**Nguyễn hữu thắng 21002238**

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

1. **Thiết lập Docker**

* Tải docker desktop từ trang web chính thức cho windown
* Tạo một thư mục có tên là prj
* Open thư mục bằng vscode
* Tạo một Dockerfile không có đuôi tệp
* Ghi vào file câu lệnh sau: FROM ubuntu:22.04
* Ý nghĩa câu lệnh trên là xây dựng docker image dựa trên docker image ubuntu:22.04
* Ta gõ tiếp: WORKDIR /app

Câu lệnh này giúp xác định thư mục làm việc chính.

* Tiếp theo. Mở một terminal gõ câu lênh: docker build -t myprj .

Trong đó docker là câu lệnh giúp gọi ứng dụng có tên là docker có trên đường dẫn hệ thống ra

Các chữ tiếp theo là truyền đối số dòng lệnh giúp ứng dụng docker đọc được

build có nghĩa là ta đang xây dựng docker image từ dockerfile. -t là cờ giúp đặt tên cho dockerfile là myprj. Dấu chấm có nghĩa là tìm dockerfile tại thư mục hiện tại.

* Khi chạy xong câu lệnh trên. Mở docker desktop ra. Đã có một docker image có tên myprj được tạo ra
* Ta tiếp tục gõ trên terminal câu lệnh: docker run -it myprj bash
* Khi đó. Một docker container được tạo ra từ docker image myprj. Và khi này, ta đang ở bên trong docker container.
* Ta gõ câu lệnh: apt update để cập nhật hệ thống.
* Câu lệnh đã chạy thành công. Khi này ta gõ dòng tiếp theo vào dockerfile: RUN apt update

1. **Cài đặt Spark**

* Apache Spark cần môi trường thực thi. Vậy trước tiên ta cần cài jdk: apt install openjdk-21-jdk
* Câu lệnh trên hoạt động bình thường. khi này ta ghi câu lệnh này vào dockerfile: RUN apt install openjdk-21-jdk
* Ta thử build image từ dockerfile. Ta gặp một lỗi là:

ERROR: failed to solve: process "/bin/sh -c apt install openjdk-21-jdk" did not complete successfully: exit code: 1

* Khi ta đứng trong container thì ta có thể nhấn yes hay no. nhưng khi ta build image thì không ai nhấn cho ta cả.
* Cách sử lý là thêm -y vào trước. câu lệnh sẽ là: RUN apt -y install openjdk-21-jdk
* Khi đó. Build lại image sẽ hoạt động bình thường mà không gặp lỗi gì.
* Ta tiếp tục với một số câu lệnh để Cài đặt Apache Spark trong vùng chứa Docker.
* RUN apt -y install wget
* Câu lệnh trên giúp tải wget từ kho chứa apt về máy. wget dùng để tải file từ internet.
* Tiếp theo:   
  RUN wget https://dlcdn.apache.org/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz

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

RUN rm spark-3.5.1-bin-hadoop3.tgz

Các câu lệnh này giúp tải spark phiên bản 3.5.1. giải nén. Và xóa file nén sau khi đã giải nén

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

Di chuyển và đổi tên thư mục vừa giải nén thành thư mục tại vị trí /opt/spark

* Tiếp theo:

ENV SPARK\_HOME=/opt/spark

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

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

Đặt biến môi trường SPARK\_HOME, thêm thư muc bin và sbin của spark vào đường dẫn hệ thống

* Ta build lại image và chạy  container.
* Khi đã ở bên trong container. Ta gõ: spark-shell
* Ta được:



* Như vậy Spark đã được cài đặt và hoạt động chính xác.

1. **Thiết lập cơ sở dữ liệu:**

* ta tiếp tục ghi vào dockerfile những câu lệnh sau:

RUN apt -y install sqlite3

RUN apt -y install python3

RUN RUN apt install nano

* tải sqlite3, python3, và cài trình soạn thảo văn bản nano để phục vụ sau này
* ta build lại image từ dockerfile đã thêm. Và chạy lại Docker container.
* Ta gõ: nano mot\_trieu\_hang.py
* Sau đó gõ đoạn code này vào:

import sqlite3

def create\_and\_populate\_db(db\_name, num\_rows):

    conn = sqlite3.connect(db\_name)

    c = conn.cursor()

    c.execute('''CREATE TABLE IF NOT EXISTS mydata

                (id INTEGER PRIMARY KEY, ho TEXT, ten\_dem TEXT, ten TEXT)''')

    for i in range(num\_rows):

        c.execute("INSERT INTO mydata (ho, ten\_dem, ten) VALUES (?, ?, ?)", ('Nguyen\_'+str(i), 'Huu\_'+str(i), 'Thang\_'+str(i)))

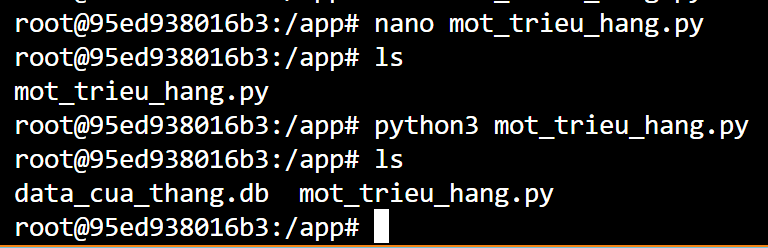
    conn.commit()

    conn.close()

create\_and\_populate\_db('data\_cua\_thang.db', 1000000)

sau đó nhấn ctrl X để lưu.

Tiếp theo gõ: python3 mot\_trieu\_hang.py



* Sau khi chạy xong đoạn mã python trên. Ta đã điền được vào cơ sở dữ liệu data\_cua\_thang.db

Một triệu hàng.

1. **Cài đặt các phụ thuộc**

* Ta tiếp tục thêm các câu lệnh giúp cài đặt các phụ thuộc:

RUN apt -y install python3-pip

RUN pip install pyspark

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

* Đầu tiên ta cài pip cho python3. Sau đó tải pyspark về. và cài JDBC cho sqlite, để giúp kết nối và thực hiện truy vấn từ Spark đến cơ sở dữ liệu.

1. **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: RUN mv sqlite-jdbc-3.34.0.jar /opt/spark/jars
* Ta dùng trình soạn thảo văn bản nano. tạo lại file data\_cua\_thang.db như ở phần 3. Tạo file doc\_10\_dong.py và ghi đoạn code này vào:

from pyspark.sql import SparkSession

spark = SparkSession.builder \

    .appName("Read SQLite with PySpark") \

    .config("spark.driver.extraClassPath", "/path/to/sqlite-jdbc-3.34.0.jar") \

    .getOrCreate()

db\_path = "jdbc:sqlite:data\_cua\_thang.db"

df = spark.read.format("jdbc") \

    .option("url", db\_path) \

    .option("dbtable", "mydata") \

    .option("driver", "org.sqlite.JDBC") \

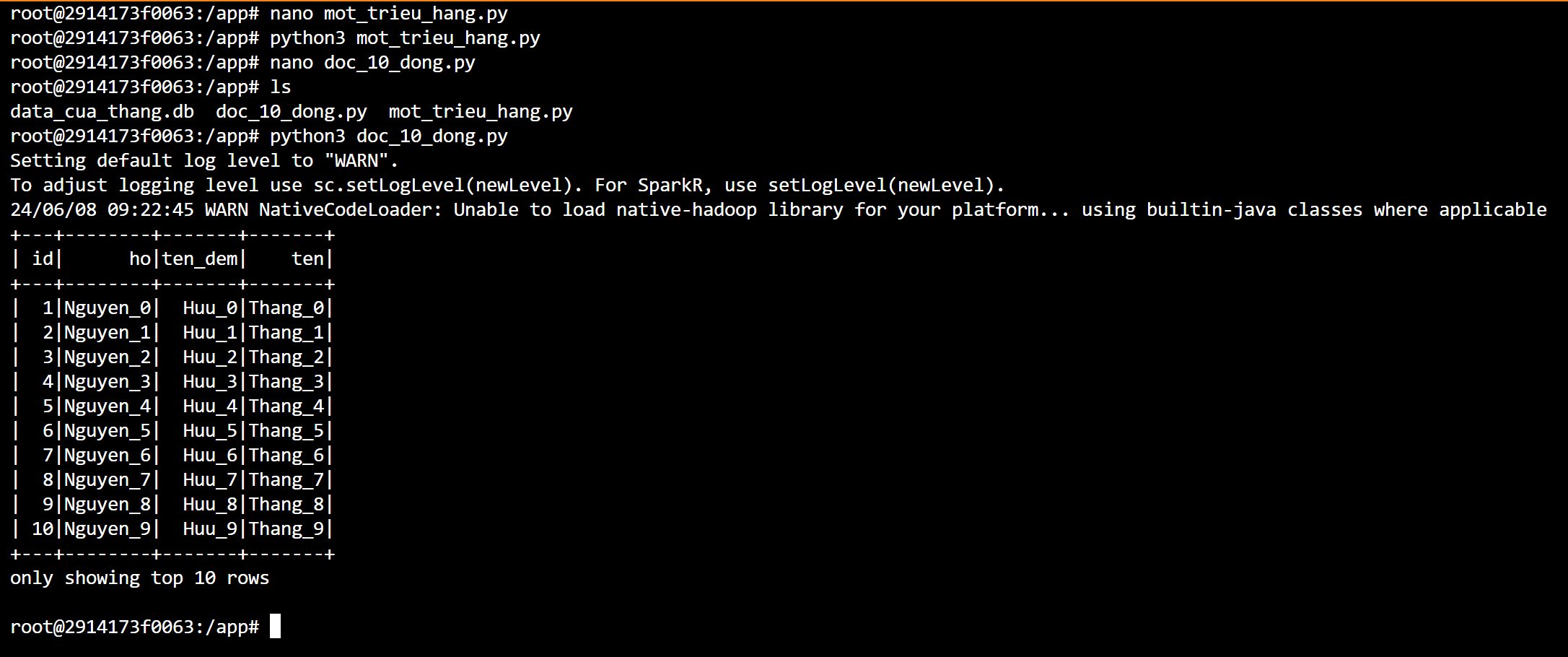
    .load()

df.show(10)

spark.stop()

sau đó gõ: python3 doc\_10\_dong.py

ta được kết quả



Như vậy SparkSQL có thể truy cập và truy vấn cơ sở dữ liệu

1. **Chạy truy vấn**

* Tôi sửa lại file mot\_trieu\_hang.py. thêm yếu tố random vào cho việc truy vấn trở nên khách qua. Tôi sửa như sau:

import sqlite3

import random

def create\_and\_populate\_db(db\_name, num\_rows):

    conn = sqlite3.connect(db\_name)

    c = conn.cursor()

    c.execute('''CREATE TABLE IF NOT EXISTS mydata

                (id INTEGER PRIMARY KEY, ho TEXT, ten\_dem TEXT, ten TEXT)''')

    for i in range(num\_rows):

        c.execute("INSERT INTO mydata (ho, ten\_dem, ten) VALUES (?, ?, ?)", ('Nguyen\_'+str(random.randint(0, 10)), 'Huu\_'+str(random.randint(0, 10)), 'Thang\_'+str(random.randint(0, 10))))

    conn.commit()

    conn.close()

create\_and\_populate\_db('data\_cua\_thang.db', 1000000)

và chạy đoạn code này: python3 mot\_trieu\_hang.py

* Tiếp tục dùng trình soạn thảo nano. Tạo file truyvan.py

from pyspark.sql import SparkSession

# Tạo SparkSession

spark = SparkSession.builder \

    .appName("SQLite with SparkSQL") \

    .config("spark.driver.extraClassPath", "/opt/spark/jars/sqlite-jdbc-3.34.0.jar") \

    .getOrCreate()

# Đường dẫn tới file SQLite

db\_path = "jdbc:sqlite:/app/data\_cua\_thang.db"

# Đọc dữ liệu từ SQLite

df = spark.read.format("jdbc") \

    .option("url", db\_path) \

    .option("dbtable", "mydata") \

    .option("driver", "org.sqlite.JDBC") \

    .load()

# Đăng ký DataFrame dưới dạng một bảng tạm thời

df.createOrReplaceTempView("mydata")

# Truy vấn 1: Lấy 10 dòng đầu tiên

print("Truy vấn 1: Lấy 10 dòng đầu tiên")

spark.sql("SELECT \* FROM mydata LIMIT 10").show()

# Truy vấn 2: Đếm số lượng dòng trong bảng

print("Truy vấn 2: Đếm số lượng dòng trong bảng")

spark.sql("SELECT COUNT(\*) FROM mydata").show()

# Truy vấn 3: Lấy các dòng có tên là 'Thang\_1'

print("Truy vấn 3: Lấy các dòng có tên là 'Thang\_1'")

spark.sql("SELECT \* FROM mydata WHERE ten = 'Thang\_1'").show()

# Truy vấn 4: Đếm số lượng họ 'Nguyen\_1'

print("Truy vấn 4: Đếm số lượng họ 'Nguyen\_1'")

spark.sql("SELECT COUNT(\*) FROM mydata WHERE ho = 'Nguyen\_1'").show()

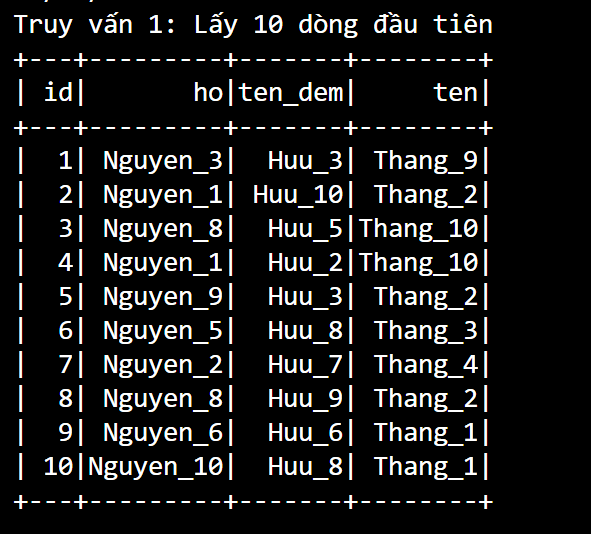
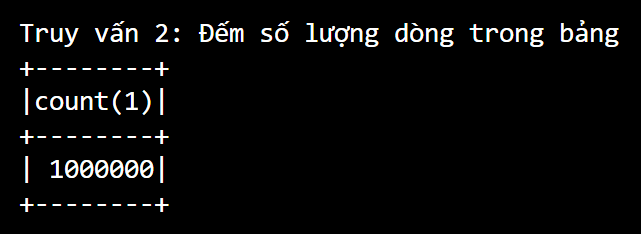
# Truy vấn 5: Lấy 10 họ đầu tiên và đếm số lượng tương ứng

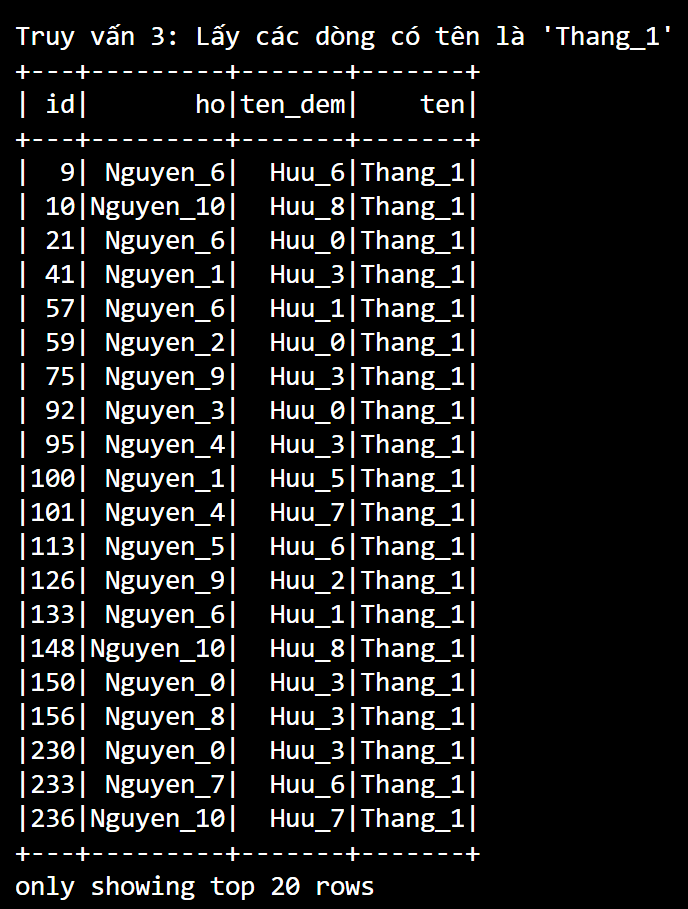
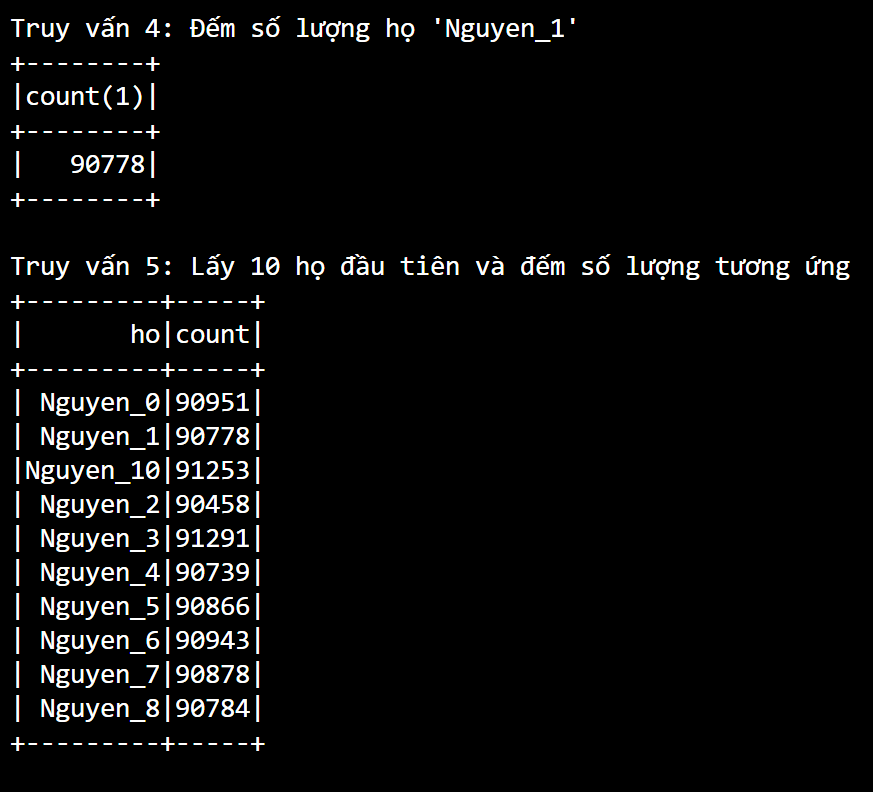
print("Truy vấn 5: Lấy 10 họ đầu tiên và đếm số lượng tương ứng")

spark.sql("SELECT ho, COUNT(\*) as count FROM mydata GROUP BY ho ORDER BY ho LIMIT 10").show()

# Kết thúc SparkSession

spark.stop()

kết quả:  
 

* Các thao tác crud:

Để cho dễ quan sát. Tạo một cơ sở dữ liệu mới có một bảng dữ liệu có 10 dòng

Ta dùng nano. Tạo một file muoi\_dong.py. sau đó thực thi file này.

import sqlite3

import random

def create\_and\_populate\_db(db\_name, num\_rows):

    conn = sqlite3.connect(db\_name)

    c = conn.cursor()

    c.execute('''CREATE TABLE IF NOT EXISTS mydata

                (id INTEGER PRIMARY KEY, ho TEXT, ten\_dem TEXT, ten TEXT)''')

    for i in range(num\_rows):

        c.execute("INSERT INTO mydata (ho, ten\_dem, ten) VALUES (?, ?, ?)", ('Nguyen\_'+str(random.randint(0, 10)), 'Huu\_'+str(random.randint(0, 10)), 'Thang\_'+str(random.randint(0, 10))))

    conn.commit()

    conn.close()

create\_and\_populate\_db('data\_cua\_thang.db', 10)

* Chạy tiếp đoạn code sau để xem dữ liệu trong bảng:

from pyspark.sql import SparkSession

spark = SparkSession.builder \

    .appName("SQLite with SparkSQL") \

    .config("spark.driver.extraClassPath", "/opt/spark/jars/sqlite-jdbc-3.34.0.jar") \

    .getOrCreate()

db\_path = "jdbc:sqlite:/app/data\_cua\_thang.db"

df = spark.read.format("jdbc") \

    .option("url", db\_path) \

    .option("dbtable", "mydata") \

    .option("driver", "org.sqlite.JDBC") \

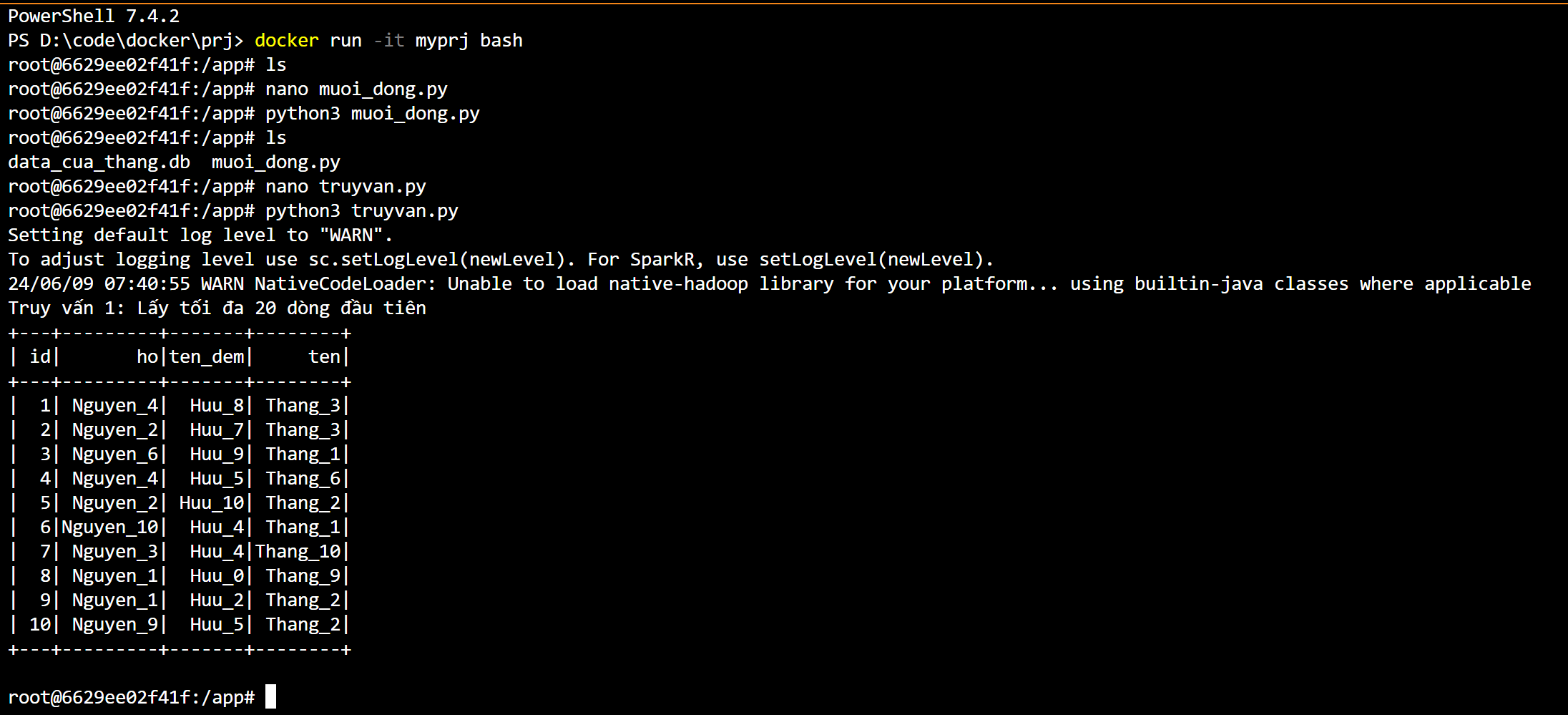
    .load()

df.createOrReplaceTempView("mydata")

print("Truy vấn 1: Lấy tối đa 20 dòng đầu tiên")

spark.sql("SELECT \* FROM mydata LIMIT 20").show()

spark.stop()

* Ta thu được kết quả:  
  
* Tiếp theo ta tạo một file crud.py có nội dung như sau và chạy:
* from pyspark.sql import SparkSession
* from pyspark.sql.functions import lit, when
* import os

spark = SparkSession.builder \

    .appName("Spark crud") \

    .getOrCreate()

db\_path = "jdbc:sqlite:/app/data\_cua\_thang.db"

df = spark.read.format("jdbc") \

    .option("url", db\_path) \

    .option("dbtable", "mydata") \

    .option("driver", "org.sqlite.JDBC") \

    .load()

df.show(10)

new\_data = [(11, "Nguyen\_11", "Huu\_11", "Thang\_11"),

            (12, "Nguyen\_12", "Huu\_12", "Thang\_12")]

new\_df = spark.createDataFrame(new\_data, ["id", "ho", "ten\_dem", "ten"])

new\_df.show()

df = df.union(new\_df)

df.createOrReplaceTempView("mydata")

print("Truy vấn 2: Đếm số lượng dòng trong bảng")

spark.sql("SELECT COUNT(\*) FROM mydata").show()

print("READ: Hiển thị dòng với id = 11")

spark.sql("SELECT \* FROM mydata WHERE id = 11").show()

print("UPDATE: Cập nhật dữ liệu")

df = df.withColumn("ten", when(df.id == 1, "Updated\_Thang").otherwise(df.ten))

df.createOrReplaceTempView("mydata")

spark.sql("SELECT \* FROM mydata WHERE id = 1").show()

print("DELETE: Xóa dòng với id = 3")

df = df.where(df.id != 3)

df.createOrReplaceTempView("mydata")

spark.sql("SELECT \* FROM mydata WHERE id != 3").show()

df.show(10)

print("Truy vấn 2: Đếm số lượng dòng trong bảng")

spark.sql("SELECT COUNT(\*) FROM mydata").show()

df.write.format("jdbc") \

    .option("url", "jdbc:sqlite:/app/x.db?busy\_timeout=5000") \

    .option("dbtable", "mydata") \

    .option("driver", "org.sqlite.JDBC") \

    .mode("overwrite") \

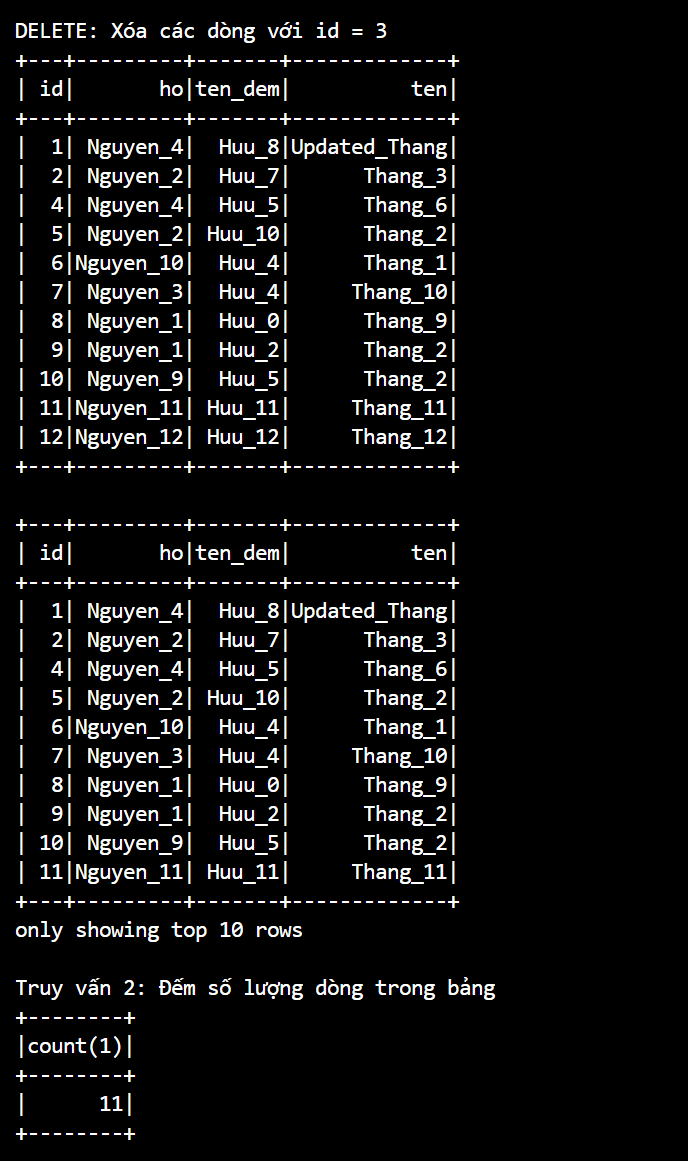
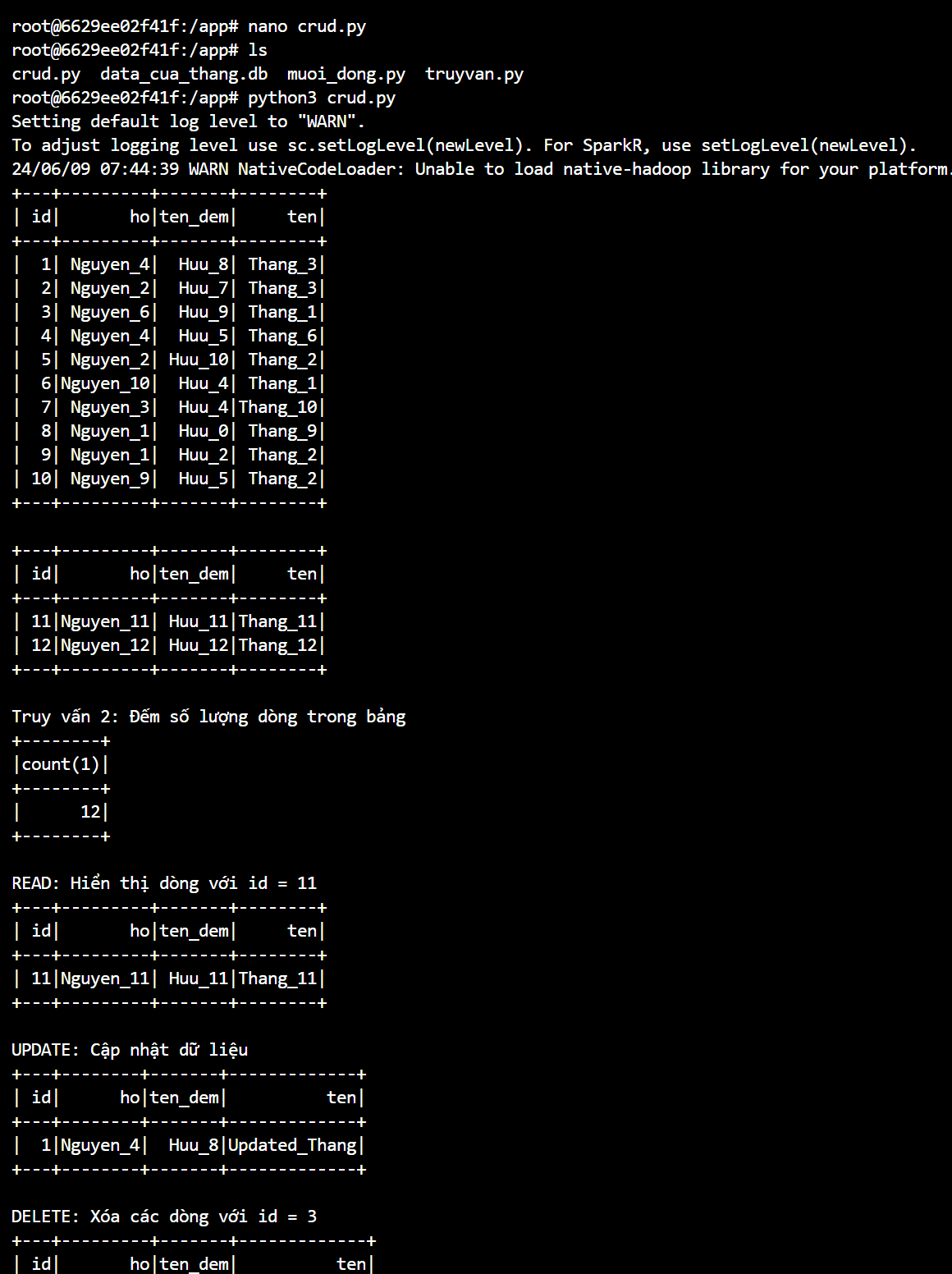
    .save()

spark.stop()

os.system('rm /app/data\_cua\_thang.db')

os.system('mv /app/x.db /app/data\_cua\_thang.db')

việc ghi file đè lên file cũ em gặp lỗi. vậy em giải quyết bằng cách là ghi sang file mới. xong xóa file cũ đi và đổi tên.

* Ta được kết quả:  
  
* Vậy là em đã hòa thành việc thao tác CRUD (Tạo, Đọc, Cập nhật, Xóa). Thông qua thông qua Spark
* mệnh đề WHERE ảnh hưởng đến hiệu suất của một số truy vấn ta tạo một file python where.py với nội dung như sau:

from pyspark.sql import SparkSession

import time

# Khởi tạo SparkSession

spark = SparkSession.builder \

    .appName("SQLite with SparkSQL") \

    .config("spark.driver.extraClassPath", "/opt/spark/jars/sqlite-jdbc-3.34.0.jar") \

    .getOrCreate()

db\_path = "jdbc:sqlite:/app/data\_cua\_thang.db"

# Tải dữ liệu từ cơ sở dữ liệu SQLite

df = spark.read.format("jdbc") \

    .option("url", db\_path) \

    .option("dbtable", "mydata") \

    .option("driver", "org.sqlite.JDBC") \

    .load()

df.createOrReplaceTempView("mydata")

# Hàm thực hiện và tính thời gian truy vấn

def execute\_query(query):

    start\_time = time.time()

    result = spark.sql(query).collect()

    end\_time = time.time()

    print(f"Query: {query}")

    print(f"Execution Time: {end\_time - start\_time} seconds")

    print(f"Result Count: {len(result)}\n")

# Truy vấn 1: Truy vấn cơ bản không có WHERE

query1 = "SELECT \* FROM mydata"

execute\_query(query1)

# Truy vấn 2: Truy vấn với mệnh đề WHERE đơn giản

query2 = "SELECT \* FROM mydata WHERE ho = 'Nguyen\_9'"

execute\_query(query2)

# Truy vấn 3: Truy vấn với mệnh đề WHERE phức tạp hơn

query3 = "SELECT \* FROM mydata WHERE ho = 'Nguyen\_9' AND ten = 'Thang\_9'"

execute\_query(query3)

# Truy vấn 4: Sử dụng WHERE trên cột ten

query4 = "SELECT \* FROM mydata WHERE ten = 'Thang\_3'"

execute\_query(query4)

# Truy vấn 5: Sử dụng WHERE trên các cột ho và ten\_dem

query5 = "SELECT \* FROM mydata WHERE ho = 'Nguyen\_5' AND ten\_dem = 'Huu\_7'"

execute\_query(query5)

# Truy vấn 6: Sử dụng WHERE trên cột ho hoặc ten

query6 = "SELECT \* FROM mydata WHERE ho = 'Nguyen\_3' OR ten = 'Thang\_2'"

execute\_query(query6)

# Truy vấn 7: Sử dụng WHERE phức tạp với AND và OR

query7 = "SELECT \* FROM mydata WHERE (ho = 'Nguyen\_4' AND ten\_dem = 'Huu\_9') OR ten = 'Thang\_6'"

execute\_query(query7)

# Truy vấn 8: Sử dụng WHERE với điều kiện LIKE

query8 = "SELECT \* FROM mydata WHERE ten LIKE 'Thang\_%'"

execute\_query(query8)

# Truy vấn 9: Sử dụng WHERE với điều kiện BETWEEN

query9 = "SELECT \* FROM mydata WHERE id BETWEEN 5 AND 15"

execute\_query(query9)

# Truy vấn 10: Sử dụng WHERE với điều kiện IN

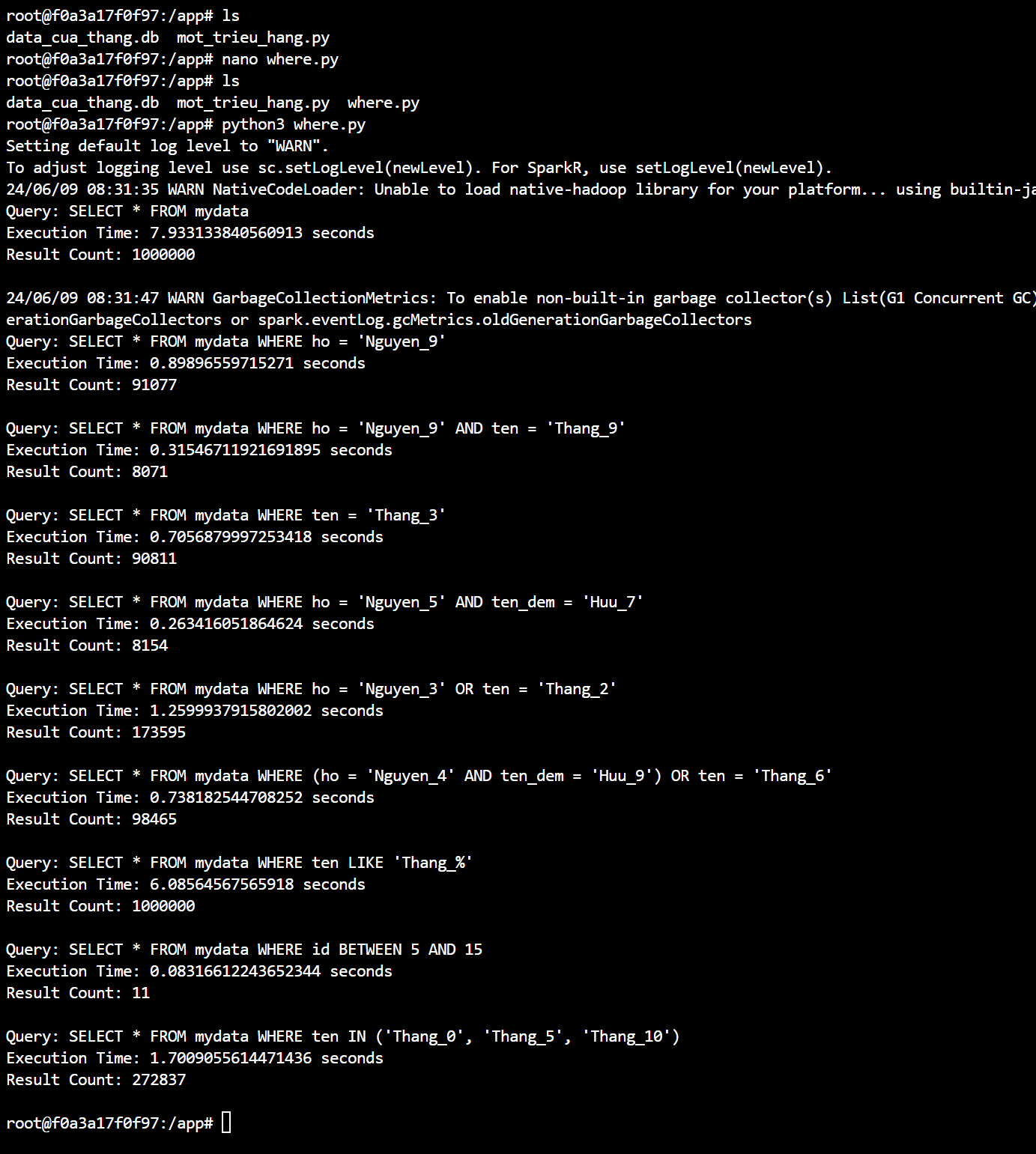
query10 = "SELECT \* FROM mydata WHERE ten IN ('Thang\_0', 'Thang\_5', 'Thang\_10')"

execute\_query(query10)

# Dừng phiên Spark

spark.stop()

* và ta nhận được kết quả:



* một số nhận xét:
* Truy vấn 1: Đây là truy vấn chậm nhất vì nó truy xuất tất cả các hàng trong bảng mà không có bộ lọc nào. Điều này yêu cầu Spark phải tải toàn bộ dữ liệu, xử lý và trả về tất cả các bản ghi.
* Truy vấn 2: Truy vấn này nhanh hơn rất nhiều vì nó chỉ trả về các hàng mà ho là 'Nguyen\_9', giảm đáng kể lượng dữ liệu cần xử lý.
* Truy vấn 3: Truy vấn này nhanh hơn nữa nhờ điều kiện lọc kép, chỉ trả về các hàng thỏa mãn cả hai điều kiện, giảm thêm dữ liệu cần xử lý.
* Truy vấn 4: Tương tự như truy vấn 2, nhưng sử dụng điều kiện lọc trên cột ten, thời gian thực hiện tương tự và số lượng kết quả gần bằng nhau.
* Truy vấn 5: Truy vấn này cũng rất nhanh vì nó lọc dựa trên hai điều kiện, chỉ trả về các hàng thỏa mãn cả hai điều kiện.
* Truy vấn 6: Truy vấn này chậm hơn các truy vấn trước đó vì nó sử dụng điều kiện OR, khiến lượng dữ liệu trả về nhiều hơn do bất kỳ hàng nào thỏa mãn một trong hai điều kiện đều được trả về.
* Truy vấn 7: Truy vấn này phức tạp hơn nhưng vẫn có hiệu suất tốt do sử dụng cả điều kiện AND và OR, tuy nhiên nó trả về nhiều hàng hơn các truy vấn chỉ dùng AND.
* Truy vấn 8: Truy vấn này khá chậm vì sử dụng điều kiện LIKE không giới hạn, kết quả là tất cả các hàng trong bảng được trả về, tương tự truy vấn đầu tiên.
* Truy vấn 9: Đây là truy vấn nhanh nhất vì nó chỉ lọc một tập rất nhỏ các hàng dựa trên điều kiện BETWEEN trên cột id.
* Truy vấn 10: Truy vấn này chậm hơn do sử dụng điều kiện IN với nhiều giá trị, kết quả là nhiều hàng hơn được trả về.
* **Kết luận**
* Hiệu quả của mệnh đề WHERE: Các truy vấn với mệnh đề WHERE cụ thể (đặc biệt là các điều kiện AND) cho thấy hiệu suất cao hơn so với các truy vấn không có bộ lọc hoặc với điều kiện tổng quát.
* Điều kiện phức tạp: Các điều kiện sử dụng OR hoặc LIKE không giới hạn thường chậm hơn do lượng dữ liệu trả về lớn hơn.
* Điều kiện đơn giản và rõ ràng: Các điều kiện rõ ràng và đơn giản, đặc biệt là trên các cột được lập chỉ mục, thường mang lại hiệu suất tốt nhất.
* Việc tối ưu hóa các truy vấn SQL bằng cách sử dụng mệnh đề WHERE hiệu quả là rất quan trọng để cải thiện hiệu suất truy vấn và giảm thời gian xử lý.

1. **Kết luận**

* Qua project này, ta nắm được cách xây dựng và chạy một docker container, cùng với đó là thực hiện sử dụng spark trong docker, biết cách sử dụng cơ sở dữ liệu trong docker và thực hiện một vài truy vấn, ngoài ra nhận thấy được sự ảnh hưởng của where đối với hiệu năng truy vấn.