TÀI LIỆU HƯỚNG DẪN

DỰ ÁN CUỐI KỲ: CHẠY SQLSPARK trong DOCKER CONTAINER

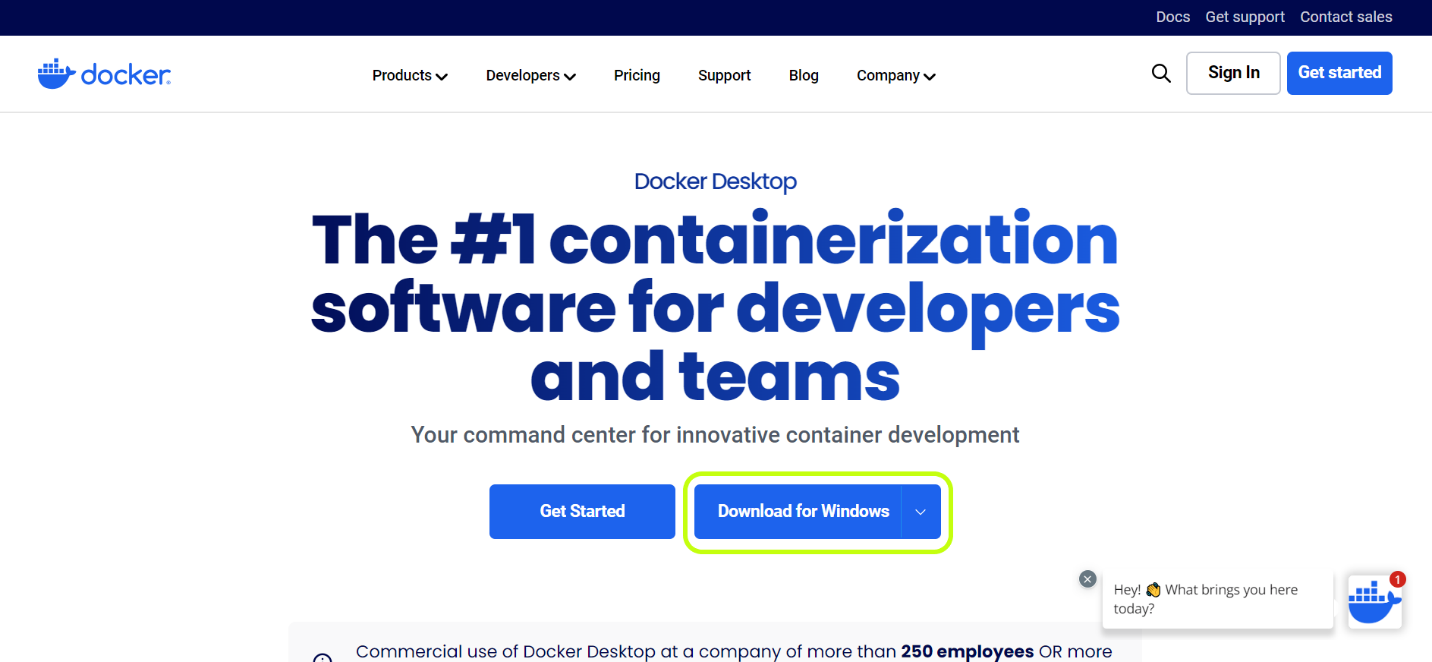
# Mô tả

Tài liệu này nhằm mục đích hướng dẫn thiết lập và chạy SparkSQL trong Docker container. Các phần phụ thuộc cần thiết trong container được cài đặt để thực hiện các truy vấn SQL trên cơ sở dữ liệu (SQLite) cũng được lưu trữ trong cùng một container.

# Điều kiện

Trước khi triển khai dự án, hãy đảm bảo máy tính của bạn có đủ bộ nhớ và đã cài đặt Docker (chạy hiệu quả với RAM 16GB).

Tải Docker Desktop tại [Docker Desktop](https://www.docker.com/products/docker-desktop/) và chọn phiên bản phù hợp với hệ điều hành của máy.



Giao diện website tải Docker Desktop

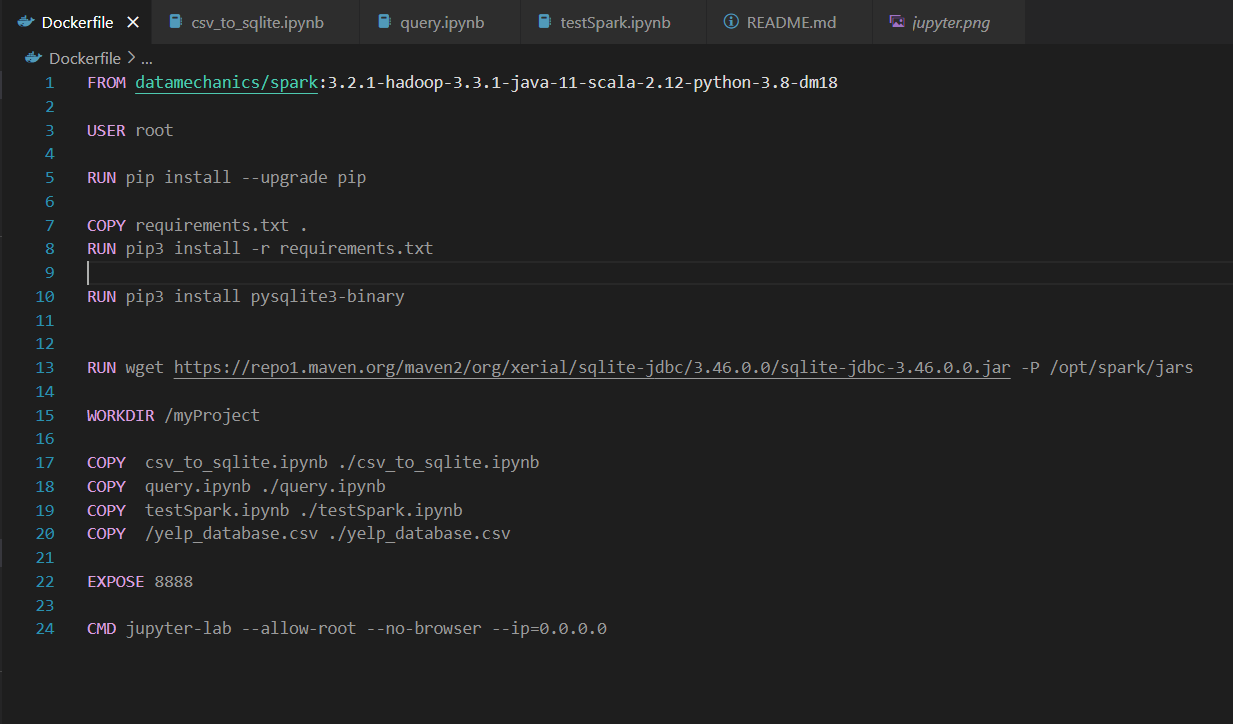
# Cài đặt docker

## Bước 1: Xây dựng Dockerfile

Để triển khai Spark với Docker, chúng tôi muốn cài đặt một số gói (connectors) để không cần cài đặt riêng. Nếu kiểm tra Docker hub, chúng tôi thấy rằng có nhiều tùy chọn có sẵn cho Spark.

Dự án đã lựa chọn image [datamechanics/spark](https://hub.docker.com/r/datamechanics/spark) vì nó cung cấp tất cả các connectors cần thiết mà chúng ta cần:

* Jupyter Notebook
* Tích hợp hỗ trợ Python & PySpark
* pip và conda (vì vậy rất dễ cài đặt các gói bổ sung)



Docker file

Đầu tiên, nó sẽ cài đặt datamechanics spark image, sau đó

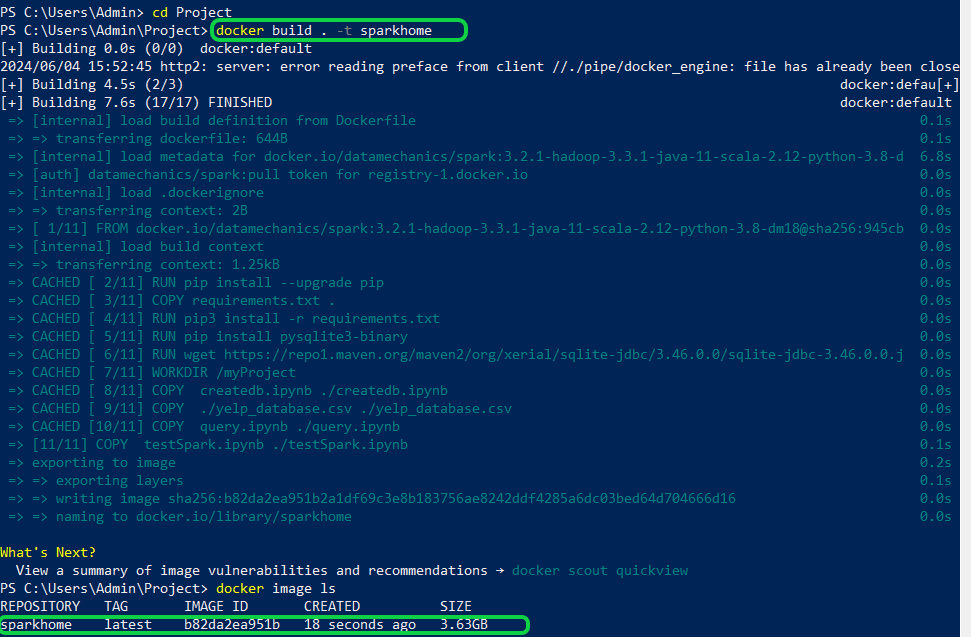
sẽ cài đặt các gói cần thiết có trong ***requirements.txt,*** đặt working directory và đặt cổng ra cho Jupter Notebook là 8888. Cuối cùngdẫn đến jupyter lab.

Như vậy ta đã viết được một docker file chuẩn bị các cài đặt cơ bản cho một container.

## Bước 2: Xây dựng Docker Image

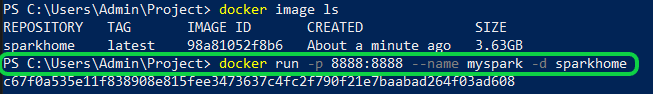


Câu lệnh khởi tạo một docker image có tên là **sparkhome** từ ***Dockerfile***.



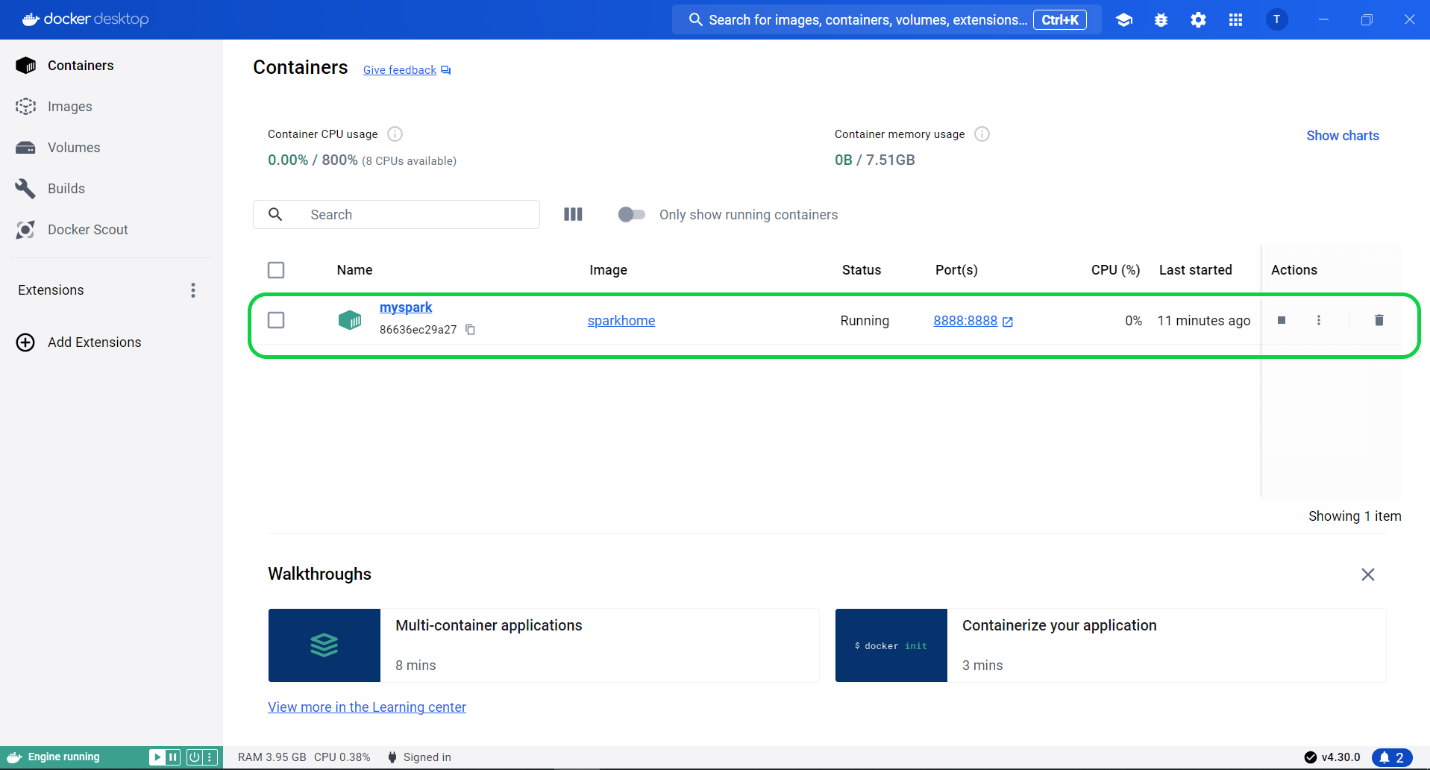
## Bước 3: Khởi tạo docker container

 Câu lệnh này sẽ khởi động một container Docker từ image **sparkhome**, mở cổng **8888** cho phép truy cập từ bên ngoài, đặt tên cho container là **myspark**, và chạy container ở chế độ nền.



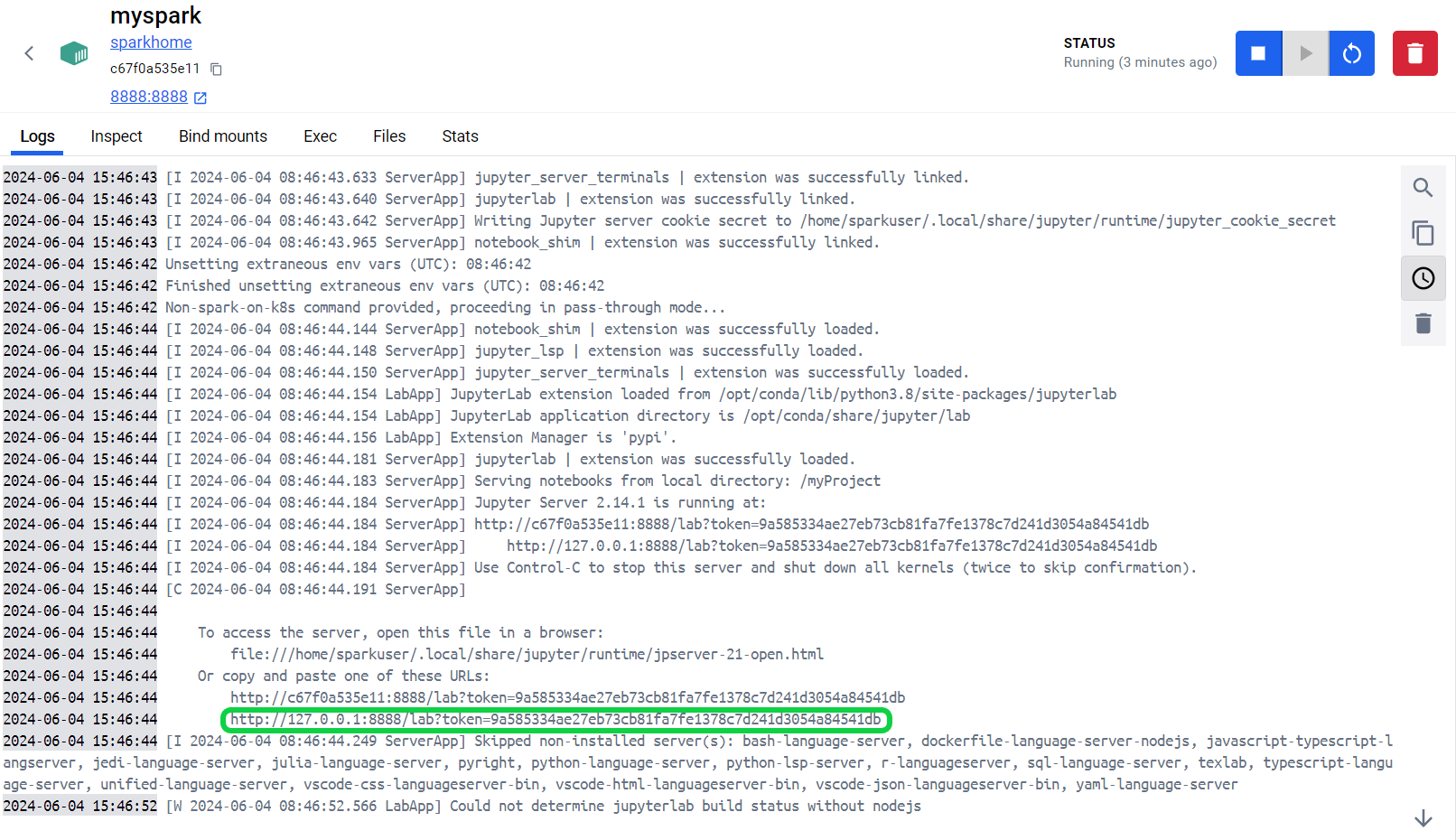
## Bước 4: Run container

Tại option **Container** của Docker Desktop sẽ xuất hiện một container có tên là **myspark.**



Container trên giao diện Docker Desktop

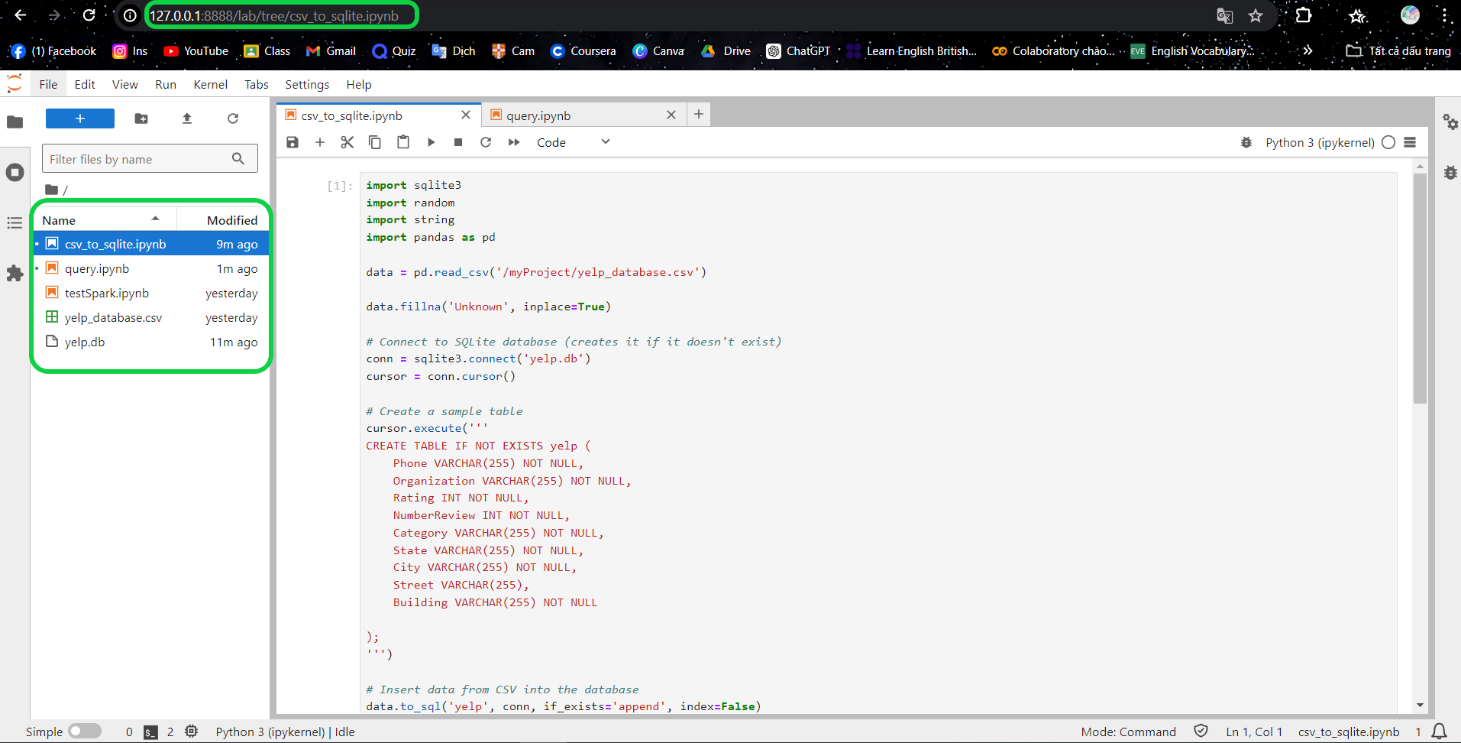
Trong option **logs** của container myspark sẽ có liên kết đến Jupyter lab



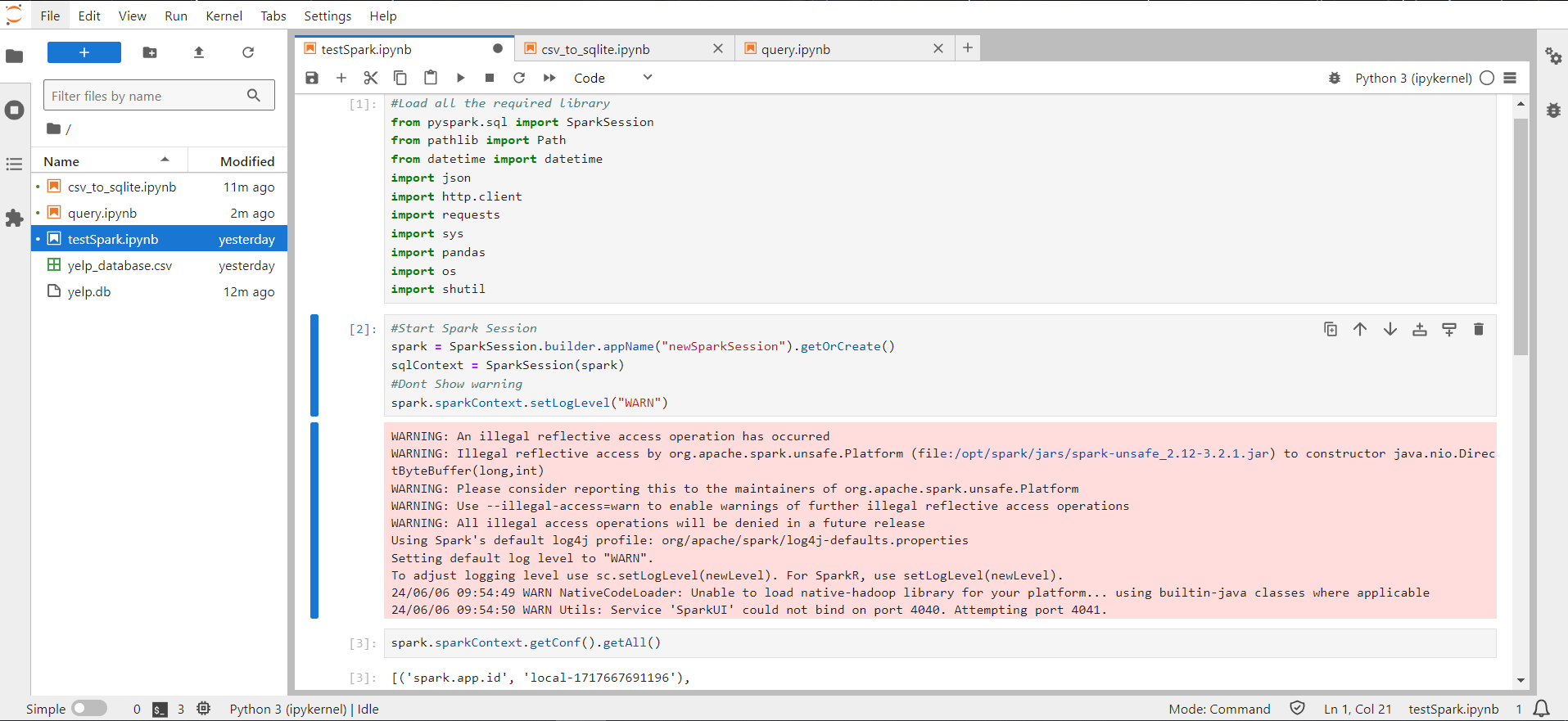
Liên kết đến Jupyter lab

## Bước 5

* Sao chép và dán link vào trình duyệt để chạy phiên pyspark từ trình duyệt.



* Tại Jupyter lab chạy file `testSpark.ipynb` để kiểm tra Spark có hoạt động không.

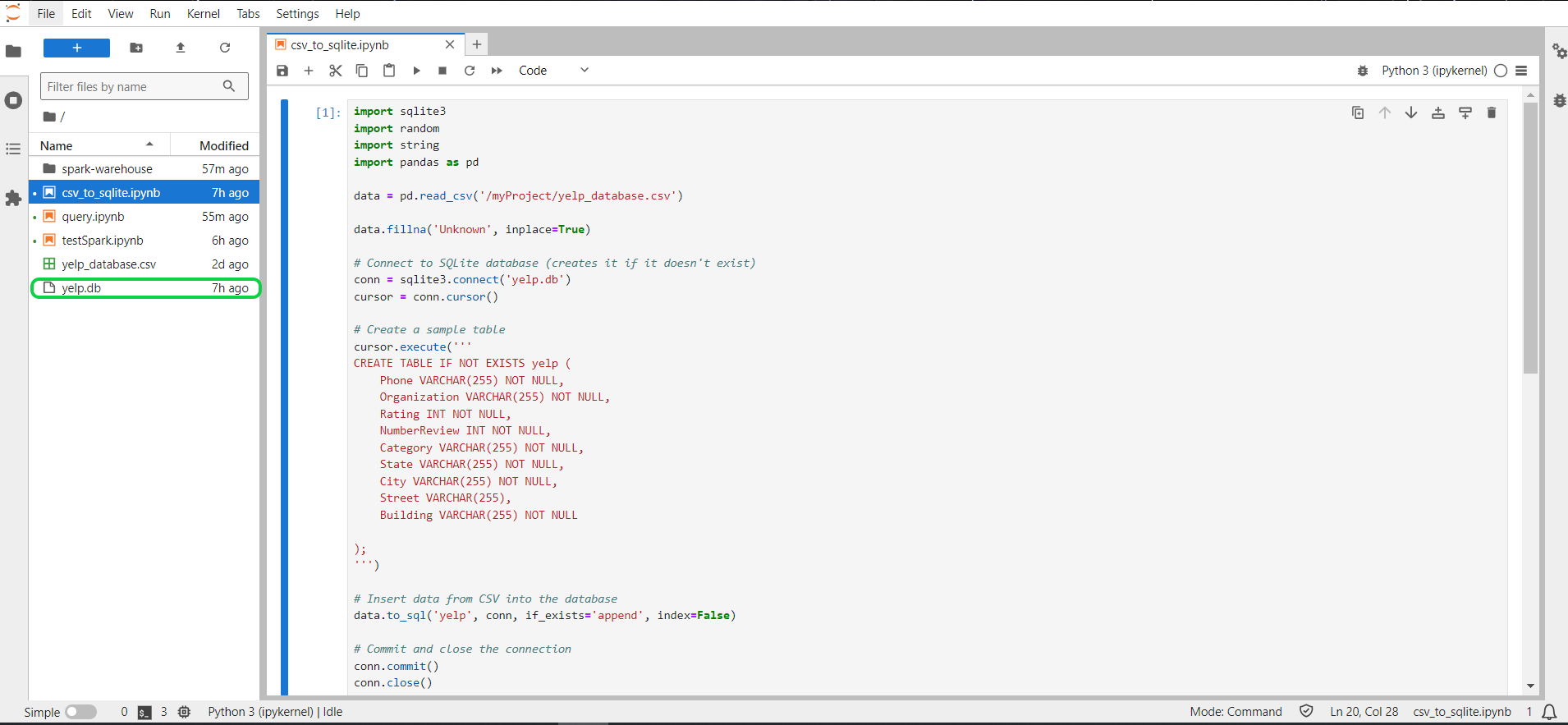


# SparkSQL

## Bước 1: Tạo cơ sở dữ liệu

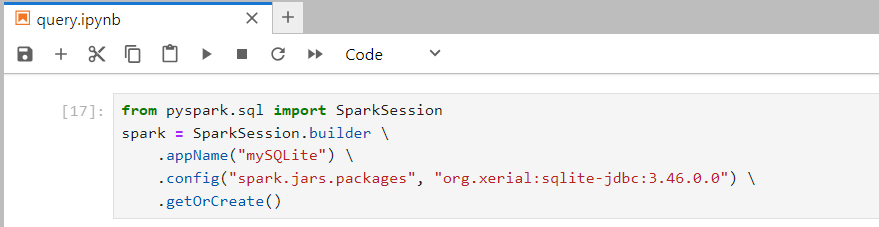
File ***csv-to-sqlite.ipynb*** với mục đích tạo ra một cơ sở dữ liệu SQLite bằng sqlite3 và chèn dữ liệu từ một file .csv vào cơ sở dữ liệu đó. Từ đó có thể dễ dàng thực hiện các thao tác truy vấn và phân tích dữ liệu sau này.

* Đầu tiên ta tạo một database có tên là **yelp**. Sau đó tạo một Table cũng tên là **yelp** bao gồm 9 cột: Phone, Organization, Rating, NumberReview, Category, State, City, Street, Building.
* Insert vào Table file ***yelp\_database.csv*** bao gồm 1000000 dòng. Đây là dataset thống kê đánh giá của dữ liệu người dùng cho các doanh nghiệp tại US.
* Chạy file ***csv-to-sqlite.ipynb*** để tạo tệp ***yelp.db***.



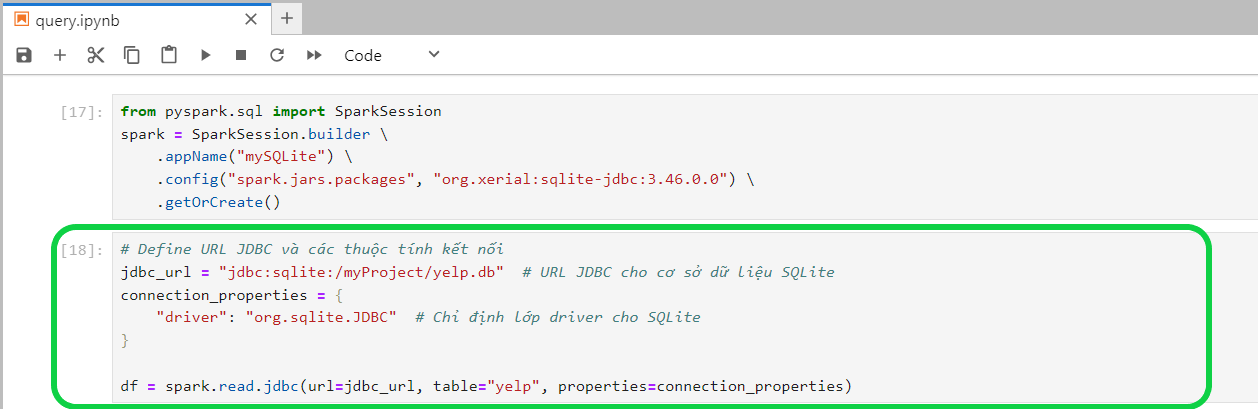
## Bước 2: Kết nối với cơ sở dữ liệu

* Tạo một phiên SparkSession để làm việc với dữ liệu trong môi trường Spark.



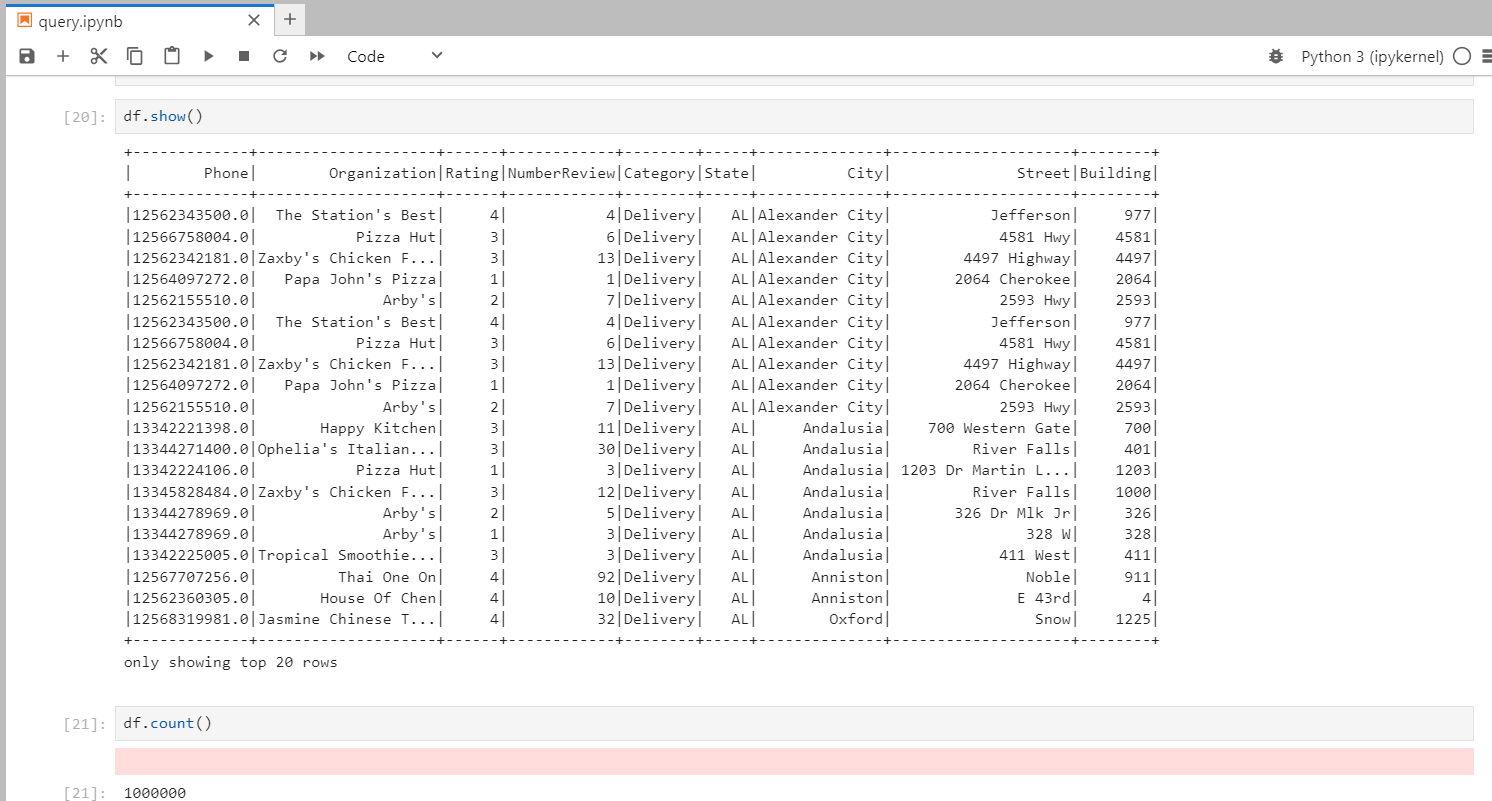
Đây là một phiên làm việc trong Spark, cho phép tạo DataFrame và thực hiện các truy vấn trên dữ liệu.

* Để kết nối và tương tác với các cơ sở dữ liệu từ Spark, cho phép đọc, ghi và xử lý dữ liệu từ nguồn dữ liệu SQLite, dự án sử dụng API JDBC driver 3.46.0.0



Cơ sở dữ liệu SQLite được lưu trữ tại “/myProject/yelp.db” và bảng là **yelp**, sau đó tạo một DataFrame từ dữ liệu đó để xử lý và phân tích dữ liệu trong môi trường Spark.

**df** sẽ chứa dữ liệu từ bảng **yelp** trong cơ sở dữ liệu SQLite được xác định. Các truy vấn trên dữ liệu được thực hiện trong DataFrame **df**.



## Bước 3: Thực hiện truy vấn SQL bằng SparkSQL

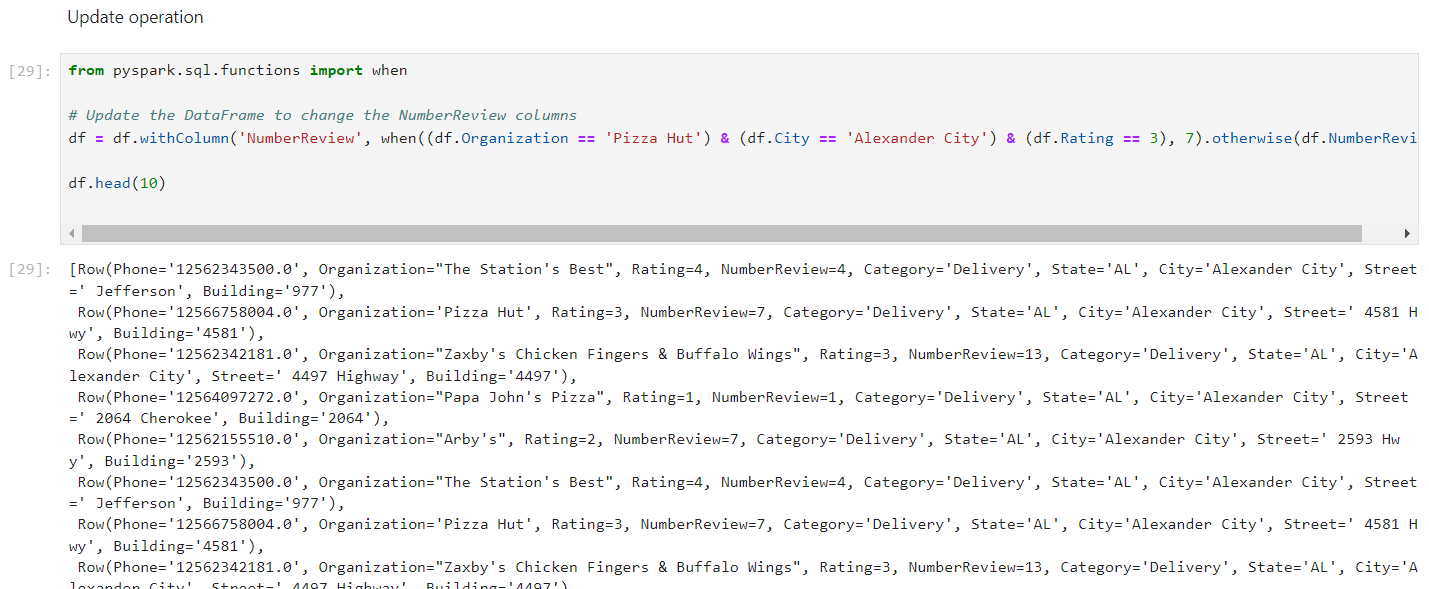
* Thao tác tạo



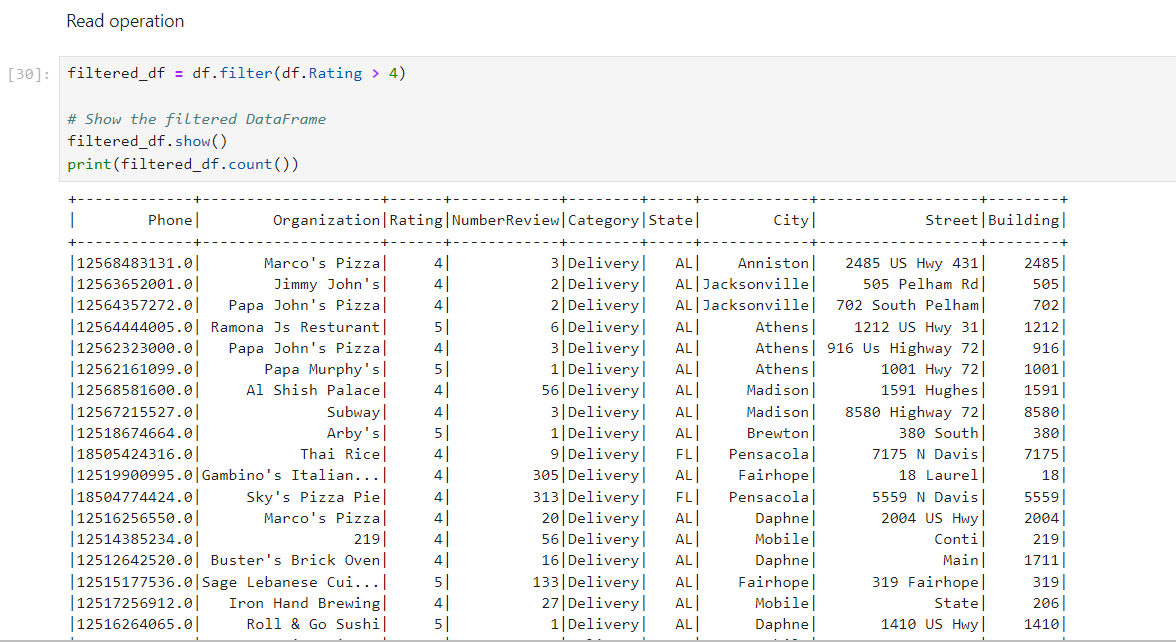
* Thao tác xóa



* Thao tác cập nhật

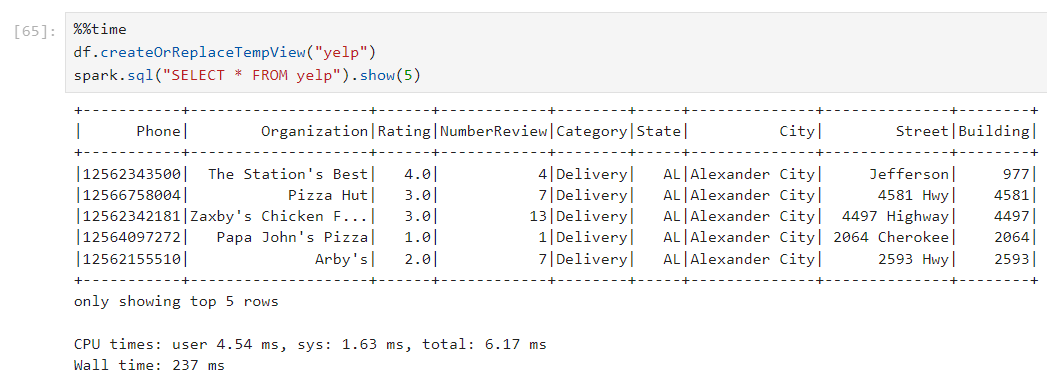


* Thao tác đọc

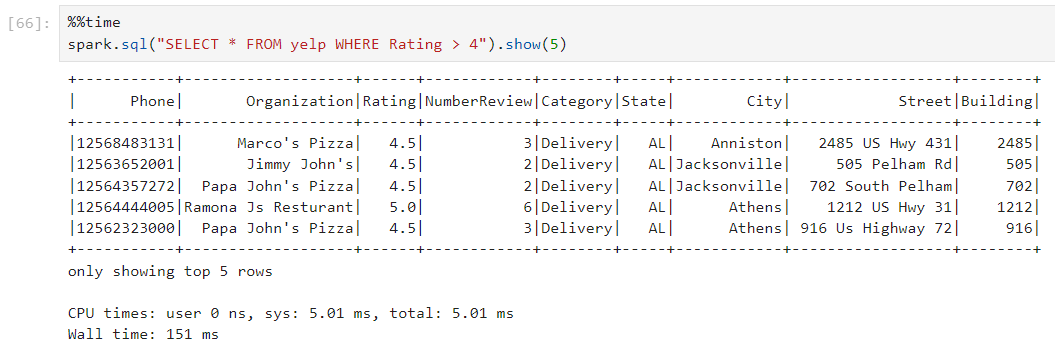


## Bước 4: So sánh hiệu suất truy vấn liên quan đến WHERE

## Khi không có WHERE

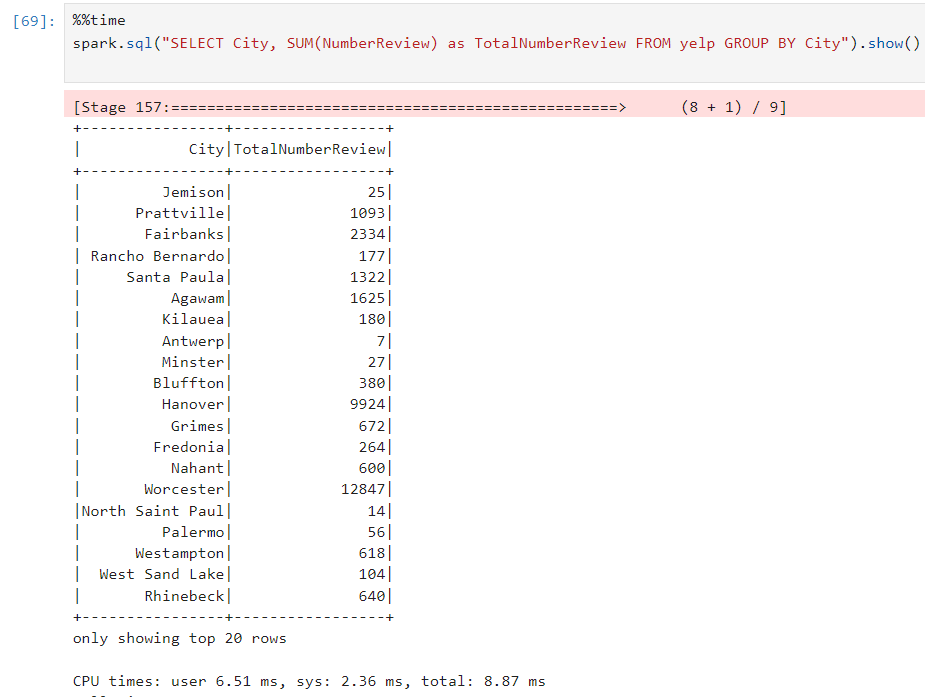


* Khi có WHERE

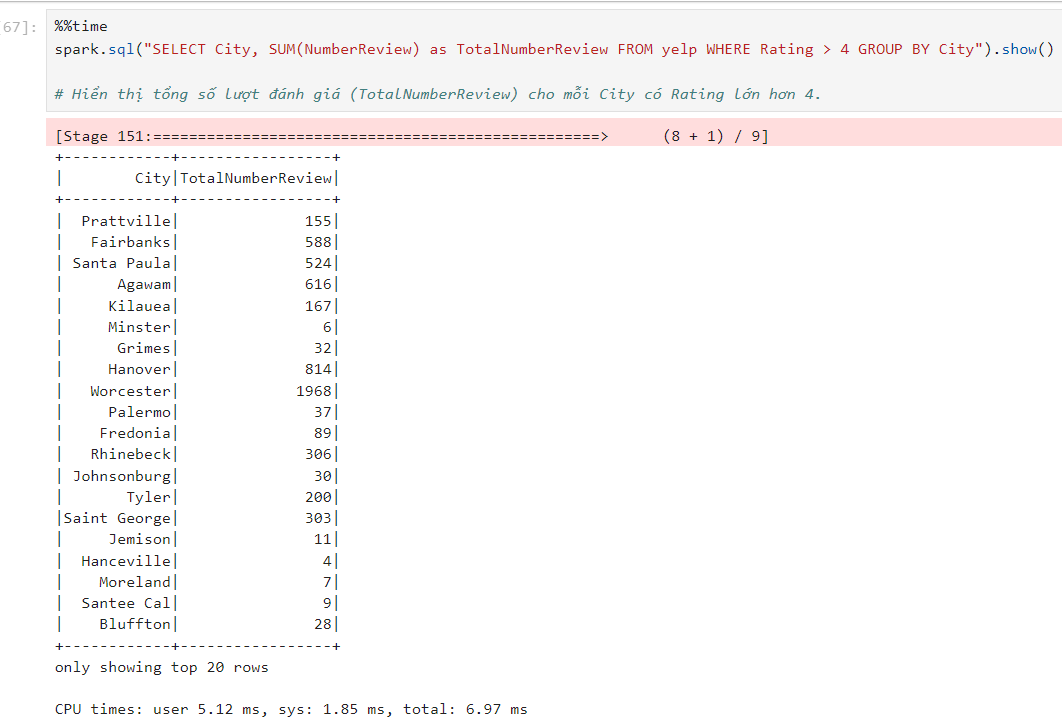


Mệnh đề WHERE giúp cơ sở dữ liệu làm việc hiệu quả hơn bằng cách giảm lượng dữ liệu cần xử lý, tận dụng chỉ mục và giảm tải hệ thống. Điều này dẫn đến thời gian thực thi nhanh hơn so với truy vấn không có WHERE.

* Không có WHERE, có GROUP BY



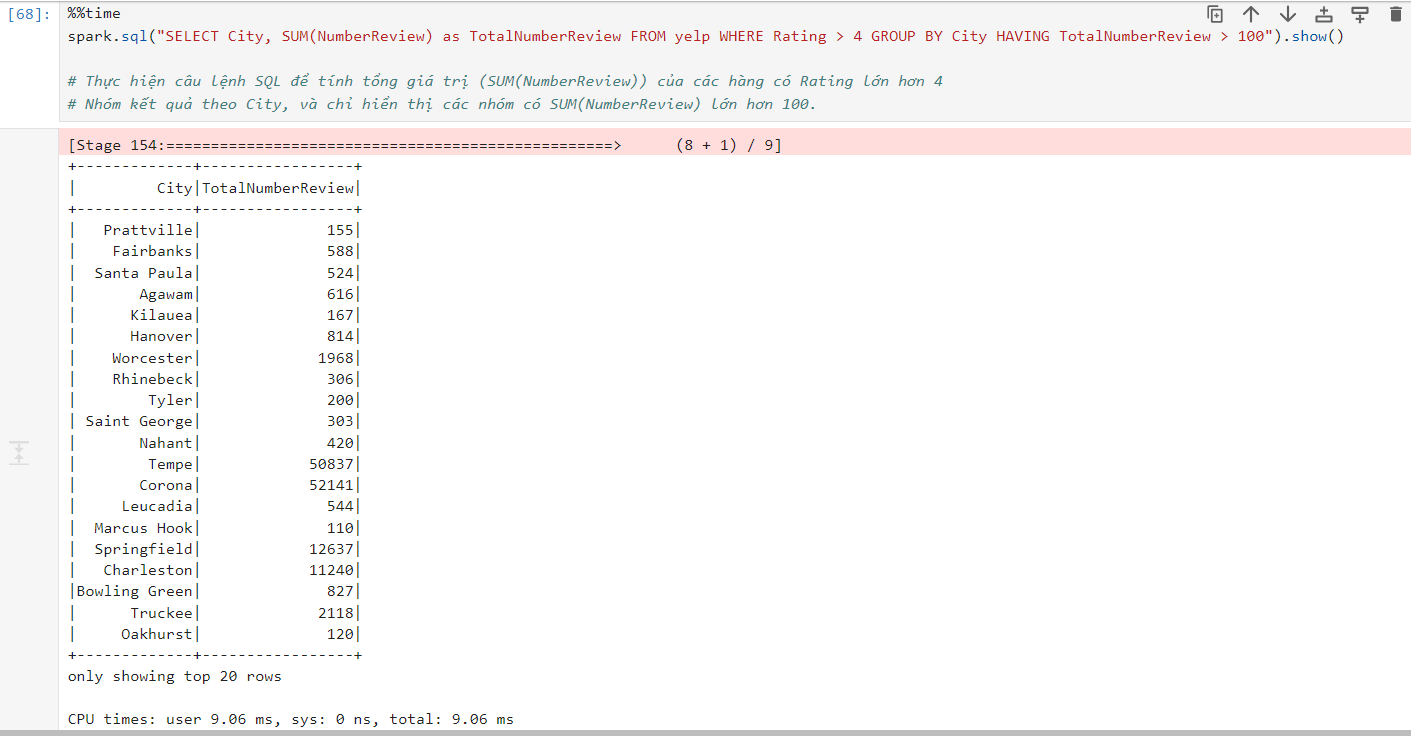
* Có WHERE, có GROUP BY



Mệnh đề WHERE khi kết hợp với GROUP BY giúp tăng tốc độ thực thi truy vấn bằng cách giảm số lượng bản ghi cần nhóm, tối ưu hóa kế hoạch thực thi và giảm tải hệ thống.

Nếu không có mệnh đề WHERE, cơ sở dữ liệu phải xử lý tất cả các dữ liệu trong bảng trước khi thực hiện việc nhóm, điều này sẽ tốn nhiều thời gian và tài nguyên hơn, đặc biệt nếu bảng có rất nhiều dữ liệu.

* Khi có HAVING



Khi sử dụng cả WHERE và HAVING trong một truy vấn, mệnh đề WHERE sẽ lọc các hàng dữ liệu có Rating > 4 trước, sau đó GROUP BY nhóm các dữ liệu theo City và cuối cùng, mệnh đề HAVING sẽ áp dụng điều kiện trên các nhóm này.

Việc kiểm tra điều kiện HAVING trên mỗi nhóm sau khi nhóm đã được tạo ra làm tăng thêm bước xử lý, kéo dài thời gian thực thi.

Tuy nhiên, việc có WHERE đã giúp tối ưu hiệu suất truy vấn hơn rất nhiều.