**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỆ THỐNG THÔNG TIN**



**BÁO CÁO BÀI TẬP QUÁ TRÌNH**

**MÔN HỌC ĐIỆN TOÁN ĐÁM MÂY**

**ĐỀ TÀI:**

**Azure With PySpark DataFrame**

Giảng viên hướng dẫn: ThS. Hà Lê Hoài Trung

Tên thành viên:

1. Phan Nguyễn Lâm Hà MSSV: 21522030

**TP. HỒ CHÍ MINH, NĂM 2024**

**LỜI CẢM ƠN**

Thành thật bày tỏ lòng biết ơn sâu sắc đến toàn thể thầy và các bạn trong lớp IS402.P11.HTCL, khoa Hệ Thống Thông Tin, trường Đại học Công nghệ Thông tin - Đại học Quốc gia TP.HCM, đã dành sự quan tâm và hỗ trợ cho chúng tôi trong suốt quá trình học tập và thực hiện dự án.

Đặc biệt, chúng tôi muốn bày tỏ lòng biết ơn đặc biệt đến thầy Hà Lê Hoài Trung. Sự tận tâm và sự hướng dẫn kỹ càng từ thầy đã giúp chúng tôi vượt qua những thách thức trong dự án và phát triển không chỉ về kiến thức mà còn về kỹ năng thực hành.

Trong suốt quá trình thực hiện dự án, chúng tôi đã luôn nỗ lực để tiến bộ và hoàn thiện từ những sai sót. Chúng tôi mong muốn nhận được sự phản hồi chân thành từ phía thầy và các để có thêm kinh nghiệm và kiến thức cần thiết cho các dự án sắp tới.

Xin được bày tỏ lòng biết ơn sâu sắc và cam kết sẽ tiếp tục nỗ lực hơn nữa để phát triển và trở thành những chuyên gia có ích trong lĩnh vực điện toán đám mây.

**Nhóm thực hiện**

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

*……., ngày……...tháng……năm 2024*

**Người nhận xét**

*(Ký tên và ghi rõ họ tên****)***

**MỤC LỤC**

[1. Apache Spark 5](#_Toc185401426)

[1.1. Spark là gì? 5](#_Toc185401427)

[1.2. Spark Application 6](#_Toc185401428)

[1.3. Ứng dụng Spark so với ứng dụng Pyspark 8](#_Toc185401429)

[1.4. Phần cốt lõi của một chương trình pyspark 9](#_Toc185401430)

[1.5. Thêm gói PySpark 9](#_Toc185401431)

[1.6. Bắt đầu phiên Spark 10](#_Toc185401432)

[1.7. Xác định một tập hợp các biến đổi và hành động 10](#_Toc185401433)

[2. Spark DataFrame 11](#_Toc185401434)

[2.1. Giới thiệu về Spark DataFrames 11](#_Toc185401435)

[2.2. Spark DataFrames so với Spark Datasets 12](#_Toc185401436)

[2.3. Phân vùng của một Spark DataFrame 14](#_Toc185401437)

[3. Biến Đổi Spark DataFrame 15](#_Toc185401438)

[3.1. Kích hoạt tính toán bằng hành động 15](#_Toc185401439)

[3.2. Hiểu biến đổi hẹp và biến đổi rộng 16](#_Toc185401440)

[3.3. Phương thức Lọc Dataframe 19](#_Toc185401441)

[3.3.1. Lọc các hàng của DataFrame 20](#_Toc185401442)

[3.3.2. Chuyển từ khóa in sang cách viết theo kiểu pythonic 21](#_Toc185401443)

[3.3.3. Phủ định các điều kiện logic 22](#_Toc185401444)

[3.3.4. Lọc giá trị null (tức là dữ liệu bị thiếu) 23](#_Toc185401445)

[3.3.5. Lọc ngày và giờ trong DataFrame 24](#_Toc185401446)

[3.4. Phương thức quản lý cột Dataframe 24](#_Toc185401447)

[3.4.1. Thêm cột mới 25](#_Toc185401448)

[3.4.2. Loại bỏ cột 25](#_Toc185401449)

[3.4.3. Tái sắp xếp cột 25](#_Toc185401450)

[3.4.4. Tính toán giá trị mới 25](#_Toc185401451)

[3.4.5. Tính toán hoặc thêm cột dựa trên điều kiện 26](#_Toc185401452)

[3.5. Sắp xếp các dòng của DataFrame 26](#_Toc185401453)

[3.5.1. Sắp xếp tăng dần hoặc giảm dần 26](#_Toc185401454)

[3.5.2. Sắp xếp nhiều cột 26](#_Toc185401455)

[3.5.3. Sắp xếp dựa trên điều kiện 27](#_Toc185401456)

[3.6. Xoá các giá trị trùng lặp từ DataFrame 27](#_Toc185401457)

[3.6.1. Xoá các hàng trùng lặp dựa trên một hoặc nhiều cột cụ thể 27](#_Toc185401458)

[3.7. Các kỹ thuật khác để xử lý giá trị null 28](#_Toc185401459)

[3.7.1. Thay thế giá trị null bằng giá trị khác 28](#_Toc185401460)

[3.7.2. Loại bỏ các cột chứa giá trị null 28](#_Toc185401461)

[3.7.3. Thực hiện kiểm tra tồn tại giá trị null 29](#_Toc185401462)

[3.8. Các phép toán liên kết (Union) 29](#_Toc185401463)

[3.8.1. Sử dụng phép toán union() 29](#_Toc185401464)

[3.8.2. Lưu ý 30](#_Toc185401465)

[3.9. Các phép toán nối (Join) 30](#_Toc185401466)

[3.9.1. Sử dụng phép toán join() 30](#_Toc185401467)

[3.9.2. Loại nối (Join types) 31](#_Toc185401468)

[3.9.3. Lưu ý 32](#_Toc185401469)

[3.10. Các phép toán pivot (Pivot) 32](#_Toc185401470)

[3.10.1. Sử dụng phép toán pivot() 32](#_Toc185401471)

[3.10.2. Lưu ý 33](#_Toc185401472)

[3.11. Thu thập và phân rã 33](#_Toc185401473)

[3.11.1. Thu thập (Collecting) 33](#_Toc185401474)

[3.11.2. Phân rã (Explode) 34](#_Toc185401475)

[3.11.3. Lưu ý 35](#_Toc185401476)

[4. Demo Cài Đặt Spark và Chạy Local PySpark 35](#_Toc185401477)

# Apache Spark

## Spark là gì?

Spark là một công cụ xử lý dữ liệu quy mô lớn hỗ trợ cả các máy đơn và các cụm máy. Ngày nay, Spark đã trở thành tiêu chuẩn không chính thức cho việc cấu trúc và quản lý các ứng dụng dữ liệu lớn.

Nó có một số tính năng mà các công cụ tiền nhiệm của nó không có, như khả năng xử lý trong bộ nhớ và xử lý dòng .Nhưng, tính năng quan trọng nhất của tất cả, là Spark là một nền tảng thống nhất cho việc xử lý dữ liệu lớn

Điều này có nghĩa là Spark đi kèm với nhiều thư viện và công cụ tích hợp sẵn xử lý các khía cạnh khác nhau của công việc với dữ liệu lớn. Nó có một SQL engine tích hợp sẵn để thực hiện xử lý dữ liệu quy mô lớn; một thư viện hoàn chỉnh cho học máy có khả năng mở rộng (MLib); một engine xử lý dòng cho phân tích dữ liệu dòng; và nhiều hơn nữa.

Nói chung, các công ty lớn có nhiều nhu cầu dữ liệu khác nhau, và kết quả là, các kỹ sư và nhà phân tích có thể phải kết hợp và tích hợp nhiều công cụ và kỹ thuật lại với nhau, để họ có thể xây dựng nhiều đường ống dữ liệu khác nhau để đáp ứng những nhu cầu này. Nhưng cách tiếp cận này có thể tạo ra một vấn đề phụ thuộc rất nghiêm trọng, tạo ra một rào cản lớn cho việc hỗ trợ quy trình làm việc này. Đây là một trong những lý do lớn khiến Spark trở nên thành công. Nó loại bỏ một phần lớn của vấn đề này, bằng cách đã bao gồm gần như tất cả những gì có thể cần sử dụng.

Spark được thiết kế để bao phủ một loạt các khối công việc trước đây yêu cầu các hệ thống phân tán riêng biệt... Bằng cách hỗ trợ các khối công việc này trong cùng một engine, Spark làm cho việc kết hợp các loại xử lý khác nhau trở nên dễ dàng và không tốn kém, điều này thường cần thiết trong các đường ống phân tích dữ liệu sản xuất. Ngoài ra, nó giảm bớt gánh nặng quản lý của việc duy trì các công cụ riêng biệt.

## Spark Application

Máy tính cá nhân có thể làm rất nhiều việc, nhưng nó không thể xử lý một lượng lớn dữ liệu một cách hiệu quả. Trong tình huống như vậy, chúng ta cần nhiều máy tính làm việc cùng nhau, kết hợp tài nguyên của chúng để xử lý khối lượng hoặc phức tạp của dữ liệu. Spark là framework điều phối các tính toán trên tập hợp các máy tính này. Do đó, một phần quan trọng của cấu trúc của Spark liên quan sâu đến các mô hình tính toán phân tán.

Có lẽ không có một cụm máy tính ở nhà. Vì vậy, khi làm theo các ví dụ trong cuốn sách này, sẽ chạy Spark trên một máy tính đơn (tức là chế độ máy đơn). Nhưng hãy quên điều này một chút.

Trong mọi ứng dụng Spark, luôn có một máy tính đóng vai trò làm trạm điều khiển (driver node), và nhiều máy tính khác đóng vai trò làm các trạm thực thi (worker nodes). Trạm điều khiển chịu trách nhiệm quản lý ứng dụng Spark, tức là yêu cầu tài nguyên, phân phối các nhiệm vụ cho các trạm thực thi, thu thập và biên soạn kết quả,.... Các trạm thực thi chịu trách nhiệm thực hiện các nhiệm vụ được giao cho họ, và họ cần gửi kết quả của các nhiệm vụ này trở lại cho trạm điều khiển.

Mỗi ứng dụng Spark được phân phối thành hai quy trình khác nhau và độc lập: 1) một quy trình trạm điều khiển

2) và một tập hợp các quy trình thực thi .

Quy trình trạm điều khiển, hoặc chương trình trạm điều khiển, là nơi ứng dụng bắt đầu, và nó được thực thi bởi trạm điều khiển. Chương trình trạm điều khiển này chịu trách nhiệm cho:

1) duy trì thông tin về ứng dụng Spark ;

2) phản hồi cho chương trình hoặc đầu vào của người dùng;

3) và phân tích, phân phối và lên lịch làm việc trên các trạm thực thi .

Mỗi khi một ứng dụng Spark bắt đầu, quy trình trạm điều khiển phải giao tiếp với quản lý cụm, để có được các trạm thực thi để thực hiện các nhiệm vụ cần thiết. Nói cách khác, quản lý cụm quyết định xem Spark có thể sử dụng một số tài nguyên (tức là một số máy tính) của cụm hay không. Nếu quản lý cụm cho phép Spark sử dụng các nút mà nó cần, chương trình trạm điều khiển sẽ phân chia ứng dụng thành nhiều nhiệm vụ nhỏ và sẽ giao các nhiệm vụ này cho các trạm thực thi.

Các quy trình thực thi là các quy trình diễn ra trong mỗi một trong các trạm thực thi. Mỗi quy trình thực thi được tạo thành từ một tập hợp các nhiệm vụ, và trạm thực thi chịu trách nhiệm thực hiện và thực thi các nhiệm vụ này được giao cho nó, bởi chương trình trạm điều khiển. Sau khi thực hiện các nhiệm vụ này, trạm thực thi sẽ gửi kết quả trở lại cho trạm điều khiển (hoặc chương trình trạm điều khiển). Nếu cần, các trạm thực thi có thể giao tiếp với nhau trong khi thực hiện các nhiệm vụ của mình. A diagram of a cluster manager

Description automatically generated

## Ứng dụng Spark so với ứng dụng Pyspark

Gói pyspark chỉ là một công cụ để viết các ứng dụng Spark bằng ngôn ngữ lập trình Python. Điều này có nghĩa là, mỗi ứng dụng pyspark là một ứng dụng Spark được viết bằng Python.

Với khái niệm này trong tâm trí, có thể hiểu rằng một ứng dụng pyspark là một mô tả của một ứng dụng Spark. Khi chúng ta biên dịch (hoặc thực thi) chương trình Python của mình, mô tả này được dịch thành một ứng dụng Spark thô sẽ được thực thi bởi Spark.

Để viết một ứng dụng pyspark, viết một script Python sử dụng thư viện pyspark. Khi thực thi script Python này với trình thông dịch Python, ứng dụng sẽ tự động được chuyển đổi thành mã Spark và sẽ được gửi đến Spark để được thực thi trên cụm máy tính;

## Phần cốt lõi của một chương trình pyspark

Trong phần này, tôi muốn chỉ ra các phần cốt lõi tạo nên mỗi chương trình pyspark. Điều này có nghĩa là mỗi chương trình pyspark mà viết sẽ có những "phần cốt lõi" này, bao gồm:

1. Import gói pyspark (hoặc các modules):



2. Khởi đầu phiên Spark:

A black screen with a black background

Description automatically generated

3. Xác định một tập hợp các biến đổi (transformations) và hành động (actions) trên các Spark DataFrames:

A computer screen with text

Description automatically generated

Lưu ý rằng có thể có nhiều hơn một tập hợp các biến đổi và hành động trong một chương trình pyspark, nhưng những phần trên là những phần cốt lõi phổ biến mà mỗi chương trình pyspark thường có.

## Thêm gói PySpark

Spark đi kèm với rất nhiều chức năng được cài đặt. Nhưng, để sử dụng chúng trong chương trình pyspark , phải nhập hầu hết các chức năng này vào phiên làm việc của mình. Điều này có nghĩa là phải nhập các gói cụ thể (hoặc "modules") của pyspark vào phiên làm việc Python của mình.

Ví dụ, hầu hết các hàm được sử dụng để xác định các biến đổi và tổng hợp trong Spark DataFrames đến từ module pyspark.sql.functions.

Đó là lý do tại sao chúng ta thường bắt đầu các script Python của mình bằng cách nhập các hàm từ module này, như sau:



Hoặc, nhập toàn bộ module với từ khóa import, như sau:



## Bắt đầu phiên Spark

Mỗi ứng dụng Spark bắt đầu với một phiên Spark. Cơ bản, Phiên Spark là điểm nhập của ứng dụng . Điều này có nghĩa là, trong mọi chương trình pyspark viết, luôn nên bắt đầu bằng cách xác định phiên Spark của mình. Chúng ta làm điều này bằng cách sử dụng phương thức getOrCreate() từ module pyspark.sql.SparkSession.builder.

Chỉ cần lưu kết quả của phương thức này vào bất kỳ đối tượng Python nào. Thông thường, người ta thường đặt tên cho đối tượng này là spark, như trong ví dụ dưới đây. Như vậy, có thể truy cập tất cả thông tin và phương thức của Spark từ đối tượng spark này.

A black background with white text

Description automatically generated

## Xác định một tập hợp các biến đổi và hành động

Mỗi chương trình pyspark được tạo thành từ một tập hợp các biến đổi và hành động trên một tập hợp các Spark DataFrames.

Chúng là cấu trúc dữ liệu cơ bản cung cấp dữ liệu cho tất cả các chương trình pyspark. Nói cách khác, trong mỗi chương trình pyspark, chúng ta đều biến đổi nhiều Spark DataFrames để có được kết quả mong muốn.

Ví dụ, trong đoạn mã dưới đây, chúng tôi bắt đầu với Spark DataFrame được lưu trong đối tượng students, và áp dụng nhiều biến đổi trên nó để xây dựng DataFrame ar\_department. Cuối cùng, chúng tôi áp dụng hành động .show() trên DataFrame ar\_department:

A computer screen with white and green text

Description automatically generated

# Spark DataFrame

## Giới thiệu về Spark DataFrames

Trong chương này, hiểu cách Spark biểu diễn và quản lý các bảng (hoặc dữ liệu dạng bảng). Các ngôn ngữ lập trình và framework khác nhau sử dụng các tên khác nhau để mô tả một bảng. Nhưng, trong Apache Spark, chúng được gọi là Spark DataFrames.

Trong pyspark, các DataFrames này được lưu trong các đối tượng Python thuộc lớp `pyspark.sql.dataframe.DataFrame`, và tất cả các phương thức có trong lớp này, thường được gọi là API DataFrame của Spark. Đây là API quan trọng nhất của Spark, vì hầu hết các ứng dụng Spark sẽ sử dụng nhiều API này để xây dựng các biến đổi dữ liệu và luồng dữ liệu

## Spark DataFrames so với Spark Datasets

Spark có hai khái niệm về dữ liệu có cấu trúc: DataFrames và Datasets. Tóm lại, một Spark Dataset là một bộ sưu tập dữ liệu phân tán (Tài liệu Chính thức của Apache Spark 2022). Ngược lại, một Spark DataFrame là một Spark Dataset được tổ chức thành các cột có tên.

Điều này có nghĩa là, Spark DataFrames rất giống với các bảng như chúng ta biết trong cơ sở dữ liệu quan hệ - RDBMS, hoặc trong bảng tính (như Excel). Vì vậy trong một Spark DataFrame, mỗi cột có tên, và tất cả chúng có cùng số hàng. Hơn nữa, tất cả các hàng trong một cột phải lưu trữ cùng một loại dữ liệu, nhưng mỗi cột có thể lưu trữ một loại dữ liệu khác nhau.

Trong khi đó, Spark Datasets được coi là một bộ sưu tập của bất kỳ loại dữ liệu nào. Vì vậy một Dataset có thể là một bộ sưu tập dữ liệu không có cấu trúc, như các tệp nhật ký, cây JSON và XML, v.v. Spark Datasets có thể được tạo và biến đổi thông qua API Dataset của Spark. Nhưng API này chỉ có sẵn trong API Scala và Java của Spark. Vì lý do này, chúng ta không làm việc trực tiếp trên Datasets với pyspark, chỉ là DataFrames. Điều này là đúng, vì đa số các ứng dụng, chúng ta muốn sử dụng DataFrames, và không phải Datasets, để biểu diễn dữ liệu của chúng ta.

Tuy nhiên, điều gì làm cho một Spark DataFrame khác biệt so với các dataframe khác? Như DataFrame của pandas? Hoặc cấu trúc dữ liệu data.frame của R? Là khía cạnh phân tán của nó. Spark DataFrames dựa trên Spark Datasets, và các Datasets này là bộ sưu tập dữ liệu được phân tán trên cụm. Ví dụ, hãy giả sử có bảng sau được lưu dưới dạng một Spark DataFrame:

A screenshot of a computer

Description automatically generated

Nếu đang chạy Spark trên một cụm 4 nút (một trong số đó là nút điều khiển và ba nút còn lại là các nút công việc). Mỗi nút công việc của cụm sẽ lưu trữ một phần của dữ liệu này. Vì vậy, , như là người lập trình, sẽ nhìn thấy, quản lý và biến đổi bảng này như nó là một bảng duy nhất và thống nhất. Nhưng đằng sau màn hình, Spark sẽ chia nhỏ dữ liệu này và lưu trữ nó thành nhiều phần trên cụm Spark. Hình trình bày khái niệm này một cách trực quanA diagram of a server

Description automatically generated

## Phân vùng của một Spark DataFrame

Một Spark DataFrame luôn được chia thành nhiều phần nhỏ, và, những phần này luôn được phân tán trên cụm máy. Mỗi phần nhỏ của dữ liệu tổng thể này được coi là một phân vùng của DataFrame.

Đối với đa số trường hợp, không cần thao tác các phân vùng này bằng tay hoặc từng cái một, vì Spark tự động thực hiện công việc này cho .

M nút của cụm sẽ giữ một phần của DataFrame tổng thể. Nếu chúng ta chuyển đổi phân phối này thành phân vùng, điều này có nghĩa là mỗi nút của cụm có thể giữ một hoặc nhiều phân vùng của Spark DataFrame.

Nếu chúng ta tổng hợp tất cả các phân vùng có mặt trong một nút của cụm, chúng ta sẽ có một phần của DataFrame tổng thể. Hình ở dưới minh họa cho khái niệm này:

A diagram of a file

Description automatically generated

Nếu Spark DataFrame không lớn, mỗi nút của cụm có thể chỉ lưu trữ một phân vùng của DataFrame này. Ngược lại, tùy thuộc vào độ phức tạp và kích thước của DataFrame, Spark sẽ chia DataFrame này thành nhiều phân vùng hơn số lượng nút trong cụm. Trong trường hợp này, mỗi nút của cụm sẽ giữ nhiều hơn 1 phân vùng của DataFrame tổng thể.

# Biến Đổi Spark DataFrame

## Kích hoạt tính toán bằng hành động

Do đó, Spark sẽ tránh thực hiện bất kỳ tính toán nặng nề nào cho đến khi việc tính toán đó thực sự cần thiết. Nhưng làm thế nào hoặc khi nào Spark sẽ đối mặt với quyết định này? Khi nó gặp một hành động. Một hành động là công cụ có để kích hoạt Spark để thực sự thực hiện các biến đổi đã định nghĩa.

Một hành động chỉ dẫn Spark tính toán kết quả từ một loạt các biến đổi

Có bốn loại hành động trong Spark:

1. Hiển thị kết quả trên console;
2. Ghi dữ liệu vào một tập tin hoặc nguồn dữ liệu nào đó;
3. Thu thập dữ liệu từ một Spark DataFrame sang các đối tượng native trong python (hoặc Java, Scala, R, v.v.);
4. Đếm số hàng trong một Spark DataFrame;

Đã biết loại hành động đầu tiên, vì chúng ta đã sử dụng nó trước đó với phương thức show(). Phương thức show() này là một hành động, bởi vì đang yêu cầu Spark hiển thị một số kết quả cho . Vì vậy, chúng ta có thể khiến Spark thực sự tính toán các biến đổi xác định DataFrame big\_values bằng cách yêu cầu Spark hiển thị DataFrame này cho chúng ta.



Đặc biệt hơn, đôi khi muốn thu thập dữ liệu của một Spark DataFrame để sử dụng nó bên trong python. Nói cách khác, đôi khi cần làm một số công việc mà Spark không thể tự làm được. Để làm điều đó, thu thập một phần của dữ liệu đang được tạo bởi Spark và lưu trữ nó trong một đối tượng python thông thường để sử dụng trong một chương trình python tiêu chuẩn.

Đó là những gì phương thức collect() làm. Nó chuyển tất cả dữ liệu của DataFrame Spark thành một danh sách python tiêu chuẩn mà có thể dễ dàng truy cập bằng python. Cụ thể hơn, nhận được một danh sách python chứa các giá trị Row():





Phương thức take() rất giống với collect(). Nhưng thường áp dụng take() khi cần thu thập chỉ một phần nhỏ của DataFrame (và không phải là toàn bộ), như là n hàng đầu tiên.





## Hiểu biến đổi hẹp và biến đổi rộng

Trong Spark, có hai loại biến đổi: biến đổi hẹp (narrow) và biến đổi rộng (wide). Hãy nhớ rằng, một Spark DataFrame được chia thành nhiều phần nhỏ (gọi là các phân vùng), và những phần này được phân tán trên cụm. Sự khác biệt cơ bản giữa các biến đổi hẹp và rộng là liệu biến đổi có buộc Spark đọc dữ liệu từ nhiều phân vùng để tạo ra một phần của kết quả của biến đổi đó hay không.

Cụ thể hơn, các biến đổi hẹp là các biến đổi đơn giản trong đó mỗi dữ liệu đầu vào (hoặc mỗi phân vùng của DataFrame đầu vào) chỉ đóng góp vào một phân vùng của kết quả.

Trong biến đổi hẹp, mỗi phần tử dữ liệu đầu vào chỉ đóng góp vào một phân vùng duy nhất của kết quả, điều này làm cho quá trình biến đổi không cần phải đọc dữ liệu từ nhiều phân vùng khác nhau. Tuy nhiên, trong các biến đổi rộng, mỗi phần tử dữ liệu đầu vào có thể đóng góp vào nhiều phân vùng của kết quả, và do đó, đòi hỏi Spark phải đọc dữ liệu từ nhiều phân vùng và thực hiện các thao tác tổ chức lại dữ liệu trước khi tạo ra kết quả cuối cùng.

A diagram of a diagram

Description automatically generated

Nói cách khác, mỗi phân vùng của DataFrame đầu vào sẽ được sử dụng (riêng biệt) để tạo ra một phần cá nhân của kết quả của biến đổi . Theo một góc nhìn khác, có thể hiểu các biến đổi hẹp như những biến đổi mà Spark không cần phải đọc toàn bộ DataFrame đầu vào để tạo ra một phần nhỏ và duy nhất của kết quả .

Một ví dụ cổ điển về biến đổi hẹp là bộ lọc (filter). Ví dụ, giả sử có ba học sinh (Anne, Carls và Mike), và mỗi người đều có một túi đầy các quả bóng màu xanh, cam và đỏ. Bây giờ, giả sử yêu cầu họ thu thập tất cả các quả bóng màu đỏ từ những túi này và kết hợp chúng trong một túi duy nhất.

Để thực hiện nhiệm vụ này, Mike không cần phải biết trong túi của Carls hay Anne có những quả bóng gì. Anh ta chỉ cần thu thập các quả bóng màu đỏ chỉ có trong túi của mình. Cuối cùng, mỗi học sinh sẽ có một phần của kết quả cuối cùng (tức là tất cả các quả bóng màu đỏ trong túi của mình), và họ chỉ cần kết hợp tất cả các phần này để có được kết quả tổng cộng.

Điều tương tự áp dụng cho bộ lọc trong DataFrame của Spark. Khi lọc tất cả các hàng mà cột "state" có giá trị "Alaska", Spark sẽ lọc tất cả các hàng trong mỗi phân vùng một cách riêng biệt, và sau đó, sẽ kết hợp tất cả các kết quả để có được kết quả cuối cùng.

Ngược lại, các biến đổi rộng là ngược lại. Trong các biến đổi rộng, Spark cần sử dụng nhiều hơn 1 phân vùng của DataFrame đầu vào để tạo ra một phần nhỏ của kết quả.

A diagram of data processing

Description automatically generated

Khi loại biến đổi này xảy ra, mỗi node worker của cụm cần chia sẻ phân vùng của mình với các node khác. Nói cách khác, điều xảy ra là một quá trình chia lại phân vùng. Mỗi node worker gửi phân vùng của mình đến các node khác, để họ có thể truy cập vào nó trong khi thực hiện các nhiệm vụ được giao.

Quá trình chia lại phân vùng là một chủ đề rất phổ biến trong Apache Spark, vì nó có thể là một nguồn gốc nghiêm trọng của không hiệu quả trong ứng dụng Spark . Cụ thể, khi các quá trình chia lại phân vùng xảy ra, Spark cần ghi dữ liệu trở lại ổ cứng của máy tính, và đây không phải là một thao tác nhanh chóng. Điều này không có nghĩa là các biến đổi rộng là xấu hoặc chậm, chỉ là các quá trình chia lại phân vùng mà chúng tạo ra có thể là một vấn đề.

Một ví dụ cổ điển về hoạt động rộng là tổng hợp theo nhóm. Ví dụ, giả sử chúng ta có một DataFrame chứa doanh số bán hàng hàng ngày của nhiều cửa hàng phân tán trên khắp đất nước, và chúng ta muốn tính tổng doanh số bán hàng cho mỗi thành phố/khu vực. Để tính tổng doanh số của một thành phố cụ thể, như "São Paulo", Spark sẽ cần tìm tất cả các hàng tương ứng với thành phố này trước khi thêm các giá trị, và những hàng này có thể phân tán trên nhiều phân vùng của cụm.

## Phương thức Lọc Dataframe

Để minh họa một số ví dụ tiếp theo trong chương này, chúng ta sẽ sử dụng một DataFrame khác được gọi là transf. Dữ liệu mô tả DataFrame này là miễn phí và được cung cấp dưới dạng tệp CSV.

Dưới đây là mã để nhập dữ liệu từ tệp CSV transf.csv, để tái tạo DataFrame transf trong Phiên Spark

A screen shot of a computer

Description automatically generated

### Lọc các hàng của DataFrame

Để lọc các hàng cụ thể của DataFrame, pyspark cung cấp hai phương thức DataFrame tương đương gọi là where() và filter(). Nói cách khác, chúng đều thực hiện cùng một việc và hoạt động theo cùng một cách. Các phương thức này nhận vào một biểu thức logic biểu diễn những gì muốn lọc.

Như một ví dụ đầu tiên, giả sử muốn kiểm tra tất cả các hàng từ DataFrame transf nơi transferValue nhỏ hơn 1000. Để làm điều này, có thể sử dụng đoạn mã sau:

A black background with a yellow light

Description automatically generated with medium confidence

Việc viết biểu thức logic SQL đơn giản trong một chuỗi là cách dễ dàng và "sạch sẽ" nhất để tạo ra biểu thức lọc trong pyspark. Tuy nhiên, cũng có thể viết cùng một biểu thức một cách "pythonic" hơn, sử dụng hàm col() từ module pyspark.sql.functions.

A black screen with white text

Description automatically generated

Vẫn có một phương án verbose hơn, không yêu cầu sử dụng hàm col(). Với phương pháp này, chỉ cần tham chiếu đến cột cụ thể bằng toán tử chấm (.), như trong ví dụ dưới đây:

A black screen with red and white numbers

Description automatically generated

### Chuyển từ khóa in sang cách viết theo kiểu pythonic

Python thực sự có một khóa in giống như SQL, nhưng, khóa này không hoạt động như mong đợi trong pyspark. Để viết một biểu thức logic, sử dụng cách viết theo kiểu python, để lọc các dòng nơi một cột bằng một trong các giá trị được liệt kê, có thể sử dụng phương thức isin().

Phương thức này thuộc về lớp Column, vì vậy, luôn nên sử dụng isin() sau tên cột hoặc hàm col(). Trong ví dụ dưới đây, chúng ta đang lọc các dòng nơi destinationBankNumber là 290 hoặc 666:

A black background with colorful text

Description automatically generated

A screen shot of a computer

Description automatically generated

### Phủ định các điều kiện logic

Trong một số trường hợp, việc mô tả các dòng không muốn trong bộ lọc của mình dễ dàng hơn. Đó là, muốn phủ định (hoặc đảo ngược) biểu thức logic của mình. Đối với điều này, SQL cung cấp từ khóa not, đặt trước biểu thức logic muốn phủ định.

Ví dụ, chúng ta có thể lọc tất cả các dòng của transf nơi clientNumber không bằng 3284. Hãy nhớ, các phương thức filter() và where() là tương đương nhau (nghĩa là cả hai đều có nghĩa là cùng một điều).

A black screen with a black background

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Để dịch biểu thức này theo cách viết theo kiểu python, chúng ta sử dụng toán tử ~. Tuy nhiên, vì chúng ta đang phủ định biểu thức logic là một thể thống nhất, quan trọng phải bao quanh toàn bộ biểu thức với dấu ngoặc.

A black background with white text

Description automatically generated

### Lọc giá trị null (tức là dữ liệu bị thiếu)

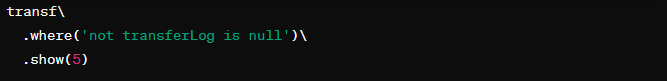
Đôi khi, các giá trị null đóng một vai trò quan trọng trong việc lọc . có thể muốn thu thập tất cả các giá trị null này để có thể điều tra tại sao chúng bị null ban đầu, hoặc, muốn loại bỏ hoàn toàn chúng khỏi DataFrame .

Bởi vì đây là một loại giá trị đặc biệt trong Spark, với một ý nghĩa đặc biệt (sự "thiếu" một giá trị), cần sử dụng cú pháp đặc biệt để lọc đúng các giá trị này trong DataFrame . Trong SQL, có thể sử dụng từ khoá is để lọc các giá trị này:

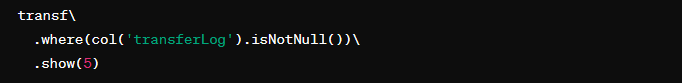
A black screen with a black border

Description automatically generated

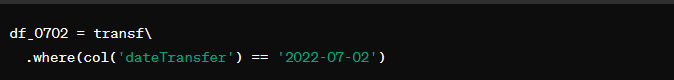
Tuy nhiên, nếu muốn loại bỏ các giá trị này khỏi DataFrame , chỉ cần phủ định (hoặc đảo ngược) biểu thức trên bằng từ khoá not, như sau:



Hoặc, có thể sử dụng isNotNull() để loại bỏ giá trị null:



### Lọc ngày và giờ trong DataFrame



A number and date on a black background

Description automatically generated

Tuy nhiên, là một phương pháp tốt hơn để viết những giá trị cụ thể này sử dụng các lớp date và datetime có sẵn của Python:

A computer screen with white text and numbers

Description automatically generated

## Phương thức quản lý cột Dataframe

Khi làm việc với DataFrame trong PySpark, sẽ thường xuyên cần thao tác với các cột, bao gồm thêm cột mới, đổi tên cột, loại bỏ cột không cần thiết và tái sắp xếp các cột. Dưới đây là một số phương pháp phổ biến để quản lý các cột của DataFrame:

### Thêm cột mới

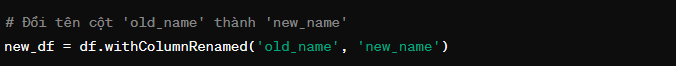
có thể thêm cột mới vào DataFrame bằng cách sử dụng phương thức `withColumn()` và truyền vào tên cột mới và giá trị hoặc biểu thức tương ứng.

A screen shot of a computer code

Description automatically generated

**Đổi tên cột**

Để đổi tên cột, có thể sử dụng phương thức `withColumnRenamed()`. Ví dụ:



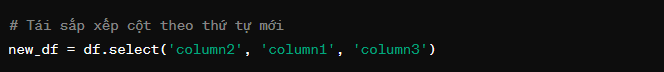
### Loại bỏ cột

Để loại bỏ cột, có thể sử dụng phương thức `drop()`. Ví dụ:



### Tái sắp xếp cột

Đôi khi, cần tái sắp xếp các cột trong DataFrame của mình. có thể sử dụng phương thức `select()` để lựa chọn các cột theo thứ tự mong muốn. Ví dụ:



### Tính toán giá trị mới

có thể tính toán giá trị mới cho một cột bằng cách sử dụng các hàm tính toán có sẵn trong PySpark. Ví dụ:

A screen shot of a computer code

Description automatically generated

### Tính toán hoặc thêm cột dựa trên điều kiện

cũng có thể tính toán hoặc thêm cột dựa trên các điều kiện có sẵn trong DataFrame. Ví dụ:

A screen shot of a computer program

Description automatically generated

## Sắp xếp các dòng của DataFrame

Khi làm việc với DataFrame trong PySpark, thường cần sắp xếp các dòng dữ liệu theo một hoặc nhiều cột cụ thể. Dưới đây là cách thực hiện điều này:

### Sắp xếp tăng dần hoặc giảm dần

có thể sắp xếp các dòng của DataFrame theo một hoặc nhiều cột cụ thể bằng cách sử dụng phương thức `orderBy()`. Ví dụ:

A computer screen shot of a black background

Description automatically generated

### Sắp xếp nhiều cột

Nếu cần, có thể sắp xếp theo nhiều cột bằng cách truyền nhiều tên cột vào phương thức `orderBy()`. Ví dụ:

A black background with white text

Description automatically generated

### Sắp xếp dựa trên điều kiện

cũng có thể sắp xếp DataFrame dựa trên một hoặc nhiều điều kiện. Ví dụ:

A black screen with white text

Description automatically generated

## Xoá các giá trị trùng lặp từ DataFrame

Khi làm việc với DataFrame trong PySpark, có thể gặp trường hợp cần xoá các hàng trùng lặp dựa trên giá trị của một hoặc nhiều cột. Dưới đây là cách thực hiện điều này:

Xoá các hàng trùng lặp dựa trên tất cả các cột

Để xoá các hàng trùng lặp dựa trên tất cả các cột, có thể sử dụng phương thức `dropDuplicates()`. Ví dụ:

A black background with white text

Description automatically generated

### Xoá các hàng trùng lặp dựa trên một hoặc nhiều cột cụ thể

Nếu chỉ muốn xoá các hàng trùng lặp dựa trên một hoặc nhiều cột cụ thể, có thể chỉ định danh sách các cột đó khi sử dụng phương thức `dropDuplicates()`. Ví dụ:

A screen shot of a computer code

Description automatically generated

## Các kỹ thuật khác để xử lý giá trị null

Trong quá trình làm việc với DataFrame trong PySpark, có một số kỹ thuật khác để xử lý giá trị null một cách hiệu quả:

### Thay thế giá trị null bằng giá trị khác

có thể thay thế các giá trị null bằng một giá trị khác bằng cách sử dụng phương thức `fillna()`. Ví dụ:

A black screen with white text

Description automatically generated

### Loại bỏ các cột chứa giá trị null

Nếu muốn loại bỏ các cột có chứa giá trị null, có thể sử dụng phương thức `drop()` hoặc chỉ định các cột cụ thể cần loại bỏ bằng cách sử dụng phương thức `dropna()`. Ví dụ:

A computer screen with white text

Description automatically generated

### Thực hiện kiểm tra tồn tại giá trị null

Đôi khi chỉ cần kiểm tra xem DataFrame có chứa giá trị null không. có thể sử dụng phương thức `isNull()` để thực hiện điều này. Ví dụ:

A screen shot of a computer code

Description automatically generated

## Các phép toán liên kết (Union)

Trong PySpark, có thể kết hợp hai hoặc nhiều DataFrame lại với nhau bằng cách sử dụng phép toán liên kết (union). Phép toán này kết hợp các hàng từ các DataFrame thành một DataFrame duy nhất.

### Sử dụng phép toán union()

có thể sử dụng phương thức `union()` để kết hợp hai DataFrame lại với nhau. Ví dụ:

A computer screen with text on it

Description automatically generated

Kết quả:

A black screen with a black background

Description automatically generated

### Lưu ý

- Hai DataFrame cần có cùng số cột và kiểu dữ liệu của các cột phải tương đương.

- Phép toán `union()` sẽ loại bỏ các hàng trùng lặp tức thì. Nếu muốn giữ lại các hàng trùng lặp, có thể sử dụng phép toán `unionAll()`. Tuy nhiên, `unionAll()` hiện đã được deprecated và nên sử dụng phương thức `union()`.

## Các phép toán nối (Join)

Trong PySpark, có thể nối hai hoặc nhiều DataFrame lại với nhau bằng cách sử dụng phép toán nối (join). Phép toán này cho phép kết hợp các dữ liệu từ các DataFrame khác nhau dựa trên các cột chung.

### Sử dụng phép toán join()

có thể sử dụng phương thức `join()` để nối hai DataFrame lại với nhau dựa trên các cột chung. Ví dụ:

A computer screen with text

Description automatically generated

Kết quả:

A black screen with a black background

Description automatically generated

### Loại nối (Join types)

Phép toán `join()` cũng cho phép chỉ định loại nối (join type), bao gồm `inner`, `outer`, `left`, và `right`. Mặc định là `inner join`. Ví dụ:

A computer screen with text

Description automatically generated

Kết quả:

A black screen with a black background

Description automatically generated

### Lưu ý

- Khi sử dụng phép toán `join()`, các cột kết nối (join columns) phải có cùng kiểu dữ liệu.

- Loại nối (join type) ảnh hưởng đến kết quả của nối, vì vậy hãy chắc chắn chọn loại nối phù hợp với yêu cầu .

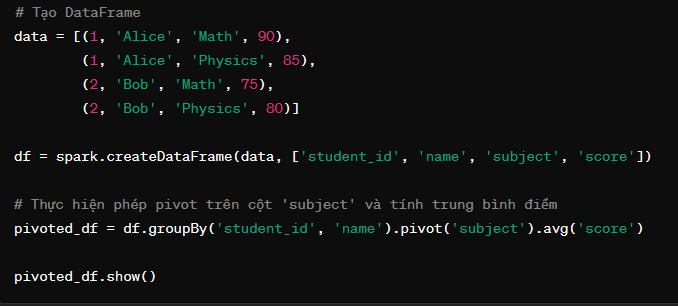
Phép toán nối là một công cụ mạnh mẽ cho phép kết hợp các dữ liệu từ nhiều nguồn khác nhau để thực hiện phân tích hoặc xây dựng các bộ dữ liệu phức tạp hơn.

## Các phép toán pivot (Pivot)

Trong PySpark, phép toán pivot cho phép chuyển đổi dữ liệu từ dạng dài (long) thành dạng rộng (wide), tạo ra các cột mới dựa trên các giá trị trong một cột cụ thể.

### Sử dụng phép toán pivot()

có thể sử dụng phương thức `pivot()` để thực hiện phép pivot trên DataFrame. Ví dụ:



Kết quả:

A black screen with red and green squares

Description automatically generated

### Lưu ý

- Phép pivot chuyển đổi các giá trị trong cột đã chọn thành các cột mới.

- Trong ví dụ trên, cột 'subject' được chuyển đổi thành các cột 'Math' và 'Physics'.

- Các giá trị mới trong các cột được tạo ra bằng cách thực hiện các hàm tổng hợp như `avg()`, `sum()`, `max()`, v.v.

Phép toán pivot là một công cụ hữu ích khi muốn biến đổi dữ liệu từ dạng dài thành dạng rộng để phân tích hoặc trực quan hóa dễ dàng hơn.

## Thu thập và phân rã

### Thu thập (Collecting)

Phép toán `collect()` được sử dụng để thu thập tất cả các phần tử của một DataFrame và trả về chúng dưới dạng danh sách Python.

A screenshot of a computer

Description automatically generated

Kết quả:

A computer code with white text

Description automatically generated with medium confidence

### Phân rã (Explode)

Phép toán `explode()` được sử dụng để phân rã các mảng hoặc cấu trúc thành các hàng đơn lẻ.

A computer screen with white text

Description automatically generatedKết quả:

A black screen with a black background

Description automatically generated

### Lưu ý

- Phép thu thập (collect) chuyển đổi DataFrame thành một danh sách Python của các đối tượng Row.

- Phép phân rã (explode) chuyển đổi một cột có cấu trúc (ví dụ: mảng) thành các hàng riêng lẻ cho mỗi phần tử trong cấu trúc đó.

- `explode()` cần phải được kết hợp với hàm `select()` để chọn các cột cần phân rã và đặt tên cho cột mới.

# Demo Cài Đặt Spark và Chạy Local PySpark

* Video Hướng Dẫn Setup Spark: <https://www.youtube.com/watch?v=49yQ-bdj4Ww>
* Link DataSet: <https://www.kaggle.com/datasets/fatihilhan/global-superstore-dataset>