**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.

**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.

**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.

pyspark --packages com.databricks:spark-csv\_2.11:1.4.0

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

Auction.show(2)

Auction.count()

Auction.printSchema()

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

exit()

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

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

... sql queries...

**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.

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

invalidate metadata;

show tables;

Hive/impala:

select order\_item\_order\_id as order\_id, sum(order\_item\_quantity) as number\_of\_order\_items from order\_items group by order\_item\_order\_id order by number\_of\_order\_items desc limit 1

select \* from orders where order\_id IN (select order\_item\_order\_id as order\_id from order\_items group by order\_item\_order\_id having sum(order\_item\_quantity) = 24)

OR

select orders.order\_id from orders join order\_items on orders.order\_id = order\_items.order\_item\_id group by orders.order\_id having sum(order\_items.order\_item\_quantity) = 24

Subqueries are not supported in the HAVING clause.

Pyspark

ordersDFM =

sqlc.read.format("jdbc").option("url","jdbc:mysql://localhost/retail\_db").o

ption("driver","com.mysql.jdbc.Driver").option("dbtable","orders").optio

n("user","retail\_dba").option("password","cloudera").load()

orderItemsDFM

orderItemsDFM.groupBy(“order\_item\_order\_id”).sum(“order\_item\_quantity”).withColumnRenamed("sum(order\_item\_quantity)", "number\_of\_order\_items").filter(number\_of\_order\_items = 24).show()

orderItemsDFM.groupBy("order\_item\_order\_id").sum("order\_item\_quantity").withColumnRenamed("sum(order\_item\_quantity)", "number\_of\_order\_items").orderBy("number\_of\_order\_items").show(1)

ordersDFM.registerTempTable("orders\_temp\_table")

orderItemsDFM.registerTempTable("order\_items\_temp\_table")

sqlContext.sql(“select order\_item\_order\_id as order\_id, sum(order\_item\_quantity) as number\_of\_order\_items from order\_items\_temp\_table group by order\_item\_order\_id order by number\_of\_order\_items desc limit 1

”).show()

sqlContext.sql(“select \* from orders\_temp\_table where order\_id IN (select order\_item\_order\_id as order\_id from order\_items\_temp\_table group by order\_item\_order\_id having sum(order\_item\_quantity) = 24)”).show()

attempt at data frame:

orderItemsDFM.groupBy("order\_item\_order\_id").sum("order\_item\_quantity").withColumnRenamed("sum(order\_item\_quantity)", "number\_of\_order\_items").orderBy("number\_of\_order\_items").show(1)