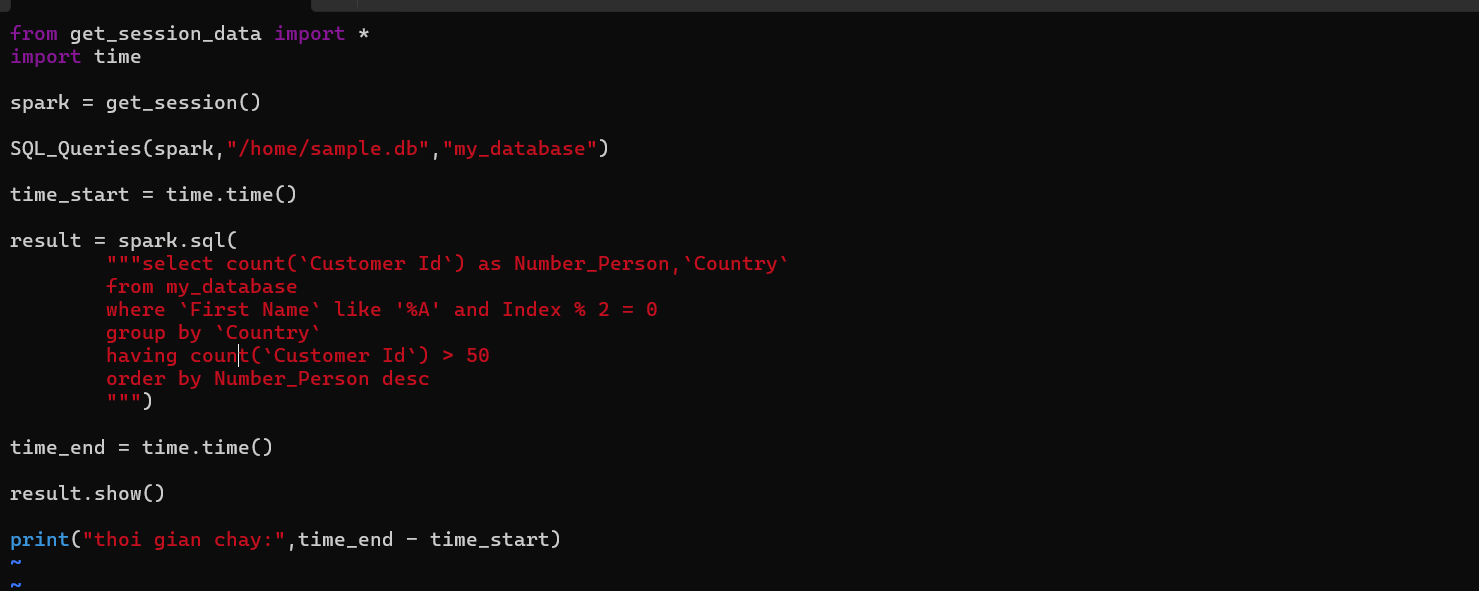
****

**Ảnh 25: Nội dung exam8.py**

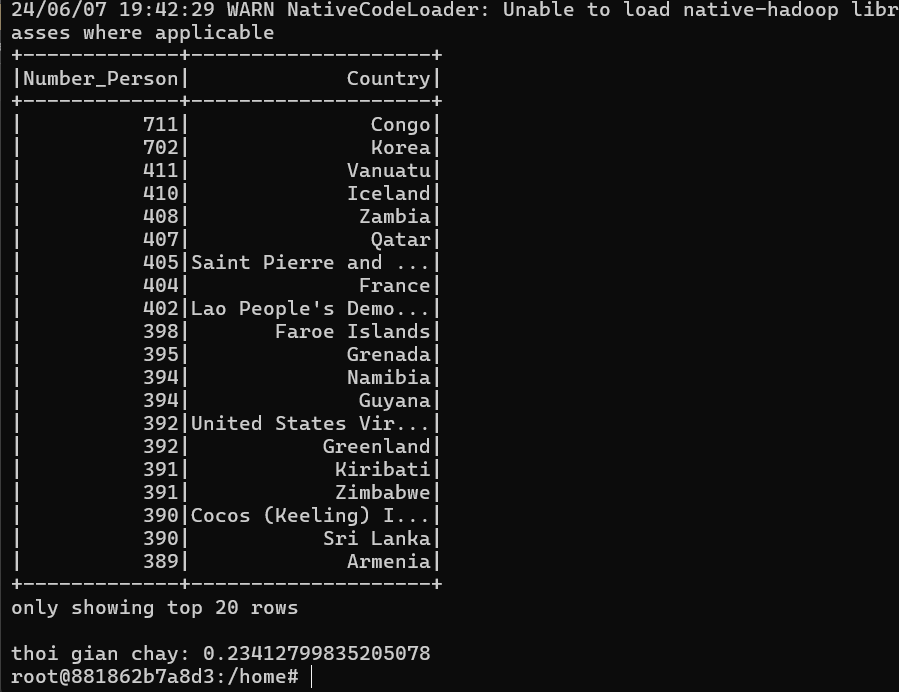
****

**Ảnh 26: Kết quả exam8.py**

-Nội dung của file **exam8.py** sẽ sử dụng các điều kiện **having count(`Customer Id) > 50 order by desc** sắp xếp các giá trị từ lớn đến bé. Còn file **exam9.py** sẽ thay thế điều kiện **where `Last Name` like 'B%'** bằng điều kiện **Index % 2 = 0.** Có thể thấy, khi sử dụng điều kiện liên quan đến tính toán, tốc độ truy vấn tang đáng kể. Với thời gian **0.207s** của **exam8.py** và **0.234s** của **exam9**.py, tốc độ truy vấn tăng khoảng **18%**

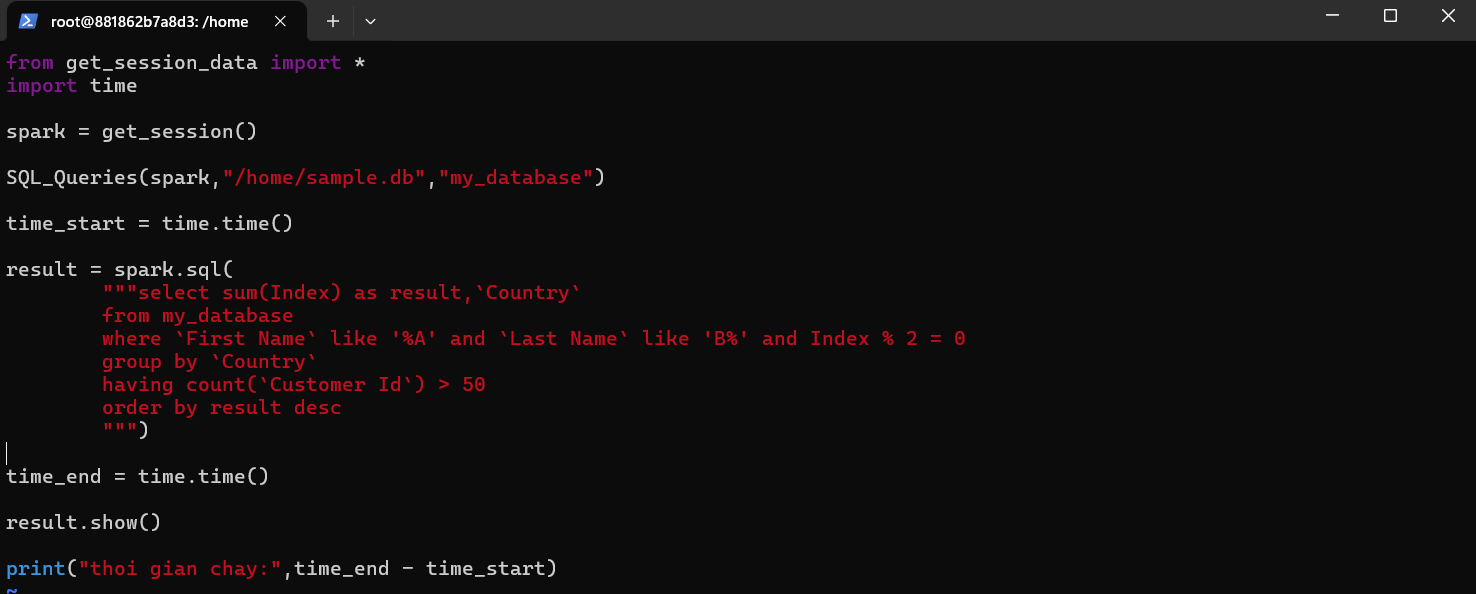
****

**Ảnh 27: Nội dung exam9.py**

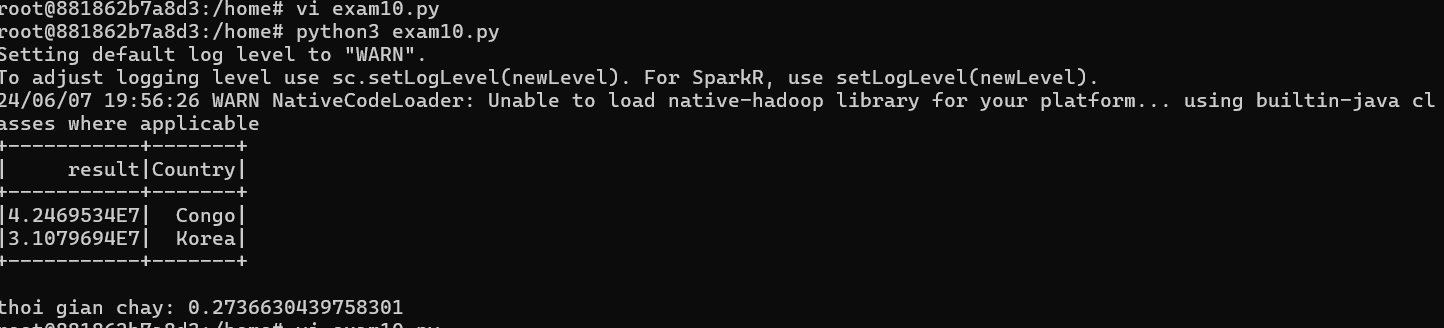
****

**Ảnh 28: Kết quả exam9.py**

**-**Nội dung 2 file **exam10.py** và **exam11.py** được thay đổi để xem nếu một vài điều kiện khác thay đổi, có ảnh hưởng nhiều đến hiệu suất truy vấn khi cùng một điều kiện **WHERE** không. Ở đây, ta lần lượt sử dụng điều kiện **HAVING** đếm số lượng người phù hợp **>50** và **>40**. Khi chạy thử cho thấy kết quả truy vấn trả về lần lượt là **0.273s** và **0.283s**. Sự khác biệt này là không đáng kể, chỉ khoảng **3%**

****

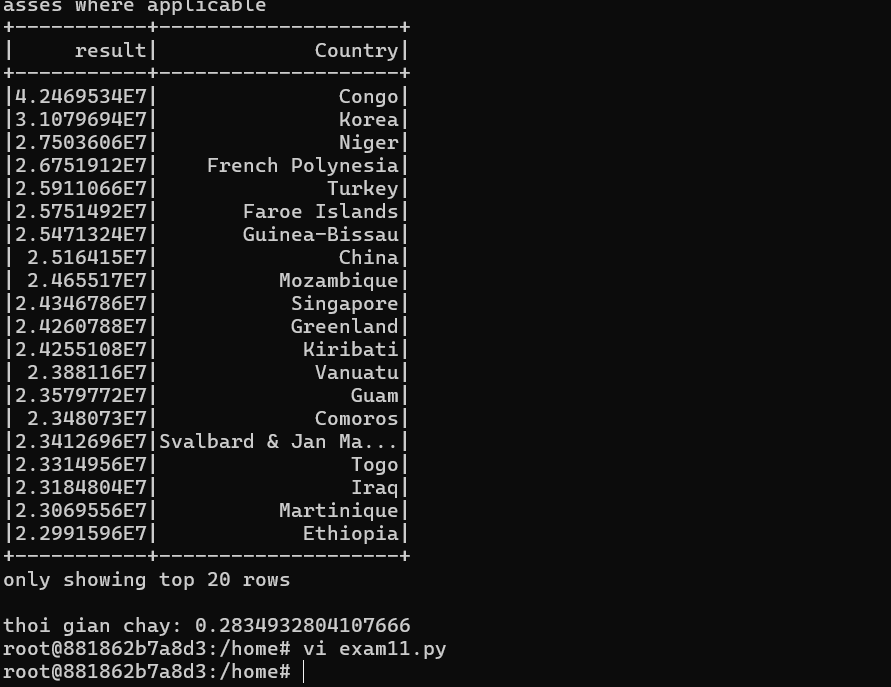
**Ảnh 29: Nội dung exam 10.py**

****

**Ảnh 30: Kết quả exam10.py**



**Ảnh 31: Nội dung exam11.py**



**Ảnh 32: Kết quả exam11.py**

**6.4. Kết luận**

-Càng nhiều điều kiện trong **WHERE** thì càng làm ảnh hưởng tới tốc đô truy vấn dữ liệu.

-Các điều kiện trong **WHERE** có liên quan đến tính toán thường có ảnh hưởng lớn tới tốc độ truy vấn.

-Việc **SELECT** cũng làm ảnh hưởng tới tốc độ truy vấn khi có **WHERE**, vậy nên cần lựa chọn các dữ liệu cần thiết để truy vấn.

-Các điều kiện tính toán như **COUNT, SUM** với **GROUP BY** cũng gây ra ảnh hưởng khi khi truy vấn với **WHERE**, ta thấy khi dùng **SUM,** tốc độ truy vấn tăng đáng kể.

-Điều kiện **HAVING, ORDER BY** hoạt động ảnh hưởng khá nhỏ tới hiệu suất khi sử dụng **WHERE**