

Tìm hiểu

***1.Spark Properties:***

Spark properties 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 cài đặt trực tiếp trên SparkConf và được chuyển tới SparkContext của bạn. 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 () .

**Spark Properties** bao gồm :

Dynamically Loading Spark Properties

* Viewing Spark Properties
* Available Properties
* Application Properties
  + Runtime Environment
  + Shuffle Behavior
  + Spark UI
  + Compression and Serialization
  + Memory Management
  + Execution Behavior
  + Executor Metrics
  + Networking
  + Scheduling
  + Barrier Execution Mode
  + Dynamic Allocation
  + Thread Configurations
  + Security
  + Spark SQL
    - Runtime SQL Configuration
    - Static SQL Configuration
  + Spark Streaming
  + SparkR
  + GraphX
  + Deploy
  + Cluster Managers
    - YARN
    - Mesos
    - Kubernetes
    - Standalone Mode
* Environment Variables
* Configuring Logging
* Overriding configuration directory
* Inheriting Hadoop Cluster Configuration
* Custom Hadoop/Hive Configuration
* Custom Resource Scheduling and Configuration Overview

Spark cung cấp 3 vị trí để tinh chỉnh hệ thống của bạn như :

Spark properties kiểm soát hầu hết các tham số ứng dụng và có thể được thiết lập bằng cách sử dụng đối tượng SparkConf hoặc thông qua các thuộc tính hệ thống Java.

Các biến môi trường có thể được sử dụng để đặt cài đặt cho mỗi máy, chẳng hạn như địa chỉ IP, thông qua tập lệnh conf / spark-env.sh trên mỗi node.

Logging có thể được định cấu hình thông qua log4j.properties.

***2.*** ***Apache Spark RDD:***

Resilient Distributed Datasets (RDD) là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phần vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy chủ (cluster).

RDDs có thể chứa bất kỳ kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa. Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra qua điều khiển xác định trên dữ liệu trong bộ nhớ hoặc RDDs, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Có hai cách để tạo RDDs:

* Tạo từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala.
* Lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase hoặc các cơ sở dữ liệu quan hệ.

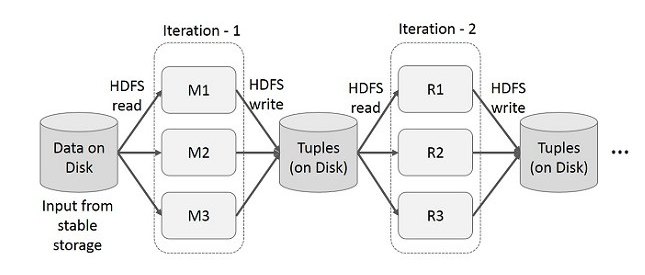
**Thực thi trên MapRedure**

MapReduce được áp dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với thuật toán xử lý phân tán song song trên một cụm. Nó cho phép người dùng viết các tính toán song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về xử lý/phân phối công việc và khả năng chịu lỗi.

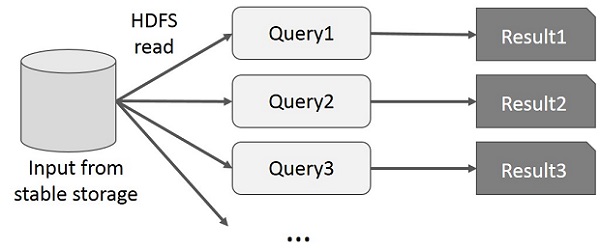
Tuy nhiên, trong hầu hết các framework hiện tại, cách duy nhất để sử dụng lại dữ liệu giữa các tính toán (Ví dụ: giữa hai công việc MapReduce) là ghi nó vào storage (Ví dụ: HDFS). Mặc dù framework này cung cấp nhiều hàm thư viện để truy cập vào tài nguyên tính toán của cụm Cluster, điều đó vẫn là chưa đủ.

Cả hai ứng dụng Lặp (Iterative) và Tương tác (Interactive) đều yêu cầu chia sẻ truy cập và xử lý dữ liệu nhanh hơn trên các công việc song song. Chia sẻ dữ liệu chậm trong MapReduce do sao chép tuần tự và tốc độ I/O của ổ đĩa. Về hệ thống lưu trữ, hầu hết các ứng dụng Hadoop, cần dành hơn 90% thời gian để thực hiện các thao tác đọc-ghi HDFS.

* Iterative Operation trên MapReduce:



* Interactive Operations trên MapReduce:



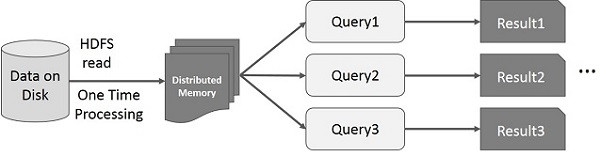
**Thực thi trên Spark RDD**

Để khắc phục được vấn đề về MapRedure, các nhà nghiên cứu đã phát triển một framework chuyên biệt gọi là Apache Spark. Ý tưởng chính của Spark là Resilient Distributed Datasets (RDD); nó hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa, nó lưu trữ trạng thái của bộ nhớ dưới dạng một đối tượng trên các công việc và đối tượng có thể chia sẻ giữa các công việc đó. Việc xử lý dữ liệu trong bộ nhớ nhanh hơn 10 đến 100 lần so với network và disk.

* Iterative Operation trên Spark RDD:



* Interactive Operations trên Spark RDD:



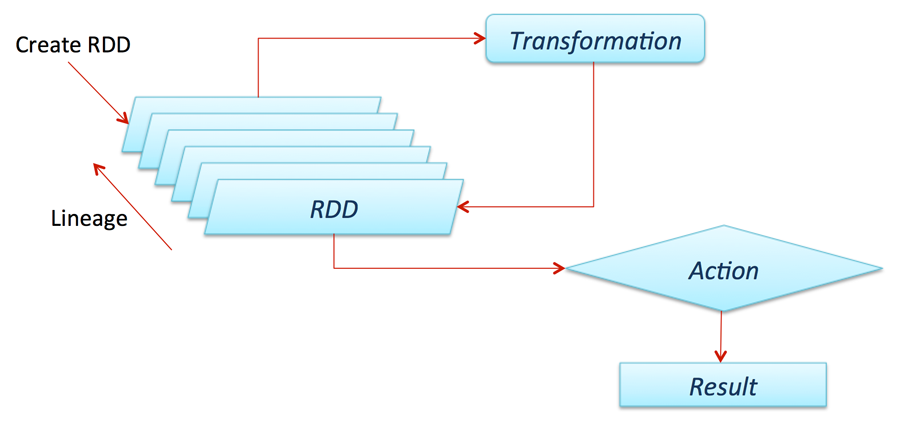
**Các loại RDD :**



* Các RDD biểu diễn một tập hợp cố định, đã được phân vùng các record để có thể xử lý song song.
* Các record trong RDD có thể là đối tượng Java, Scale hay Python tùy lập trình viên chọn. Không giống như DataFrame, mỗi record của DataFrame phải là một dòng có cấu trúc chứa các field đã được định nghĩa sẵn.
* RDD đã từng là API chính được sử dụng trong series Spark 1.x và vẫn có thể sử dụng trong version 2.X nhưng không còn được dùng thường xuyên nữa.
* RDD API có thể được sử dụng trong Python, Scala hay Java:
  + Scala và Java: Perfomance tương đương trên hầu hết mọi phần. (Chi phí lớn nhất là khi xử lý các raw object)
  + Python: Mất một lượng performance, chủ yếu là cho việc serialization giữa tiến trình Python và JVM

**Các transformation và action với RDD**

RDD cung cấp các transformation và action hoạt động giống như DataFrame lẫn DataSets. Transformation xử lý các thao tác lazily và Action xử lý thao tác cần xử lý tức thời.

**Một số transformation:**

Nhiều phiên bản transformation của RDD có thể hoạt động trên các Structured API, **transformation** xử lý lazily, tức là chỉ giúp dựng execution plans, dữ liệu chỉ được truy xuất thực sự khi thực hiện **action**

* distinct: loại bỏ trùng lắp trong RDD
* filter: tương đương với việc sử dụng where trong SQL – tìm các record trong RDD xem những phần tử nào thỏa điều kiện. Có thể cung cấp một hàm phức tạp sử dụng để filter các record cần thiết – Như trong Python, ta có thể sử dụng hàm lambda để truyền vào filter
* map: thực hiện một công việc nào đó trên toàn bộ RDD. Trong Python sử dụng lambda với từng phần tử để truyền vào map
* flatMap: cung cấp một hàm đơn giản hơn hàm map. Yêu cầu output của map phải là một structure có thể lặp và mở rộng được.
* sortBy: mô tả một hàm để trích xuất dữ liệu từ các object của RDD và thực hiện sort được từ đó.
* randomSplit: nhận một mảng trọng số và tạo một random seed, tách các RDD thành một mảng các RDD có số lượng chia theo trọng số.

**Một số action :**

Action thực thi ngay các transformation đã được thiết lập để thu thập dữ liệu về driver để xử lý hoặc ghi dữ liệu xuống các công cụ lưu trữ.

* reduce: thực hiện hàm reduce trên RDD để thu về 1 giá trị duy nhất
* count: đếm số dòng trong RDD
* countApprox: phiên bản đếm xấp xỉ của count, nhưng phải cung cấp timeout vì có thể không nhận được kết quả.
* countByValue: đếm số giá trị của RDD
* chỉ sử dụng nếu map kết quả nhỏ vì tất cả dữ liệu sẽ được load lên memory của driver để tính toán
* chỉ nên sử dụng trong tình huống số dòng nhỏ và số lượng item khác nhau cũng nhỏ.
* countApproxDistinct: đếm xấp xỉ các giá trị khác nhau
* countByValueApprox: đếm xấp xỉ các giá trị
* first: lấy giá trị đầu tiên của dataset
* max và min: lần lượt lấy giá trị lớn nhất và nhỏ nhất của dataset
* take và các method tương tự: lấy một lượng giá trị từ trong RDD. take sẽ trước hết scan qua một partition và sử dụng kết quả để dự đoán số lượng partition cần phải lấy thêm để thỏa mãn số lượng lấy.
* top và takeOrdered: top sẽ hiệu quả hơn takeOrdered vì top lấy các giá trị đầu tiên được sắp xếp ngầm trong RDD.
* takeSamples: lấy một lượng giá trị ngẫu nhiên trong RDD

**Một số kỹ thuật đối với RDD**

- Lưu trữ file:

* Thực hiện ghi vào các file plain-text
* Có thể sử dụng các codec nén từ thư viện của Hadoop
* Lưu trữ vào các database bên ngoài yêu cầu ta phải lặp qua tất cả partition của RDD – Công việc được thực hiện ngầm trong các high-level API
* sequenceFile là một flat file chứa các cặp key-value, thường được sử dụng làm định dạng input/output của MapReduce. Spark có thể ghi các sequenceFile bằng các ghi lại các cặp key-value
* Đồng thời, Spark cũng hỗ trợ ghi nhiều định dạng file khác nhau, cho phép define các class, định dạng output, config và compression scheme của Hadoop.

**Caching: Tăng tốc xử lý bằng cache**

* Caching với RDD, Dataset hay DataFrame có nguyên lý như nhau.
* Chúng ta có thể lựa chọn cache hay persist một RDD, và mặc định, chỉ xử lý dữ liệu trong bộ nhớ

**Checkpointing: Lưu trữ lại các bước xử lý để phục hồi**

* Checkpointing lưu RDD vào đĩa cứng để các tiến trình khác để thể sử dụng lại RDD point này làm partition trung gian thay vì tính toán lại RDD từ các nguồn dữ liệu gốc
* Checkpointing cũng tương tự như cache, chỉ khác nhau là lưu trữ vào đĩa cứng và không dùng được trong API của DataFrame
* Cần sử dụng nhiều để tối ưu tính toán.

***3.Spark DataFrame:***

**Spark DataFrame là gì**

Khi Spark ver 1.3 ra đời, Spark đã đẻ thêm tính năng có tên là **Spark DataFrame** .

Nó có thể thiết lập Schema cho Spark RDD và có thể tạo Object DataFrame.

**Vậy Spark DataFrame có gì hơn so với RDD ?**

1. Giống như viết SQL, đầy đủ chức năng như select, where ... đặc biệt là join với các DataFrame khác.
2. Sử dụng các method như filter, select để trích xuất dữ liệu theo cột, hàng.
3. Xử gọn các loại data như Log ... với groupBy → agg
4. Thêm 1 cột dễ dàng với UDF(User Defined Function)
5. Giống như SQL, Spark DataFrame đã hỗ trợ Pivot (Spark 1.6 trở lên) rất hữu ích cho việc lập bảng biểu, báo cáo.

Tóm lại là dễ xài, đơn giản hơn RDD mà hiệu suất, khả năng optimized truy vấn tốt hơn RDD.

Chính vì vậy với các trường hợp thông thường, các bạn nên xài DataFrame.

**Tạo DataFrame như thế nào ?**

Giả sử chúng ta có cấu trúc file Log bao gồm 3 cột : Date, User\_ID, Campaign\_ID



**Cách 1: Tạo từ RDD**

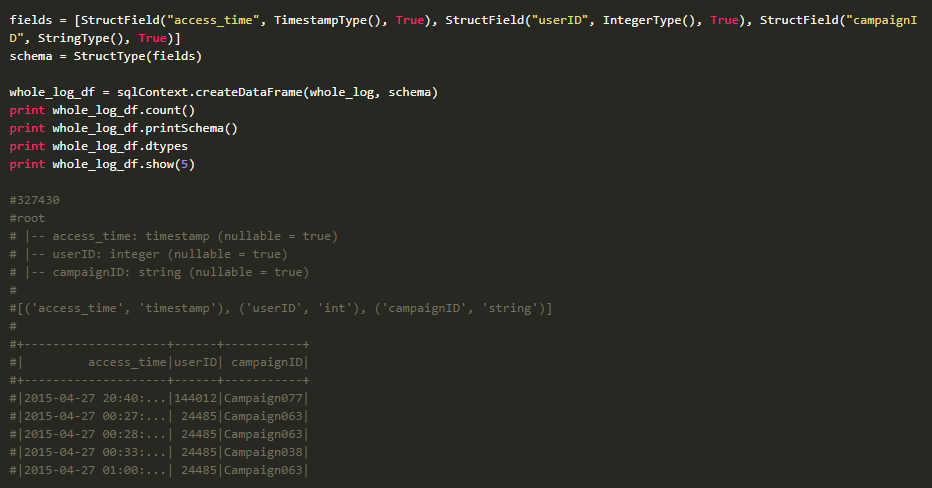
Nếu bạn đã có RDD với tên column và type tương ứng (TimestampType, IntegerType, StringType)thì bạn có thể dễ dàng tạo DataFrame bằng :



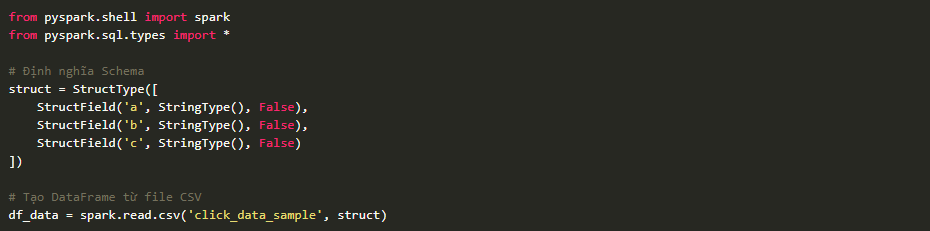
Với *printSchema() ,* *dtypes* sẽ in thông tin của schema

và *count()* trả về số record

Và nếu chỉ muốn in n record đầu tiên thì sử dụng *show(n)*

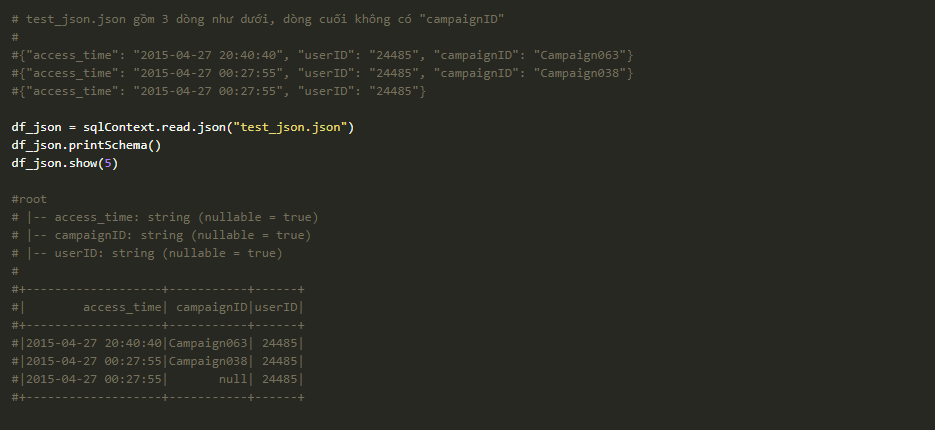


**Cách 2: Tạo trực tiếp từ file CSV**



**Cách 3: Giao lưu trực tiếp từ file json**

Bằng cách sử dụng *sqlContext.read.json*. Mỗi dòng của file json sẽ được coi là 1 object. Trong trường hợp object thiếu data thì sẽ null tại đó.



**Cách 4: Giao thông trực tiếp từ parquet**

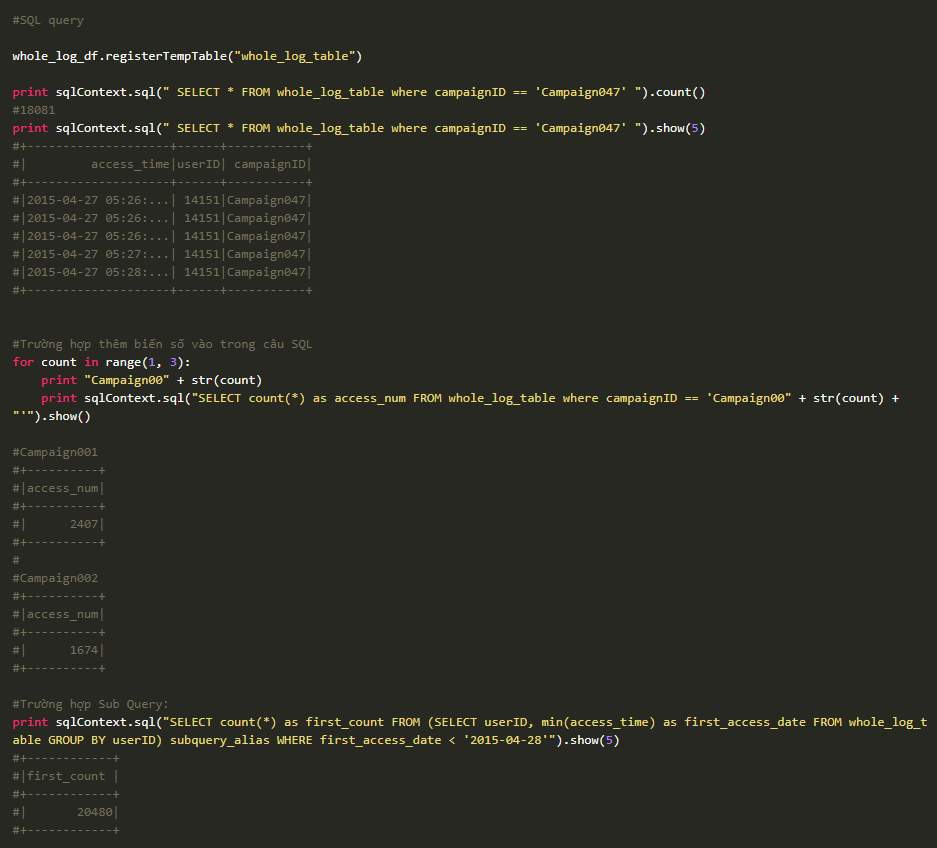
Sử dụng *sqlContext.read.parquet*



**Thao Tác Với DataFrame :**

***1. Query bằng SQL***

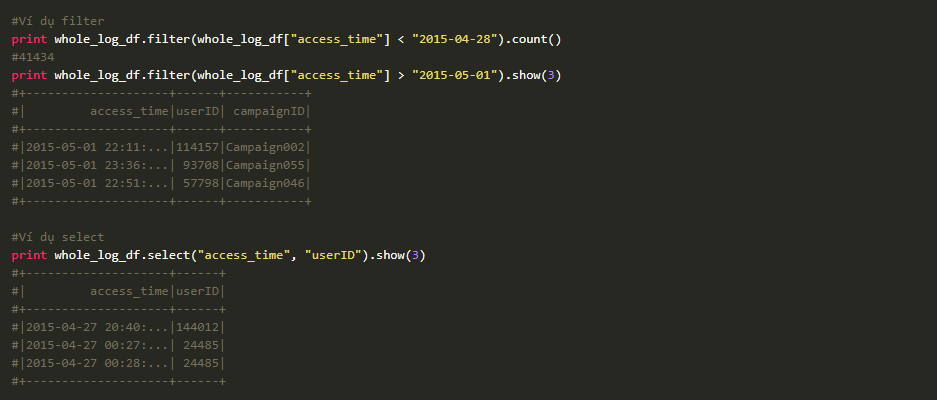
Bằng cách sử dụng registerTempTable, bạn sẽ có một table được tham chiếu đến Dataframe đó, bạn có thể sử dụng tên table này để viết query SQL. Nếu bạn sử dụng sqlContext.sql('query SQL') thì giá trị trả về cũng là Dataframe.



***2. Tìm kiếm sử dụng filter, select***

Đối với DataFrame , tìm kiếm kèm điều kiện rất đơn giản. Giống với câu query ở trên nhưng filter, select dễ dàng hơn rất nhiều. Vậy **filter** và **select** khác nhau thế nào ?

Cùng là để tìm kiếm nhưng filter trả về những row thoả mãn điều kiện, trong đó select lấy dữ liệu theo column.



***3.*** ***Sử dụng groupBy***

GroupBy có chức năng giống với reduceByKey của RDD, nhưng nó còn cung cấp 1 rổ method. Ở đây mình sẽ run thử code count, agg.

***groupBy→count***

Ví dụ sau sẽ lấy key là **campaignID** và tiến hành **groupBy**. Sau đó dùng **count()** để lấy số record ứng với mỗi key.



***groupBy→pivot***

Pivot thì từ Spark v1.6 trở lên được đưa vào, có chức năng giống với pivot trong SQL. Thử áp dụng xem sao nhỉ :

Trước khi pivot (agged\_df)

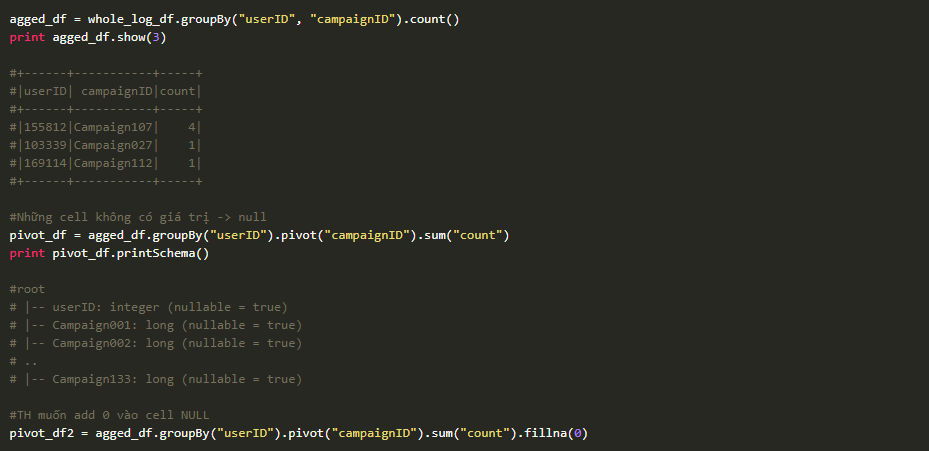
Số hàng = số UserID(=75,545) \* campainID(=133)

Số cột = 3

Sau khi pivot(pivot\_df)

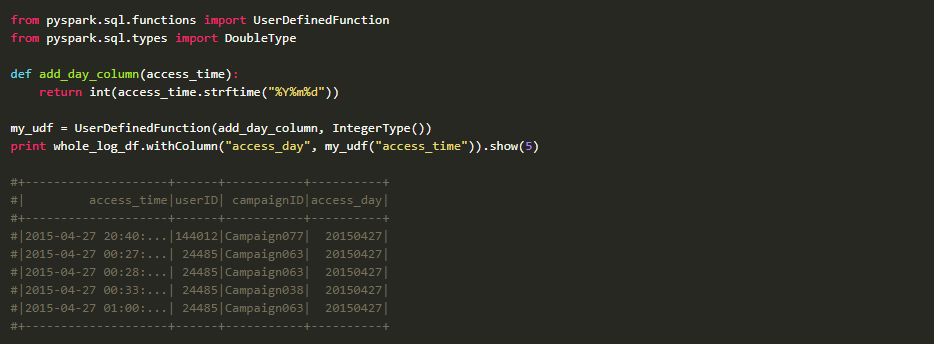
Số hàng = số UserID(=75,545)

Số cột = UserID + số CampainID = 1 + 133 = 134

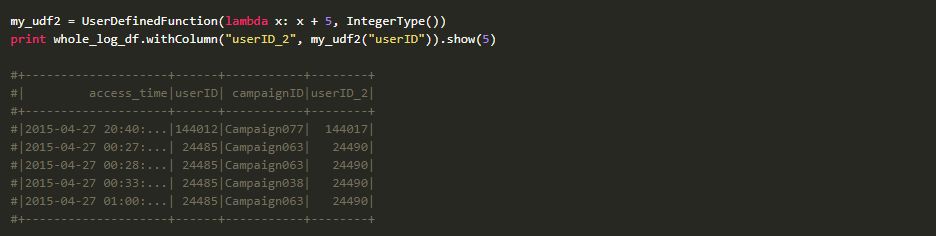


***4.*** ***Thêm cột sử dụng UDF***

Trong Spark DataFrame có thể sử dụng UDF, với ứng dụng chính là thêm cột. Như đã nói ở trên, bản chất DataFrame là immutable nên khi thêm cột thì 1 DataFrame mới sẽ được tạo.



*Cũng có thể sử dụng**lambda*



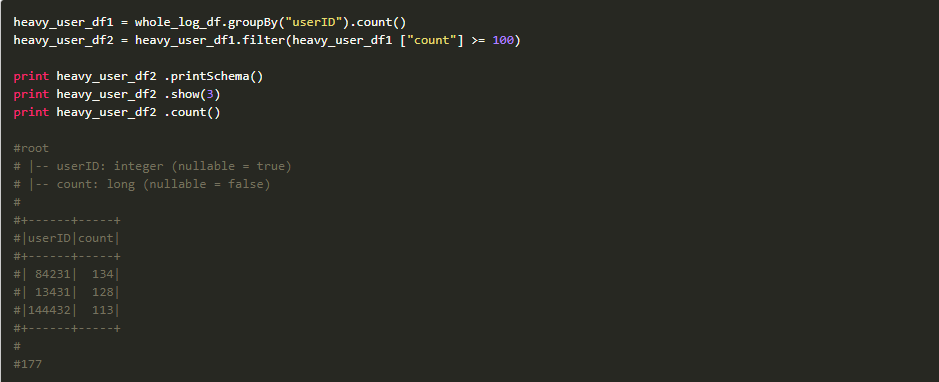
Ngược lại, muốn xóa cột thì sử dụng df.drop()



***5.*** ***Join 2 DataFrame***

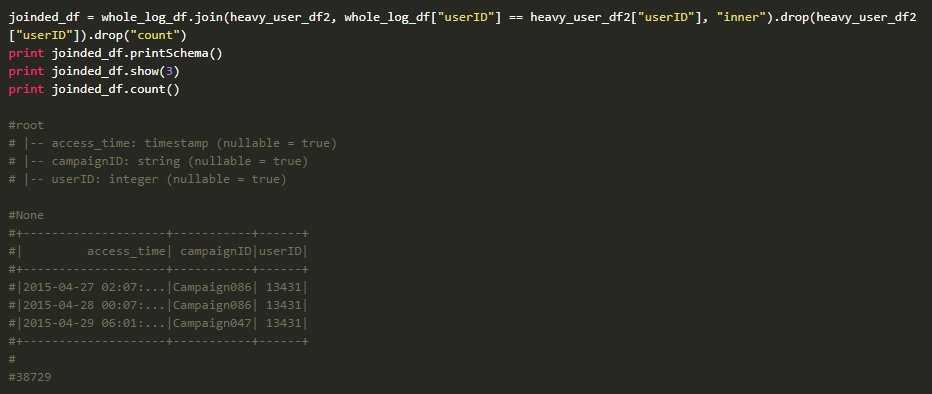
Tính năng rất quan trọng khi xử lý dữ liệu chính là join.

Mình sẽ sử dụng data đầu vào ban đầu để tạo ra 1 DataFrame mới chứa những userID có count >100



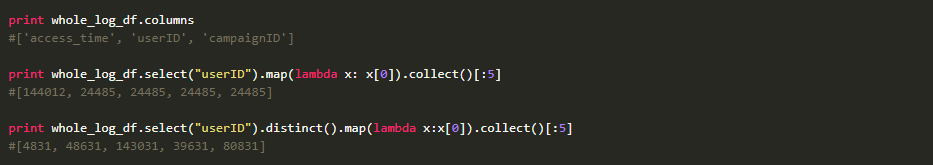
Được heavy\_user\_df2 rồi tiến hành join (mặc định là inner join).

Các kiểu join bao gồm : inner, outer, left\_outer, rignt\_outer



***6. Lấy data theo cột trong DataFrame***

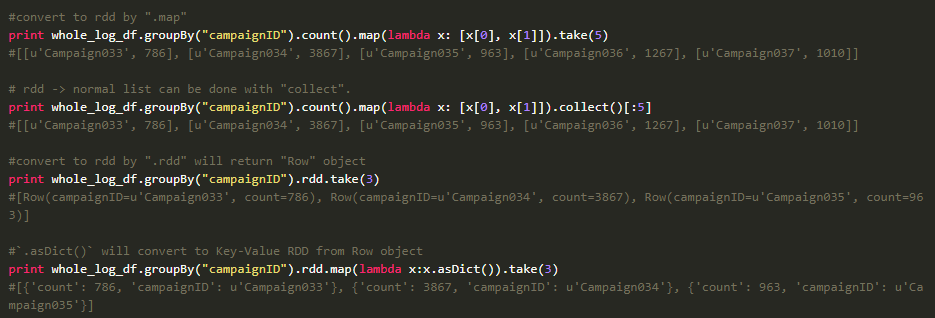
* Lấy lable của cột (df.columns) -> Trả về list tên cột (not DataFrame )
* Lấy riêng 1 cột (df.select("userID").map(lambda x: x[0]).collect()) -> Trả về list userID (not RDD/Dataframe)
* Lấy cột distinct, chỉ cần thêm .distinct()



***7.*** ***Từ DataFrame , tạo RDD***

Có 2 cách chính là:

* Sử dụng map() : mỗi hàng của DataFrame được chuyển sang RDD theo dạng list
* Sử dụng .rdd: mỗi hàng của DataFrame được chuyển sang RDD Row Object (Tức là mỗi hàng sẽ là 1 Object) Tiếp theo sử dụng .asDict() với Row Object để chuyển về RDD Key-Value.



Nguồn tham khảo :

<https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd>

<https://spark.apache.org/docs/latest/configuration.html>

<https://codetudau.com/xu-ly-du-lieu-voi-spark-dataframe/index.html>

<https://helpex.vn/article/rdd-trong-spark-la-gi-va-tai-sao-chung-ta-can-no-5c6afe5bae03f628d053a84c>