**HU Extension Assignment 06 E63 Big Data Analytics**

Issued on: March 05, 2016 Due by 11:30PM EST, March 11, 2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document. We cannot retype text that is in JPG images. Please, always submit a copy of original, working scripts and/or class files you used as separate files. Sometimes we need to run your code and retyping is too costly. Please, submit to the class drop box. For issues and comments visit the class Discussion Board. You can solve these problems using any language of your choice.

**Problem 1.** Go to an online newspaper and select paragraphs from two articles in a similar field, about politics, art, movies, or any other topic of your choice. Let those paragraphs be moderately small, a few lines and around 100 words. Save those paragraphs as .txt files and then import them into two Spark RDD objects, paragraphA and paragraphB. Use Spark transformation functions to transform those initial RDD-s into RDD-s that contain only words. List for us the first 10 words in each RDD. Subsequently create RDD-s that contain only unique words in each of paragraphs. Then create an RDD that contains only words that are present in paragraphA but not in paragraphB. Finally create an RDD that contains only the words common to two paragraphs.

I choose two paragraphs from http://www.nytimes.com/2016/03/07/us/politics/election-results.html

ParagraphA:

Senator Marco Rubio of Florida won at the Puerto Rico primary on Sunday, giving him a much-needed victory after a string of losses that threatened to push the Republican presidential nomination further out of his reach. In Maine, Senator Bernie Sanders of Vermont won the Democratic caucuses on Sunday, according to The Associated Press. The victory in Puerto Rico came a day after Mr. Rubio was shut out of first-place finishes in all four of Saturday’s Republican nominating contests. Senator Ted Cruz of Texas won the caucuses in Kansas and Maine, and Donald J. Trump won the Kentucky caucuses and the Louisiana primary.

ParagraphB:

Mr. Rubio — who in recent days has focused on his home state, which he must win in its March 15 primary if he hopes to continue his campaign — made a detour to Puerto Rico on Saturday, campaigning in San Juan and holding a news conference, mostly in Spanish, in a last-minute push to win the primary. Puerto Rico, which has roughly 3.7 million residents, could help Mr. Rubio in his home state’s primary. More than one million Puerto Ricans live in Florida, concentrated most heavily around Orlando, and many were closely watching their island’s contest Sunday. Puerto Rico is embroiled in a debt crisis, with the territory facing a deficit of more than $70 billion.

Then I wrote Scala program to achieve what’s asked in the problem:

package e63.course.assignment6

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

object Problem1 {

def main(args: Array[*String*]) {

// set input file paths

val paragraphAFileName = args(0)

val paragraphBFileName = args(1)

// Create a Spark Configuration

val sparkConf = new SparkConf().setAppName("Assignment6\_Problem1")

// Create a Scala Spark Context.

val sparkContext = new SparkContext(sparkConf);

// Load the input data into RDDs

val paragraphA = sparkContext.textFile(paragraphAFileName)

val paragraphB = sparkContext.textFile(paragraphBFileName)

// get an RDD of tokens for each of the input files

val paragraphATokens = paragraphA.flatMap(line => line.split(" "))

val paragraphBTokens = paragraphB.flatMap(line => line.split(" "))

// print out number of tokens

println("\nTotal number of tokens in paragraphA: " + paragraphATokens.count());

println("\nTotal number of tokens in paragraphB: " + paragraphBTokens.count());

// To transform the RDDs into RDDs that contain only words

// filter in only the words from the tokens

// '.' and ',' are treated separately because those characters immediately follow a word,

// in which case we want to keep the word and get rid of '.' and ','

val paragraphAWords = paragraphATokens.filter(word => word.matches("[A-Za-z0-9.,]+")).map(word => word.toLowerCase().replaceAll("[.,]", ""))

val paragraphBWords = paragraphBTokens.filter(word => word.matches("[A-Za-z0-9.,]+")).map(word => word.toLowerCase().replaceAll("[.,]", ""))

// list first 10 words from paragraphA and paragrahB

println("\nFirst 10 words from paragraphA:")

paragraphAWords.take(10).foreach(println)

println("\nFirst 10 words from paragraphB:")

paragraphBWords.take(10).foreach(println)

// print out number of words

println("\nTotal number of words in paragraphA: " + paragraphAWords.count());

println("\nTotal number of words in paragraphB: " + paragraphBWords.count());

//create RDDs that contain only unique words from each RDD

val paragraphAUniqueWords = paragraphAWords.distinct()

val paragraphBUniqueWords = paragraphBWords.distinct()

// print out number of unique words

println("\nTotal number of unique words in paragraphA: " + paragraphAUniqueWords.count());

println("\nTotal number of unique words in paragraphB: " + paragraphBUniqueWords.count());

// create an RDD with words that are in paragraphA but not in paragraphB

val wordsInParAButNotInParB = paragraphAUniqueWords.subtract(paragraphBUniqueWords)

// list out information of words that are in paragraphA but not in paragraphB

println("\nTotal number of words in paragraphA that are not in paragraphB: " + wordsInParAButNotInParB.count())

println("\nList of words in paragraphA that are not in paragraphB:")

wordsInParAButNotInParB.collect().foreach(println)

// create an RDD with words that are common in paragraphA and paragraphB

val wordsCommonInParAAndParB = paragraphAUniqueWords.intersection(paragraphBUniqueWords)

// list out information of words that are common in paragraphA and pragraphB

println("\nTotal number of common words in paragraphA and paragraphB: " + wordsCommonInParAAndParB.count())

println("\nList of words that are common in paragraphA and paragraphB")

wordsCommonInParAAndParB.collect().foreach(println)

}

}

Then I added the required dependencies in build.sbt file:

// additional libraries

libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-core" % "1.6.0" % "provided"

)

libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-sql" % "1.6.0" % "provided"

)

Then I built the program using sbt:

rpulekar-m1:Assignment6 rpulekar$ pwd

/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6

rpulekar-m1:Assignment6 rpulekar$ sbt clean package

[info] Loading global plugins from /Users/rpulekar/.sbt/0.13/plugins

[info] Set current project to Assignment6 (in build file:/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/)

[success] Total time: 0 s, completed Mar 10, 2016 8:43:16 PM

[info] Updating {file:/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/}assignment6...

[info] Resolving org.fusesource.jansi#jansi;1.4 ...

[info] Done updating.

[info] Compiling 3 Scala sources to /Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/target/scala-2.10/classes...

[warn] Multiple main classes detected. Run 'show discoveredMainClasses' to see the list

[info] Packaging /Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/target/scala-2.10/assignment6\_2.10-0.0.1.jar ...

[info] Done packaging.

[success] Total time: 7 s, completed Mar 10, 2016 8:43:23 PM

rpulekar-m1:Assignment6 rpulekar$

Made sure that the output file is generated:

rpulekar-m1:Assignment6 rpulekar$ ls -lha target/scala-2.10/

total 48

drwxr-xr-x 4 rpulekar 1327142227 136B Mar 10 20:43 .

drwxr-xr-x 5 rpulekar 1327142227 170B Mar 10 20:43 ..

-rw-r--r-- 1 rpulekar 1327142227 22K Mar 10 20:43 assignment6\_2.10-0.0.1.jar

drwxr-xr-x 3 rpulekar 1327142227 102B Mar 10 20:43 classes

rpulekar-m1:Assignment6 rpulekar$

Then I copied over the jar file to the VM:

rpulekar-m1:shared\_dir\_via\_scp rpulekar$ scp -i ../private\_keys/cloudera\_id\_dsa assignment6\_2.10-0.1-SNAPSHOT.jar cloudera@192.168.71.152:~/shared

assignment6\_2.10-0.1-SNAPSHOT.jar 100% 279 0.3KB/s 00:00

rpulekar-m1:shared\_dir\_via\_scp rpulekar$

On the VM, I assembled two files: one for paragraphA, other for paragraph

[cloudera@localhost ~]$ ls assignment6\_problem1\_input/

paragraphA\_file.txt paragraphB\_file.txt

[cloudera@localhost ~]$

Then I submitted the program to spark

[cloudera@localhost ~]$ spark-submit --class e63.course.assignment6.Problem1 shared/assignment6\_2.10-0.0.1.jar file:///home/cloudera/assignment6\_problem1\_input/paragraphA\_file.txt file:///home/cloudera/assignment6\_problem1\_input/paragraphB\_file.txt

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

16/03/10 17:51:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/03/10 17:51:30 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.71.152 instead (on interface eth0)

16/03/10 17:51:30 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another

Total number of tokens in paragraphA: 103

Total number of tokens in paragraphB: 117

**First 10 words in paragraphA RDD:**

senator

marco

rubio

of

florida

won

at

the

puerto

rico

**First 10 words in paragraphB RDD:**

mr

rubio

who

in

recent

days

has

focused

on

his

Total number of words in paragraphA: 100

Total number of words in paragraphB: 111

Total number of unique words in paragraphA: 61

Total number of unique words in paragraphB: 76

Total number of words in paragraphA that are not in paragraphB: 45

Total number of common words in paragraphA and paragraphB: 16

Total number of unique words in paragraphA: 61

Total number of unique words in paragraphB: 76

Total number of words **in paragraphA that are not in paragraphB**: 45

List of words **in paragraphA that are not in paragraphB**:

shut

trump

donald

reach

texas

losses

him

that

ted

four

won

caucuses

press

day

all

louisiana

kansas

senator

at

nominating

presidential

string

republican

according

giving

sanders

out

came

democratic

finishes

after

kentucky

maine

bernie

marco

victory

was

j

threatened

nomination

contests

associated

vermont

cruz

further

Total number of **common words in paragraphA and paragraphB**: 16

List of words that are **common in paragraphA and paragraphB**

florida

the

his

puerto

on

mr

in

primary

sunday

a

rico

to

of

push

and

rubio

Deliverables attached with the submission:

- **Problem1.scala** (which is the scala program for this problem)

**Problem 2**. Consider attached file emps.txt. It contains: name, age and salary of three employees. Create RDD emps by importing that file into Spark. Next create a new RDD emps\_fields by transforming the content of every line in RDD emps into a tuple with three individual elements by splitting the lines on commas. Now comes something new. Spark has a class Row and you need to import it in your script or program. Row comes from the same package as class SQLContext. Row class creates rows with named and typed fields. You need to apply “constructor” Row to every tuple in RDD emps\_fields, like:

employees = emps\_fields.map(lambda e: Row(name = e[0], age = int(e[1]), salary = float(e[2])))

e[0], e[1] and e[2] are the first, second and third elements of the tuple e representing a row (line) in RDD emps\_fields. Note that int and float are types of fields in new rows. Newly create RDD employees is now made of Row elements and is ready to be transformed into a DataFrame. You generate a DataFrame by passing an RDD of Row elements to the method createDataFrame() of class SQLContext. Do it. Show the content of new DataFrame. Transform this DataFrame into a Temporary Table and select names of all employees who have a salary greater than 3500.

I transferred the file emps.txt to the VM

rpulekar-m1:shared\_dir\_via\_scp rpulekar$ scp -i ../private\_keys/cloudera\_id\_dsa emps.txt cloudera@192.168.71.152:~/shared

emps.txt 100% 60 0.1KB/s 00:00

rpulekar-m1:shared\_dir\_via\_scp rpulekar$

[cloudera@localhost ~]$ ls assignment6\_problem2\_input/

emps.txt

[cloudera@localhost ~]$

Then I wrote this Scala program to do what’s asked in problem 2:

package e63.course.assignment6

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import scala.Tuple3

import org.apache.spark.sql.Row

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.types.StructType

import org.apache.spark.sql.types.StructField

import org.apache.spark.sql.types.StringType

import org.apache.spark.sql.types.StructField

import org.apache.spark.sql.types.DoubleType

import org.apache.spark.sql.types.LongType

import org.apache.spark.sql.types.IntegerType

import org.apache.spark.sql.types.FloatType

object Problem2 {

def main(args: Array[*String*]) {

// set input files

val empsFilePath = args(0)

// Create a Spark Configuration

val sparkConf = new SparkConf().setAppName("Assignment6\_Problem2")

// Create a Scala Spark Context.

val sparkContext = new SparkContext(sparkConf);

// create a sql context

val sqlContext = new SQLContext(sparkContext)

// create RDD emps of the input text file where every record in RDD will be a line in

// the txt file

val emps = sparkContext.textFile(empsFilePath)

// list out contents of emps RDD

println("\n emps RDD contents:")

emps.take(10).foreach(println)

// create RDD emps\_fields where every record will be a tuple of 3 elements

val emps\_fields = emps.map(line => new Tuple3((line.split(", ")(0)), (line.split(", ")(1)), (line.split(", ")(2))))

// list out contents of emps\_fields RDD

println("\n emps\_fields RDD contents:")

emps\_fields.take(10).foreach(println)

// create employees RDD where every record will be a Row

val employees = emps\_fields.map(e => Row(e.\_1, e.\_2.toInt, e.\_3.toFloat))

// list out contents of employees RDD

println("\n employees RDD contents")

employees.take(10).foreach(println)

// create a schema for the input txt file

val empsSchema = StructType(StructField("name", StringType, false) ::

StructField("age", IntegerType, false) ::

StructField("salary", FloatType, false) :: Nil)

// create a dataframe from the employees RDD and schema

val employeesDataFrame = sqlContext.createDataFrame(employees, empsSchema)

// print the content of employees data frame

println("\nContent of employees data frame:")

employeesDataFrame.select("name", "age", "salary").show()

// transform the data frame into temporary table

employeesDataFrame.registerTempTable("employees\_temp\_table")

// select names of employees who have salary greater than 3500 from employees data

// frame

println("Will now select from employeesDataFrame, names of employees who have salary greater than 3500")

employeesDataFrame.filter("salary>3500").select("name").show()

// select names of employees who have salary greater than 3500 from employees

// temporary table

println("Will now select from employees\_temp\_table, names of employees who have salary greater than 3500")

sqlContext.sql("select name from employees\_temp\_table where salary>3500").show()

}

}

Then I compiled the program and created a jar with sbt

rpulekar-m1:Assignment6 rpulekar$ pwd

/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6

rpulekar-m1:Assignment6 rpulekar$ sbt clean package

[info] Loading global plugins from /Users/rpulekar/.sbt/0.13/plugins

[info] Set current project to Assignment6 (in build file:/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/)

[success] Total time: 0 s, completed Mar 10, 2016 8:43:16 PM

[info] Updating {file:/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/}assignment6...

[info] Resolving org.fusesource.jansi#jansi;1.4 ...

[info] Done updating.

[info] Compiling 3 Scala sources to /Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/target/scala-2.10/classes...

[warn] Multiple main classes detected. Run 'show discoveredMainClasses' to see the list

[info] Packaging /Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/target/scala-2.10/assignment6\_2.10-0.0.1.jar ...

[info] Done packaging.

[success] Total time: 7 s, completed Mar 10, 2016 8:43:23 PM

rpulekar-m1:Assignment6 rpulekar$

Made sure that the output file is generated:

rpulekar-m1:Assignment6 rpulekar$ ls -lha target/scala-2.10/

total 48

drwxr-xr-x 4 rpulekar 1327142227 136B Mar 10 20:43 .

drwxr-xr-x 5 rpulekar 1327142227 170B Mar 10 20:43 ..

-rw-r--r-- 1 rpulekar 1327142227 22K Mar 10 20:43 assignment6\_2.10-0.0.1.jar

drwxr-xr-x 3 rpulekar 1327142227 102B Mar 10 20:43 classes

rpulekar-m1:Assignment6 rpulekar$

Then I copied over the jar file to the VM:

rpulekar-m1:shared\_dir\_via\_scp rpulekar$ scp -i ../private\_keys/cloudera\_id\_dsa assignment6\_2.10-0.1-SNAPSHOT.jar cloudera@192.168.71.152:~/shared

assignment6\_2.10-0.1-SNAPSHOT.jar 100% 279 0.3KB/s 00:00

rpulekar-m1:shared\_dir\_via\_scp rpulekar$

Then I submitted the program to spark:

[cloudera@localhost ~]$ spark-submit --class e63.course.assignment6.Problem2 shared/assignment6\_2.10-0.0.1.jar file:///home/cloudera/assignment6\_problem2\_input/emps.txt

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

16/03/10 21:48:39 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/03/10 21:48:40 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.71.152 instead (on interface eth0)

16/03/10 21:48:40 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

16/03/10 21:48:42 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.

emps RDD contents:

Michael, 29, 3000.30

Andy, 30, 2500.25

Justin, 19, 4000.99

emps\_fields RDD contents:

(Michael,29,3000.30)

(Andy,30,2500.25)

(Justin,19,4000.99)

employees RDD contents

[Michael,29,3000.3]

[Andy,30,2500.25]

[Justin,19,4000.99]

Content of employees data frame:

+-------+---+-------+

| name|age| salary|

+-------+---+-------+

|Michael| 29| 3000.3|

| Andy| 30|2500.25|

| Justin| 19|4000.99|

+-------+---+-------+

Will now select from employeesDataFrame, names of employees who have salary greater than 3500

+------+

| name|

+------+

|Justin|

+------+

Will now select from employees\_temp\_table, names of employees who have salary greater than 3500

+------+

| name|

+------+

|Justin|

+------+

Deliverables:

- **Problem2.scala** (which is the scala program for this problem)

**Problem 3**. Attached file ebay.csv contains information of eBay’s auction history. The Excel file has 9 columns and they represent:

The eBay online auction dataset has the following data fields:

* auctionid - unique identifier of an auction
* bid - the proxy bid placed by a bidder
* bidtime - the time (in days) that the bid was placed, from the start of the auction
* bidder - eBay username of the bidder
* bidderrate - eBay feedback rating of the bidder
* openbid - the opening bid set by the seller
* price - the closing price that the item sold for (equivalent to the second highest bid + an increment)
* item – name of the item being sold
* daystolive – length of the auction.

Using Spark DataFrames you will explore the data with following 4 questions:

1. How many auctions were held?
2. How many bids were made per item?
   * What's the minimum, maximum, and average bid (price) per item?
   * What is the minimum, maximum and average number of bids per item?
3. Show the bids with price > 100

Import data into an RDD object. Transform that RDD into an RDD of Row-s by assign schema (column names and types). Transform that new RDD into a DataFrame. Call that DataFrame Auction. Show (print) the schema of the DataFrame. Make above queries using DatFrame API. You recall how we applied methods: select(), groupBy(), count() and others to the DataFrame in class. Use those methods.

Next transform your Auction DataFrame into a table and make the same 4 inquiries using regular SQL queries.

Persist your DataFrame as a Parquet file and show that you could exit your pyspark shell and come back in it and you will be still able to read the data from that file and create a DataFrame and an SQL like table that you can issue queries agains.

I transferred ebay.csv file to the VM:

rpulekar-m1:shared\_dir\_via\_scp rpulekar$ scp -i ../private\_keys/cloudera\_id\_dsa ebay.csv cloudera@192.168.71.152:~/shared

ebay.csv 100% 572KB 571.9KB/s 00:00

rpulekar-m1:shared\_dir\_via\_scp rpulekar$

Then I wrote the following Scala program to do what’s asked in problem 3:

package e63.course.assignment6

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.Row

import org.apache.spark.sql.types.StructType

import org.apache.spark.sql.types.StructField

import org.apache.spark.sql.types.StringType

import org.apache.spark.sql.types.StructField

import org.apache.spark.sql.types.DoubleType

import org.apache.spark.sql.types.LongType

import org.apache.spark.sql.types.IntegerType

import org.apache.spark.sql.types.FloatType

**object** Problem3 {

**def** main(args: Array[*String*]) {

// set input files

**val** ebayFilePath = args(0)

// Create a Spark Configuration

**val** sparkConf = **new** SparkConf().setAppName("Assignment6\_Problem3")

// Create a Scala Spark Context.

**val** sparkContext = **new** SparkContext(sparkConf);

// create a sql context

**val** sqlContext = **new** SQLContext(sparkContext)

// import data into an RDD

**val** ebayRDD = sparkContext.textFile(ebayFilePath)

// transform the RDD into another RDD made of tuples

**val** ebayRDDOfTuples = ebayRDD.map(l => **new** **Tuple9**(l.split(",")(0), l.split(",")(1), l.split(",")(2), l.split(",")(3), l.split(",")(4), l.split(",")(5), l.split(",")(6), l.split(",")(7), l.split(",")(8)))

// transform the RDD into RDD of Rows

**val** ebayRDDOfRows = ebayRDDOfTuples.map(e => Row(e.\_1.toLong, e.\_2.toFloat, e.\_3.toFloat, e.\_4, e.\_5.toInt, e.\_6.toFloat, e.\_7.toFloat, e.\_8, e.\_9.toInt))

// print first 10 rows to make sure RDDs have been generated correctly

println("ebayRDDOfRows:")

ebayRDDOfRows.take(10).foreach(println)

// create schema for ebay.csv file

**val** ebaySchema = **StructType**(**StructField**("auctionid", **LongType**, **false**) ::

**StructField**("bid", **FloatType**, **false**) ::

**StructField**("bidtime", **FloatType**, **false**) ::

**StructField**("bidder", **StringType**, **false**) ::

**StructField**("bidderrate", **IntegerType**, **false**) ::

**StructField**("openbid", **FloatType**, **false**) ::

**StructField**("price", **FloatType**, **false**) ::

**StructField**("item", **StringType**, **false**) ::

**StructField**("daystolive", **IntegerType**, **false**) :: **Nil**)

// transform the RDD into dataframe with the help of above created schema

**val** Auction = sqlContext.createDataFrame(ebayRDDOfRows, ebaySchema)

// print schema of Auction dataframe

println("Printing Auction dataframe schema")

Auction.printSchema()

// print first 10 rows of Auction data frame to make sure it is generated correctly

println("Printing first 10 rows of Auction data frame to make sure data frame is created correctly")

Auction.select("auctionid", "bid", "bidtime", "bidder", "bidderrate", "openbid", "price", "item", "daystolive").show(10)

// how many auctions were held? (querying the dataframe)

println("Below is answer to the question: How many auctions were held? (querying the dataframe)")

println(Auction.select("auctionid").distinct().count())

// How many bids were made per item? (querying the dataframe)

println("Below is answer to the question: How many bids were made per item? (querying the dataframe)")

Auction.groupBy("item").count().show()

// What's the minimum bid (price) per item? (querying the dataframe)

println("Below is answer to the question: What's the minimum bid (price) per item? (querying the dataframe)")

Auction.groupBy("item").min("bid").show()

// What's the maximum bid (price) per item? (querying the dataframe)

println("Below is answer to the question: What's the maximum bid (price) per item? (querying the dataframe)")

Auction.groupBy("item").max("bid").show()

// What's the average bid (price) per item? (querying the dataframe)

println("Below is answer to the question: What's the average bid (price) per item? (querying the dataframe)")

Auction.groupBy("item").avg("bid").show()

// create a data frame of item, auctionid and count of bids for that combination of item-auctionid

**val** AuctionBidCountDataFrame = Auction.groupBy("item", "auctionid").count()

// What is the minimum number of bids per item? (querying the dataframe)

// Here I have shown min number of bids per item across the auctions

// So first I have grouped by item and auctionid on line 93.

// Then I found the min number of bids placed in an auction for each item

println("Below is answer to the question: What is the minimum number of bids per item? (querying the dataframe)")

AuctionBidCountDataFrame.groupBy("item").min("count").show()

// What is the maximum number of bids per item? (querying the dataframe)

// Here I have shown max number of bids per item across the auctions

// So first I have grouped by item and auctionid on line 93.

// Then I found the max number of bids placed in an auction for each item

println("Below is answer to the question: What is the maximum number of bids per item? (querying the dataframe)")

AuctionBidCountDataFrame.groupBy("item").max("count").show()

// What is the average number of bids per item? (querying the dataframe)

// Here I have shown avg number of bids per item across the auctions

// So first I have grouped by item and auctionid on line 93.

// Then I found the avg number of bids placed in an auction for each item

println("Below is answer to the question: What is the average number of bids per item? (querying the dataframe)")

AuctionBidCountDataFrame.groupBy("item").avg("count").show()

// Show the bids with price > 100 (querying the dataframe)

// Assumption: 'price' in the problem statement refers to closing price

// If it meant bid price I would have used .filter("bid>100") instead

println("Below is answer to the question: Show the bids with price > 100 (querying the dataframe)")

Auction.filter("price > 100").show()

// convert auction dataframe into a temporary table

Auction.registerTempTable("Auction\_temp\_table")

// How many auctions were held? (querying the temp sql table)

println("Below is answer to the question: How many auctions were held? (querying the temp sql table)")

sqlContext.sql("select count(distinct auctionid) from Auction\_temp\_table").show()

// How many bids were made per item? (querying the temp sql table)

println("Below is answer to the question: How many bids were made per item? (querying the temp sql table)")

sqlContext.sql("select item, count(\*) as number\_of\_bids from Auction\_temp\_table group by item").show()

// What's the minimum bid (price) per item? (querying the temp sql table)

println("Below is answer to the question: What's the minimum bid (price) per item? (querying the temp sql table)")

sqlContext.sql("select item, min(bid) as minimum\_bid\_price from Auction\_temp\_table group by item").show()

// What's the maximum bid (price) per item? (querying the temp sql table)

println("Below is answer to the question: What's the maximum bid (price) per item? (querying the temp sql table)")

sqlContext.sql("select item, max(bid) as maximum\_bid\_price from Auction\_temp\_table group by item").show()

// What's the average bid (price) per item? (querying the temp sql table)

println("Below is answer to the question: What's the average bid (price) per item? (querying the temp sql table)")

sqlContext.sql("select item, avg(bid) as average\_bid\_price from Auction\_temp\_table group by item").show()

// What is the minimum number of bids per item? (querying the temp sql table)

// Here I have shown min number of bids per item across the auctions

// So first I have grouped by item and auctionid

// and then I found the min number of bids placed in an auction for each item

println("Below is answer to the question: What is the minimum number of bids per item? (querying the temp sql table)")

sqlContext.sql("select item, min(number\_of\_bids) as min\_number\_of\_bids from (select item, auctionid, count(\*) as number\_of\_bids from Auction\_temp\_table group by item, auctionid order by item) auction\_bid\_count group by item").show()

// What is the maximum number of bids per item? (querying the temp sql table)

// Here I have shown max number of bids per item across the auctions

// So first I have grouped by item and auctionid

// and then I found the max number of bids placed in an auction for each item

println("Below is answer to the question: What is the maximum number of bids per item? (querying the temp sql table)")

sqlContext.sql("select item, max(number\_of\_bids) as max\_number\_of\_bids from (select item, auctionid, count(\*) as number\_of\_bids from Auction\_temp\_table group by item, auctionid order by item) auction\_bid\_count group by item").show()

// What is the average number of bids per item? (querying the temp sql table)

// Here I have shown avg number of bids per item across the auctions

// So first I have grouped by item and auctionid

// and then I found the avg number of bids placed in an auction for each item

println("Below is answer to the question: What is the average number of bids per item? (querying the temp sql table)")

sqlContext.sql("select item, avg(number\_of\_bids) as avg\_number\_of\_bids from (select item, auctionid, count(\*) as number\_of\_bids from Auction\_temp\_table group by item, auctionid order by item) auction\_bid\_count group by item").show()

// Show the bids with price > 100 (querying the temp sql table)

// Assumption: 'price' in the problem statement refers to closing price

// If it meant bid price I would have used ...where bid>100") instead

println("Below is answer to the question: Show the bids with price > 100 (querying the temp sql table)")

sqlContext.sql("select \* from Auction\_temp\_table where price>100").show()

}

}

Then I compiled the program and created a jar with sbt

rpulekar-m1:Assignment6 rpulekar$ pwd

/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6

rpulekar-m1:Assignment6 rpulekar$ sbt clean package

[info] Loading global plugins from /Users/rpulekar/.sbt/0.13/plugins

[info] Set current project to Assignment6 (in build file:/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/)

[success] Total time: 0 s, completed Mar 10, 2016 8:43:16 PM

[info] Updating {file:/Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/}assignment6...

[info] Resolving org.fusesource.jansi#jansi;1.4 ...

[info] Done updating.

[info] Compiling 3 Scala sources to /Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/target/scala-2.10/classes...

[warn] Multiple main classes detected. Run 'show discoveredMainClasses' to see the list

[info] Packaging /Users/rpulekar/work/big-data-analytics-harvard/scala\_workspace\_for\_course/Assignment6/target/scala-2.10/assignment6\_2.10-0.0.1.jar ...

[info] Done packaging.

[success] Total time: 7 s, completed Mar 10, 2016 8:43:23 PM

rpulekar-m1:Assignment6 rpulekar$

Made sure that the output file is generated:

rpulekar-m1:Assignment6 rpulekar$ ls -lha target/scala-2.10/

total 48

drwxr-xr-x 4 rpulekar 1327142227 136B Mar 10 20:43 .

drwxr-xr-x 5 rpulekar 1327142227 170B Mar 10 20:43 ..

-rw-r--r-- 1 rpulekar 1327142227 22K Mar 10 20:43 assignment6\_2.10-0.0.1.jar

drwxr-xr-x 3 rpulekar 1327142227 102B Mar 10 20:43 classes

rpulekar-m1:Assignment6 rpulekar$

Then I copied over the jar file to the VM:

rpulekar-m1:shared\_dir\_via\_scp rpulekar$ scp -i ../private\_keys/cloudera\_id\_dsa assignment6\_2.10-0.1-SNAPSHOT.jar cloudera@192.168.71.152:~/shared

assignment6\_2.10-0.1-SNAPSHOT.jar 100% 279 0.3KB/s 00:00

rpulekar-m1:shared\_dir\_via\_scp rpulekar$

Then I submitted the program to spark:

[cloudera@localhost ~]$ spark-submit --class e63.course.assignment6.Problem3 shared/assignment6\_2.10-0.0.1.jar file:///home/cloudera/assignment6\_problem3\_input/ebay.csv

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

16/03/10 21:52:22 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/03/10 21:52:22 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.71.152 instead (on interface eth0)

These are the answers of all questions:

ebayRDDOfRows:

[8213034705,95.0,2.927373,jake7870,0,95.0,117.5,xbox,3]

[8213034705,115.0,2.943484,davidbresler2,1,95.0,117.5,xbox,3]

[8213034705,100.0,2.951285,gladimacowgirl,58,95.0,117.5,xbox,3]

[8213034705,117.5,2.998947,daysrus,10,95.0,117.5,xbox,3]

[8213060420,2.0,0.065266,donnie4814,5,1.0,120.0,xbox,3]

[8213060420,15.25,0.123218,myreeceyboy,52,1.0,120.0,xbox,3]

[8213060420,3.0,0.186539,parakeet2004,5,1.0,120.0,xbox,3]

[8213060420,10.0,0.18669,parakeet2004,5,1.0,120.0,xbox,3]

[8213060420,24.99,0.187049,parakeet2004,5,1.0,120.0,xbox,3]

[8213060420,20.0,0.249491,bluebubbles\_1,25,1.0,120.0,xbox,3]

**Printing Auction dataframe schema**

root

|-- auctionid: long (nullable = false)

|-- bid: float (nullable = false)

|-- bidtime: float (nullable = false)

|-- bidder: string (nullable = false)

|-- bidderrate: integer (nullable = false)

|-- openbid: float (nullable = false)

|-- price: float (nullable = false)

|-- item: string (nullable = false)

|-- daystolive: integer (nullable = false)

Printing first 10 rows of Auction data frame to make sure data frame is created correctly

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

| auctionid| bid| bidtime| bidder|bidderrate|openbid|price|item|daystolive|

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

|8213034705| 95.0|2.927373| jake7870| 0| 95.0|117.5|xbox| 3|

|8213034705|115.0|2.943484| davidbresler2| 1| 95.0|117.5|xbox| 3|

|8213034705|100.0|2.951285|gladimacowgirl| 58| 95.0|117.5|xbox| 3|

|8213034705|117.5|2.998947| daysrus| 10| 95.0|117.5|xbox| 3|

|8213060420| 2.0|0.065266| donnie4814| 5| 1.0|120.0|xbox| 3|

|8213060420|15.25|0.123218| myreeceyboy| 52| 1.0|120.0|xbox| 3|

|8213060420| 3.0|0.186539| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420| 10.0| 0.18669| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420|24.99|0.187049| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420| 20.0|0.249491| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

only showing top 10 rows

How many auctions were held? (DataFrame API):

println(Auction.select("auctionid").distinct().count())

627

How many bids were made per item? (DataFrame API)

Auction.groupBy("item").count().show()

+-------+-----+

| item|count|

+-------+-----+

| xbox| 2784|

| palm| 5917|

|cartier| 1953|

+-------+-----+

What's the minimum bid (price) per item? (dataframe API)

Auction.groupBy("item").min("bid").show()

+-------+--------+

| item|min(bid)|

+-------+--------+

| xbox| 0.01|

| palm| 0.01|

|cartier| 1.0|

+-------+--------+

What's the maximum bid (price) per item? (dataframe API)

Auction.groupBy("item").max("bid").show()

+-------+--------+

| item|max(bid)|

+-------+--------+

| xbox| 501.77|

| palm| 290.0|

|cartier| 5400.0|

+-------+--------+

What's the average bid (price) per item? (dataframe API)

Auction.groupBy("item").avg("bid").show()

+-------+------------------+

| item| avg(bid)|

+-------+------------------+

| xbox| 85.3979382128535|

| palm|146.12732120350637|

|cartier| 569.8761238992|

+-------+------------------+

What is the minimum number of bids per item? (dataframe API)

val AuctionBidCountDataFrame = Auction.groupBy("item", "auctionid").count()

AuctionBidCountDataFrame.groupBy("item").min("count").show()

+-------+----------+

| item|min(count)|

+-------+----------+

| xbox| 2|

| palm| 1|

|cartier| 2|

+-------+----------+

What is the maximum number of bids per item? (dataframe API)

val AuctionBidCountDataFrame = Auction.groupBy("item", "auctionid").count()

AuctionBidCountDataFrame.groupBy("item").max("count").show()

+-------+----------+

| item|max(count)|

+-------+----------+

| xbox| 75|

| palm| 54|

|cartier| 46|

+-------+----------+

What is the average number of bids per item? (dataframe API)

val AuctionBidCountDataFrame = Auction.groupBy("item", "auctionid").count()

AuctionBidCountDataFrame.groupBy("item").avg("count").show()

+-------+------------------+

| item| avg(count)|

+-------+------------------+

| xbox| 18.81081081081081|

| palm|17.250728862973762|

|cartier|14.360294117647058|

+-------+------------------+

Show the bids with price > 100 (querying the dataframe)

Auction.filter("price > 100").show()

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

| auctionid| bid| bidtime| bidder|bidderrate|openbid|price|item|daystolive|

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

|8213034705| 95.0|2.927373| jake7870| 0| 95.0|117.5|xbox| 3|

|8213034705|115.0|2.943484| davidbresler2| 1| 95.0|117.5|xbox| 3|

|8213034705|100.0|2.951285|gladimacowgirl| 58| 95.0|117.5|xbox| 3|

|8213034705|117.5|2.998947| daysrus| 10| 95.0|117.5|xbox| 3|

|8213060420| 2.0|0.065266| donnie4814| 5| 1.0|120.0|xbox| 3|

|8213060420|15.25|0.123218| myreeceyboy| 52| 1.0|120.0|xbox| 3|

|8213060420| 3.0|0.186539| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420| 10.0| 0.18669| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420|24.99|0.187049| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420| 20.0|0.249491| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 22.0| 0.24956| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 24.0|0.249653| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 26.0|0.249757| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 80.0| 0.59059| sa4741| 3| 1.0|120.0|xbox| 3|

|8213060420| 75.0|0.657384| jhnsn2273| 51| 1.0|120.0|xbox| 3|

|8213060420| 85.0|0.657917| jhnsn2273| 51| 1.0|120.0|xbox| 3|

|8213060420| 83.0|0.816447| sa4741| 3| 1.0|120.0|xbox| 3|

|8213060420|100.0|1.005903| sa4741| 3| 1.0|120.0|xbox| 3|

|8213060420|100.0|1.012697| jhnsn2273| 51| 1.0|120.0|xbox| 3|

|8213060420|110.0|1.013056| jhnsn2273| 51| 1.0|120.0|xbox| 3|

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

only showing top 20 rows

How many auctions were held? (querying the temp sql table)

sqlContext.sql("select count(distinct auctionid) from Auction\_temp\_table").show()

+---+

|\_c0|

+---+

|627|

+---+

How many bids were made per item? (querying the temp sql table)

sqlContext.sql("select item, count(\*) as number\_of\_bids from Auction\_temp\_table group by item").show()

+-------+--------------+

| item|number\_of\_bids|

+-------+--------------+

| xbox| 2784|

| palm| 5917|

|cartier| 1953|

+-------+--------------+

What's the minimum bid (price) per item? (querying the temp sql table)

sqlContext.sql("select item, min(bid) as minimum\_bid\_price from Auction\_temp\_table group by item").show()

+-------+-----------------+

| item|minimum\_bid\_price|

+-------+-----------------+

| xbox| 0.01|

| palm| 0.01|

|cartier| 1.0|

+-------+-----------------+

What's the maximum bid (price) per item? (querying the temp sql table)

sqlContext.sql("select item, max(bid) as maximum\_bid\_price from Auction\_temp\_table group by item").show()

+-------+-----------------+

| item|maximum\_bid\_price|

+-------+-----------------+

| xbox| 501.77|

| palm| 290.0|

|cartier| 5400.0|

+-------+-----------------+

What's the average bid (price) per item? (querying the temp sql table)

sqlContext.sql("select item, avg(bid) as average\_bid\_price from Auction\_temp\_table group by item").show()

+-------+------------------+

| item| average\_bid\_price|

+-------+------------------+

| xbox| 85.3979382128535|

| palm|146.12732120350637|

|cartier| 569.8761238992|

+-------+------------------+

What is the minimum number of bids per item? (querying the temp sql table)

sqlContext.sql("select item, min(number\_of\_bids) as min\_number\_of\_bids from (select item, auctionid, count(\*) as number\_of\_bids from Auction\_temp\_table group by item, auctionid order by item) auction\_bid\_count group by item").show()

+-------+------------------+

| item|min\_number\_of\_bids|

+-------+------------------+

|cartier| 2|

| palm| 1|

| xbox| 2|

+-------+------------------+

What is the maximum number of bids per item? (querying the temp sql table)

sqlContext.sql("select item, max(number\_of\_bids) as max\_number\_of\_bids from (select item, auctionid, count(\*) as number\_of\_bids from Auction\_temp\_table group by item, auctionid order by item) auction\_bid\_count group by item").show()

+-------+------------------+

| item|max\_number\_of\_bids|

+-------+------------------+

|cartier| 46|

| palm| 54|

| xbox| 75|

+-------+------------------+

What is the average number of bids per item? (querying the temp sql table)

sqlContext.sql("select item, avg(number\_of\_bids) as avg\_number\_of\_bids from (select item, auctionid, count(\*) as number\_of\_bids from Auction\_temp\_table group by item, auctionid order by item) auction\_bid\_count group by item").show()

+-------+------------------+

| item|avg\_number\_of\_bids|

+-------+------------------+

|cartier|14.360294117647058|

| palm|17.250728862973762|

| xbox| 18.81081081081081|

+-------+------------------+

Show the bids with price > 100 (querying the temp sql table)

sqlContext.sql("select \* from Auction\_temp\_table where price>100").show()

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

| auctionid| bid| bidtime| bidder|bidderrate|openbid|price|item|daystolive|

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

|8213034705| 95.0|2.927373| jake7870| 0| 95.0|117.5|xbox| 3|

|8213034705|115.0|2.943484| davidbresler2| 1| 95.0|117.5|xbox| 3|

|8213034705|100.0|2.951285|gladimacowgirl| 58| 95.0|117.5|xbox| 3|

|8213034705|117.5|2.998947| daysrus| 10| 95.0|117.5|xbox| 3|

|8213060420| 2.0|0.065266| donnie4814| 5| 1.0|120.0|xbox| 3|

|8213060420|15.25|0.123218| myreeceyboy| 52| 1.0|120.0|xbox| 3|

|8213060420| 3.0|0.186539| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420| 10.0| 0.18669| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420|24.99|0.187049| parakeet2004| 5| 1.0|120.0|xbox| 3|

|8213060420| 20.0|0.249491| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 22.0| 0.24956| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 24.0|0.249653| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 26.0|0.249757| bluebubbles\_1| 25| 1.0|120.0|xbox| 3|

|8213060420| 80.0| 0.59059| sa4741| 3| 1.0|120.0|xbox| 3|

|8213060420| 75.0|0.657384| jhnsn2273| 51| 1.0|120.0|xbox| 3|

|8213060420| 85.0|0.657917| jhnsn2273| 51| 1.0|120.0|xbox| 3|

|8213060420| 83.0|0.816447| sa4741| 3| 1.0|120.0|xbox| 3|

|8213060420|100.0|1.005903| sa4741| 3| 1.0|120.0|xbox| 3|

|8213060420|100.0|1.012697| jhnsn2273| 51| 1.0|120.0|xbox| 3|

|8213060420|110.0|1.013056| jhnsn2273| 51| 1.0|120.0|xbox| 3|

+----------+-----+--------+--------------+----------+-------+-----+----+----------+

only showing top 20 rows

[cloudera@localhost ~]$

For the parquet part of the problem:

I started the pyspark program in this way:

[cloudera@localhost ~]$ pyspark --packages com.databricks:spark-csv\_2.11:1.4.0

Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)

[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

Ivy Default Cache set to: /home/cloudera/.ivy2/cache

The jars for the packages stored in: /home/cloudera/.ivy2/jars

Then I created Auction data frame on pyspark command line as:

>>> Auction = sqlContext.read.load("file:///home/cloudera/assignment6\_problem3\_input/ebay.csv",format="csv",header="true",inferSchema="true")

16/03/10 22:25:24 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.1.0

Then I did the below to make sure that data frame is indeed created:

>>> Auction.show(2)

+----------+-----+--------+-------------+----------+-------+-----+----+----------+

| auctionid| bid| bidtime| bidder|bidderrate|openbid|price|item|daystolive|

+----------+-----+--------+-------------+----------+-------+-----+----+----------+

|8213034705| 95.0|2.927373| jake7870| 0| 95.0|117.5|xbox| 3|

|8213034705|115.0|2.943484|davidbresler2| 1| 95.0|117.5|xbox| 3|

+----------+-----+--------+-------------+----------+-------+-----+----+----------+

only showing top 2 rows

>>> Auction.count()

10654

>>>

Then I did the below to save the data frame as a parquet file

>>> Auction.write.save("auction.parquet",format="parquet")

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/parquet-hadoop-bundle-1.5.0-cdh5.5.1.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/parquet-format-2.1.0-cdh5.5.1.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/parquet-pig-bundle-1.5.0-cdh5.5.1.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/hive-jdbc-1.1.0-cdh5.5.2-standalone.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/hive-exec-1.1.0-cdh5.5.2.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [shaded.parquet.org.slf4j.helpers.NOPLoggerFactory]

>>>

Then I did the below to make sure parquet file is created

[cloudera@localhost assignment6\_problem3\_input]$ hadoop fs -ls

Found 2 items

drwxr-xr-x - cloudera supergroup 0 2016-03-06 20:41 assignment6\_problem1\_input

drwxr-xr-x - cloudera supergroup 0 2016-03-10 22:33 auction.parquet

[cloudera@localhost assignment6\_problem3\_input]$

Then I exited out of pyspark:

>>> exit()

[cloudera@localhost ~]$

Went into Pyspark again:

Did the below to construct data frame again by reading the parquet file:

>>> Auction = sqlContext.read.parquet("auction.parquet")

16/03/10 22:37:16 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.1.0

16/03/10 22:37:16 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException

16/03/10 22:37:17 WARN DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/parquet-hadoop-bundle-1.5.0-cdh5.5.1.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/parquet-format-2.1.0-cdh5.5.1.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/parquet-pig-bundle-1.5.0-cdh5.5.1.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/hive-jdbc-1.1.0-cdh5.5.2-standalone.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/hive-exec-1.1.0-cdh5.5.2.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [shaded.parquet.org.slf4j.helpers.NOPLoggerFactory]

>>>

Did the below to load data frame into temporary sql table:

>>> Auction.registerTempTable("Auction\_temp\_table")

>>>

Then I issued a query against dataframe:

>>> Auction.groupBy("item").count().show()

[Stage 4:===============> (54 + 1) / 199[Stage 4:=====================> (77 + 1) / 199[Stage 4:===========================> (100 + 1) / 199[Stage 4:==================================> (125 + 1) / 199[Stage 4:=========================================> (151 + 1) / 199[Stage 4:==================================================> (182 + 1) / 199 +-------+-----+

| item|count|

+-------+-----+

| xbox| 2784|

| palm| 5917|

|cartier| 1953|

+-------+-----+

>>>

Then I issued a query against temporary sql table:

>>> sqlContext.sql("select item, count(\*) as number\_of\_bids from Auction\_temp\_table group by item").show()

[Stage 8:=======================> (82 + 1) / 199[Stage 8:================================> (119 + 1) / 199[Stage 8:=============================================> (164 + 1) / 199 +-------+--------------+

| item|number\_of\_bids|

+-------+--------------+

| xbox| 2784|

| palm| 5917|

|cartier| 1953|

+-------+--------------+

>>>

This proves that even after exiting Pyspark and coming back, I can read from parquet file and construct a usable dataframe and temporary sql table out of it

Deliverables:

- Problem3.scala (which is the scala program for this problem)

**Problem 4**. Use Sqoop to import all tables in MySQL demo database retail\_db into Hive. Use Spark DataFrame API or Spark Temporary Tables to find orders with the largest number of order items per order.

I went to Cloudera Quickstart VM

I went to the terminal and gave this command:

[cloudera@quickstart ~]$ sqoop import-all-tables -m 1 --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba --password=cloudera --compression-codec=snappy --as-parquetfile --warehouse-dir=/user/hive/warehouse --hive-import

The command executed successfully. Trailing output of the command:

Map output records=1345

Input split bytes=87

Spilled Records=0

Failed Shuffles=0

Merged Map outputs=0

GC time elapsed (ms)=252

CPU time spent (ms)=2980

Physical memory (bytes) snapshot=181051392

Virtual memory (bytes) snapshot=1522274304

Total committed heap usage (bytes)=60620800

File Input Format Counters

Bytes Read=0

File Output Format Counters

Bytes Written=0

16/03/10 23:10:01 INFO mapreduce.ImportJobBase: Transferred 46.1318 KB in 35.5327 seconds (1.2983 KB/sec)

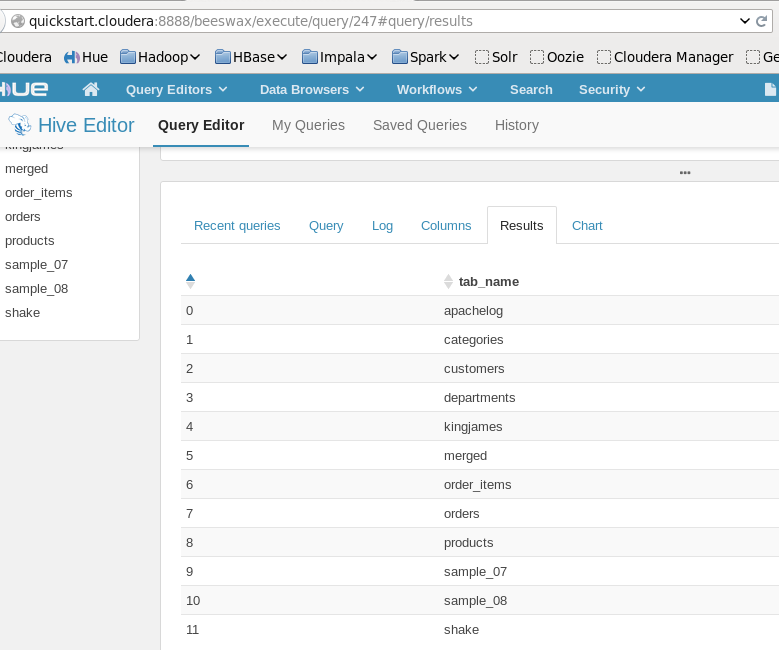
16/03/10 23:10:01 INFO mapreduce.ImportJobBase: Retrieved 1345 records.

[cloudera@quickstart ~]$

I went to Hue’s Hive editor and gave

show tables command

And saw the tables there:



Then I copied hive-site.xml from /etc/hive/conf to /etc/spark/conf/

[cloudera@quickstart ~]$ ls /etc/spark/conf/

docker.properties.template slaves.template

fairscheduler.xml.template spark-defaults.conf

hive-site.xml spark-defaults.conf.template

log4j.properties.template spark-env.sh

metrics.properties.template spark-env.sh.template

slaves

[cloudera@quickstart ~]$

Then I started hiveserver2 as daemon:

[cloudera@quickstart ~]$ hiveserver2 &

[3] 42509

[cloudera@quickstart ~]$

Then I started pyspark with the command:

[cloudera@quickstart ~]$ pyspark --driver-class-path libraries/mysql-connector-java-5.1.38/mysql-connector-java-5.1.38-bin.jar

To solve the problem: find orders with the largest number of order items per order

I have used 2 approaches:

Approach 1 (**with HiveContext**):

>>> hiveContext = HiveContext(sc)

>>>

I found out the max number of order items in an order:

>>> hiveContext.sql("select count(order\_item\_quantity) as number\_of\_order\_items from order\_items group by order\_item\_order\_id order by number\_of\_order\_items desc").show(1)

+---------------------+

|number\_of\_order\_items|

+---------------------+

| 5|

+---------------------+

only showing top 1 row

Then I found out orders which have 5 order items in them:

>>> hiveContext.sql("select orders.order\_id,orders.order\_date,orders.order\_customer\_id,orders.order\_status, count(\*) as num\_of\_order\_items from orders join order\_items on orders.order\_id = order\_items.order\_item\_order\_id group by orders.order\_id,orders.order\_date,orders.order\_customer\_id,orders.order\_status having num\_of\_order\_items=5 order by order\_id").show()

+--------+-------------+-----------------+---------------+------------------+

|order\_id| order\_date|order\_customer\_id| order\_status|num\_of\_order\_items|

+--------+-------------+-----------------+---------------+------------------+

| 5|1374735600000| 11318| COMPLETE| 5|

| 10|1374735600000| 5648|PENDING\_PAYMENT| 5|

| 11|1374735600000| 918| PAYMENT\_REVIEW| 5|

| 12|1374735600000| 1837| CLOSED| 5|

| 15|1374735600000| 2568| COMPLETE| 5|

| 17|1374735600000| 2667| COMPLETE| 5|

| 24|1374735600000| 11441| CLOSED| 5|

| 28|1374735600000| 656| COMPLETE| 5|

| 29|1374735600000| 196| PROCESSING| 5|

| 58|1374735600000| 9213|PENDING\_PAYMENT| 5|

| 64|1374735600000| 5579|PENDING\_PAYMENT| 5|

| 73|1374735600000| 8504|PENDING\_PAYMENT| 5|

| 77|1374735600000| 7915|PENDING\_PAYMENT| 5|

| 84|1374735600000| 6789| PROCESSING| 5|

| 107|1374822000000| 1845| COMPLETE| 5|

| 110|1374822000000| 2746| COMPLETE| 5|

| 112|1374822000000| 5375| CANCELED| 5|

| 116|1374822000000| 8763| CLOSED| 5|

| 122|1374822000000| 2071| PROCESSING| 5|

| 140|1374822000000| 4257|PENDING\_PAYMENT| 5|

+--------+-------------+-----------------+---------------+------------------+

only showing top 20 rows

So basically these are the orders which have maximum number of order items in them.

I have only shown 20 orders here,

There are 11353 of those as revealed by this query:

>>> hiveContext.sql("select orders.order\_id,orders.order\_date,orders.order\_customer\_id,orders.order\_status, count(\*) as num\_of\_order\_items from orders join order\_items on orders.order\_id = order\_items.order\_item\_order\_id group by orders.order\_id,orders.order\_date,orders.order\_customer\_id,orders.order\_status having num\_of\_order\_items=5").count()

11353

Note: I had to split this into 2 queries since it seems like nested queries are not supported in HiveContext. I tried different combinations of joins too, but couldn’t get this to work in one query.

Approach 2 (with DataFrames):

I created a dataframe from orders table and made sure that it is created correctly:

>>> ordersDFM = sqlContext.read.format("jdbc").option("url","jdbc:mysql://localhost/retail\_db").option("driver","com.mysql.jdbc.Driver").option("dbtable","orders").option("user","retail\_dba").option("password","cloudera").load()

>>> ordersDFM.printSchema()

root

|-- order\_id: integer (nullable = false)

|-- order\_date: timestamp (nullable = false)

|-- order\_customer\_id: integer (nullable = false)

|-- order\_status: string (nullable = false)

>>>

I created a dataframe from order\_items table and made sure it is created correctly:

>>> orderItemsDFM = sqlContext.read.format("jdbc").option("url","jdbc:mysql://localhost/retail\_db").option("driver","com.mysql.jdbc.Driver").option("dbtable","order\_items").option("user","retail\_dba").option("password","cloudera").load()

>>> orderItemsDFM.printSchema()

root

|-- order\_item\_id: integer (nullable = false)

|-- order\_item\_order\_id: integer (nullable = false)

|-- order\_item\_product\_id: integer (nullable = false)

|-- order\_item\_quantity: integer (nullable = false)

|-- order\_item\_subtotal: double (nullable = false)

|-- order\_item\_product\_price: double (nullable = false)

>>>

Then I found max number of order items for any order:

>>> orderItemsDFM.groupBy("order\_item\_order\_id").count().groupBy().max("count").show()

+----------+

|max(count)|

+----------+

| 5|

+----------+

Then I gave the query (using dataframe API) to find out orders with max number of order items:

>>> ordersDFM.join(orderItemsDFM, ordersDFM['order\_id']==orderItemsDFM['order\_item\_order\_id'], "inner").groupBy(ordersDFM['order\_id'],ordersDFM['order\_date'],ordersDFM['order\_customer\_id'],ordersDFM['order\_status']).count().filter("count=5").orderBy(ordersDFM['order\_id']).show()

+--------+--------------------+-----------------+---------------+-----+

|order\_id| order\_date|order\_customer\_id| order\_status|count|

+--------+--------------------+-----------------+---------------+-----+

| 5|2013-07-25 00:00:...| 11318| COMPLETE| 5|

| 10|2013-07-25 00:00:...| 5648|PENDING\_PAYMENT| 5|

| 11|2013-07-25 00:00:...| 918| PAYMENT\_REVIEW| 5|

| 12|2013-07-25 00:00:...| 1837| CLOSED| 5|

| 15|2013-07-25 00:00:...| 2568| COMPLETE| 5|

| 17|2013-07-25 00:00:...| 2667| COMPLETE| 5|

| 24|2013-07-25 00:00:...| 11441| CLOSED| 5|

| 28|2013-07-25 00:00:...| 656| COMPLETE| 5|

| 29|2013-07-25 00:00:...| 196| PROCESSING| 5|

| 58|2013-07-25 00:00:...| 9213|PENDING\_PAYMENT| 5|

| 64|2013-07-25 00:00:...| 5579|PENDING\_PAYMENT| 5|

| 73|2013-07-25 00:00:...| 8504|PENDING\_PAYMENT| 5|

| 77|2013-07-25 00:00:...| 7915|PENDING\_PAYMENT| 5|

| 84|2013-07-25 00:00:...| 6789| PROCESSING| 5|

| 107|2013-07-26 00:00:...| 1845| COMPLETE| 5|

| 110|2013-07-26 00:00:...| 2746| COMPLETE| 5|

| 112|2013-07-26 00:00:...| 5375| CANCELED| 5|

| 116|2013-07-26 00:00:...| 8763| CLOSED| 5|

| 122|2013-07-26 00:00:...| 2071| PROCESSING| 5|

| 140|2013-07-26 00:00:...| 4257|PENDING\_PAYMENT| 5|

+--------+--------------------+-----------------+---------------+-----+

only showing top 20 rows

I have shown only 20 rows.

There are totally 11353 rows as shown by this query:

>>> ordersDFM.join(orderItemsDFM, ordersDFM['order\_id']==orderItemsDFM['order\_item\_order\_id'], "inner").groupBy(ordersDFM['order\_id'],ordersDFM['order\_date'],ordersDFM['order\_customer\_id'],ordersDFM['order\_status']).count().filter("count=5").orderBy(ordersDFM['order\_id']).count()

11353

Note: I had to split this into 2 queries since I was not able to use ‘in’ clause with DataFrame API. I tried different combinations of joins too, but couldn’t get this to work in one query.