Bài tập 2: Tìm hiểu Spark

1. Spark properties.

Thuộc tính Spark kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được đặt trực tiếp trên SparkConf được chuyển đến SparkContext.SparkConf cho phép bạn định cấu hình một số thuộc tính chung (ví dụ: URL chính và tên ứng dụng), cũng như các cặp khóa-giá trị tùy ý thông qua phương thức set().Ví dụ: chúng ta có thể khởi tạo một ứng dụng với hai luồng như sau:

Lưu ý rằng code chạy với local [2], nghĩa là hai luồng - thể hiện sự song song "tối thiểu", có thể giúp phát hiện lỗi chỉ tồn tại khi chúng tôi chạy trong ngữ cảnh phân tán.

```
val conf = new SparkConf()
        .setMaster("local[2]")
        .setAppName("CountingSheep")
val sc = new SparkContext(conf)
```

Lưu ý rằng ta có thể có nhiều hơn 1 luồng ở chế độ cục bộ và trong các trường hợp như Spark Streaming, chúng ta thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề nào.

Các thuộc tính chỉ định một số khoảng thời gian nên được cấu hình với một đơn vị thời gian. Định dạng sau được chấp nhận:

```
25ms (milliseconds)
5s (seconds)
10m or 10min (minutes)
3h (hours)
5d (days)
1y (years)
```

Hình 1: Thuộc tính chỉ định thời gian.

Thuộc tính chỉ định kích thước byte phải được cấu hình với đơn vị kích thước. Định dạng sau được chấp nhận:

```
1b (bytes)
1k or 1kb (kibibytes = 1024 bytes)
1m or 1mb (mebibytes = 1024 kibibytes)
1g or 1gb (gibibytes = 1024 mebibytes)
1t or 1tb (tebibytes = 1024 gibibytes)
1p or 1pb (pebibytes = 1024 tebibytes)
```

Hình 2: Thuộc tính chỉ kích thước.

Trong khi các số không có đơn vị thường được hiểu là byte, một số ít được hiểu là KiB hoặc MiB. Việc chỉ định đơn vị là mong muốn nếu có thể.

Trong một số trường hợp, bạn có thể muốn tránh mã hóa cứng các cấu hình nhất định trong a SparkConf. Ví dụ: nếu bạn muốn chạy cùng một ứng dụng với các bản gốc khác nhau hoặc số lượng bộ nhớ khác nhau. Spark cho phép bạn chỉ cần tạo một conf trống:

```
val sc = new SparkContext(new SparkConf())
```

Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:

./bin/spark-submit --name "My app" --master local[4] --conf spark.eventLog.enabled=false

--conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar

Công cụ spark-submit và trình Spark hỗ trợ hai cách để tải cấu hình động. Đầu tiên là các tùy chọn dòng lệnh, chẳng hạn như --master, như được hiển thị ở trên. spark-submit có thể chấp nhận bất kỳ thuộc tính Spark nào bằng cách sử dụng cờ --conf, nhưng sử dụng cờ đặc biệt cho các thuộc tính đóng một phần trong việc khởi chạy ứng dụng Spark. Đang chạy ./bin/spark-submit –help sẽ hiển thị toàn bộ danh sách các tùy chọn này. bin/spark-submit cũng sẽ đọc các tùy chọn cấu hình từ đó conf/spark-defaults.conf, trong đó mỗi dòng bao gồm một khóa và một giá trị được phân tách bằng khoảng trắng. Ví dụ:

```
spark.master            spark://5.6.7.8:7077
spark.executor.memory   4g
spark.eventLog.enabled  true
spark.serializer        org.apache.spark.serializer.KryoSerializer
```

Mọi giá trị được chỉ định dưới dạng cờ hoặc trong tệp thuộc tính sẽ được chuyển đến ứng dụng và được hợp nhất với những giá trị được chỉ định thông qua SparkConf. Các thuộc tính được đặt trực tiếp trên SparkConf được ưu tiên cao nhất, sau đó các cờ được chuyển đến spark-submit hoặc spark-shell, sau đó là các tùy chọn

trong tệp spark-defaults.conf. Một vài khóa cấu hình đã được đổi tên kể từ các phiên bản Spark trước đó; trong những trường hợp như vậy, các tên khóa cũ hơn vẫn được chấp nhận, nhưng được ưu tiên thấp hơn bất kỳ trường hợp nào của khóa mới hơn.

Các thuộc tính của Spark chủ yếu có thể được chia thành hai loại: một là liên quan đến triển khai, như “spark.driver.memory”, “spark.executor.instances”, loại thuộc tính này có thể không bị ảnh hưởng khi thiết lập theo chương trình SparkConftrong thời gian chạy, hoặc hành vi tùy thuộc vào trình quản lý cụm và chế độ triển khai mà bạn chọn, vì vậy bạn nên đặt thông qua tệp cấu hình hoặc spark-submit tùy chọn dòng lệnh; một cái khác chủ yếu liên quan đến kiểm soát thời gian chạy Spark, như “spark.task.maxFailures”, loại thuộc tính này có thể được đặt theo một trong hai cách.

a) Dynamically Loading Spark Properties

Trong một số trường hợp, bạn có thể muốn tránh mã hóa cứng các cấu hình nhất định trong SparkConf. Ví dụ: nếu bạn muốn chạy cùng một ứng dụng với các bản chính khác nhau hoặc số lượng bộ nhớ khác nhau. Spark cho phép bạn chỉ cần tạo một conf trống:

```
val sc = new SparkContext(new SparkConf())
```

Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:

```
./bin/spark-submit --name "My app" --master local[4] --conf spark.eventLog.enabled=false
--conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar
```

Spark shell và công cụ spark-submit hỗ trợ hai cách để tải cấu hình động. Đầu tiên là các tùy chọn dòng lệnh, chẳng hạn như --master, như hình trên. spark-submit có thể chấp nhận bất kỳ thuộc tính Spark nào sử dụng cờ --conf / -c, nhưng sử dụng cờ đặc biệt cho các thuộc tính đóng một vai trò trong việc khởi chạy ứng dụng Spark. Chạy ./bin/spark-submit --help sẽ hiển thị toàn bộ danh sách các tùy chọn này.

bin / spark-submit cũng sẽ đọc các tùy chọn cấu hình từ conf / spark-defaults.conf, trong đó mỗi dòng bao gồm một khóa và một giá trị được phân tách bằng khoảng trắng. Ví dụ:

```
spark.master            spark://5.6.7.8:7077
spark.executor.memory   4g
spark.eventLog.enabled  true
spark.serializer        org.apache.spark.serializer.KryoSerializer
```

Mọi giá trị được chỉ định dưới dạng cờ hoặc trong tệp thuộc tính sẽ được chuyển đến ứng dụng và được hợp nhất với những giá trị được chỉ định thông qua SparkConf. Các thuộc tính được đặt trực tiếp trên SparkConf được ưu tiên cao nhất, sau đó các cờ được chuyển đến spark-submit hoặc spark-shell, sau đó là các tùy chọn trong tệp spark-defaults.conf. Một vài khóa cấu hình đã được đổi tên kể từ các phiên bản Spark trước đó; trong những trường hợp như vậy, các tên khóa cũ hơn vẫn được chấp nhận, nhưng được ưu tiên thấp hơn bất kỳ trường hợp nào của khóa mới hơn.

Các thuộc tính của Spark chủ yếu có thể được chia thành hai loại: một là liên quan đến triển khai, như “spark.driver.memory”, “spark.executor.instances”, loại thuộc tính này có thể không bị ảnh hưởng khi thiết lập lập trình thông qua SparkConf trong thời gian chạy, hoặc hành vi tùy thuộc vào trình quản lý cụm và chế độ triển khai

bạn chọn, vì vậy bạn nên đặt thông qua tệp cấu hình hoặc tùy chọn dòng lệnh spark-submit; một loại khác chủ yếu liên quan đến kiểm soát thời gian chạy Spark, như “spark.task.maxFailures”, loại thuộc tính này có thể được đặt theo một trong hai cách.

b) Viewing Spark Properties

Giao diện người dùng web ứng dụng tại http: // <driver>: 4040 liệt kê các thuộc tính Spark trong tab "Môi trường". Đây là một nơi hữu ích để kiểm tra để đảm bảo rằng các thuộc tính của bạn đã được đặt chính xác. Lưu ý rằng chỉ các giá trị được chỉ định rõ ràng thông qua spark-defaults.conf, SparkConf hoặc dòng lệnh mới xuất hiện. Đối với tất cả các thuộc tính cấu hình khác, bạn có thể giả sử giá trị mặc định được sử dụng.

c) Available Properties

Hầu hết các thuộc tính kiểm soát cài đặt nội bộ đều có giá trị mặc định hợp lý:

d) Environment Variables

Một số cài đặt Spark nhất định có thể được định cấu hình thông qua các biến môi trường, được đọc từ tập lệnh conf / spark-env.sh trong thư mục nơi Spark được cài đặt (hoặc conf / spark-env.cmd trên Windows). Ở chế độ Độc lập và Mesos, tệp này có thể cung cấp thông tin cụ thể cho máy như tên máy chủ. Nó cũng có nguồn gốc khi chạy các ứng dụng Spark cục bộ hoặc các tập lệnh gửi.

Lưu ý rằng conf / spark-env.sh không tồn tại theo mặc định khi Spark được cài đặt. Tuy nhiên, bạn có thể sao chép conf / spark-env.sh.template để tạo nó. Đảm bảo rằng bạn thực thi bản sao.

Vì spark-env.sh là một tập lệnh shell, một số trong số này có thể được đặt theo chương trình - ví dụ: bạn có thể tính SPARK_LOCAL_IP bằng cách tra cứu IP của một giao diện mạng cụ thể.

Lưu ý: Khi chạy Spark trên YARN ở chế độ cụm, các biến môi trường cần được đặt bằng thuộc tính spark.yarn.appMasterEnv. [EnvironmentVariableName] trong tệp conf / spark-defaults.conf của bạn. Các biến môi trường được đặt trong spark-env.sh sẽ không được phản ánh trong quy trình YARN Application Master ở chế độ cụm. Xem Thuộc tính Spark liên quan đến YARN để biết thêm thông tin.

e) Configuring Logging

Spark sử dụng log4j để ghi nhật ký. Bạn có thể định cấu hình nó bằng cách thêm tệp log4j. properties vào thư mục conf. Một cách để bắt đầu là sao chép log4j.properties.template hiện có ở đó.

2. Spark RDD

Resilient Distributed Datasets (RDD) là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp phân tán bất biến của các đối tượng. Mỗi tập dữ liệu trong RDD được chia thành các phân vùng logic, có thể được tính toán trên các nút khác nhau của cụm. RDD có thể chứa bất kỳ loại đối tượng Python, Java hoặc Scala nào, bao gồm các lớp do người dùng định nghĩa.

Về mặt hình thức, RDD là một tập hợp các bản ghi được phân vùng, chỉ đọc. RDD có thể được tạo thông qua các hoạt động xác định trên dữ liệu trên bộ lưu trữ ổn định hoặc các RDD khác. RDD là một tập hợp các phần tử chịu được lỗi có thể hoạt động song song.

Có hai cách để tạo RDD - song song tập hợp hiện có trong chương trình trình điều khiển của bạn hoặc tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài, chẳng hạn như hệ thống tệp chia sẻ, HDFS, HBase hoặc bất kỳ nguồn dữ liệu nào cung cấp Định dạng đầu vào Hadoop.

Spark sử dụng khái niệm RDD để đạt được các hoạt động MapReduce nhanh hơn và hiệu quả hơn. Trước tiên, chúng ta hãy thảo luận về cách các hoạt động MapReduce diễn ra và tại sao chúng không hiệu quả như vậy.

a) Chia sẻ dữ liệu bằng Spark RDD

Chia sẻ dữ liệu chậm trong MapReduce do sao chép, tuần tự hóa và IO đĩa . Hầu hết các ứng dụng Hadoop, chúng dành hơn 90% thời gian để thực hiện các thao tác đọc-ghi HDFS.

Nhận thức được vấn đề này, các nhà nghiên cứu đã phát triển một framework chuyên biệt có tên là Apache Spark. Ý tưởng chính của tia lửa

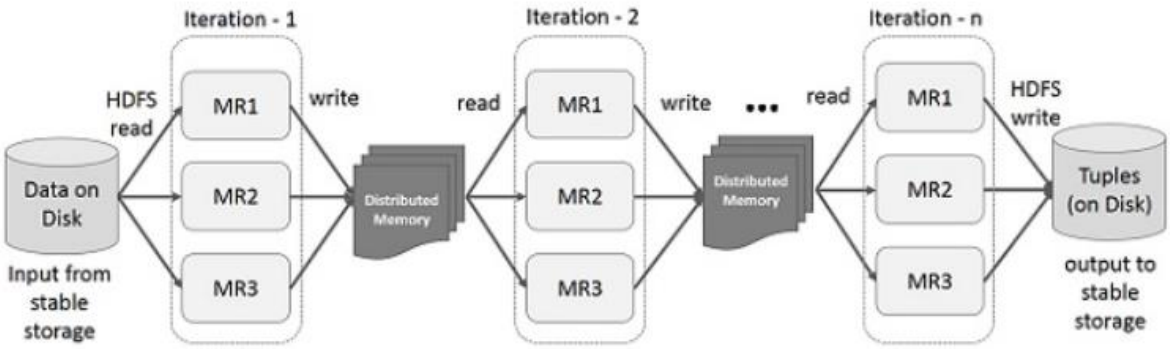
là R esilient D istributed D atasets (RDD); nó hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa là, nó lưu trữ trạng thái bộ nhớ như một đối tượng trên các công việc và đối tượng có thể chia sẻ giữa các công việc đó. Chia sẻ dữ liệu trong bộ nhớ nhanh hơn mạng và Đĩa từ 10 đến 100 lần.

Bây giờ chúng ta hãy thử tìm hiểu cách các hoạt động lặp lại và tương tác diễn ra trong Spark RDD.

b) Hoạt động lặp lại trên Spark RDD

Hình minh họa dưới đây cho thấy các hoạt động lặp lại trên Spark RDD. Nó sẽ lưu trữ các kết quả trung gian trong một bộ nhớ phân tán thay vì Ổ lưu trữ ổn định (Đĩa) và làm cho hệ thống nhanh hơn.

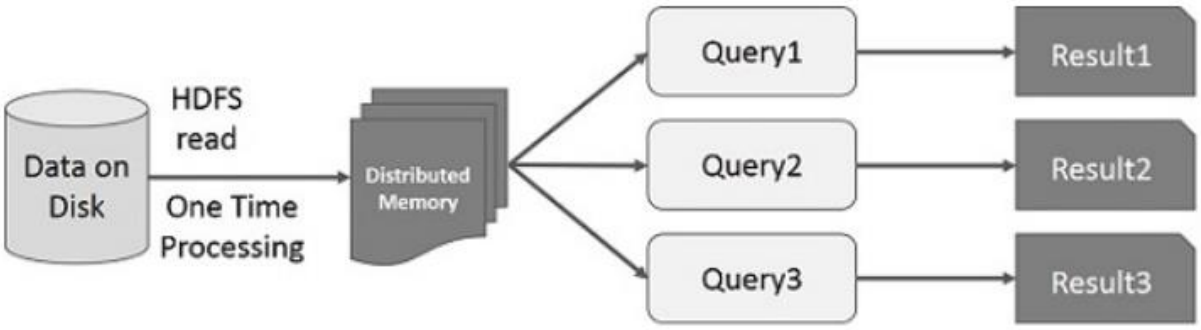
Lưu ý - Nếu bộ nhớ phân tán (RAM) không đủ để lưu trữ các kết quả trung gian (Trạng thái công việc), thì nó sẽ lưu các kết quả đó trên đĩa.



Hình 3: Hoạt động lặp lại trên Spark RDD.

c) Hoạt động tương tác trên Spark RDD

Hình minh họa này cho thấy các hoạt động tương tác trên Spark RDD. Nếu các truy vấn khác nhau được chạy lặp lại trên cùng một tập dữ liệu, thì dữ liệu cụ thể này có thể được lưu trong bộ nhớ để có thời gian thực thi tốt hơn.



Hình 4: Hoạt động tương tác trên Spark RDD.

Theo mặc định, mỗi RDD đã chuyển đổi có thể được tính toán lại mỗi khi bạn chạy một hành động trên đó. Tuy nhiên, bạn cũng có thể duy trì một RDD trong bộ nhớ, trong trường hợp đó Spark sẽ giữ các phần tử xung quanh trên cụm để truy cập nhanh hơn nhiều, vào lần tiếp theo bạn truy vấn nó. Ngoài ra còn có hỗ trợ cho các RDD lâu dài trên đĩa hoặc được sao chép qua nhiều nút.

Bộ sưu tập song song được tạo ra bằng cách gọi SparkContext của parallelize phương pháp trên một bộ sưu tập iterable hoặc hiện tại trong chương trình của bạn. Các phần tử của bộ sưu tập được sao chép để tạo thành một tập dữ liệu phân tán có thể được vận hành song song. Ví dụ: đây là cách tạo một tập hợp song song chứa các số từ 1 đến 5:

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

Hoặc có thể đọc dữ liệu từ một file bên ngoài như:

```
distFile = sc.textFile("data.txt")
```

Một số lưu ý khi đọc tệp với Spark:

- Nếu sử dụng một đường dẫn trên hệ thống tệp cục bộ, tệp cũng phải có thể truy cập được tại cùng một đường dẫn trên các nút công nhân. Sao chép tệp cho tất cả công nhân hoặc sử dụng hệ thống tệp chia sẻ được gắn kết trên mạng.

- Tất cả các phương thức nhập dựa trên tệp của Spark, bao gồm textFile, hỗ trợ chạy trên thư mục, tệp nén và cả ký tự đại diện. Ví dụ, bạn có thể sử dụng textFile("/my/directory"), textFile("/my/directory/*.txt") và textFile("/my/directory/*.gz").

- Các phương pháp textFile cũng có một đối số tùy chọn thứ hai để kiểm soát số lượng các phân vùng của tập tin. Theo mặc định, Spark tạo một phân vùng cho mỗi khối của tệp (các khối là 128MB theo mặc định trong HDFS), nhưng bạn cũng có thể yêu cầu số lượng phân vùng cao hơn bằng cách chuyển một giá trị lớn hơn. Lưu ý rằng bạn không thể có ít phân vùng hơn khối.

Ngoài các tệp văn bản, API Python của Spark cũng hỗ trợ một số định dạng dữ liệu khác:

- SparkContext.wholeTextFiles cho phép bạn đọc một thư mục chứa nhiều tệp văn bản nhỏ và trả về mỗi tệp dưới dạng cặp (tên tệp, nội dung). Điều này trái ngược với textFile, sẽ trả về một bản ghi trên mỗi dòng trong mỗi tệp.

- RDD.saveAsPickleFile và SparkContext.pickleFile hỗ trợ lưu RDD ở một định dạng đơn giản bao gồm các đối tượng Python có sẵn. Lô hàng được sử dụng trong tuần tự hóa dưa chua, với kích thước lô mặc định là 10.
- SequenceFile và các định dạng đầu vào / đầu ra Hadoop

3. Spark DataFrame.

Có thể chuyện đổi đổi các dạng dữ liệu như Hive , CSV, Json, RDBMS, XML,Parquet, Cassandra, RDDs thông qua Spark sql để cho vào DataFrame trong Spark

Spark SQL là một mô-đun Spark để xử lý dữ liệu có cấu trúc. Không giống như API Spark RDD cơ bản, các giao diện do Spark SQL cung cấp cung cấp cho Spark nhiều thông tin hơn về cấu trúc của cả dữ liệu và tính toán đang được thực hiện. Bên trong, Spark SQL sử dụng thông tin bổ sung này để thực hiện các tối ưu hóa bổ sung. Có một số cách để tương tác với Spark SQL bao gồm SQL và API tập dữ liệu. Khi tính toán một kết quả, cùng một công cụ thực thi được sử dụng, không phụ thuộc vào API / ngôn ngữ nào bạn đang sử dụng để diễn đạt tính toán. Sự thống nhất này có nghĩa là các nhà phát triển có thể dễ dàng chuyển đổi qua lại giữa các API khác nhau, dựa trên đó cung cấp cách tự nhiên nhất để thể hiện một chuyển đổi nhất định.

DataFrame là một tập hợp dữ liệu phân tán, được tổ chức thành các cột được đặt tên. Về mặt khái niệm, nó tương đương với các bảng quan hệ có kỹ thuật tối ưu hóa tốt.

Một DataFrame có thể được xây dựng từ một loạt các nguồn khác nhau như bảng Hive, tệp Dữ liệu có cấu trúc, cơ sở dữ liệu bên ngoài hoặc RDD hiện có. API này được thiết kế cho các ứng dụng Khoa học dữ liệu và Dữ liệu lớn hiện đại lấy cảm hứng từ DataFrame trong Lập trình R và Gấu trúc trong Python.

Dưới đây là một số tính năng đặc trưng của DataFrame:

- Khả năng xử lý dữ liệu có kích thước từ Kilobyte đến Petabyte trên một cụm nút đơn đến cụm lớn.

- Hỗ trợ các định dạng dữ liệu khác nhau (Avro, csv, tìm kiếm đàn hồi và Cassandra) và hệ thống lưu trữ (HDFS, bảng HIVE, mysql, v.v.).
- Tối ưu hóa hiện đại và tạo mã thông qua trình tối ưu hóa Spark SQL Catalyst (khung chuyển đổi cây).
- Có thể dễ dàng tích hợp với tất cả các công cụ và khuôn khổ Dữ liệu lớn thông qua Spark-Core.
- Cung cấp API cho Lập trình Python, Java, Scala và R.

Điểm khởi đầu để lập trình Spark với API Dataset và DataFrame.

Một SparkSession có thể được sử dụng để tạo DataFrame, đăng ký DataFramedưới dạng bảng, thực thi SQL qua bảng, bảng bộ nhớ cache và đọc tệp parquet. Để tạo một SparkSession, hãy sử dụng mẫu trình tạo sau:

```
spark = SparkSession.builder
    .master("local")

    .appName("Word Count")

    .config("spark.some.config.option", "some-value")
    .getOrCreate()
```

Builder là một thuộc tính lớp để tạo các SparkSession.

appName(name) là đặt tên cho ứng dụng, tên này sẽ được hiển thị trong giao diện người dùng web Spark. Nếu không có tên ứng dụng nào được đặt, tên được tạo ngẫu nhiên sẽ được sử dụng.

config(key,value,conf) đặt tùy chỉnh cấu hình Các tùy chọn được thiết lập bằng cách sử dụng phương pháp này được tự động truyền đến cấu hình riêng của cả SparkConf và SparkSession.

```
{
  {"id" : "1201", "name" : "satish", "age" : "25"}
  {"id" : "1202", "name" : "krishna", "age" : "28"}
  {"id" : "1203", "name" : "amith", "age" : "39"}
  {"id" : "1204", "name" : "javed", "age" : "20"}
  {"id" : "1205", "name" : "prudvi", "age" : "19"}
}
```

DataFrame cung cấp DataFrame cung cấp một ngôn ngữ dành riêng cho miền để thao tác dữ liệu có cấu trúc. Ở đây, chúng tôi bao gồm một số ví dụ cơ bản về xử lý dữ liệu có cấu trúc bằng DataFrames.

Đọc tài liệu JSON

- Đầu tiên, chúng ta phải đọc tài liệu JSON. Dựa trên điều này, tạo một DataFrame có tên (dfs). Sử dụng lệnh sau để đọc tài liệu JSON có tên là worker.json. Dữ liệu được hiển thị dưới dạng bảng với các trường - id, tên và tuổi. bằng câu lệnh

Câu lệnh dùng để đọc dữ liệu từ file people.json phía trên

```
df = spark.read.json("examples/src/main/resources/people.json")
```

Để hiện dữ liệu ra ta dùng câu lệnh bên dưới để thể hiện ra dữ liệu sau khi đọc

```
df.show()
```

```
+----+------+--------+
|age | id   |  name  |
+----+------+--------+
| 25 | 1201 | satish |
| 28 | 1202 | krishna|
| 39 | 1203 | amith  |
| 23 | 1204 | javed  |
| 23 | 1205 | prudvi |
+----+------+--------+
```

Kết quả ta thu được như bên trên

Nếu muốn chọn 1 cột trong dataFrame mà ta có ta sẽ dùng lệnh như sau

```
dfs.select("name").show()
```

Và kết quả ta sẽ được như sau

```
+--------+
|  name  |
+--------+
| satish |
| krishna|
| amith  |
| javed  |
| prudvi |
+--------+
```

Ngoài chọn 1 cột ta có thể lọc theo điều kiện với mỗi cột trong data frame

```
df.filter(df['age'] > 21).show()
```

Như câu điều kiện này ta sẽ lọc bảng với cột tuổi có tuổi lớn hơn 21 từ DataFrame ban đầu ta sẽ có kết quả như sau:

```
+----+------+--------+
|age | id   | name   |
+----+------+--------+
| 25 | 1201 | satish |
| 28 | 1202 | krishna|
| 39 | 1203 | amith  |
+----+------+--------+
```

Ngoài ra ta có thể thực hiện GroupBy và đếm các phần tử của từng cột bằng câu lệnh sau

```
df.groupBy("age").count().show()
```

Câu lệnh sẽ nhóm tuổi và đếm số lượng tuổi xuất hiện trong DataFrame

```
+----+-----+
|age |count|
+----+-----+
| 23 |  2  |
| 25 |  1  |
| 28 |  1  |
| 39 |  1  |
+----+-----+
```

Ta có thể tao tác với các dữ liệu trên dataFrame và lọc tùy theo các yêu cầu được đặt ra

```
sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
```

Ngoài ra Spark có thể thao tác với các câu truy vấn của SQL bằng phương thức **spark.sql(“ câu lệnh truy vấn trong SQL”)**

TÀI LIỆU THAM KHẢO

https://spark.apache.org/docs/2.3.0/configuration.html

https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds

https://spark.apache.org/docs/latest/sql-programming-guide.html

https://www.tutorialspoint.com/spark_sql/spark_sql_dataframes.htm