Ryan Ballenger
CSCIE63 Big Data Analytics
Assignment 6

Assignment 6 Solutions

Problem 1.

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.

Paragraph A
Fans of the "Civil War" series know that Spider-Man has a fairly important part to play in the story, so it makes sense that Disney and Marvel wanted to include this superhero staple in the big battle to come. Whether or not it will gel with the complicated plot this story arc will have to tackle, remains to be seen. Until then, enjoy your new Spider-Man, people.

Paragraph B
Gone is the Andrew Garfield Spider-Man of yesterday, the new Spidey will be played by young actor Tom Holland. We don't get to see the face of the new actor in the newly revealed footage in the "Captain America: Civil War" trailer. But we do get to hear him, and check out his brand new suit. Complete with comic-book inspired, moving spidey eyes.

Commands and output

***only snippets of output as requested in my previous HW feedback**

```
// Compile my java program, named WordCount, which creates the RDDs

[cloudera@localhost mini-examples-java-v2]$ mvn clean && mvn compile && mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building example 0.0.1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ spark-example ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.057 s
[INFO] Finished at: 2016-03-11T11:33:24-08:00
[INFO] Final Memory: 5M/29M
[INFO] ------------------------------------------------------------------------
[INFO] Scanning for projects...
…

// Run the program WordCount to create the RDDs from
// paragraph A and B, paragraphA.txt and paragraphB.txt

[cloudera@localhost mini-examples-java-v2]$ spark-submit --class
edu.hu.examples.WordCount ./target/spark-example-0.0.1.jar paragraphA.txt paragraphB.txt
test1 test2
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]
```

```
16/03/11 11:40:12 INFO spark.SparkContext: Running Spark version 1.5.0-cdh5.5.2
16/03/11 11:40:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
16/03/11 11:40:14 WARN util.Utils: Your hostname, localhost.localdomain resolves to a
loopback address: 127.0.0.1; using 172.16.40.129 instead (on interface eth0)
16/03/11 11:40:14 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
16/03/11 11:40:14 INFO spark.SecurityManager: Changing view acls to: cloudera
16/03/11 11:40:14 INFO spark.SecurityManager: Changing modify acls to: cloudera
16/03/11 11:40:14 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui
acls disabled; users with view permissions: Set(cloudera); users with modify permissions:
Set(cloudera)
...
```

First 10 Words

First 10 words of JavaRDD<String> paragraphA (from output/test1/part-r-00000)

```
fans
of
the
civil
war
series
know
that
spiderman
has
```

First 10 words of JavaRDD<String> paragraphB (from output/test2/part-r-00000)

```
gone
is
the
andrew
garfield
spiderman
of
yesterday
the
new
```

First 10 words of JavaRDD<String> uniqueParagraphA (from output/uniqueA/part-r-00000)

```
play
```

```
this
it
have
include
until
big
with
sense
in
```

First 10 words of JavaRDD<String> uniqueParagraphB (from output/uniqueB/part-r-00000)

```
hear
captain
inspired
is
complete
check
trailer
gone
holland
tom
```

First 10 words of JavaRDD<String> paraAandNotB (from output/A_not_B/part-r-00000)

```
play
complicated
this
not
it
people
whether
part
that
a
```

First 10 words of JavaRDD<String> commonWords (from output/common/part-r-00000)

```
civil
be
with
will
to
in
war
of
```

spiderman
new

Java Code for WordCount

```
package edu.hu.examples;

import java.util.Arrays;
import java.util.*;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;

import scala.Tuple2;

public class WordCount {
	public static void main(String[] args) throws Exception {
		String inputFile = args[0];
			String inputFileTwo = args[1];
		String outputFile = args[2];
			String outputFileTwo = args[3];

		// Create a Java Spark Context.
		SparkConf conf = new SparkConf().setAppName("wordCount");
		JavaSparkContext sc = new JavaSparkContext(conf);

		// Load our input data.
		JavaRDD<String> paragraphA = sc.textFile(inputFile);
			JavaRDD<String> paragraphB = sc.textFile(inputFileTwo);


			// RDDs that contain only words
		paragraphA = paragraphA.flatMap(new FlatMapFunction<String, String>() {
			public Iterable<String> call(String x) {
				List<String> words = Arrays.asList(x.split(" "));
				       for(int i=0; i<words.size(); i++){
						    String word =
words.get(i).replaceAll("[^A-Za-z0-9]", "").toLowerCase();
						    words.set(i, word);
				       }
				      return words;
			}
		});
```

```
                    paragraphB = paragraphB.flatMap(new FlatMapFunction<String,
String>() {
                    public Iterable<String> call(String x) {
                                  List<String> words = Arrays.asList(x.split(" "));
                                 for(int i=0; i<words.size(); i++){
                                  String word =
words.get(i).replaceAll("[^A-Za-z0-9]", "").toLowerCase();
                                        words.set(i, word);
                                 }
                                 return words;
                    }
                    });

        // unique words in each paragraph
        JavaRDD<String> uniqueParagraphA = paragraphA.distinct();
        JavaRDD<String> uniqueParagraphB = paragraphB.distinct();

        // present in paragraph A but not in paragraph B
        JavaRDD<String> paraAandNotB = paragraphA.subtract(paragraphB).distinct();

        // only words common in both paragraphs
        JavaRDD<String> commonWords = paragraphA.intersection(paragraphB).distinct();

        // Save the word count back out to a text file, causing evaluation.
        paragraphA.saveAsTextFile("output/allWordsA");
        paragraphB.saveAsTextFile("output/allWordsB");
        uniqueParagraphA.saveAsTextFile("output/uniqueA");
        uniqueParagraphB.saveAsTextFile("output/uniqueB");
        paraAandNotB.saveAsTextFile("output/A_not_B");
        commonWords.saveAsTextFile("output/common");
       }
}
```

Problem 2.

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.

Commands and output

Only snippets of output as requested by feedback in my previous HW.

// my_script.py is submitted and successfully runs

```
[cloudera@localhost ~]$ spark-submit my_script.py
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]
16/03/11 12:13:04 INFO spark.SparkContext: Running Spark version 1.5.0-cdh5.5.2
16/03/11 12:13:05 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
16/03/11 12:13:06 WARN util.Utils: Your hostname, localhost.localdomain resolves to a
loopback address: 127.0.0.1; using 172.16.40.129 instead (on interface eth0)
16/03/11 12:13:06 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
16/03/11 12:13:06 INFO spark.SecurityManager: Changing view acls to: cloudera
16/03/11 12:13:06 INFO spark.SecurityManager: Changing modify acls to: cloudera
16/03/11 12:13:06 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui
acls disabled; users with view permissions: Set(cloudera); users with modify permissions:
Set(cloudera)
16/03/11 12:13:08 INFO slf4j.Slf4jLogger: Slf4jLogger started
16/03/11 12:13:08 INFO Remoting: Starting remoting
...
```

Content of the data frame

Outputted during the spark-submit my_script.py command from above:

```
+---+-------+-------+
|age|   name| salary|
+---+-------+-------+
| 29|Michael| 3000.3|
| 30|   Andy|2500.25|
| 19| Justin|4000.99|
+---+-------+-------+
```

Query

Select names of employees who have a salary greater than 3500. The query is also outputted during the above command.

```
+------+
|  name|
+------+
|Justin|
+------+
```

Python Code for Problem 2.

```
import re
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext, Row

conf = SparkConf().setMaster("local").setAppName("MyApp")
sc = SparkContext(conf = conf)

# create SQLContext with the SparkContext parameter
sqlContext = SQLContext(sc)

# RDD from importing file emps.txt
emps = sc.textFile("/user/cloudera/emps.txt")

# splitting on the commas to make 3 individual elements
emps_fields = emps.map(lambda line: (line.split(", ")))

# create rows
employees = emps_fields.map(lambda e: Row(name = e[0], age =
int(e[1]), salary = float(e[2])))

# create a dataframe with the rowed data
empsDF = sqlContext.createDataFrame(employees)
```

```
# show the dataframe
empsDF.show()

# create a temporary table
empsDF.registerTempTable("emps")

# select employees with salaries > 3500
sqlContext.sql("select name from emps where salary > 3500").show()
```

Problem 3.

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:

- How many auctions were held?
- 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?
- 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.

Schema of the dataframe

```
>>> # print the schema of the dataframe
... Auction.printSchema()
root
 |-- auctionid: long (nullable = true)
 |-- bid: double (nullable = true)
 |-- bidder: string (nullable = true)
 |-- bidderrate: double (nullable = true)
 |-- bidtime: double (nullable = true)
 |-- daystolive: double (nullable = true)
 |-- item: string (nullable = true)
 |-- openbid: double (nullable = true)
 |-- price: double (nullable = true)
```

Spark Dataframe Queries

// I submit my script my_script.py which includes the 4 dataframe queries.
// Following the output snippet, the results of the 4 queries, which were within the output,
// are displayed.

```
[cloudera@quickstart Documents]$ spark-submit my_script.py
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/jars/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]
16/03/11 13:46:56 INFO spark.SparkContext: Running Spark version 1.5.0-cdh5.5.0
16/03/11 13:46:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
16/03/11 13:46:58 WARN util.Utils: Your hostname, quickstart.cloudera resolves to a loopback
address: 127.0.0.1; using 172.16.40.132 instead (on interface eth1)
```

```
16/03/11 13:46:58 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
...

// How many auctions were held?

Number of auctions: 627

// How many bids were made per item?
// I found the bid counts by item, as instructed, as opposed to by “auctionid” which could be
// another interpretation of the question.

+-------+----------+
|   item|count(bid)|
+-------+----------+
|   xbox|      2784|
|   palm|      5917|
|cartier|      1953|
+-------+----------+

// Minimum, maximum, and average bid (price) per item
// The min., max., and avg. prices per item are determined. The final bid price is analyzed as
// opposed to the amount of each submitted bid, since the question specifies “price”.

+-------+----------+----------+-----------------+
|   item|max(price)|min(price)|       avg(price)|
+-------+----------+----------+-----------------+
|   xbox|    501.77|      31.0|144.27594109195397|
|   palm|     290.0|     175.0| 231.1302670272107|
|cartier|    5400.0|      26.0| 925.0479057859695|
+-------+----------+----------+-----------------+

// Bids with price > 100
// For this query, bids with “bid” > 100 are found, since “bid” represents the price of
// the specific bid and not the closing price of the item (“price”).

+----------+------+-----------------+----------+--------+----------+----+-------+-----+
| auctionid|   bid|           bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+------+-----------------+----------+--------+----------+----+-------+-----+
|8213034705| 115.0|     davidbresler2|       1.0|2.943484|       3.0|xbox|   95.0|117.5|
|8213034705| 117.5|           daysrus|      10.0|2.998947|       3.0|xbox|   95.0|117.5|
|8213060420| 110.0|         jhnsn2273|      51.0|1.013056|       3.0|xbox|    1.0|120.0|
|8213060420| 105.0|          pagep123|       2.0| 2.79934|       3.0|xbox|    1.0|120.0|
|8213060420| 110.0|         pagep123|       2.0|2.799676|       3.0|xbox|    1.0|120.0|
|8213060420| 115.0|         pagep123|       2.0|2.800197|       3.0|xbox|    1.0|120.0|
|8213060420| 115.0|        skcardina|       1.0|2.968495|       3.0|xbox|    1.0|120.0|
|8213060420| 117.5|        skcardina|       1.0|2.972766|       3.0|xbox|    1.0|120.0|
|8213060420| 120.0|  djnoeproductions|      17.0|2.999722|       3.0|xbox|    1.0|120.0|
|8213067838| 120.0|          godb_24|       0.0|2.946516|       3.0|xbox|  29.99|132.5|
|8213067838| 105.0|       unique82me|       0.0|2.989051|       3.0|xbox|  29.99|132.5|
|8213067838| 110.0|       unique82me|       0.0|2.989178|       3.0|xbox|  29.99|132.5|
|8213067838| 115.0|       unique82me|       0.0|2.989317|       3.0|xbox|  29.99|132.5|
|8213067838| 120.0|       unique82me|       0.0|2.989387|       3.0|xbox|  29.99|132.5|
|8213067838| 125.0|       unique82me|       0.0|2.989468|       3.0|xbox|  29.99|132.5|
|8213067838| 130.0|       unique82me|       0.0| 2.99316|       3.0|xbox|  29.99|132.5|
|8213067838| 130.0|*champaignbubbles*|     202.0|2.996516|       3.0|xbox|  29.99|132.5|
|8213067838| 132.5|*champaignbubbles*|     202.0|2.996632|       3.0|xbox|  29.99|132.5|
|8213067838| 132.5|*champaignbubbles*|     202.0|2.997789|       3.0|xbox|  29.99|132.5|
|8213073509|101.09|          djfelony|     265.0|2.639931|       3.0|xbox|    1.0|114.5|
+----------+------+------------------+----------+--------+----------+----+-------+-----+
only showing top 20 rows
```

SQL queries

// All my queries are run at once with a script named my_script.py. Below is a snippet of the
// output followed by the results of the queries, which were also part of the output.

// Command and output snippet

```
[cloudera@quickstart ~]$ spark-submit my_script.py
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/jars/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]
16/03/11 17:19:56 INFO spark.SparkContext: Running Spark version 1.5.0-cdh5.5.0
16/03/11 17:19:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
16/03/11 17:19:58 WARN util.Utils: Your hostname, quickstart.cloudera resolves to a loopback
address: 127.0.0.1; using 172.16.40.133 instead (on interface eth2)
```

```
16/03/11 17:19:58 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
16/03/11 17:19:58 INFO spark.SecurityManager: Changing view acls to: cloudera
16/03/11 17:19:58 INFO spark.SecurityManager: Changing modify acls to: cloudera
16/03/11 17:19:58 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui
acls disabled; users with view permissions: Set(cloudera); users with modify permissions:
...

// How many auctions were held?
+---+
|_c0|
+---+
|627|
+---+

// How many bids were made per item?
+-------+----+
|   item| _c1|
+-------+----+
|   xbox|2784|
|   palm|5917|
|cartier|1953|
+-------+----+


// Minimum, maximum, and average bid (price) per item
// As mentioned earlier, this query finds the corresponding prices, which are the final bid prices,
// and not the value of each bid, since "price" is specified in the question.
+-------+------+-----+------------------+
|   item|   Max|  Min|               Avg|
+-------+------+-----+------------------+
|   xbox|501.77| 31.0|144.27594109195397|
|   palm| 290.0|175.0| 231.1302670272107|
|cartier|5400.0| 26.0| 925.0479057859695|
+-------+------+-----+------------------+


// Show the bids with price > 100
// The query finds the bids with "bid" > 100 meaning that specific bid was greater than 100 and
// not the price, which is the final price of the auction.
+----------+------+-----------------+----------+--------+----------+----+-------+-----+
| auctionid|   bid|           bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+------+-----------------+----------+--------+----------+----+-------+-----+
|8213034705| 115.0|     davidbresler2|       1.0|2.943484|       3.0|xbox|   95.0|117.5|
|8213034705| 117.5|           daysrus|      10.0|2.998947|       3.0|xbox|   95.0|117.5|
|8213060420| 110.0|         jhnsn2273|      51.0|1.013056|       3.0|xbox|    1.0|120.0|
|8213060420| 105.0|          pagep123|       2.0| 2.79934|       3.0|xbox|    1.0|120.0|
|8213060420| 110.0|          pagep123|       2.0|2.799676|       3.0|xbox|    1.0|120.0|
|8213060420| 115.0|          pagep123|       2.0|2.800197|       3.0|xbox|    1.0|120.0|
|8213060420| 115.0|         skcardina|       1.0|2.968495|       3.0|xbox|    1.0|120.0|
|8213060420| 117.5|         skcardina|       1.0|2.972766|       3.0|xbox|    1.0|120.0|
|8213060420| 120.0|  djnoeproductions|      17.0|2.999722|       3.0|xbox|    1.0|120.0|
|8213067838| 120.0|           godb_24|       0.0|2.946516|       3.0|xbox|  29.99|132.5|
|8213067838| 105.0|        unique82me|       0.0|2.989051|       3.0|xbox|  29.99|132.5|
|8213067838| 110.0|        unique82me|       0.0|2.989178|       3.0|xbox|  29.99|132.5|
|8213067838| 115.0|        unique82me|       0.0|2.989317|       3.0|xbox|  29.99|132.5|
|8213067838| 120.0|        unique82me|       0.0|2.989387|       3.0|xbox|  29.99|132.5|
|8213067838| 125.0|        unique82me|       0.0|2.989468|       3.0|xbox|  29.99|132.5|
|8213067838| 130.0|        unique82me|       0.0| 2.99316|       3.0|xbox|  29.99|132.5|
|8213067838| 130.0|*champaignbubbles*|     202.0|2.996516|       3.0|xbox|  29.99|132.5|
|8213067838| 132.5|*champaignbubbles*|     202.0|2.996632|       3.0|xbox|  29.99|132.5|
|8213067838| 132.5|*champaignbubbles*|     202.0|2.997789|       3.0|xbox|  29.99|132.5|
|8213073509|101.09|           djfelony|     265.0|2.639931|       3.0|xbox|    1.0|114.5|
+----------+------+------------------+----------+--------+----------+----+-------+-----+
only showing top 20 rows
```

Parquet File

```
// From pyspark, my dataframe is persisted as a Parquet file.
// A snippet of the output is shown after the command.

>>> Auction.write.save("auction.parquet",format="parquet")
16/03/11 17:38:48 INFO parquet.ParquetRelation: Using default output committer for Parquet: parquet.hadoop.ParquetOutputCommitter
16/03/11 17:38:48 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
16/03/11 17:38:48 INFO datasources.DefaultWriterContainer: Using user defined output committer class parquet.hadoop.ParquetOutputCommitter
16/03/11 17:38:48 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
16/03/11 17:38:48 INFO spark.SparkContext: Starting job: save at NativeMethodAccessorImpl.java:-2
```

```
16/03/11 17:38:48 INFO scheduler.DAGScheduler: Got job 1 (save at
NativeMethodAccessorImpl.java:-2) with 1 output partitions
16/03/11 17:38:48 INFO scheduler.DAGScheduler: Final stage: ResultStage 1(save at
NativeMethodAccessorImpl.java:-2)
16/03/11 17:38:48 INFO scheduler.DAGScheduler: Parents of final stage: List()
16/03/11 17:38:48 INFO scheduler.DAGScheduler: Missing parents: List()
16/03/11 17:38:48 INFO scheduler.DAGScheduler: Submitting ResultStage 1
...

// Pyspark is exited and restarted.
>>> quit()
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/static/sql,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/SQL/execution/json,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/SQL/execution,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/SQL/json,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/SQL,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/metrics/json,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/stages/stage/kill,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/api,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped o.s.j.s.ServletContextHandler{/,null}
16/03/11 17:42:07 INFO handler.ContextHandler: stopped
o.s.j.s.ServletContextHandler{/static,null}
...

// The parquet file is read and stored in Auction2.
>>> Auction2 = sqlContext.read.parquet("auction.parquet")
16/03/11 17:44:14 INFO hive.HiveContext: Initializing execution hive, version 1.1.0
16/03/11 17:44:14 INFO client.ClientWrapper: Inspected Hadoop version: 2.6.0-cdh5.5.0
16/03/11 17:44:14 INFO client.ClientWrapper: Loaded
org.apache.hadoop.hive.shims.Hadoop23Shims for Hadoop version 2.6.0-cdh5.5.0
16/03/11 17:44:14 INFO hive.metastore: Trying to connect to metastore with URI
thrift://127.0.0.1:9083
16/03/11 17:44:14 INFO hive.metastore: Opened a connection to metastore, current
connections: 1
16/03/11 17:44:14 INFO hive.metastore: Connected to metastore.
```

```
16/03/11 17:44:16 WARN shortcircuit.DomainSocketFactory: The short-circuit local reads
feature cannot be used because libhadoop cannot be loaded.
16/03/11 17:44:16 INFO session.SessionState: Created HDFS directory:
file:/tmp/spark-94b8be2f-755e-466d-ad7a-fcb7e2ab363f/scratch/cloudera
16/03/11 17:44:16 INFO session.SessionState: Created local directory:
/tmp/7758f258-9801-41d7-8c99-9d10453bbe45_resources
16/03/11 17:44:16 INFO session.SessionState: Created HDFS directory:
file:/tmp/spark-94b8be2f-755e-466d-ad7a-fcb7e2ab363f/scratch/cloudera/7758f258-9801-41d7-
8c99-9d10453bbe45
...

// A table named Auction2 is created from the parquet-derived dataframe.
>>> Auction2.registerTempTable("Auction2")

// An SQL query is run on the Auction2 table to show it works.
>>> sqlContext.sql("select * from Auction2").show(3)
16/03/11 17:47:32 INFO storage.BlockManagerInfo: Removed broadcast_0_piece0 on
localhost:36551 in memory (size: 22.3 KB, free: 534.5 MB)
16/03/11 17:47:32 INFO spark.ContextCleaner: Cleaned accumulator 2
16/03/11 17:47:32 INFO parse.ParseDriver: Parsing command: select * from Auction2
16/03/11 17:47:33 INFO parse.ParseDriver: Parse Completed
16/03/11 17:47:33 INFO storage.MemoryStore: ensureFreeSpace(92440) called with
curMem=0, maxMem=560497950
16/03/11 17:47:33 INFO storage.MemoryStore: Block broadcast_1 stored as values in memory
(estimated size 90.3 KB, free 534.4 MB)
16/03/11 17:47:33 INFO storage.MemoryStore: ensureFreeSpace(21233) called with
curMem=92440, maxMem=560497950
16/03/11 17:47:33 INFO storage.MemoryStore: Block broadcast_1_piece0 stored as bytes in
memory (estimated size 20.7 KB, free 534.4 MB)
16/03/11 17:47:33 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on
localhost:36551 (size: 20.7 KB, free: 534.5 MB)
16/03/11 17:47:33 INFO spark.SparkContext: Created broadcast 1 from showString at
NativeMethodAccessorImpl.java:-2
16/03/11 17:47:34 INFO Configuration.deprecation: mapred.min.split.size is deprecated.
Instead, use mapreduce.input.fileinputformat.split.minsize
...
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|       3.0|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|       3.0|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|       3.0|xbox|   95.0|117.5|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
```

only showing top 3 rows

Python Code for Problem 3

```
import re
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext, Row, functions

#pyspark.sql.functions

conf = SparkConf().setMaster("local").setAppName("MyApp")
sc = SparkContext(conf = conf)
theFile = sc.textFile("/user/cloudera/ebay.csv")

# create SQL context
sqlContext = SQLContext(sc)

ebay_fields = theFile.map(lambda line: (line.split(",")))

# create rows from the ebay fields
ebay_rows = ebay_fields.map(lambda e: Row(auctionid = int(e[0]),
                                          bid = float(e[1]),
                                          bidtime = float(e[2]),
                                          bidder=e[3],
                                          bidderrate=float(e[4]),
                                          openbid=float(e[5]),
                                          price=float(e[6]),
                                          item=e[7],
                                          daystolive=float(e[8])
                                          ))

# create the dataframe for the ebay data
Auction = sqlContext.createDataFrame(ebay_rows)

# print the schema of the dataframe
Auction.printSchema()

#####
# Dataframe queries

# How many auctions were held?
```

```
print "\nNumber of auctions: " +
str(Auction.select(Auction.auctionid).distinct().count())

# How many bids were made per item?
Auction.groupBy("item").agg({"bid":"count"}).show()

# minimum, maximum, and average bid (price) per item
maxP = Auction.groupBy("item").agg({"price":"max"})
minP = Auction.groupBy("item").agg({"price":"min"})
avgP = Auction.groupBy("item").agg({"price":"avg"})

combo = maxP.join(minP, maxP.item == minP.item,
'outer').drop(minP.item)
combo = combo.join(avgP, combo.item == avgP.item,
'outer').drop(avgP.item)
combo.select("item", "max(price)", "min(price)", "avg(price)").show()

# bids with price > 100
Auction.filter(Auction.bid > 100.0).show()


######
#SQL queries

Auction.registerTempTable("Auction")

#number of auctions
sqlContext.sql("select count(distinct(auctionid)) from
Auction").show()

# number of bids per item
sqlContext.sql("select item, count(*) from Auction Group By
item").show()

# minimum, maximum, and average bid (price) per item
sqlContext.sql("select item, max(price) as Max, min(price) as Min,
avg(price) as Avg from Auction Group By item").show()

# bids > 100
sqlContext.sql("select * from Auction where bid > 100.0").show()


#####
```

```
# Parquet File storage and access

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

Auction2 = sqlContext.read.parquet("auction.parquet")

Auction2.registerTempTable("Auction2")

sqlContext.sql("select * from Auction2").show(3)
```

Problem 4.

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.

```
// All the tables in the retail_db are imported into Hive with Squoop
[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
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
16/03/11 18:07:10 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.5.0
16/03/11 18:07:10 WARN tool.BaseSqoopTool: Setting your password on the command-line is
insecure. Consider using -P instead.
16/03/11 18:07:10 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can
override
16/03/11 18:07:10 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
16/03/11 18:07:10 WARN tool.BaseSqoopTool: It seems that you're doing hive import directly
into default
16/03/11 18:07:10 WARN tool.BaseSqoopTool: hive warehouse directory which is not
supported. Sqoop is
16/03/11 18:07:10 WARN tool.BaseSqoopTool: firstly importing data into separate directory and
then
16/03/11 18:07:10 WARN tool.BaseSqoopTool: inserting data into hive. Please consider
removing
16/03/11 18:07:10 WARN tool.BaseSqoopTool: --target-dir or --warehouse-dir into
/user/hive/warehouse in
16/03/11 18:07:10 WARN tool.BaseSqoopTool: case that you will detect any issues.
```

```
16/03/11 18:07:10 INFO manager.MySQLManager: Preparing to use a MySQL streaming
resultset.
...

// Start the hive metadata server for Spark to locate Hive
[cloudera@quickstart ~]$ hiveserver2 &
[1] 25296

// Copy the hive-site.xml file to be accessible by Spark
[cloudera@quickstart ~]$ sudo cp /etc/hive/conf/hive-site.xml /etc/spark/conf/

// Create a Hive Context in pyspark and submit the query for largest orders
>>> from pyspark.sql import SQLContext, HiveContext
>>> hivecontext = HiveContext(sc)

>>> hivecontext.sql("select order_item_order_id, count(*) as count from order_items Group By
order_item_order_id Order By count DESC").show()
...
16/03/11 18:46:20 INFO scheduler.TaskSetManager: Starting task 199.0 in stage 2.0 (TID 201,
localhost, partition 199,PROCESS_LOCAL, 1914 bytes)
16/03/11 18:46:20 INFO executor.Executor: Running task 199.0 in stage 2.0 (TID 201)
16/03/11 18:46:20 INFO scheduler.TaskSetManager: Finished task 198.0 in stage 2.0 (TID 200)
in 25 ms on localhost (199/200)
16/03/11 18:46:20 INFO storage.ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of
1 blocks
16/03/11 18:46:20 INFO storage.ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/03/11 18:46:20 INFO executor.Executor: Finished task 199.0 in stage 2.0 (TID 201). 3414
bytes result sent to driver
16/03/11 18:46:20 INFO scheduler.DAGScheduler: ResultStage 2 (showString at
NativeMethodAccessorImpl.java:-2) finished in 11.069 s
16/03/11 18:46:20 INFO scheduler.DAGScheduler: Job 1 finished: showString at
NativeMethodAccessorImpl.java:-2, took 14.646290 s
16/03/11 18:46:20 INFO scheduler.TaskSetManager: Finished task 199.0 in stage 2.0 (TID 201)
in 28 ms on localhost (200/200)
16/03/11 18:46:20 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks
have all completed, from pool
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|              13431|    5|
|              45831|    5|
|              14231|    5|
|               3231|    5|
|              15031|    5|
|               3831|    5|
|              16431|    5|
|               4431|    5|
|              19831|    5|
|               4831|    5|
|              20631|    5|
|               6231|    5|
|              22231|    5|
|               8031|    5|
|              23231|    5|
|               9031|    5|
|              23631|    5|
|              10231|    5|
|              24431|    5|
|              30631|    5|
+-------------------+-----+
only showing top 20 rows
```