HU Extension Assignment 06 E63 Big Data Analytics

Issued on: March 05, 2016 Due by 11:30PM EST, March 11, 2016
Submission is late by 40 minutes

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.

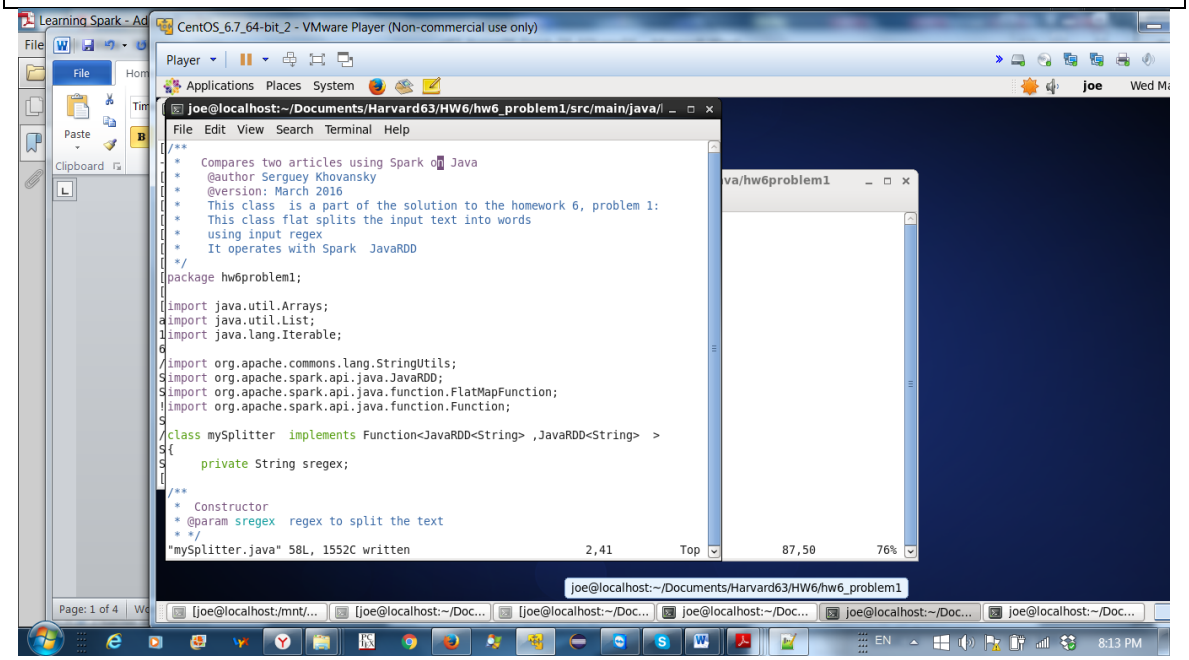
Save those paragraphs as .txt files and then import them into two Spark RDD objects, `paragraphA` and `paragraphB`.
Use Spark transformation functions to transform those initial RDD-s into RDD-s that contain only words.

Solution:

I import and split the files into words using a class mySplitter, which is saved as a stand alone file named mySplitter.java

```
class mySplitter  implements Function<JavaRDD<String> ,JavaRDD<String>  >
{
    private String sregex;

/**
 *  Constructor
 * @param sregex  regex to split the text
 * */
     mySplitter(String sregex)
     {
       this.sregex = sregex;
     }

/**
 * Call function is implemented from Function class of Spark
 * @param input  JavaRDD<String>  object, obtained from input text file
 * @return words JavaRDD<String>  output object with words split
 *
 * */
  @Override
    public  JavaRDD<String>  call( JavaRDD<String>  input) {
          JavaRDD<String>  words = input.flatMap
       (
          new FlatMapFunction<String, String>()
```

```
        {
          public Iterable<String> call(String x)
          {
            //return Arrays.asList(x.toLowerCase().split( "\\W+") );
            return Arrays.asList(x.toLowerCase().split(sregex));
            }
        }
      );
    return words;
  }
}
```



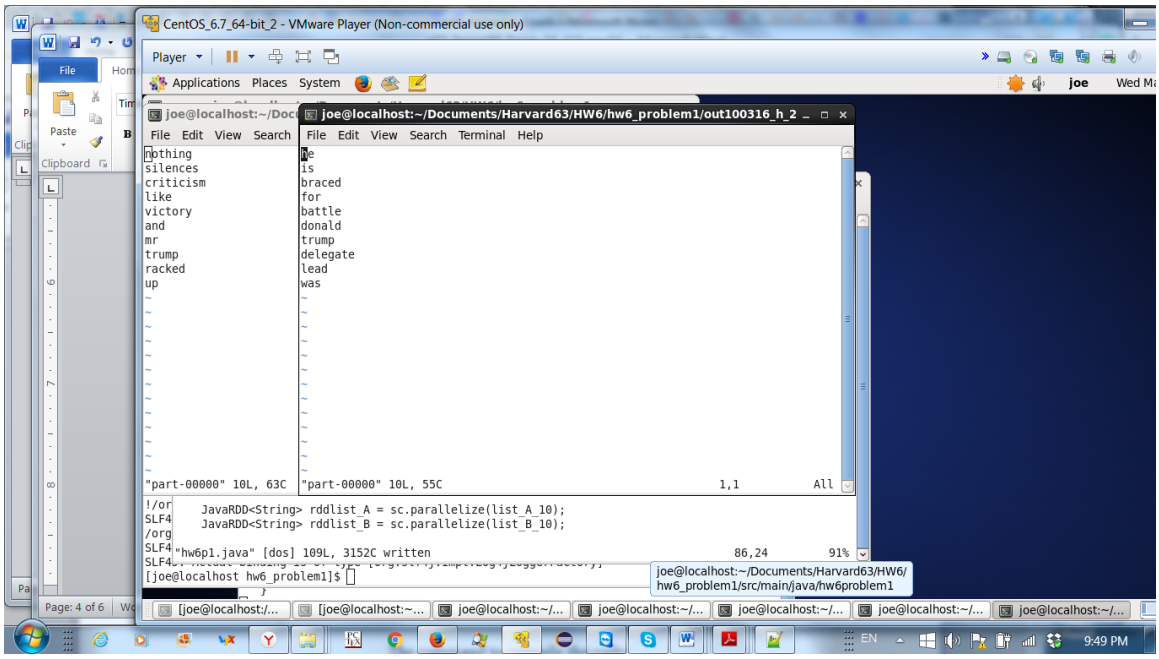
To read and split the files use the following lines in the class with main method (named hw6problem1)

```
String sregex = "\\W+";
 mySplitter  MySplitter = new mySplitter(sregex );

 JavaRDD<String> words1 =  MySplitter.call(input1);
 JavaRDD<String> words2 =  MySplitter.call(input2);
```

List for us the first 10 words in each RDD.

It is possible to create a special class for each operation, similar to mySplitter class, which I used to split the original input files, but I think that the overhead in that class would exceed the several extra lines that I need to print out here to handle the output twice.

```
List<String>  list_A_10 =  paragraphA.take(10);
List<String>  list_B_10 =  paragraphB.take(10);
JavaRDD<String> rddlist_A = sc.parallelize(list_A_10);
JavaRDD<String> rddlist_B = sc.parallelize(list_B_10);
rddlist_A.saveAsTextFile(outputFile1);
rddlist_B.saveAsTextFile(outputFile2);
```



Subsequently create RDD-s that contain only unique words in each of paragraphs.

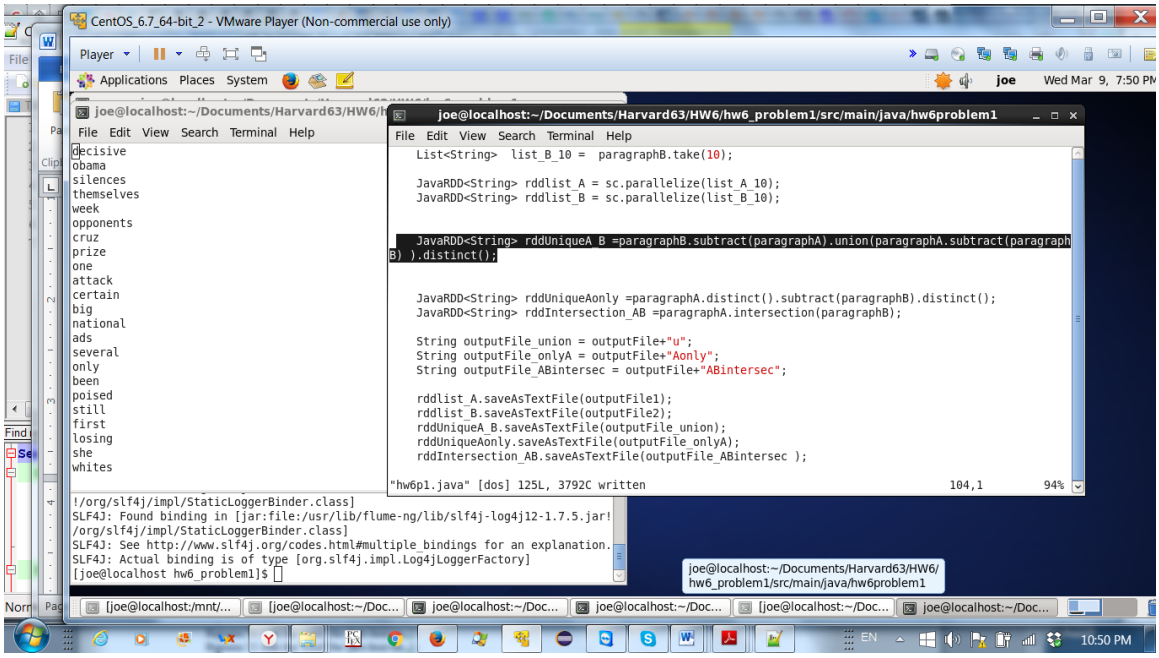
Relevant line:

```
JavaRDD<String> rddUniqueA_B
=paragraphB.subtract(paragraphA).union(paragraphA.subtract(paragraphB) ).distinct();
```

Output:

```
decisive
obama
silences
themselves
week
opponents
cruz
prize
one
attack
```

certain
big
national
ads
several
only
been
poised
still
first
losing
she
whites

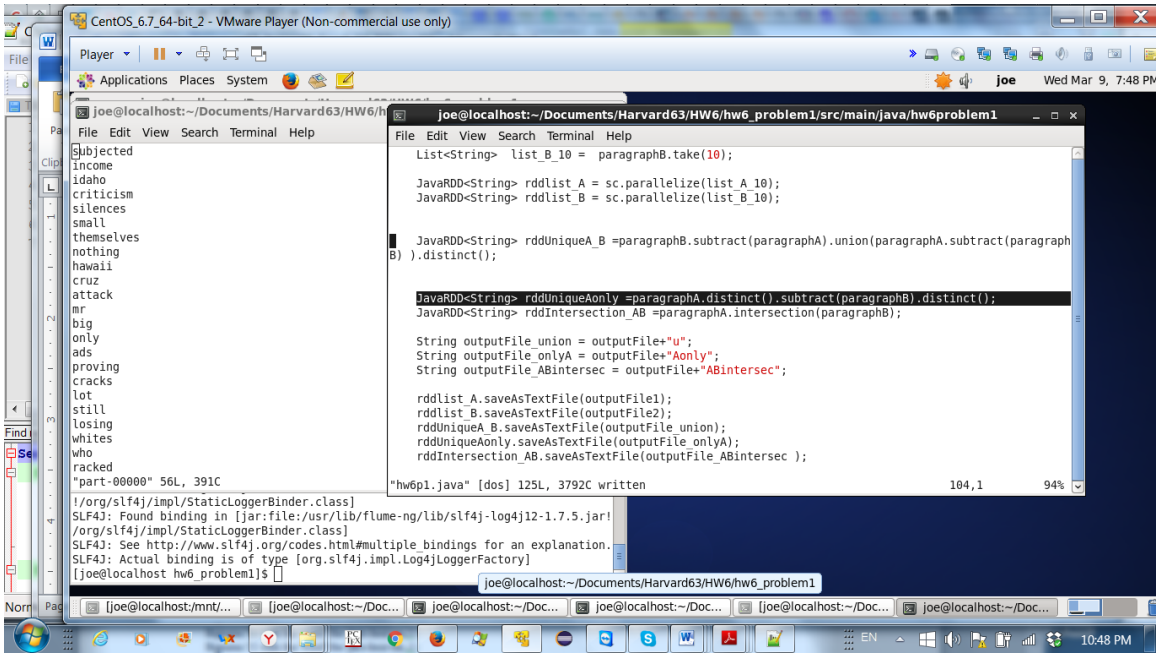


Then create an RDD that contains only words that are present in `paragraphA` but not in `paragraphB`.

Relevant part of the code:

```
JavaRDD<String> rddUniqueAonly =paragraphA.distinct().subtract(paragraphB).distinct();
```

Picture of the relevant code with the part of the output



Output, this list is incomplete

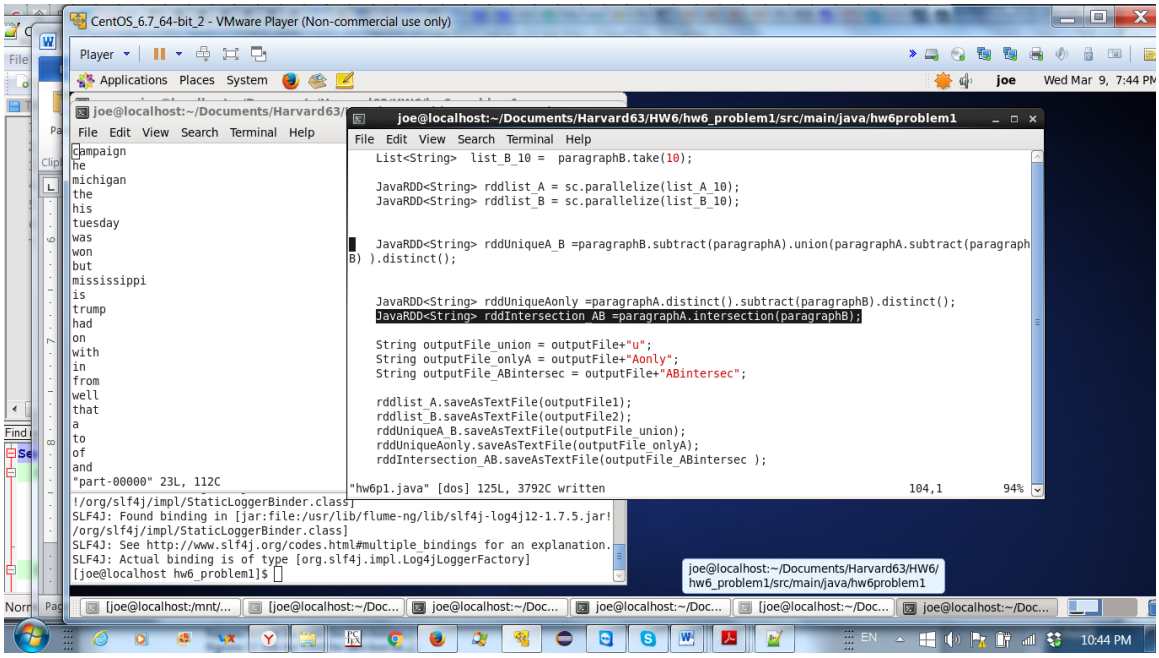
ubjected
income
idaho
criticism
silences
small
themselves
nothing
hawaii
cruz
attack
mr
big
only
ads
proving
cracks
lot
still
losing
whites
who
racked

Finally create an RDD that contains only the words common to two paragraphs.

Relevant part of the code

```
JavaRDD<String> rddIntersection_AB =paragraphA.intersection(paragraphB);
```

Output:



Output

```
campaign
he
michigan
the
his
tuesday
was
won
but
mississippi
is
trump
had
on
with
in
from
well
that
a
to
of
and
```

Compilation:

Prepare Maven

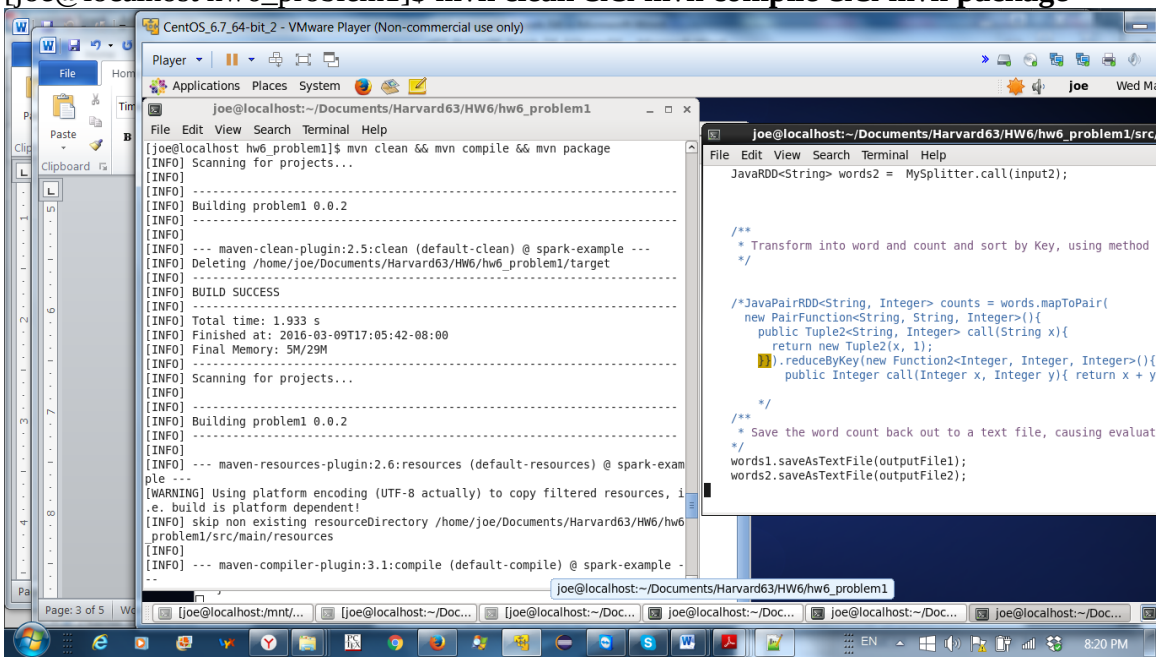
```
<project>
  <groupId>hw6problem1</groupId>
  <artifactId>spark-example</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>problem1</name>
  <packaging>jar</packaging>
  <version>0.0.2</version>
  <dependencies>
    <dependency> <!-- Spark dependency -->
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.10</artifactId>
      <version>1.1.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <build>
    <pluginManagement>
```

```
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
 </build>
</project>
```

Place pom.xml in the folder :
/home/joe/Documents/Harvard63/HW6/hw6_problem1
[joe@localhost hw6_problem1]$ ls
hw6p1_data pom.xml src target

Place the code hw6p1.java into the folder:
/home/joe/Documents/Harvard63/HW6/hw6_problem1/src/hw6problem1
[joe@localhost hw6problem1]$ ls
hw6p1.java

Run maven, being in the folder where pom.xml is located, i.e.
/home/joe/Documents/Harvard63/HW6/hw6_problem1
[joe@localhost hw6_problem1]$ **mvn clean && mvn compile && mvn package**



Run:

Folder:
/home/joe/Documents/Harvard63/HW6/hw6_problem1

[joe@localhost hw6_problem1]$ **spark-submit --class hw6problem1.hw6p1 ./target/spark-example-0.0.2.jar file:///home/joe/Documents/Harvard63/HW6/hw6_problem1/hw6p1_data/hw6p1_data_a.txt file:///home/joe/Documents/Harvard63/HW6/hw6_problem1/hw6p1_data/hw6p1_data_b.txt file:///home/joe/Documents/Harvard63/HW6/hw6_problem1/out100316_k**

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.

```
public static void main(String[] args) throws Exception {
// read the file name from command line
String inputFile = args[0];
String outputFile= args[1];
 //Create a Java Spark Context sc
 SparkConf conf = new SparkConf().setAppName("hw6p2");
 JavaSparkContext sc = new JavaSparkContext(conf);
 // Create RDD emps
  JavaRDD<String> emps = sc.textFile(inputFile);
```

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.

The relevant part of the code is.

```
    JavaRDD<Tuple3<String, String, String>> emps_f = emps.map
        (
           new Function<String,  Tuple3<String, String, String>>()
           {
             public Tuple3<String, String,String> call(String x)
             {
              return new  Tuple3(x.split(",")[0], x.split(",")[1],x.split(",")[2] );
             }
           }
        );
```

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.

First add several lines to pom.xml to be able to ”digest” SQLContext, that is located in the package org.apache.spark.sql.SQLContext

```
<dependency>
   <groupId>org.apache.spark</groupId>
   <artifactId> spark-sql_2.10 </artifactId>
   <version>1.5.1</version>
</dependency>
```

Add the following line in the code:

```
import org.apache.spark.sql.Row;
```

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.

```
JavaRDD<Row> employees = emps_f.map
        (
           new Function< Tuple3<String, String, String> , Row>()
           {
             public Row   call(Tuple3<String, String,String> x)
             {
           return  RowFactory.create( x._1(),  Integer.parseInt(x._2()), Float.parseFloat( x._3()));
              }
           }
         );
```

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.

```
//import relevant packages
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.*;
import org.apache.spark.sql.DataFrame;

// create schema of dataset
```

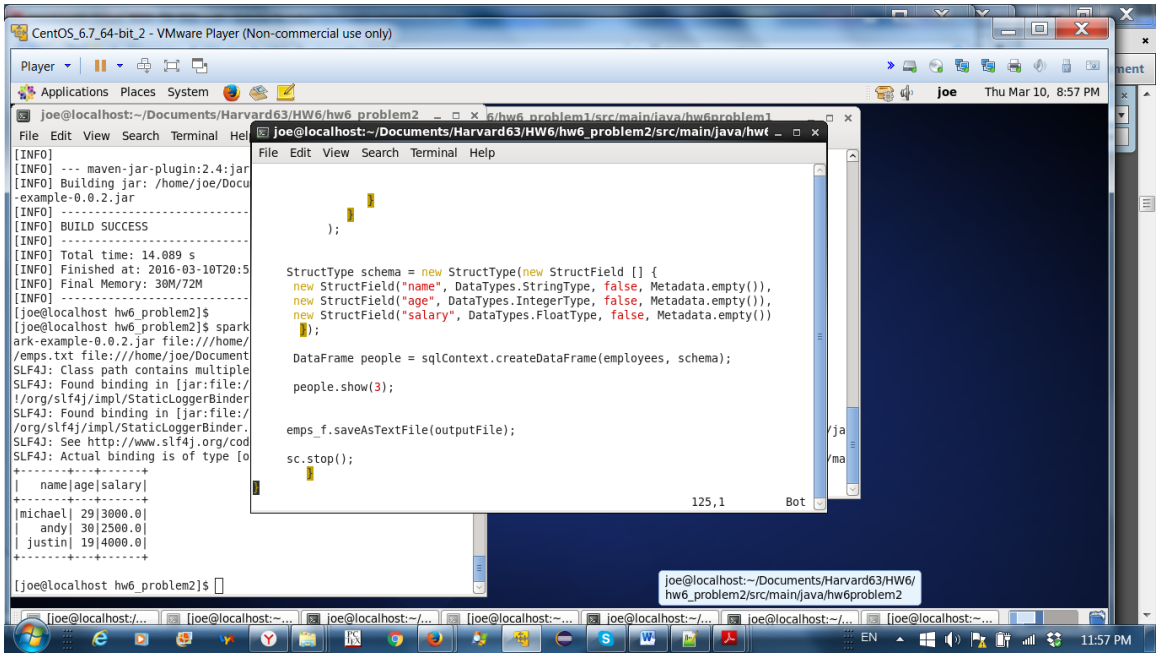
```
StructType schema = new StructType(new StructField [] {
        new StructField("name", DataTypes.StringType, false, Metadata.empty()),
        new StructField("age", DataTypes.IntegerType, false, Metadata.empty()),
        new StructField("salary", DataTypes.FloatType, false, Metadata.empty())
       });
// create dataframe
       SQLContext sqlContext = new SQLContext(sc);
      DataFrame people = sqlContext.createDataFrame(employees, schema);
```

Show the content of new DataFrame.

```
people.show(3);
```

Output:

```
+-------+---+------+
|   name|age|salary|
+-------+---+------+
|michael| 29|3000.0|
|   andy| 30|2500.0|
| justin| 19|4000.0|
+-------+---+------+
```

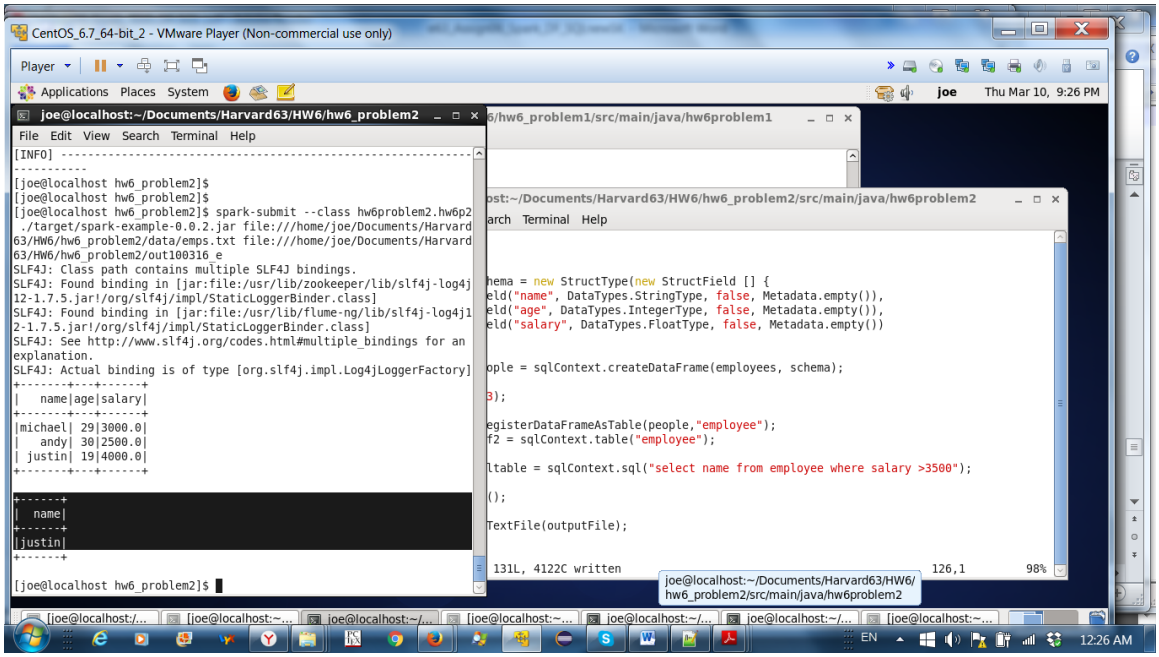


Transform this DataFrame into a Temporary Table and select names of all employees who have a salary greater than 3500.

```
sqlContext.registerDataFrameAsTable(people,"employee");
DataFrame  df2 = sqlContext.table("employee");
DataFrame sqltable = sqlContext.sql("select name from employee where salary >3500");
sqltable.show();
```

Output:

```
+------+
|  name|
+------+
|justin|
```



Running command

```
[joe@localhost hw6_problem2]$ spark-submit --class hw6problem2.hw6p2
./target/spark-example-0.0.2.jar
file:///home/joe/Documents/Harvard63/HW6/hw6_problem2/data/emps.txt
file:///home/joe/Documents/Harvard63/HW6/hw6_problem2/out100316_b
```

The code.

All detailed explanations are given in the discussion above

```
/**
 *  @author Serguey Khovansky
 *  @version: March 2016
 * This is solution of the homework 6, 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.
 *
 */

package hw6problem2;

import java.util.Arrays;
import java.util.List;
import java.lang.Iterable;

import scala.Tuple3;

import org.apache.commons.lang.StringUtils;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaPairRDD;
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 org.apache.spark.api.java.function.Function;

import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;

import org.apache.spark.sql.types.*;
import org.apache.spark.sql.DataFrame;

/**
 * The class hw6p1 make database operations with two text files using SPARK MapReduce technology
 */
public class hw6p2 {

 /**
  * @param args[]  input and output paths and location of the file input and output files
 */
  public static void main(String[] args) throws Exception {
    String inputFile = args[0];
    String outputFile= args[1];

 /**
    * Create a Java Spark Context sc
  */
    SparkConf conf = new SparkConf().setAppName("hw6p2");
    JavaSparkContext sc = new JavaSparkContext(conf);

// need for sql
```

```
   SQLContext sqlContext = new SQLContext(sc);

   /**
   *           Create RDD emps
   */
   JavaRDD<String> emps = sc.textFile(inputFile);


   /**
   *Turn into Tuple of words
   */

              JavaRDD<Tuple3<String, String, String>> emps_f = emps.map
        (
                    new Function<String,  Tuple3<String, String, String>>()
            {
                     public Tuple3<String, String,String> call(String x)
             {
              List L= Arrays.asList(x.toLowerCase().split( "\\W+") );
              return new  Tuple3(L.get(0), L.get(1),L.get(2) );
             }
            }
        );

       JavaRDD<Row> employees = emps_f.map
         (
            new Function< Tuple3<String, String, String> , Row>()
            {
             public Row   call(Tuple3<String, String,String> x)
             {
                  return  RowFactory.create( x._1(),  Integer.parseInt(x._2()), Float.parseFloat( x._3()));
              }
            }
          );


     StructType schema = new StructType(new StructField [] {
      new StructField("name", DataTypes.StringType, false, Metadata.empty()),
      new StructField("age", DataTypes.IntegerType, false, Metadata.empty()),
      new StructField("salary", DataTypes.FloatType, false, Metadata.empty())
       });

      DataFrame people = sqlContext.createDataFrame(employees, schema);

      people.show(3);

      sqlContext.registerDataFrameAsTable(people,"employee");
      DataFrame  df2 = sqlContext.table("employee");

      DataFrame sqltable = sqlContext.sql("select name from employee where salary >3500");

     sqltable.show();

     emps_f.saveAsTextFile(outputFile);

     sc.stop();
                }
}
```

Pom.xml

```
<project>
  <groupId>hw6problem2</groupId>
  <artifactId>spark-example</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>problem2</name>
  <packaging>jar</packaging>
```

```
  <version>0.0.2</version>
  <dependencies>
    <dependency> <!-- Spark dependency -->
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.10</artifactId>
      <version>1.1.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.10</artifactId>
      <version>1.5.1</version>
    </dependency>
  </dependencies>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <build>
    <pluginManagement>
      <plugins>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.1</version>
          <configuration>
            <source>${java.version}</source>
            <target>${java.version}</target>
          </configuration>
                </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

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.

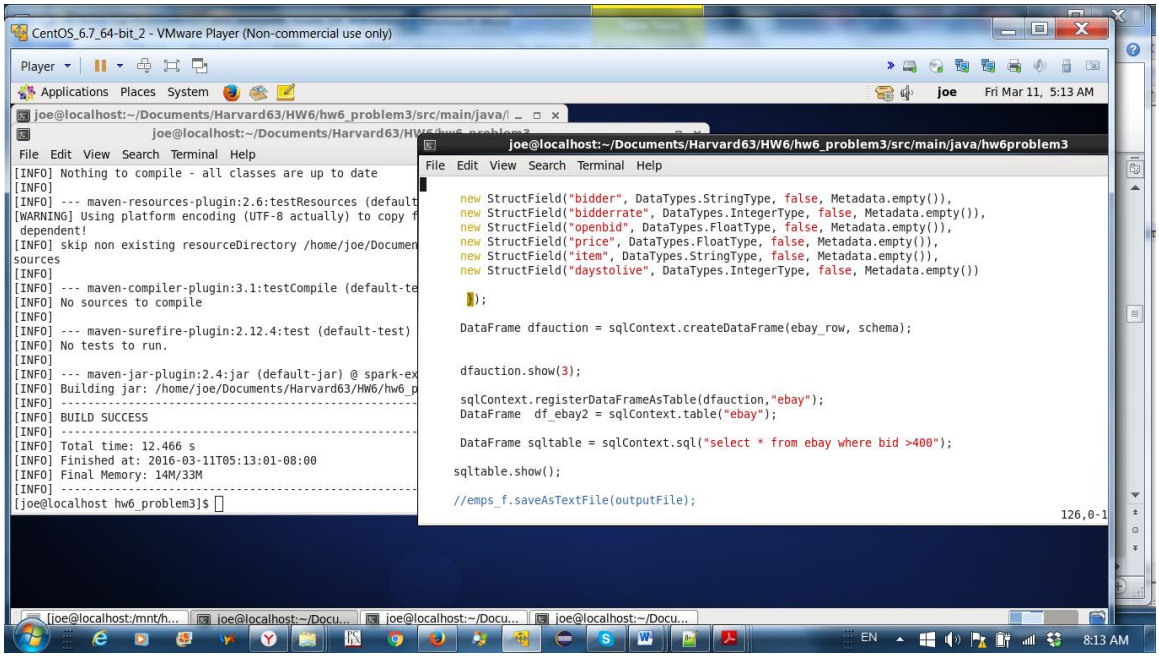
Using Spark DataFrames you will explore the data with following 4 questions:

1. How many auctions were held?
2. How many bids were made per item?
3.

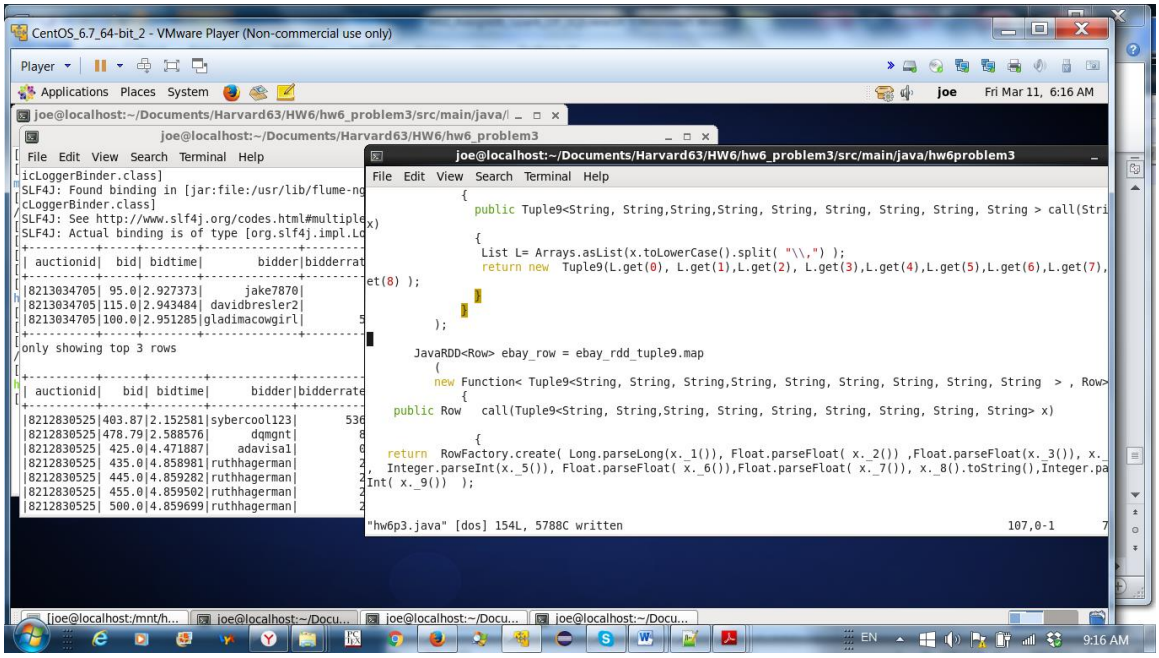
- What's the minimum, maximum, and average bid (price) per item?
- What is the minimum, maximum and average number of bids per item?

4. 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`.



Intermediate output:



Running command

```
[joe@localhost hw6_problem2]$ spark-submit --class hw6problem3.hw6p3
./target/spark-example-0.0.2.jar
file:///home/joe/Documents/Harvard63/HW6/hw6_problem3/data/ebay.csv
```

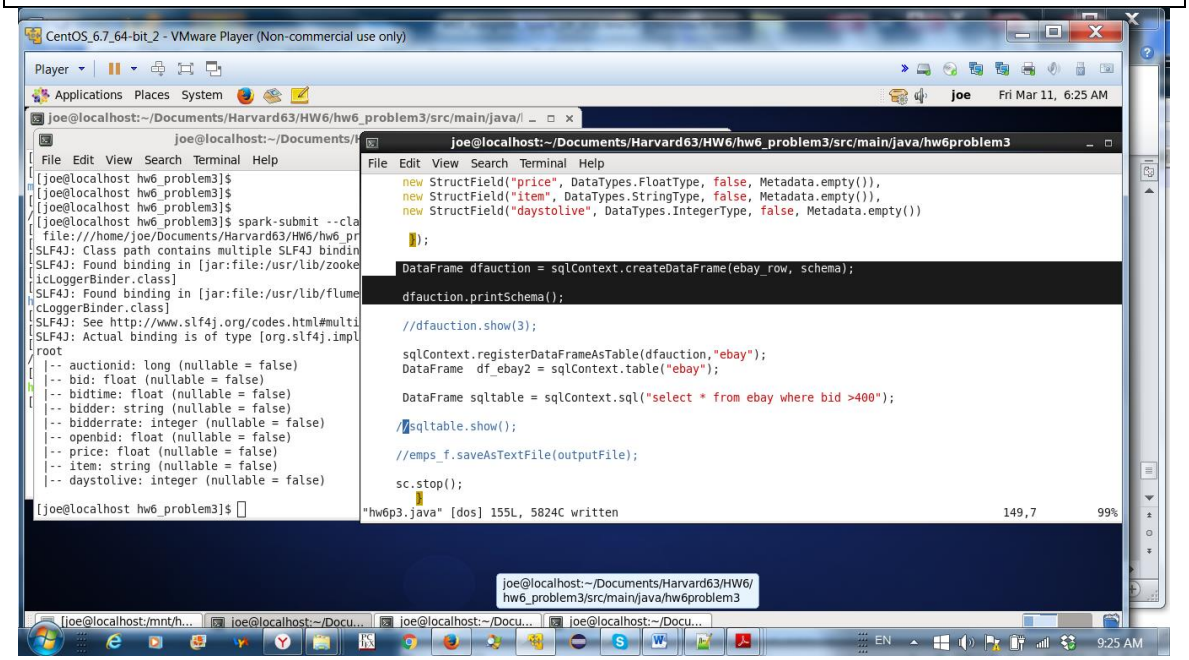
Show (print) the schema of the DataFrame.

Relevant line of code

```
DataFrame dfauction = sqlContext.createDataFrame(ebay_row, schema);
dfauction.printSchema();
```

```
[joe@localhost hw6_problem3]$ spark-submit --class hw6problem3.hw6p3
./target/spark-example-0.0.2.jar
file:///home/joe/Documents/Harvard63/HW6/hw6_problem3/data/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]
```

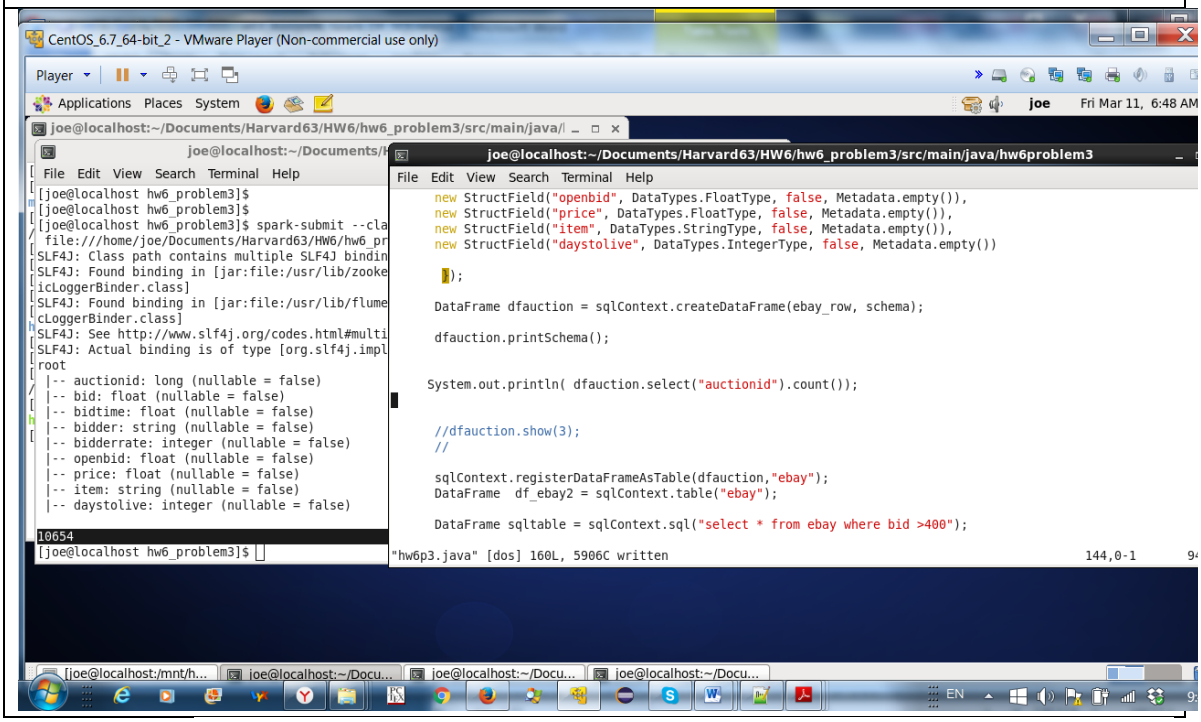
```
root
 |-- auctionid: long (nullable = false)
 |-- bid: float (nullable = false)
 |-- bidtime: float (nullable = false)
 |-- bidder: string (nullable = false)
 |-- bidderrate: integer (nullable = false)
 |-- openbid: float (nullable = false)
 |-- price: float (nullable = false)
 |-- item: string (nullable = false)
 |-- daystolive: integer (nullable = false)
```



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.

1. How many auctions were held?

```
System.out.println( dfauction.select("auctionid").count());
```



2. How many bids were made per item?

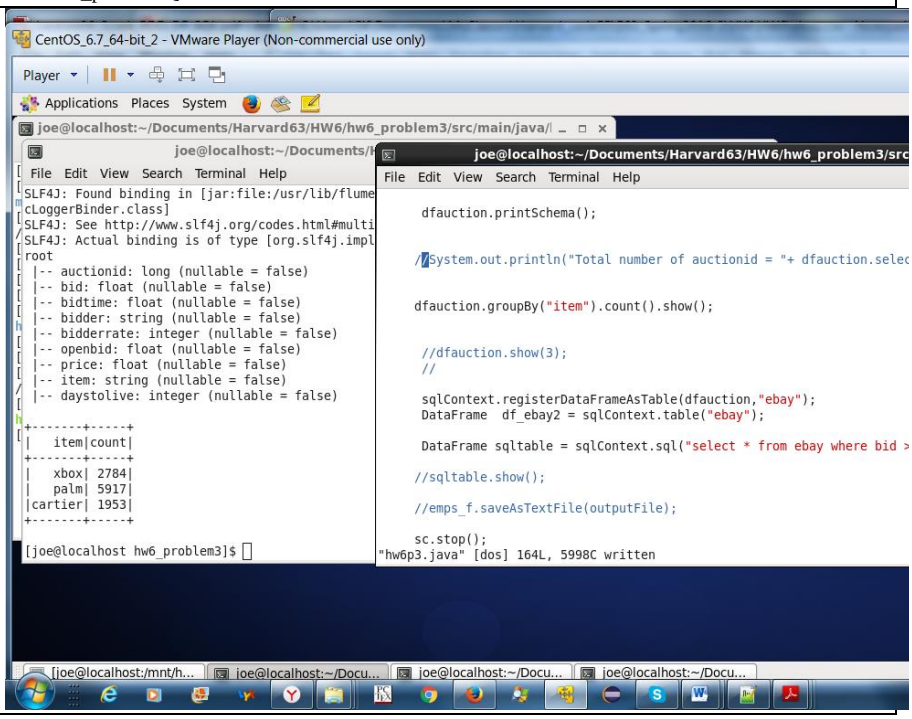
Relevant code line:

```
dfauction.groupBy("item").count().show();
```

Output:

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

+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+
```

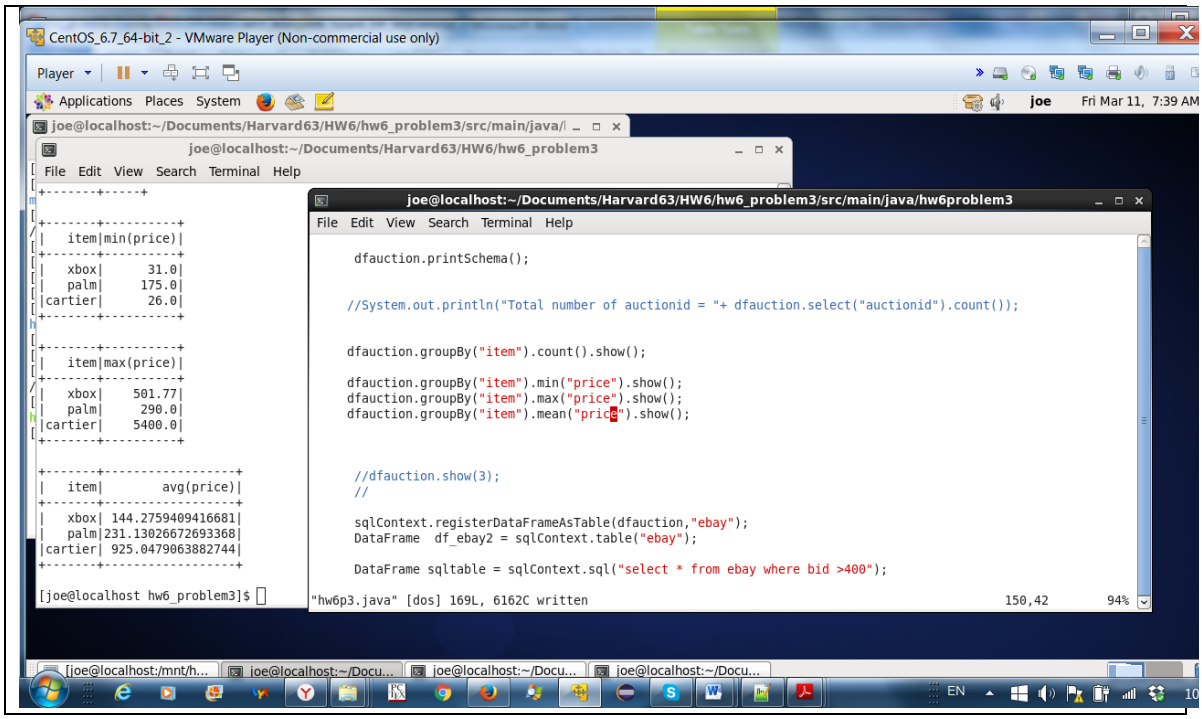
[joe@localhost hw6_problem3]$



- What's the minimum, maximum, and average bid (price) per item?

```
dfauction.groupBy("item").min("price").show();
dfauction.groupBy("item").max("price").show();
dfauction.groupBy("item").mean("price").show();
```

```
+-------+----------+
|   item|min(price)|
+-------+----------+
|   xbox|      31.0|
|   palm|     175.0|
|cartier|      26.0|
+-------+----------+

+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

+-------+------------------+
|   item|        avg(price)|
+-------+------------------+
|   xbox| 144.2759409416681|
|   palm|231.13026672693368|
|cartier| 925.0479063882744|
+-------+------------------+
```



What is the minimum, maximum and average **number of bids** per item?

Relevant code: the first line counts the number of bids per item. The second line applies operation describe() to that result. The first line is given only to understand better the outcome of the second line, which is the only relevant to the posed question.

```
dfauction.groupBy("item").count().show();
dfauction.groupBy("item").count().describe().show();
```

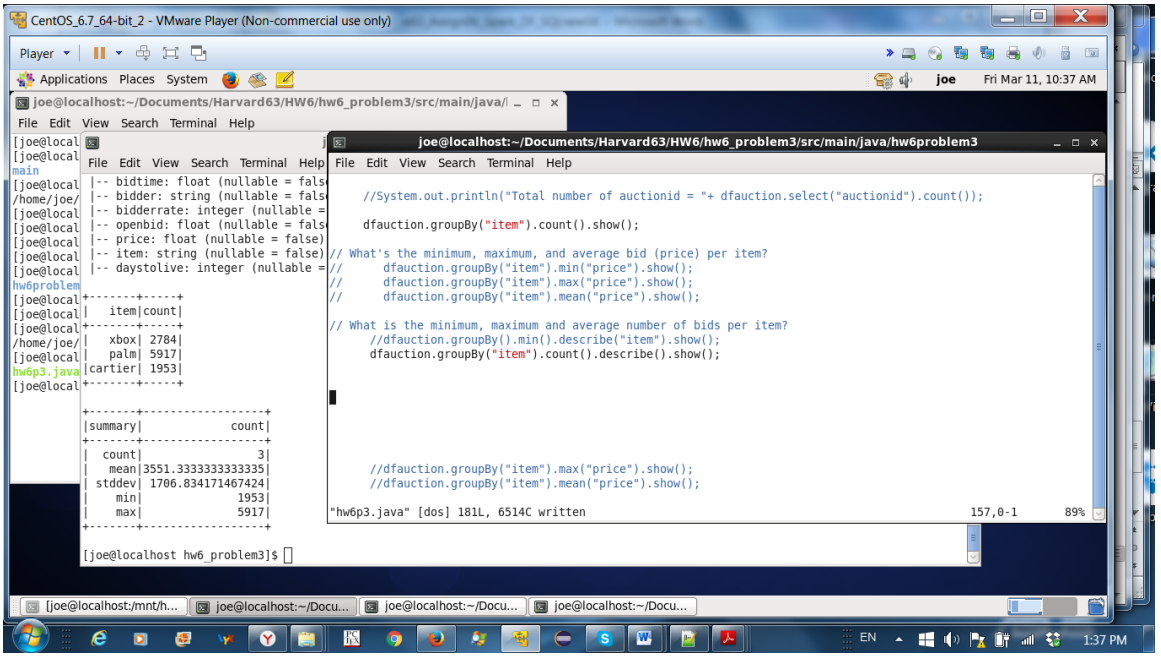
Output:

dfauction.groupBy("item").count().show();

```
+-------+-----+
|   item|count|  the"count" counts the number of bids per item
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
----+
```

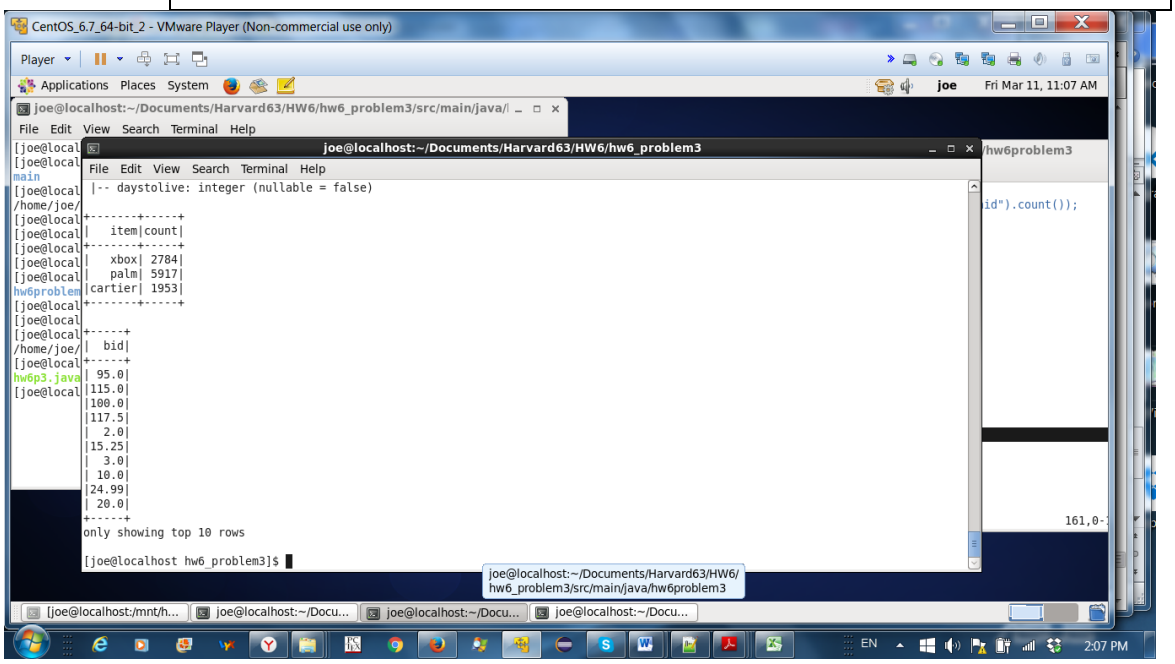
dfauction.groupBy("item").count().describe().show();

```
+-------+------------------+
|summary|     count|
+-------+------------------+
| count|                3 |
|  mean|3551.3333333333335 |
| stddev| 1706.834171467424 |
|   min|             1953 |
|   max|             5917 |
  ----------------+
```



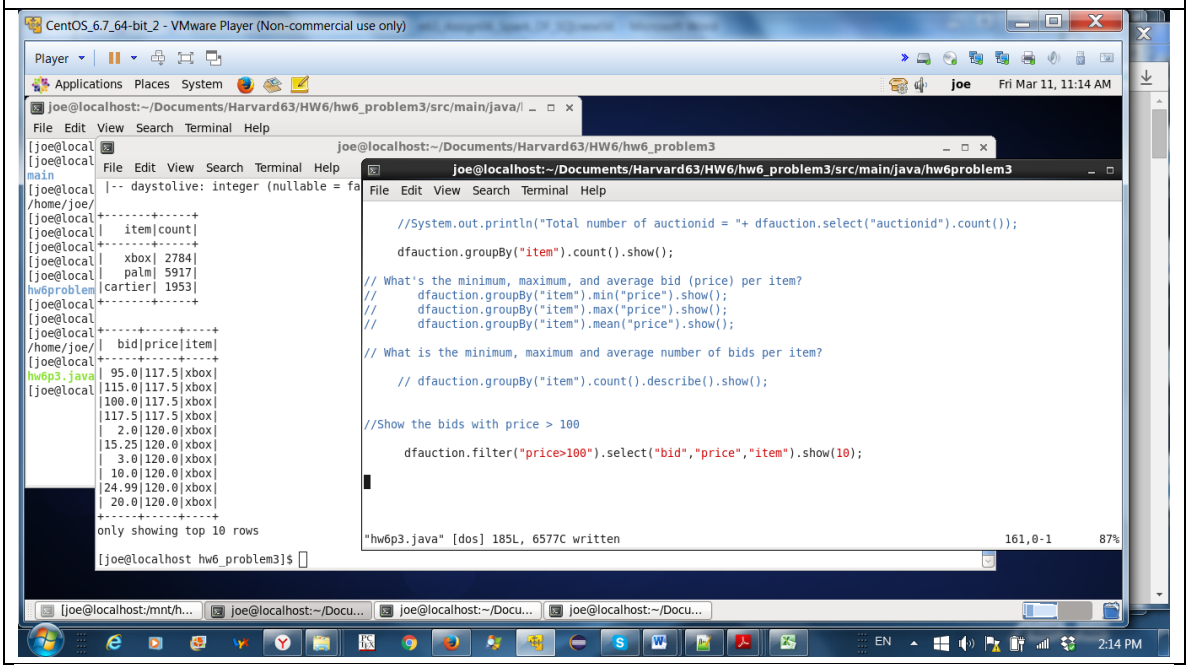
Show the bids with price > 100

```
dfauction.filter("price>100").select("bid").show(10);
```



Extension

```
dfauction.filter("price>100").select("bid","price","item").show(10);
```

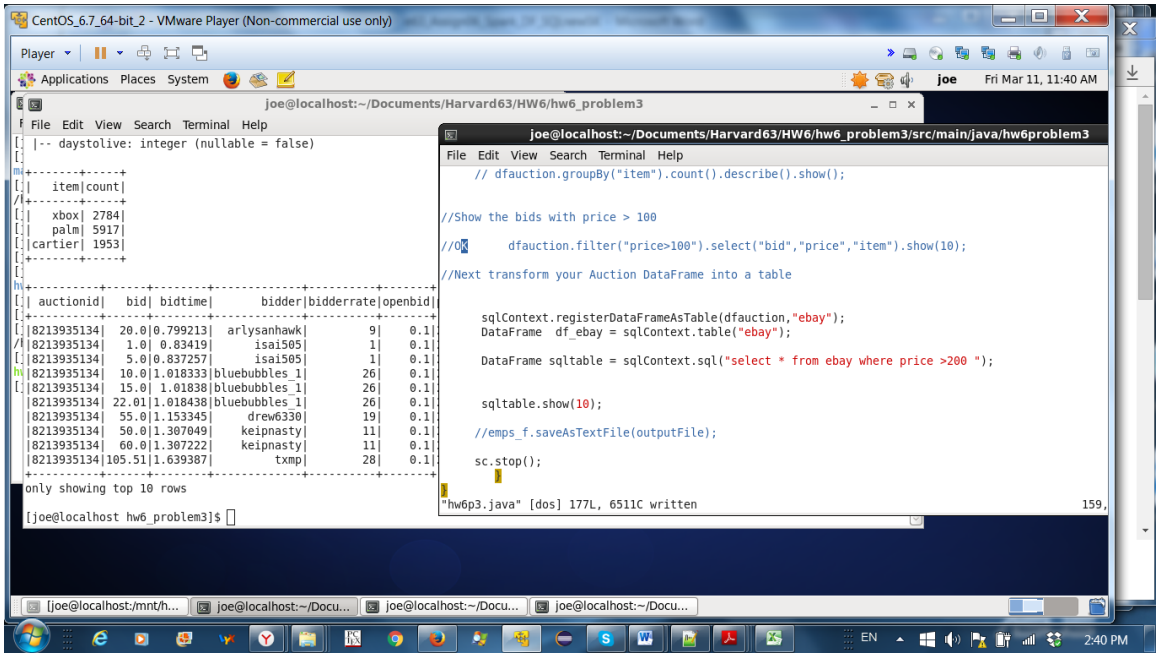


<u>Next transform your Auction DataFrame into a table</u>

Transform into a table and test it with a select query. Relevant to this quesiotns lines are boled.

```
sqlContext.registerDataFrameAsTable(dfauction,"ebay");
DataFrame  df_ebay = sqlContext.table("ebay");
DataFrame sqltable = sqlContext.sql("select * from ebay where price >200 ");
sqltable.show(10);
```

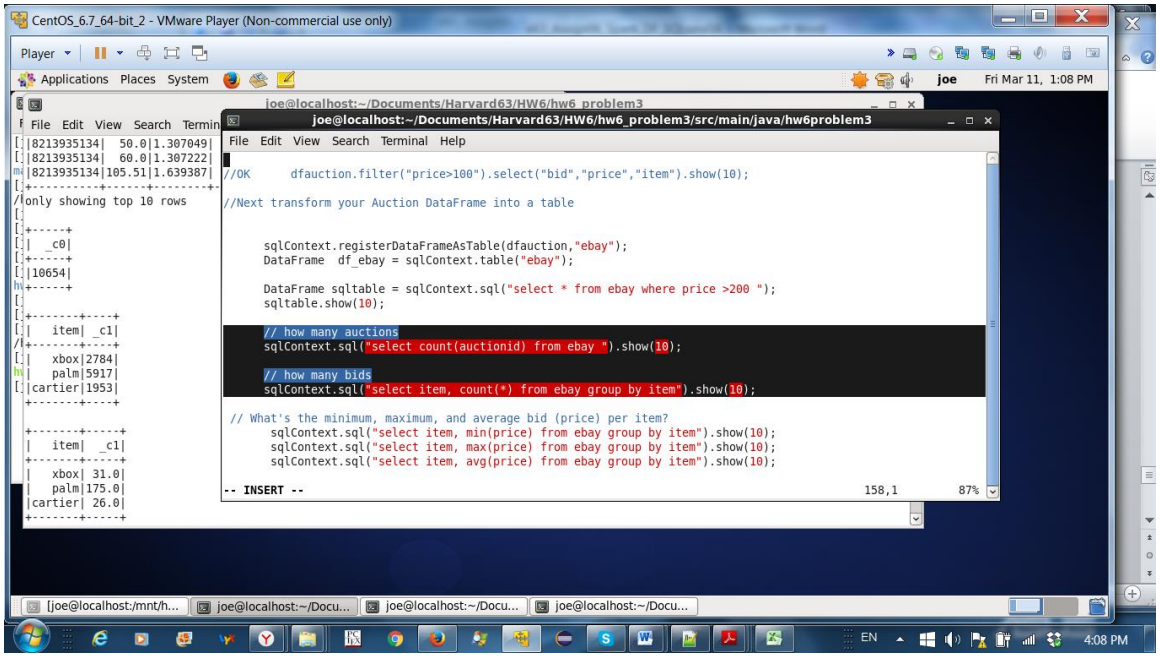
The output:



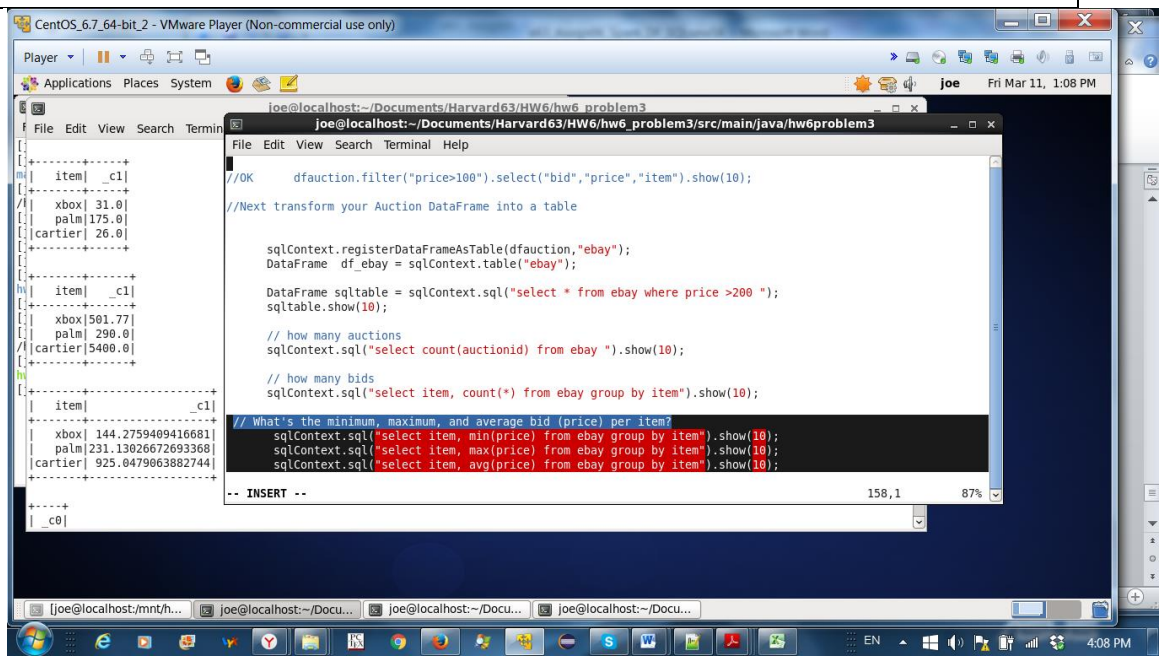
Make the same 4 inquiries using regular SQL queries.

```
// how many auctions
 sqlContext.sql("select count(auctionid) from ebay ").show(10);
```

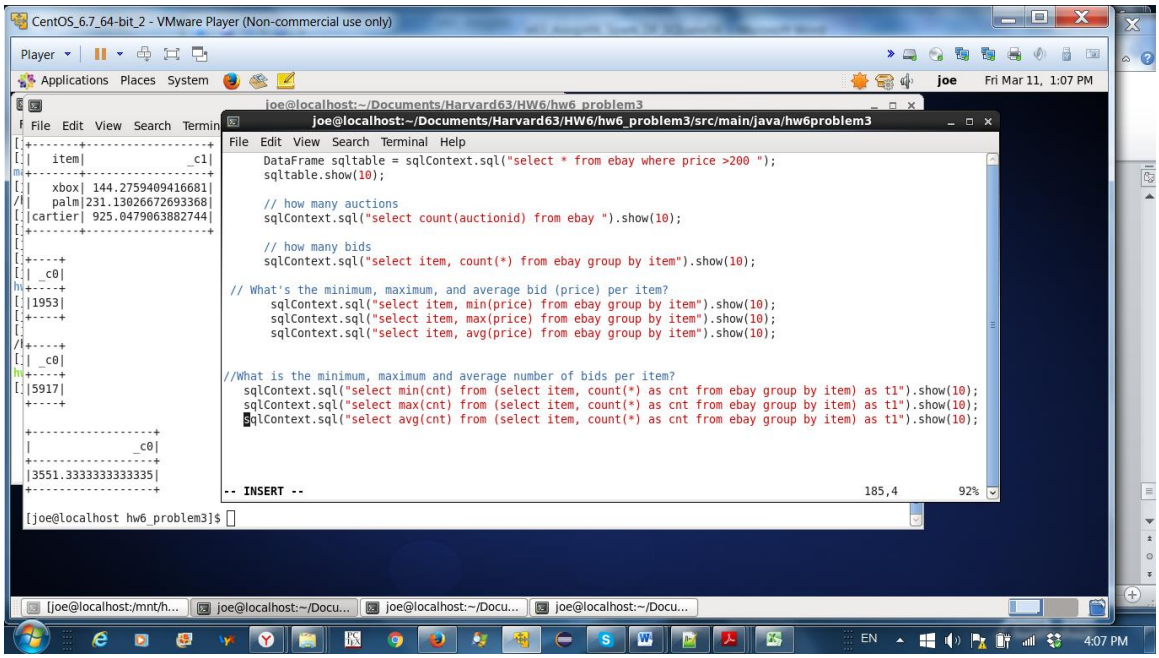
```
// how many bids
sqlContext.sql("select item, count(*) from ebay group by item").show(10);
```



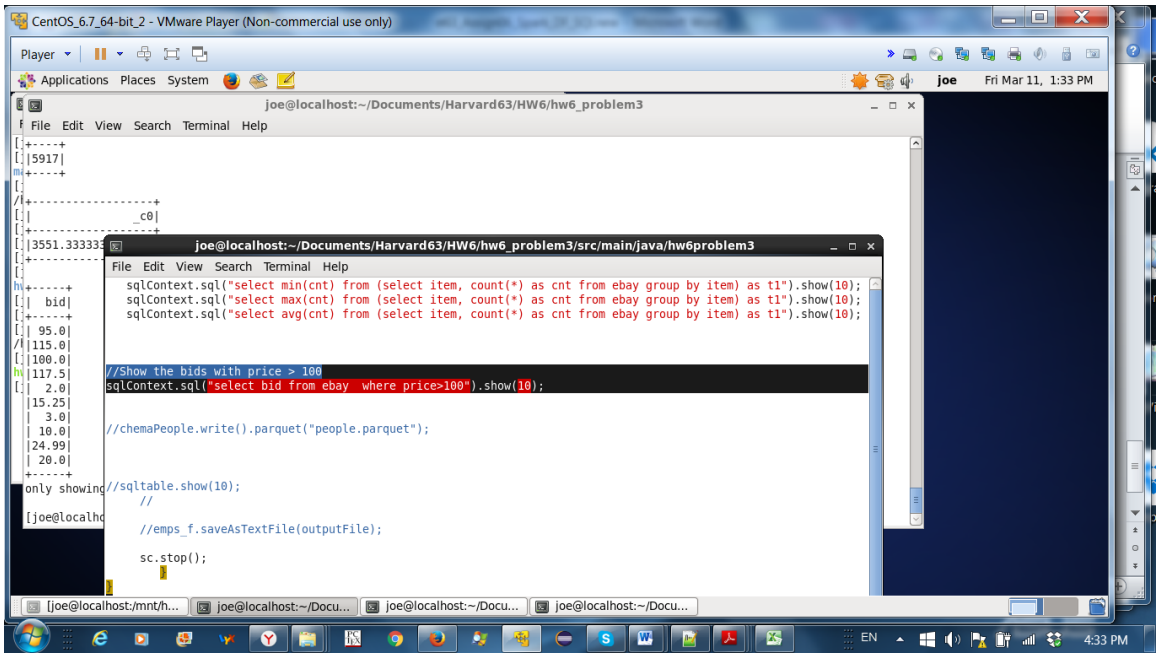

```
// What's the minimum, maximum, and average bid (price) per item?
  sqlContext.sql("select min(price) from ebay group by item").show(10);
  sqlContext.sql("select max(price) from ebay group by item").show(10);
  sqlContext.sql("select mean(price) from ebay group by item").show(10);
```



```
//What is the minimum, maximum and average number of bids per item?
sqlContext.sql("select min(cnt) from (select item, count(*) as cnt from ebay group by item) as t1").show(10);
sqlContext.sql("select max(cnt) from (select item, count(*) as cnt from ebay group by item) as t1").show(10);
sqlContext.sql("select avg(cnt) from (select item, count(*) as cnt from ebay group by item) as t1").show(10);
```

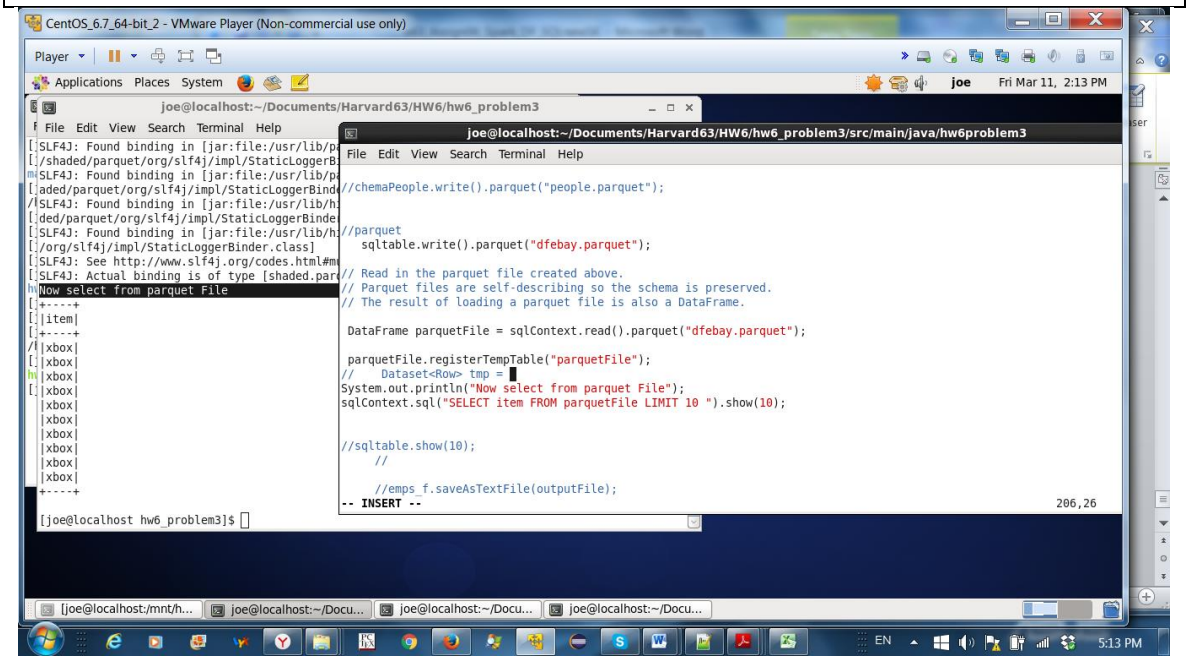



```
//Show the bids with price > 100
sqlContext.sql("select bid from ebay  where price>100").show(10);
```



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 against.

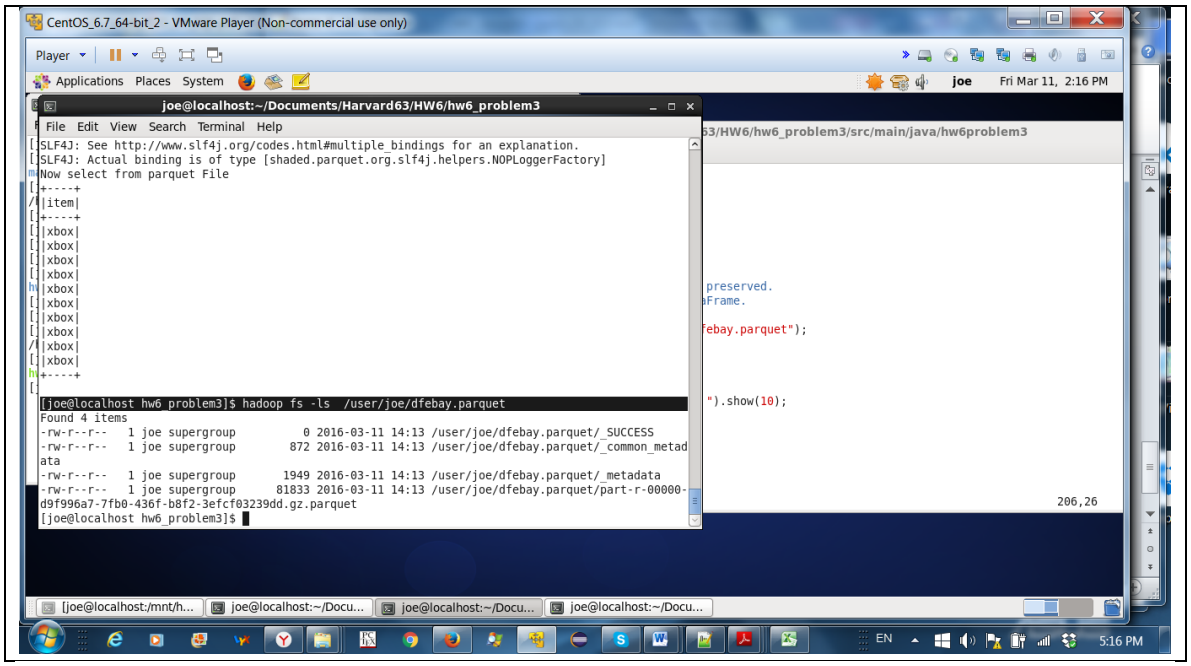
```
//parquet
   sqltable.write().parquet("dfebay.parquet");

// Read in the parquet file created above.
 DataFrame parquetFile = sqlContext.read().parquet("dfebay.parquet");

 parquetFile.registerTempTable("parquetFile");
System.out.println("Now select from parquet File");
sqlContext.sql("SELECT item FROM parquetFile LIMIT 10 ").show(10);
```



The parquet file is located in the hadoop hdfs so it certainly exists beyond the life-span of the screen I work in.

hadoop fs -ls /user/joe/dfebay.parquet

```
Found 4 items
-rw-r--r--   1 joe supergroup          0 2016-03-11 14:13 /user/joe/dfebay.parquet/_SUCCESS
-rw-r--r--   1 joe supergroup        872 2016-03-11 14:13 /user/joe/dfebay.parquet/_common_metadata
-rw-r--r--   1 joe supergroup       1949 2016-03-11 14:13 /user/joe/dfebay.parquet/_metadata
-rw-r--r--   1 joe supergroup      81833 2016-03-11 14:13 /user/joe/dfebay.parquet/part-r-00000-d9f996a7-7fb0-436f-
b8f2-3efcf03239dd.gz.parquet
```

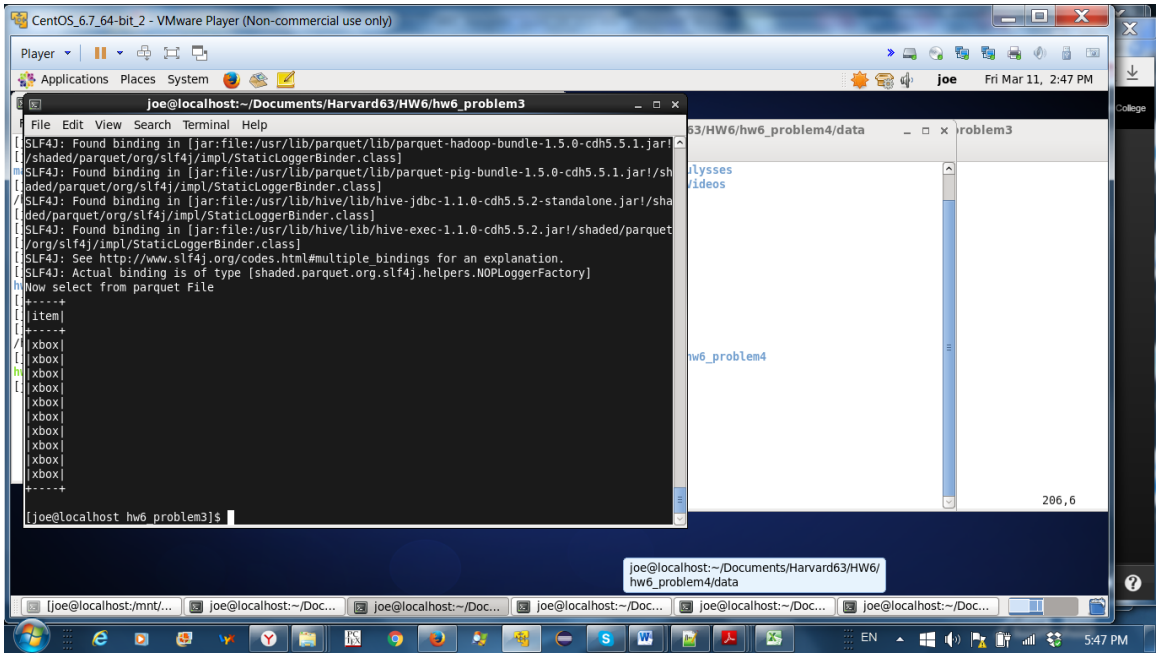


Entire output:

```
joe@localhost hw6_problem3]$
[joe@localhost hw6_problem3]$ spark-submit --class hw6problem3.hw6p3 ./target/spark-example-0.0.2.jar
file:///home/joe/Documents/Harvard63/HW6/hw6_problem3/data/ebay.csv
^C[joe@localhost hw6_problem3]$
[joe@localhost hw6_problem3]$ hadoop fs -rm -r  /user/joe/dfebay.parquet16/03/11 14:40:28 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/joe/dfebay.parquet
[joe@localhost hw6_problem3]$ spark-submit --class hw6problem3.hw6p3 ./target/spark-example-0.0.2.jar
file:///home/joe/Documents/Harvard63/HW6/hw6_problem3/data/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]
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)

Total number of auctionid = 10654
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+

+-------+----------+
|   item|min(price)|
+-------+----------+
|   xbox|      31.0|
|   palm|     175.0|
|cartier|      26.0|
+-------+----------+

+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
```

```
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

+-------+------------------+
|   item|        avg(price)|
+-------+------------------+
|   xbox| 144.2759409416681|
|   palm|231.13026672693368|
|cartier| 925.0479063882744|
+-------+------------------+

+-------+------------------+
|summary|             count|
+-------+------------------+
|  count|                 3|
|   mean|3551.3333333333335|
| stddev| 1706.834171467424|
|    min|              1953|
|    max|              5917|
+-------+------------------+

+-----+-----+----+
|  bid|price|item|
+-----+-----+----+
| 95.0|117.5|xbox|
|115.0|117.5|xbox|
|100.0|117.5|xbox|
|117.5|117.5|xbox|
|  2.0|120.0|xbox|
|15.25|120.0|xbox|
|  3.0|120.0|xbox|
| 10.0|120.0|xbox|
|24.99|120.0|xbox|
| 20.0|120.0|xbox|
+-----+-----+----+
only showing top 10 rows

+----------+------+--------+-------------+----------+-------+-----+----+----------+
| auctionid|   bid| bidtime|       bidder|bidderrate|openbid|price|item|daystolive|
+----------+------+--------+-------------+----------+-------+-----+----+----------+
|8213935134|  20.0|0.799213|  arlysanhawk|         9|    0.1|207.5|xbox|         3|
|8213935134|   1.0| 0.83419|      isai505|         1|    0.1|207.5|xbox|         3|
|8213935134|   5.0|0.837257|      isai505|         1|    0.1|207.5|xbox|         3|
|8213935134|  10.0|1.018333|bluebubbles_1|        26|    0.1|207.5|xbox|         3|
|8213935134|  15.0| 1.01838|bluebubbles_1|        26|    0.1|207.5|xbox|         3|
|8213935134| 22.01|1.018438|bluebubbles_1|        26|    0.1|207.5|xbox|         3|
|8213935134|  55.0|1.153345|     drew6330|        19|    0.1|207.5|xbox|         3|
|8213935134|  50.0|1.307049|    keipnasty|        11|    0.1|207.5|xbox|         3|
|8213935134|  60.0|1.307222|    keipnasty|        11|    0.1|207.5|xbox|         3|
|8213935134|105.51|1.639387|         txmp|        28|    0.1|207.5|xbox|         3|
+----------+------+--------+-------------+----------+-------+-----+----+----------+
only showing top 10 rows

+-----+
|  _c0|
+-----+
|10654|
+-----+

+-------+----+
|   item| _c1|
+-------+----+
|   xbox|2784|
|   palm|5917|
|cartier|1953|
+-------+----+

+-------+-----+
|   item|  _c1|
+-------+-----+
|   xbox| 31.0|
|   palm|175.0|
|cartier| 26.0|
+-------+-----+

+-------+------+
|   item|   _c1|
+-------+------+
|   xbox|501.77|
|   palm| 290.0|
```

```
|cartier|5400.0|
+-------+------+

+-------+------------------+
|   item|               _c1|
+-------+------------------+
|   xbox| 144.2759409416681|
|   palm|231.13026672693368|
|cartier| 925.0479063882744|
+-------+------------------+

+----+
| _c0|
+----+
|1953|
+----+

+----+
| _c0|
+----+
|5917|
+----+

+------------------+
|               _c0|
+------------------+
|3551.3333333333335|
+------------------+

+-----+
|  bid|
+-----+
| 95.0|
|115.0|
|100.0|
|117.5|
|  2.0|
|15.25|
|  3.0|
| 10.0|
|24.99|
| 20.0|
+-----+
only showing top 10 rows

SLF4J: Class path contains multiple SLF4J bindings.
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-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-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]
Now select from parquet File
+----+
|item|
+----+
|xbox|
|xbox|
|xbox|
|xbox|
|xbox|
|xbox|
|xbox|
|xbox|
|xbox|
|xbox|
+----+

[joe@localhost hw6_problem3]$
```



The code:

```
/**
 *  @author Serguey Khovansky
 *  @version: March 2016
 * This is solution of the homework 6, 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?
 * 3.
 * o	What's the minimum, maximum, and average bid (price) per item?
 * o	What is the minimum, maximum and average number of bids per item?
 * 4.	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.
 *
 */

package hw6problem3;

import java.util.Arrays;
import java.util.List;
import java.lang.Iterable;

import scala.Tuple9;

import org.apache.commons.lang.StringUtils;

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

import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;

import org.apache.spark.sql.types.*;
import org.apache.spark.sql.DataFrame;

/**
 * The class hw6p1 make database operations with two text files using SPARK MapReduce technology
 */
public class hw6p3 {

 /**
  * @param args[]  input and output paths and location of the file input and output files
 */
  public static void main(String[] args) throws Exception {
    String inputFile = args[0];
//    String outputFile= args[1];

 /**
    * Create a Java Spark Context sc
  */
```

```
   SparkConf conf = new SparkConf().setAppName("hw6p3");
   JavaSparkContext sc = new JavaSparkContext(conf);
   SQLContext sqlContext = new SQLContext(sc);

/**
   *       Create RDD emps
   */
   JavaRDD<String> ebay = sc.textFile(inputFile);

   /**
   *Turn into Tuple of words
   */
          JavaRDD<Tuple9<String, String, String,  String, String, String, String, String, String>> ebay_rdd_tuple9 =
ebay.map
        (
        new Function<String,  Tuple9<String, String, String,String, String, String, String, String, String>>()
              {
                public Tuple9<String, String,String,String, String, String, String, String, String > call(String x)
           {
            List L= Arrays.asList(x.toLowerCase().split( "\\,") );
            return new  Tuple9(L.get(0), L.get(1),L.get(2), L.get(3),L.get(4),L.get(5),L.get(6),L.get(7),L.get(8) );
           }
          }
        );

      JavaRDD<Row> ebay_row = ebay_rdd_tuple9.map
        (
        new Function< Tuple9<String, String, String,String, String, String, String, String, String  > , Row>()
          {
   public Row   call(Tuple9<String, String,String, String, String, String, String, String, String> x)

            {
  return  RowFactory.create( Long.parseLong(x._1()), Float.parseFloat( x._2()) ,Float.parseFloat(x._3()), x._4(),
Integer.parseInt(x._5()), Float.parseFloat( x._6()),Float.parseFloat( x._7()), x._8().toString(),Integer.parseInt( x._9())  );


             }
          }
        );




    StructType schema = new StructType(new StructField [] {
     new StructField("auctionid", DataTypes.LongType, false, Metadata.empty()),
     new StructField("bid", DataTypes.FloatType, false, Metadata.empty()),
```

```
      new StructField("bidtime", DataTypes.FloatType, false, Metadata.empty()),

      new StructField("bidder", DataTypes.StringType, false, Metadata.empty()),
      new StructField("bidderrate", DataTypes.IntegerType, false, Metadata.empty()),
      new StructField("openbid", DataTypes.FloatType, false, Metadata.empty()),
      new StructField("price", DataTypes.FloatType, false, Metadata.empty()),
      new StructField("item", DataTypes.StringType, false, Metadata.empty()),
      new StructField("daystolive", DataTypes.IntegerType, false, Metadata.empty())

       });

      DataFrame dfauction = sqlContext.createDataFrame(ebay_row, schema);

      dfauction.printSchema();


     System.out.println("Total number of auctionid = "+ dfauction.select("auctionid").count());

    dfauction.groupBy("item").count().show();

// What's the minimum, maximum, and average bid (price) per item?
      dfauction.groupBy("item").min("price").show();
      dfauction.groupBy("item").max("price").show();
      dfauction.groupBy("item").mean("price").show();

// What is the minimum, maximum and average number of bids per item?

 dfauction.groupBy("item").count().describe().show();


//Show the bids with price > 100

     dfauction.filter("price>100").select("bid","price","item").show(10);

//Next transform your Auction DataFrame into a table


      sqlContext.registerDataFrameAsTable(dfauction,"ebay");
      DataFrame  df_ebay = sqlContext.table("ebay");

      DataFrame sqltable = sqlContext.sql("select * from ebay where price >200 ");
      sqltable.show(10);

      // how many auctions
      sqlContext.sql("select count(auctionid) from ebay ").show(10);

      // how many bids
      sqlContext.sql("select item, count(*) from ebay group by item").show(10);
```

```
 // What's the minimum, maximum, and average bid (price) per item?
       sqlContext.sql("select item, min(price) from ebay group by item").show(10);
       sqlContext.sql("select item, max(price) from ebay group by item").show(10);
       sqlContext.sql("select item, avg(price) from ebay group by item").show(10);

//What is the minimum, maximum and average number of bids per item?
   sqlContext.sql("select min(cnt) from (select item, count(*) as cnt from ebay group by item) as t1").show(10);
   sqlContext.sql("select max(cnt) from (select item, count(*) as cnt from ebay group by item) as t1").show(10);
   sqlContext.sql("select avg(cnt) from (select item, count(*) as cnt from ebay group by item) as t1").show(10);

//Show the bids with price > 100
    sqlContext.sql("select bid from ebay  where price>100").show(10);

//parquet
   sqltable.write().parquet("dfebay.parquet");

// Read in the parquet file created above.
// Parquet files are self-describing so the schema is preserved.
// The result of loading a parquet file is also a DataFrame.

 DataFrame parquetFile = sqlContext.read().parquet("dfebay.parquet");

 parquetFile.registerTempTable("parquetFile");
//    Dataset<Row> tmp =
System.out.println("Now select from parquet File");
sqlContext.sql("SELECT item FROM parquetFile LIMIT 10 ").show(10);

     sc.stop();
            }
}
```

POM.xml

```
<project>
  <groupId>hw6problem3</groupId>
  <artifactId>spark-example</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>problem3</name>
  <packaging>jar</packaging>
  <version>0.0.2</version>
  <dependencies>
    <dependency> <!-- Spark dependency -->
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.10</artifactId>
      <version>1.1.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
```

```
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.10</artifactId>
    <version>1.5.1</version>
  </dependency>
        <dependency>
         <groupId>com.twitter</groupId>
         <artifactId>parquet-avro</artifactId>
         <version>1.0.0</version>
       </dependency>
  </dependencies>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <build>
    <pluginManagement>
      <plugins>
        <plugin>
              <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.1</version>
          <configuration>
            <source>${java.version}</source>
            <target>${java.version}</target>
          </configuration>
            </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

Problem 4.

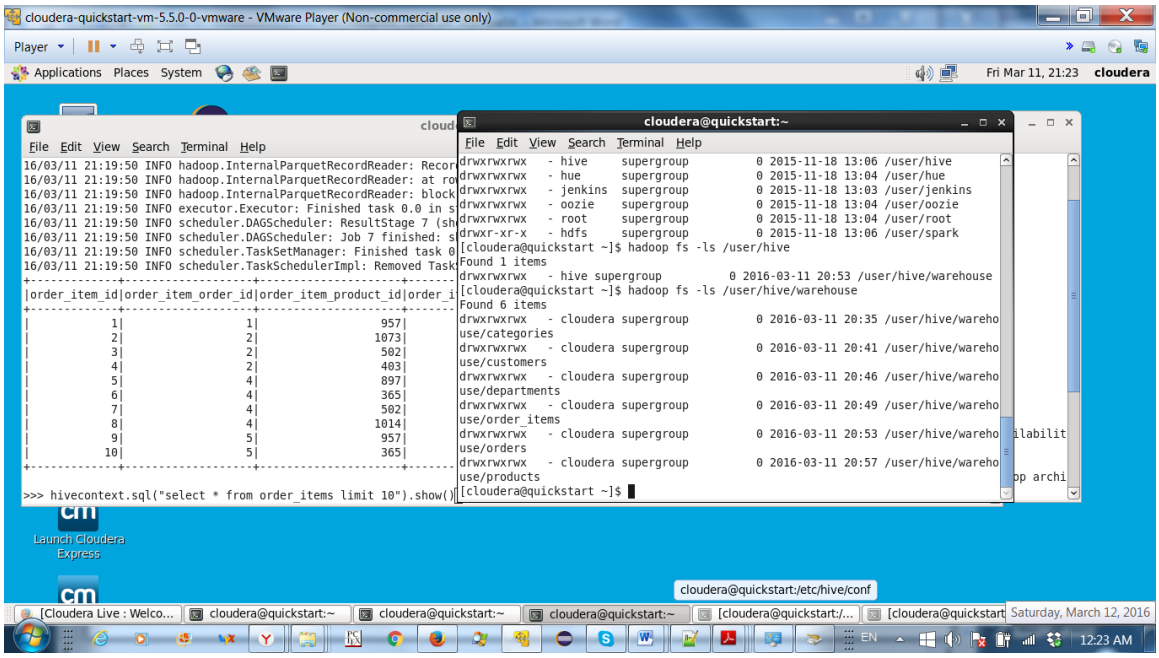
Use Sqoop to import all tables in MySQL demo database `retail_db` into Hive.
Use Spark DataFrame API or Spark Temporary Tables to find orders with the largest number of order items per order.

To make retail_db import into Hive

```
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   --hive-overwrite
```

P.S. Could not make Sqoop work as required for more than 7 hours….It began working after re-installing VM.

Required files of the database are in the Hadoop along with the results of the test query




```
// Note, that sc is produced by pyspark by defaul
from pyspark import SparkConf, SparkContext
hivecontext = HiveContext(sc)
```

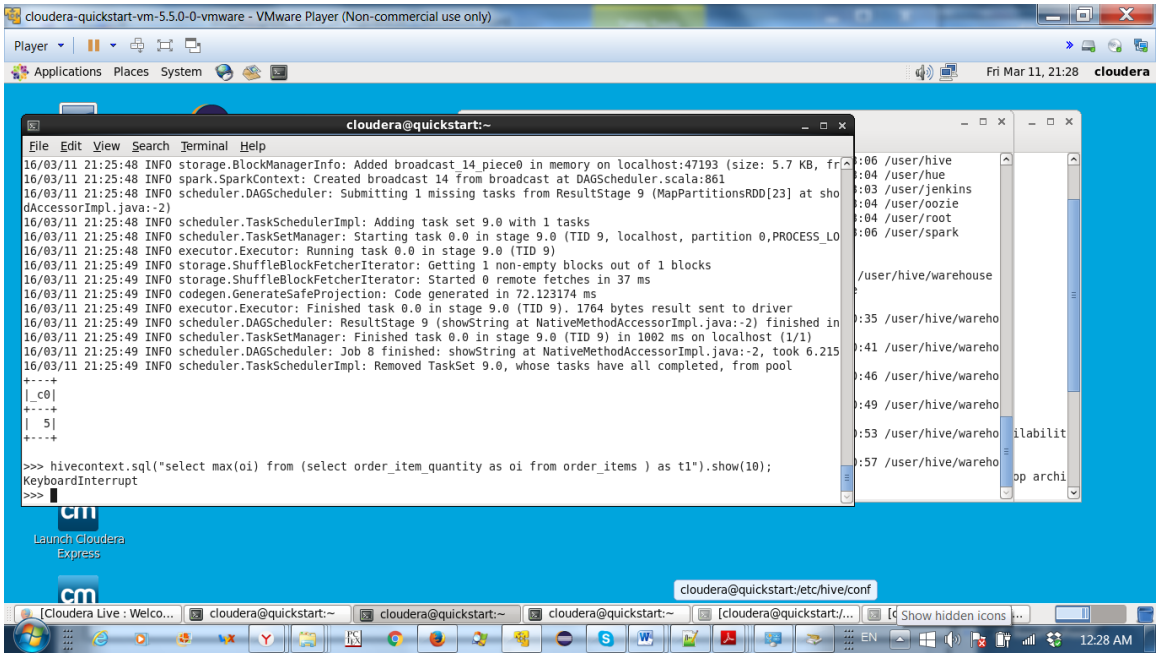
```
//test query
hivecontext.sql("select * from order_items limit 10").show()
```

<u>Find orders with the largest number of order items per order</u>

```
hivecontext.sql("select max(oi) from (select order_item_quantity as oi from order_items ) as t1").show(10);
```

Output:

```
+---+
|_c0|
+---+
|  5|
-----
```


cloudera-quickstart-vm-5.5.0-0-vmware - VMware Player (Non-commercial use only)
Player
Applications Places System
Fri Mar 11, 21:28 cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help
16/03/11 21:25:48 INFO storage.BlockManagerInfo: Added broadcast_14_piece0 in memory on localhost:47193 (size: 5.7 KB, fr
16/03/11 21:25:48 INFO spark.SparkContext: Created broadcast 14 from broadcast at DAGScheduler.scala:861
16/03/11 21:25:48 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 9 (MapPartitionsRDD[23] at sho
dAccessorImpl.java:-2)
16/03/11 21:25:48 INFO scheduler.TaskSchedulerImpl: Adding task set 9.0 with 1 tasks
16/03/11 21:25:48 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 9.0 (TID 9, localhost, partition 0,PROCESS_LO
16/03/11 21:25:48 INFO executor.Executor: Running task 0.0 in stage 9.0 (TID 9)
16/03/11 21:25:49 INFO storage.ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
16/03/11 21:25:49 INFO storage.ShuffleBlockFetcherIterator: Started 0 remote fetches in 37 ms
16/03/11 21:25:49 INFO codegen.GenerateSafeProjection: Code generated in 72.123174 ms
16/03/11 21:25:49 INFO executor.Executor: Finished task 0.0 in stage 9.0 (TID 9). 1764 bytes result sent to driver
16/03/11 21:25:49 INFO scheduler.DAGScheduler: ResultStage 9 (showString at NativeMethodAccessorImpl.java:-2) finished in
16/03/11 21:25:49 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 9.0 (TID 9) in 1002 ms on localhost (1/1)
16/03/11 21:25:49 INFO scheduler.DAGScheduler: Job 8 finished: showString at NativeMethodAccessorImpl.java:-2, took 6.215
16/03/11 21:25:49 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 9.0, whose tasks have all completed, from pool
+---+
|_c0|
+---+
| 5|
+---+
>>> hivecontext.sql("select max(oi) from (select order_item_quantity as oi from order_items) as t1").show(10);
KeyboardInterrupt
>>>
:06 /user/hive
:04 /user/hue
:03 /user/jenkins
:04 /user/oozie
:04 /user/root
:06 /user/spark
/user/hive/warehouse
:35 /user/hive/wareho
:41 /user/hive/wareho
:46 /user/hive/wareho
:49 /user/hive/wareho
:53 /user/hive/wareho
:57 /user/hive/wareho
Launch Cloudera Express
cloudera@quickstart:/etc/hive/conf
[Cloudera Live : Welco...
cloudera@quickstart:~
cloudera@quickstart:~
cloudera@quickstart:~
[cloudera@quickstart:/...
Show hidden icons
12:28 AM