Hanjiao Qiu
CSCIE63 Big Data Analytics
Assignment 06

Assignment 06 Solution

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.

Solution:

1. Read in two paragraphs from local file system. Import them into two Spark RDDs, "paragraphA" and "paragraphB". Use RDD method "flatMap()" and regex to transform them into RDDs contains only words.

```
>>> import re

>>> paragraphA = sc.textFile("file:///mnt/hgfs/VM_shared/hw06/AlphaGoA.txt")
>>> wordsA = paragraphA.flatMap(lambda line: re.sub(r"[^a-zA-Z\d]", " ",
str(line)).lower().split())
>>> wordsA.count()
159
>>> wordsA.take(20)
['a', 'huge', 'milestone', 'has', 'just', 'been', 'reached', 'in', 'the',
'field', 'of', 'artificial', 'intelligence', 'alphago', 'a', 'program',
'developed', 'by', 'google', 's']

>>> paragraphB = sc.textFile("file:///mnt/hgfs/VM_shared/hw06/AlphaGoB.txt")
>>> wordsB = paragraphB.flatMap(lambda line: re.sub(r"[^a-zA-Z\d]", " ",
str(line)).lower().split())
>>> wordsB.count()
166
>>> wordsB.take(20)
```

```
['google', 's', 'deepmind', 'ai', 'has', 'done', 'it', 'again', 'it', 's',
'beaten', 'go', 'world', 'champion', 'lee', 'sedol', 'in', 'the', 'second',
'of']
```

```
>>> import re
>>> paragraphA = sc.textFile("file:///mnt/hgfs/VM_shared/hw06/AlphaGoA.txt")
>>> wordsA = paragraphA.flatMap(lambda line: re.sub(r"[^a-zA-Z\d]", " ", str(line)).lower().split
())
>>> wordsA.count()
159
>>> wordsA.take(20)
['a', 'huge', 'milestone', 'has', 'just', 'been', 'reached', 'in', 'the', 'field', 'of', 'artific
ial', 'intelligence', 'alphago', 'a', 'program', 'developed', 'by', 'google', 's']
>>>
>>> paragraphB = sc.textFile("file:///mnt/hgfs/VM_shared/hw06/AlphaGoB.txt")
>>> wordsB = paragraphB.flatMap(lambda line: re.sub(r"[^a-zA-Z\d]", " ", str(line)).lower().split
())
>>> wordsB.count()
166
>>> wordsB.take(20)
['google', 's', 'deepmind', 'ai', 'has', 'done', 'it', 'again', 'it', 's', 'beaten', 'go', 'world
', 'champion', 'lee', 'sedol', 'in', 'the', 'second', 'of']
```

```
>>> wordsA.take(10)
['a', 'huge', 'milestone', 'has', 'just', 'been', 'reached', 'in', 'the',
'field']
>>> wordsB.take(10)
['google', 's', 'deepmind', 'ai', 'has', 'done', 'it', 'again', 'it', 's']
```

2. Create RDDs with unique words in each paragraph using function “distinct()”.

```
>>> distinctA = wordsA.distinct()
>>> distinctA.count()
105
>>> distinctA.take(10)
['deepmind', 'founder', 'being', 'developed', 'milestone', 'held', 'expressed',
'go', 'its', 'perfect']
>>> distinctB = wordsB.distinct()
>>> distinctB.count()
110
>>> distinctB.take(10)
['currently', 'four', 'go', 'before', 'poses', 'had', 'to', 'only', '4', 'has']
```

3. Create RDD contains words only in A but not in B using fuction “substract()”.

```
>>> wordsOnlyInA = distinctA.subtract(distinctB)
>>> wordsOnlyInA.count()
68
>>> wordsOnlyInA.take(10)
['founder', 'being', 'is', 'defeated', 'been', 'dol', 'unit', 'perfect',
'just', '28']
>>> wordsOnlyInA.collect()
['founder', 'being', 'is', 'defeated', 'been', 'dol', 'unit', 'perfect',
'just', '28', 'expressed', 'matches', 'amazing', 'taken', 'se', 'play',
'seconds', 'hours', 'game', 'legendary', 'with', 'about', 'limits', 'skills',
'lead', 'surprised', 'hugely', 'very', 'tense', 'hassabis', 'minutes',
'developed', 'held', 'its', 'program', 'remaining', 'his', 'historic',
```

```
'milestone', 'silver', 'didn', 'calling', 't', 'lose', 'computer', 'expect',
'respect', 'resigned', 'really', 'clock', 'korea', 'team', 'exciting', 'by',
'david', 'demis', 'reached', 'three', 'that', 'pushed', 'player', 'manner',
'huge', 'field', 'seoul', 'such', 'competing', 'think']
```

```
>>> wordsOnlyInA = distinctA.subtract(distinctB)
>>> wordsOnlyInA.count()
68
>>> wordsOnlyInA.take(10)
['founder', 'being', 'is', 'defeated', 'been', 'dol', 'unit', 'perfect', 'just', '28']
>>> wordsOnlyInA.collect()
['founder', 'being', 'is', 'defeated', 'been', 'dol', 'unit', 'perfect', 'just', '28', 'expressed', 'ma
tches', 'amazing', 'taken', 'se', 'play', 'seconds', 'hours', 'game', 'legendary', 'with', 'about', 'li
mits', 'skills', 'lead', 'surprised', 'hugely', 'very', 'tense', 'hassabis', 'minutes', 'developed', 'h
eld', 'its', 'program', 'remaining', 'his', 'historic', 'milestone', 'silver', 'didn', 'calling', 't',
'lose', 'computer', 'expect', 'respect', 'resigned', 'really', 'clock', 'korea', 'team', 'exciting', 'b
y', 'david', 'demis', 'reached', 'three', 'that', 'pushed', 'player', 'manner', 'huge', 'field', 'seoul
', 'such', 'competing', 'think']
```

4. Create RDD contains words common to two paragraphs using “intersection()”.

```
>>> intersection = distinctA.intersection(distinctB)
>>> intersection.count()
37
>>> intersection.collect()
['and', 'deepmind', 'artificial', 'go', 'for', '1', 'was', 'match', 'dan', '9',
'five', 'the', 'a', 'would', 'i', 's', 'south', 'half', 'has', 'prize',
'google', 'series', 'said', 'lee', 'million', 'after', 'on', 'first', 'to',
'intelligence', 'of', 'it', 'an', 'in', 'alphago', 'time', 'professional']
```

5. Final results for running “Assign06_P1.py”.

```
[cloudera@quickstart hw06]$ spark-submit Assign06_P1.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/Static
LoggerBinder.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]

Total words in A: 159
['a', 'huge', 'milestone', 'has', 'just', 'been', 'reached', 'in', 'the', 'field', 'of', 'artificial',
'intelligence', 'alphago', 'a', 'program', 'developed', 'by', 'google', 's']

Total words in B: 166
['google', 's', 'deepmind', 'ai', 'has', 'done', 'it', 'again', 'it', 's', 'beaten', 'go', 'world', 'ch
ampion', 'lee', 'sedol', 'in', 'the', 'second', 'of']

List first 10 words in A:
['a', 'huge', 'milestone', 'has', 'just', 'been', 'reached', 'in', 'the', 'field']

List first 10 words in B:
['google', 's', 'deepmind', 'ai', 'has', 'done', 'it', 'again', 'it', 's']

Total unique words in A: 105

List first 10 unique words in A:
['deepmind', 'founder', 'being', 'developed', 'milestone', 'held', 'expressed', 'go', 'its', 'perfect']

Total unique words in B: 110

List first 10 unique words in B:
['currently', 'four', 'go', 'before', 'poses', 'had', 'to', 'only', '4', 'has']
```

```
Total words only in A: 68
['founder', 'being', 'is', 'defeated', 'been', 'dol', 'unit', 'perfect', 'just', '28', 'expressed', 'ma
tches', 'amazing', 'taken', 'se', 'play', 'seconds', 'hours', 'game', 'legendary', 'with', 'about', 'li
mits', 'skills', 'lead', 'surprised', 'hugely', 'very', 'tense', 'hassabis', 'minutes', 'developed', 'h
eld', 'its', 'program', 'remaining', 'his', 'historic', 'milestone', 'silver', 'didn', 'calling', 't',
'lose', 'computer', 'expect', 'respect', 'resigned', 'really', 'clock', 'korea', 'team', 'exciting', 'b
y', 'david', 'demis', 'reached', 'three', 'that', 'pushed', 'player', 'manner', 'huge', 'field', 'seoul
', 'such', 'competing', 'think']

Total words in both A and B: 37
['and', 'deepmind', 'artificial', 'go', 'for', '1', 'was', 'match', 'dan', '9', 'five', 'the', 'a', 'wo
uld', 'i', 's', 'south', 'half', 'has', 'prize', 'google', 'series', 'said', 'lee', 'million', 'after',
 'on', 'first', 'to', 'intelligence', 'of', 'it', 'an', 'in', 'alphago', 'time', 'professional']
```

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.

Solution:

1. Create RDD "emps" by importing file from local file system to Spark.

```
>>> sqlContext = SQLContext(sc)
>>> emps = sc.textFile("file:///mnt/hgfs/VM_shared/hw06/emps.txt")
>>> emps.collect()
[u'Michael, 29, 3000.30', u'Andy, 30, 2500.25', u'Justin, 19, 4000.99']
```

2. Create new RDD "emps_fields" by transform every line into tuples. Since the words are split by ", ". There are two ways to extract the words. We can directly split on ", " (contains space) or split on comma and then use "strip" to remove the leading and trailing spaces in the word.

```
>>> fields = emps.map(lambda line: str(line).split(", "))
>>> fields.collect()
[['Michael', '29', '3000.30'], ['Andy', '30', '2500.25'], ['Justin', '19',
'4000.99']]
```

```
>>> emps_fields = fields.map(lambda p: (p[0], p[1], p[2]))
>>> emps_fields.collect()
[('Michael', '29', '3000.30'), ('Andy', '30', '2500.25'), ('Justin', '19',
'4000.99')]
```

```
>>> fields = emps.map(lambda line: str(line).split(", "))
>>> fields.collect()
[['Michael', '29', '3000.30'], ['Andy', '30', '2500.25'], ['Justin', '19', '4000.99']]
>>> emps_fields = fields.map(lambda p: (p[0], p[1], p[2]))
>>> emps_fields.collect()
[('Michael', '29', '3000.30'), ('Andy', '30', '2500.25'), ('Justin', '19', '4000.99')]
```

```
>>> fields = emps.map(lambda line: str(line).split(","))
>>> fields.collect()
[['Michael', ' 29', ' 3000.30'], ['Andy', ' 30', ' 2500.25'], ['Justin', ' 19',
' 4000.99']]
>>> emps_fields = fields.map(lambda p: (p[0].strip(), p[1].strip(),
p[2].strip()))
>>> emps_fields.collect()
[('Michael', '29', '3000.30'), ('Andy', '30', '2500.25'), ('Justin', '19',
'4000.99')]
```

```
>>> fields = emps.map(lambda line: str(line).split(","))
>>> fields.collect()
[['Michael', ' 29', ' 3000.30'], ['Andy', ' 30', ' 2500.25'], ['Justin', ' 19', ' 4000.99']]
>>> emps_fields = fields.map(lambda p: (p[0].strip(), p[1].strip(), p[2].strip()))
>>> emps_fields.collect()
[('Michael', '29', '3000.30'), ('Andy', '30', '2500.25'), ('Justin', '19', '4000.99')]
```

3. Construct a Row with RDD "emps_fields".

```
>>> from pyspark.sql.types import *
>>> from pyspark.sql import Row
>>> employee = emps_fields.map(lambda e : Row(name = e[0], age = int(e[1]),
salary = float(e[2])))
>>> employee.collect()
[Row(age=29, name='Michael', salary=3000.3000000000002), Row(age=30,
name='Andy', salary=2500.25), Row(age=19, name='Justin',
salary=4000.9899999999998)]
```

4. Create a DataFrame using the Row constructor above. Check the content of the data frame, its schema and do query using Spark DataFrame API.

```
>>> employeeDf = sqlContext.createDataFrame(employee)
>>> employeeDf.show()
+---+-------+-------+
|age|   name| salary|
+---+-------+-------+
| 29|Michael| 3000.3|
| 30|   Andy|2500.25|
| 19| Justin|4000.99|
+---+-------+-------+
```

```
>>> employeeDf.printSchema()
```

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
 |-- salary: double (nullable = true)


>>> employeeDf.filter(employeeDf.salary > 3500).show()
+---+------+-------+
|age|  name| salary|
+---+------+-------+
| 19|Justin|4000.99|
+---+------+-------+
```

```
>>> employeeDf = sqlContext.createDataFrame(employee)
>>> employeeDf.show()
+---+-------+-------+
|age|   name| salary|
+---+-------+-------+
| 29|Michael| 3000.3|
| 30|   Andy|2500.25|
| 19| Justin|4000.99|
+---+-------+-------+
```

```
>>> employeeDf.filter(employeeDf.salary > 3500).select("name").show()
+------+
|  name|
+------+
|Justin|
+------+
```

5. Transform the DataFrame into a Temporary Table and do the query. Select names of all employees who have a salary > 3500.

```
>>> employeeDf.registerTempTable("employeeTbl")
>>> sqlContext.sql("SELECT * FROM employeeTbl WHERE salary > 3500").show()
+---+------+-------+
|age|  name| salary|
+---+------+-------+
| 19|Justin|4000.99|
+---+------+-------+
```

```
>>> employeeDf.filter(employeeDf.salary > 3500).show()
+---+------+-------+
|age|  name| salary|
+---+------+-------+
| 19|Justin|4000.99|
+---+------+-------+
```

```
>>> sqlContext.sql("SELECT name FROM employeeTbl WHERE salary > 3500").show()
+------+
|  name|
+------+
|Justin|
+------+
```

```
>>> employeeDf.registerTempTable("employeeTbl")
>>> sqlContext.sql("SELECT name FROM employeeTbl WHERE salary > 3500").show()
+------+
|  name|
+------+
|Justin|
+------+
```

6. Final results for running “Assign06_P2.py”.

```
[cloudera@quickstart hw06]$ spark-submit Assign06_P2.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/Static
LoggerBinder.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]

Show the 'emps_fields' tuple:
[('Michael', '29', '3000.30'), ('Andy', '30', '2500.25'), ('Justin', '19', '4000.99')]

Show the RDD 'employee':
[Row(age=29, name='Michael', salary=3000.3000000000002), Row(age=30, name='Andy', salary=2500.25), Row(
age=19, name='Justin', salary=4000.9899999999998)]

Show the data frame 'employeeDf':
+---+-------+-------+
|age|   name| salary|
+---+-------+-------+
| 29|Michael| 3000.3|
| 30|   Andy|2500.25|
| 19| Justin|4000.99|
+---+-------+-------+
Show the schema of 'employeeDf':
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
 |-- salary: double (nullable = true)


Select names of all employees whose salary > 3500:
+------+
|  name|
+------+
|Justin|
+------+


Show the results queried from the temporary table:
+------+
|  name|
+------+
|Justin|
+------+
```

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`. 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.

Solution:

1. Read in the file from HDFS. We have to parse each line first, import data into an RDD object like problem 2.

```
[cloudera@quickstart hw06]$ hadoop fs -put /mnt/hgfs/VM_shared/hw06/ebay.csv
input
[cloudera@quickstart hw06]$ hadoop fs -ls input
```

```
[cloudera@quickstart hw06]$ hadoop fs -put /mnt/hgfs/VM_shared/hw06/ebay.csv input
[cloudera@quickstart hw06]$ hadoop fs -ls input
Found 5 items
-rw-r--r--   1 cloudera cloudera    5258688 2016-02-28 20:02 input/all-bible
-rw-r--r--   1 cloudera cloudera    5263360 2016-02-28 20:02 input/all-bible.tar
-rw-r--r--   1 cloudera cloudera    5284231 2016-02-28 20:02 input/all-shakespeare
-rw-r--r--   1 cloudera cloudera    5294080 2016-02-28 20:02 input/all-shakespeare.tar
-rw-r--r--   1 cloudera cloudera     585668 2016-03-09 16:10 input/ebay.csv
```

```
>>> lines = sc.textFile("input/ebay.csv").map(lambda line:
str(line).split(","))
>>> lines.count()
10654
>>> lines.take(3)
[['8213034705', '95', '2.927373', 'jake7870', '0', '95', '117.5', 'xbox', '3'],
['8213034705', '115', '2.943484', 'davidbresler2', '1', '95', '117.5', 'xbox',
'3'], ['8213034705', '100', '2.951285', 'gladimacowgirl', '58', '95', '117.5',
'xbox', '3']]
```

2. Then we transform the RDD into another RDD of Rows by assign the schema.

```
>>> from pyspark.sql import SQLContext, Row
>>> auctions = lines.map(lambda l: Row(auctionid = int(l[0]), bid =
float(l[1]), bidtime = float(l[2]), bidder = l[3], bidderrate = float(l[4]),
openbid = float(l[5]), price = float(l[6]), item = l[7], daystolive =
int(l[8])))

>>> auctions.take(1)
[Row(auctionid=8213034705, bid=95.0, bidder='jake7870', bidderrate=0.0,
bidtime=2.9273729999999998, daystolive=3, item='xbox', openbid=95.0,
price=117.5)]
```

3. Create the DataFrame "Auction" from the RDD. Show its schema and do some queries.

```
>>> sqlContext = SQLContext(sc)
>>> Auction = sqlContext.createDataFrame(auctions)
>>> Auction.printSchema()
root
 |-- auctionid: long (nullable = true)
 |-- bid: double (nullable = true)
 |-- bidder: string (nullable = true)
 |-- bidderrate: double (nullable = true)
 |-- bidtime: double (nullable = true)
 |-- daystolive: long (nullable = true)
 |-- item: string (nullable = true)
 |-- openbid: double (nullable = true)
 |-- price: double (nullable = true)

>>> Auction.show(3)
```

```
>>> sqlContext = SQLContext(sc)
>>> Auction = sqlContext.createDataFrame(auctions)
>>> Auction.printSchema()
root
 |-- auctionid: long (nullable = true)
 |-- bid: double (nullable = true)
 |-- bidder: string (nullable = true)
 |-- bidderrate: double (nullable = true)
 |-- bidtime: double (nullable = true)
 |-- daystolive: long (nullable = true)
 |-- item: string (nullable = true)
 |-- openbid: double (nullable = true)
 |-- price: double (nullable = true)

>>> Auction.show(3)
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 3 rows
```

Total lines we got from the file:

```
>>> Auction.count()
10654
```

Total number of auctions we held:

```
>>> Auction.select("auctionid").distinct().count()
627
```

Total number of bids were made per item:

```
>>> Auction.groupBy("item").count().show()
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+
```

Max, min, average bid (price) per item:

```
>>> Auction.groupBy("item").max("price").show()
+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

>>> Auction.groupBy("item").min("price").show()
+-------+----------+
|   item|min(price)|
+-------+----------+
```

```
|   xbox|      31.0|
|   palm|     175.0|
|cartier|      26.0|
+-------+----------+

>>> Auction.groupBy("item").mean("price").show()
+-------+------------------+
|   item|        avg(price)|
+-------+------------------+
|   xbox|144.27594109195397|
|   palm| 231.1302670272107|
|cartier| 925.0479057859695|
+-------+------------------+
```

```
>>> Auction.groupBy("item").max("price").show()
+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

>>> Auction.groupBy("item").min("price").show()
+-------+----------+
|   item|min(price)|
+-------+----------+
|   xbox|      31.0|
|   palm|     175.0|
|cartier|      26.0|
+-------+----------+

>>> Auction.groupBy("item").mean("price").show()
+-------+-----------------+
|   item|       avg(price)|
+-------+-----------------+
|   xbox|144.27594109195397|
|   palm| 231.1302670272107|
|cartier| 925.0479057859695|
+-------+-----------------+
```

Max, min, average number of bids per item:

Here I calculated on how many bids were made per item. I group by two columns (auctionid, item). For each item, it has many auctions. And for each auction, it has several bids. So I use (auctionid, item) as a pair, and count how many bids made for each pair.

Here we order by counts, we can see for each (auctionid, item), who has the largest number of bids. And we do aggregate function max(), min(), mean() on “count”.

```
>>> from pyspark.sql.functions import *
>>> AuctionsPerItem = Auction.groupBy("auctionid",
"item").count().orderBy(desc("count"))
>>> AuctionsPerItem.show(10)
+----------+-------+-----+
| auctionid|   item|count|
+----------+-------+-----+
|8214355679|   xbox|   75|
|8212629520|   xbox|   57|
|3023174478|   palm|   54|
```

```
|3020532816|   palm|   51|
|8212602164|   xbox|   50|
|1644594033|cartier|   46|
|3024823511|   palm|   45|
|8212359174|   xbox|   45|
|3018788243|   palm|   45|
|8212236671|   xbox|   43|
+----------+-------+-----+
only showing top 10 rows
```

```
>>> from pyspark.sql.functions import *
>>> AuctionsPerItem = Auction.groupBy("auctionid", "item").count().orderBy(desc("count"))
>>> AuctionsPerItem.show(10)
+----------+-------+-----+
| auctionid|   item|count|
+----------+-------+-----+
|8214355679|   xbox|   75|
|8212629520|   xbox|   57|
|3023174478|   palm|   54|
|3020532816|   palm|   51|
|8212602164|   xbox|   50|
|1644594033|cartier|   46|
|3024823511|   palm|   45|
|8212359174|   xbox|   45|
|3018788243|   palm|   45|
|8212236671|   xbox|   43|
+----------+-------+-----+
only showing top 10 rows
```

```
>>> AuctionsPerItem.groupBy("item").max("count").show()
+-------+----------+
|   item|max(count)|
+-------+----------+
|   xbox|        75|
|   palm|        54|
|cartier|        46|
+-------+----------+

>>> AuctionsPerItem.groupBy("item").min("count").show()
+-------+----------+
|   item|min(count)|
+-------+----------+
|   xbox|         2|
|   palm|         1|
|cartier|         2|
+-------+----------+

>>> AuctionsPerItem.groupBy("item").mean("count").show()
+-------+------------------+
|   item|        avg(count)|
+-------+------------------+
|   xbox| 18.81081081081081|
|   palm|17.250728862973762|
|cartier|14.360294117647058|
+-------+------------------+
```

```
>>> AuctionsPerItem.groupBy("item").max("count").show()
+-------+----------+
|   item|max(count)|
+-------+----------+
|   xbox|        75|
|   palm|        54|
|cartier|        46|
+-------+----------+

>>> AuctionsPerItem.groupBy("item").min("count").show()
+-------+----------+
|   item|min(count)|
+-------+----------+
|   xbox|         2|
|   palm|         1|
|cartier|         2|
+-------+----------+

>>> AuctionsPerItem.groupBy("item").mean("count").show()
+-------+------------------+
|   item|        avg(count)|
+-------+------------------+
|   xbox| 18.81081081081081|
|   palm|17.250728862973762|
|cartier|14.360294117647058|
+-------+------------------+
```

Show the bids with price > 100:

```
>>> Auction.filter(Auction.price > 100).count()
10043
>>> Auction.filter(Auction.price > 100).show(10)
```

```
>>> Auction.filter(Auction.price > 100).count()
10043
>>> Auction.filter(Auction.price > 100).show(10)
+----------+-----+-------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|       bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+-------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|     jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
|8213034705|117.5|      daysrus|      10.0|2.998947|         3|xbox|   95.0|117.5|
|8213060420|  2.0|    donnie4814|       5.0|0.065266|         3|xbox|    1.0|120.0|
|8213060420|15.25|   myreeceyboy|      52.0|0.123218|         3|xbox|    1.0|120.0|
|8213060420|  3.0|  parakeet2004|       5.0|0.186539|         3|xbox|    1.0|120.0|
|8213060420| 10.0|  parakeet2004|       5.0| 0.18669|         3|xbox|    1.0|120.0|
|8213060420|24.99|  parakeet2004|       5.0|0.187049|         3|xbox|    1.0|120.0|
|8213060420| 20.0| bluebubbles_1|      25.0|0.249491|         3|xbox|    1.0|120.0|
+----------+-----+-------------+----------+--------+----------+----+-------+-----+
only showing top 10 rows
```

4. Transform the DataFrame as a table and make the same 4 queries using SQL.

```
>>> Auction.registerTempTable("AuctionTbl")
>>> sqlContext.sql("SELECT count(DISTINCT auctionid) AS count FROM
AuctionTbl").show()
+-----+
|count|
+-----+
|  627|
```

```
+-----+

>>> sqlContext.sql("SELECT item, count(*) AS count FROM AuctionTbl GROUP BY
item").show()
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+

>>> sqlContext.sql("SELECT item, max(price) AS max, min(price) AS min,
avg(price) AS average FROM AuctionTbl GROUP BY item").show()
+-------+------+-----+------------------+
|   item|   max|  min|           average|
+-------+------+-----+------------------+
|   xbox|501.77| 31.0|144.27594109195397|
|   palm| 290.0|175.0| 231.1302670272107|
|cartier|5400.0| 26.0| 925.0479057859695|
+-------+------+-----+------------------+
```

```
>>> Auction.registerTempTable("AuctionTbl")
>>> sqlContext.sql("SELECT count(DISTINCT auctionid) AS count FROM AuctionTbl").show()
+-----+
|count|
+-----+
|  627|
+-----+

>>> sqlContext.sql("SELECT item, count(*) AS count FROM AuctionTbl GROUP BY item").show()
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+

>>> sqlContext.sql("SELECT item, max(price) AS max, min(price) AS min, avg(price) AS average FROM Auc
tionTbl GROUP BY item").show()
+-------+------+-----+------------------+
|   item|   max|  min|           average|
+-------+------+-----+------------------+
|   xbox|501.77| 31.0|144.27594109195397|
|   palm| 290.0|175.0| 231.1302670272107|
|cartier|5400.0| 26.0| 925.0479057859695|
+-------+------+-----+------------------+
```

```
>>> AuctionsPerItem = sqlContext.sql("SELECT auctionid, item, count(bid) AS
bids FROM AuctionTbl GROUP BY auctionid, item ORDER BY bids DESC")
>>> AuctionsPerItem.show(10)
+----------+-------+----+
| auctionid|   item|bids|
+----------+-------+----+
|8214355679|   xbox|  75|
|8212629520|   xbox|  57|
|3023174478|   palm|  54|
|3020532816|   palm|  51|
|8212602164|   xbox|  50|
```

```
|1644594033|cartier|  46|
|3024823511|   palm|  45|
|8212359174|   xbox|  45|
|3018788243|   palm|  45|
|8212236671|   xbox|  43|
+----------+-------+----+
only showing top 10 rows
```

```
>>> AuctionsPerItem = sqlContext.sql("SELECT auctionid, item, count(bid) AS bids FROM AuctionTbl GROU
P BY auctionid, item ORDER BY bids DESC")
>>> AuctionsPerItem.show(10)
+----------+-------+----+
| auctionid|   item|bids|
+----------+-------+----+
|8214355679|   xbox|  75|
|8212629520|   xbox|  57|
|3023174478|   palm|  54|
|3020532816|   palm|  51|
|8212602164|   xbox|  50|
|1644594033|cartier|  46|
|3024823511|   palm|  45|
|8212359174|   xbox|  45|
|3018788243|   palm|  45|
|8212236671|   xbox|  43|
+----------+-------+----+
only showing top 10 rows
```

Store the results “AuctionsPerItem” to a temporary table and do a second query.

```
>>> AuctionsPerItem.registerTempTable("AuctionsPerItemTbl")
>>> sqlContext.sql("SELECT item, max(bids) AS max, min(bids) AS min, avg(bids)
AS average FROM AuctionsPerItemTbl GROUP BY item").show()
+-------+---+---+------------------+
|   item|max|min|           average|
+-------+---+---+------------------+
|   xbox| 75|  2| 18.81081081081081|
|   palm| 54|  1|17.250728862973762|
|cartier| 46|  2|14.360294117647058|
+-------+---+---+------------------+
```

```
>>> AuctionsPerItem.registerTempTable("AuctionsPerItemTbl")
>>> sqlContext.sql("SELECT item, max(bids) AS max, min(bids) AS min, avg(bids) AS average FROM Auctio
nsPerItemTbl GROUP BY item").show()
+-------+---+---+------------------+
|   item|max|min|           average|
+-------+---+---+------------------+
|   xbox| 75|  2| 18.81081081081081|
|   palm| 54|  1|17.250728862973762|
|cartier| 46|  2|14.360294117647058|
+-------+---+---+------------------+
```

```
>>> sqlContext.sql("SELECT * FROM AuctionTbl WHERE price > 100").show(10)
```

```
>>> sqlContext.sql("SELECT * FROM AuctionTbl WHERE price > 100").show(10)
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
|8213034705|117.5|       daysrus|      10.0|2.998947|         3|xbox|   95.0|117.5|
|8213060420|  2.0|    donnie4814|       5.0|0.065266|         3|xbox|    1.0|120.0|
|8213060420|15.25|   myreeceyboy|      52.0|0.123218|         3|xbox|    1.0|120.0|
|8213060420|  3.0|  parakeet2004|       5.0|0.186539|         3|xbox|    1.0|120.0|
|8213060420| 10.0|  parakeet2004|       5.0| 0.18669|         3|xbox|    1.0|120.0|
|8213060420|24.99|  parakeet2004|       5.0|0.187049|         3|xbox|    1.0|120.0|
|8213060420| 20.0| bluebubbles_1|      25.0|0.249491|         3|xbox|    1.0|120.0|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 10 rows
```

5. Persist the DataFrame as a Parquet file. Here I also transform the intermediate RDD "AuctionsPerItem" into a Parquet file. The Parquet file will be stored in HDFS automatically.

```
>>> Auction.write.save("auction.parquet", format = "parquet")
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-
cdh5.5.0.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]
```

```
>>> Auction.write.save("auction.parquet", format = "parquet")
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-pig-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/org/s
lf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-hadoop-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-exec-1.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j/impl
/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-jdbc-1.1.0-cdh5.5.0-standalone.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]
```

```
>>> AuctionsPerItem.write.save("auctionsPerItem.parquet", format = "parquet")
```

```
>>> AuctionsPerItem.write.save("auctionsPerItem.parquet", format = "parquet")
[Stage 114:>                                                        (0 + 1) / 46]SLF4J: Class path con
tains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf
4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-pig-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/org
/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-hadoop-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-exec-1.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-jdbc-1.1.0-cdh5.5.0-standalone.jar!/shaded/parquet/o
rg/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]
```

```
[cloudera@quickstart ~]$ hadoop fs -ls auction.parquet
```

```
[cloudera@quickstart ~]$ hadoop fs -ls .
Found 6 items
drwxr-xr-x   - cloudera cloudera          0 2016-03-09 17:22 auction.parquet
drwxr-xr-x   - cloudera cloudera          0 2016-03-10 23:04 auctionsPerItem.parquet
drwxr-xr-x   - cloudera cloudera          0 2016-03-04 08:48 bible_freq
drwxr-xr-x   - cloudera cloudera          0 2016-03-09 16:10 input
drwxr-xr-x   - cloudera cloudera          0 2016-03-04 08:48 shakespeare_freq
drwxr-xr-x   - cloudera cloudera          0 2016-02-22 21:47 spark_output
[cloudera@quickstart ~]$ hadoop fs -ls auction.parquet
Found 4 items
-rw-r--r--   1 cloudera cloudera          0 2016-03-09 17:22 auction.parquet/_SUCCESS
-rw-r--r--   1 cloudera cloudera        872 2016-03-09 17:22 auction.parquet/_common_metadata
-rw-r--r--   1 cloudera cloudera       2022 2016-03-09 17:22 auction.parquet/_metadata
-rw-r--r--   1 cloudera cloudera     137184 2016-03-09 17:22 auction.parquet/part-r-00000-27ca6222-a7a8-41c1
-8760-91240d0db313.gz.parquet
[cloudera@quickstart ~]$ hadoop fs -ls auctionsPerItem.parquet
Found 49 items
-rw-r--r--   1 cloudera cloudera          0 2016-03-10 23:04 auctionsPerItem.parquet/_SUCCESS
-rw-r--r--   1 cloudera cloudera        392 2016-03-10 23:04 auctionsPerItem.parquet/_common_metadata
-rw-r--r--   1 cloudera cloudera      16683 2016-03-10 23:04 auctionsPerItem.parquet/_metadata
-rw-r--r--   1 cloudera cloudera        843 2016-03-10 23:04 auctionsPerItem.parquet/part-r-00000-abb724ba-8
286-46a8-b662-97f161b65e97.gz.parquet
```

6. Exit pyspark shell and enter again. Read the data from the Parquet File and create a DataFrame and SQL like table to do the query again.

```
>>> parquetFile = sqlContext.read.parquet("auction.parquet")
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-
cdh5.5.0.jar!/shaded/parquet/org/slf4j/impl/StaticLoggerBinder.class]
```

```
Using Python version 2.6.6 (r266:84292, Feb 22 2013 00:00:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>> parquetFile = sqlContext.read.parquet("auction.parquet")
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-pig-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/org/s
lf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-hadoop-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-exec-1.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j/impl
/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-jdbc-1.1.0-cdh5.5.0-standalone.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]
```

```
>>> parquetFile.groupBy("item").max("price").show()
+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

>>> parquetFile.groupBy("item").min("price").show()
+-------+----------+
|   item|min(price)|
+-------+----------+
|   xbox|      31.0|
|   palm|     175.0|
```

```
|cartier|      26.0|
+-------+----------+

>>> parquetFile.groupBy("item").mean("price").show()
+-------+------------------+
|   item|        avg(price)|
+-------+------------------+
|   xbox|144.27594109195397|
|   palm| 231.1302670272107|
|cartier| 925.0479057859695|
+-------+------------------+
```

Transform the DataFrame into a Temporary table and do SQL like queries.

```
>>> parquetFile.registerTempTable("AuctionParq")
>>> sqlContext.sql("SELECT count(DISTINCT auctionid) AS count FROM
AuctionParq").show()
+-----+
|count|
+-----+
|  627|
+-----+

>>> sqlContext.sql("SELECT item, count(*) AS count FROM AuctionParq GROUP BY
item").show()
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+

>>> sqlContext.sql("SELECT item, max(price) AS max, min(price) AS min,
avg(price) AS average FROM AuctionParq GROUP BY item").show()
+-------+------+-----+------------------+
|   item|   max|  min|           average|
+-------+------+-----+------------------+
|   xbox|501.77| 31.0|144.27594109195397|
|   palm| 290.0|175.0| 231.1302670272107|
|cartier|5400.0| 26.0| 925.0479057859695|
+-------+------+-----+------------------+


>>> parquetFileProcessed = sqlContext.read.parquet("auctionsPerItem.parquet")
>>> parquetFileProcessed.registerTempTable("AuctionsPerItemParq")
>>> sqlContext.sql("SELECT item, max(bids) AS max, min(bids) AS min, avg(bids)
AS average FROM AuctionsPerItemParq GROUP BY item").show()
+-------+---+---+------------------+
|   item|max|min|           average|
+-------+---+---+------------------+
|   xbox| 75|  2| 18.81081081081081|
|   palm| 54|  1|17.250728862973762|
```

```
|cartier| 46|  2|14.360294117647058|
+-------+---+---+------------------+
```

```
>>> parquetFileProcessed = sqlContext.read.parquet("auctionsPerItem.parquet")
>>> parquetFileProcessed.registerTempTable("AuctionsPerItemParq")
>>> sqlContext.sql("SELECT item, max(bids) AS max, min(bids) AS min, avg(bids) AS average FROM Auctio
nsPerItemParq GROUP BY item").show()
+-------+---+---+------------------+
|   item|max|min|           average|
+-------+---+---+------------------+
|   xbox| 75|  2| 18.81081081081081|
|   palm| 54|  1|17.250728862973762|
|cartier| 46|  2|14.360294117647058|
+-------+---+---+------------------+
```

```
>>> sqlContext.sql("SELECT * FROM AuctionParq WHERE price > 100").show(10)
```

```
>>> sqlContext.sql("SELECT * FROM AuctionParq WHERE price > 100").show(10)
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
|8213034705|117.5|       daysrus|      10.0|2.998947|         3|xbox|   95.0|117.5|
|8213060420|  2.0|    donnie4814|       5.0|0.065266|         3|xbox|    1.0|120.0|
|8213060420|15.25|   myreeceyboy|      52.0|0.123218|         3|xbox|    1.0|120.0|
|8213060420|  3.0|  parakeet2004|       5.0|0.186539|         3|xbox|    1.0|120.0|
|8213060420| 10.0|  parakeet2004|       5.0| 0.18669|         3|xbox|    1.0|120.0|
|8213060420|24.99|  parakeet2004|       5.0|0.187049|         3|xbox|    1.0|120.0|
|8213060420| 20.0| bluebubbles_1|      25.0|0.249491|         3|xbox|    1.0|120.0|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 10 rows
```

7. Final results for running “Assign06_P3_part1.py”.

```
[cloudera@quickstart hw06]$ spark-submit Assign06_P3_part1.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/StaticLogge
rBinder.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]
[Row(auctionid=8213034705, bid=95.0, bidder='jake7870', bidderrate=0.0, bidtime=2.9273729999999998, daystoli
ve=3, item='xbox', openbid=95.0, price=117.5)]
root
 |-- auctionid: long (nullable = true)
 |-- bid: double (nullable = true)
 |-- bidder: string (nullable = true)
 |-- bidderrate: double (nullable = true)
 |-- bidtime: double (nullable = true)
 |-- daystolive: long (nullable = true)
 |-- item: string (nullable = true)
 |-- openbid: double (nullable = true)
 |-- price: double (nullable = true)


Show 3 example records:
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 3 rows
```

```
Total auctions were held (Use Spark DataFrame API):
627

Total number of bids were made per item (Use Spark DataFrame API):
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+


Maximum, minimum and average bid (price) per item (Use Spark DataFrame API):
+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

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

+-------+------------------+
|   item|        avg(price)|
+-------+------------------+
|   xbox|144.27594109195397|
|   palm| 231.1302670272107|
|cartier| 925.0479057859695|
+-------+------------------+
```

```
Maximum, minimum and average number of bids per item (Use Spark DataFrame API):
+-------+----------+
|   item|max(count)|
+-------+----------+
|   xbox|        75|
|   palm|        54|
|cartier|        46|
+-------+----------+

+-------+----------+
|   item|min(count)|
+-------+----------+
|   xbox|         2|
|   palm|         1|
|cartier|         2|
+-------+----------+

+-------+------------------+
|   item|        avg(count)|
+-------+------------------+
|   xbox| 18.81081081081081|
|   palm|17.250728862973762|
|cartier|14.360294117647058|
+-------+------------------+
```

```
Show 10 bids with price > 100 (Use Spark DataFrame API):
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
|8213034705|117.5|       daysrus|      10.0|2.998947|         3|xbox|   95.0|117.5|
|8213060420|  2.0|    donnie4814|       5.0|0.065266|         3|xbox|    1.0|120.0|
|8213060420|15.25|   myreeceyboy|      52.0|0.123218|         3|xbox|    1.0|120.0|
|8213060420|  3.0|  parakeet2004|       5.0|0.186539|         3|xbox|    1.0|120.0|
|8213060420| 10.0|  parakeet2004|       5.0| 0.18669|         3|xbox|    1.0|120.0|
|8213060420|24.99|  parakeet2004|       5.0|0.187049|         3|xbox|    1.0|120.0|
|8213060420| 20.0| bluebubbles_1|      25.0|0.249491|         3|xbox|    1.0|120.0|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 10 rows
```

```
Total auctions were held (Use SQL):
+-----+
|count|
+-----+
|  627|
+-----+


Total number of bids were made per item (Use SQL):
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+


Maximum, minimum and average bid (price) per item (Use SQL):
+-------+------+-----+------------------+
|   item|   max|  min|           average|
+-------+------+-----+------------------+
|   xbox|501.77| 31.0|144.27594109195397|
|   palm| 290.0|175.0| 231.1302670272107|
|cartier|5400.0| 26.0| 925.0479057859695|
+-------+------+-----+------------------+


Maximum, minimum and average number of bids per item (Use SQL):
+-------+---+---+------------------+
|   item|max|min|           average|
+-------+---+---+------------------+
|   xbox| 75|  2| 18.81081081081081|
|   palm| 54|  1|17.250728862973762|
|cartier| 46|  2|14.360294117647058|
+-------+---+---+------------------+
```

```
Show 10 bids with price > 100 (Use SQL):
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
|8213034705|117.5|       daysrus|      10.0|2.998947|         3|xbox|   95.0|117.5|
|8213060420|  2.0|    donnie4814|       5.0|0.065266|         3|xbox|    1.0|120.0|
|8213060420|15.25|   myreeceyboy|      52.0|0.123218|         3|xbox|    1.0|120.0|
|8213060420|  3.0|  parakeet2004|       5.0|0.186539|         3|xbox|    1.0|120.0|
|8213060420| 10.0|  parakeet2004|       5.0| 0.18669|         3|xbox|    1.0|120.0|
|8213060420|24.99|  parakeet2004|       5.0|0.187049|         3|xbox|    1.0|120.0|
|8213060420| 20.0| bluebubbles_1|      25.0|0.249491|         3|xbox|    1.0|120.0|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 10 rows
```

8. Final results for running "Assign06_P3_part2.py".

```
[cloudera@quickstart hw06]$ spark-submit Assign06_P3_part2.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/Static
LoggerBinder.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]
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-pig-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/org/s
lf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-hadoop-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-exec-1.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j/impl
/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-jdbc-1.1.0-cdh5.5.0-standalone.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]

Maximum, minimum and average bid (price) per item (Use Spark DataFrame API):
+-------+----------+
|   item|max(price)|
+-------+----------+
|   xbox|    501.77|
|   palm|     290.0|
|cartier|    5400.0|
+-------+----------+

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

```
+-------+------------------+
|   item|        avg(price)|
+-------+------------------+
|   xbox|144.27594109195397|
|   palm| 231.1302670272107|
|cartier| 925.0479057859695|
+-------+------------------+


Total auctions were held (Use SQL):
+-----+
|count|
+-----+
|  627|
+-----+


Total number of bids were made per item (Use SQL):
+-------+-----+
|   item|count|
+-------+-----+
|   xbox| 2784|
|   palm| 5917|
|cartier| 1953|
+-------+-----+


Maximum, minimum and average bid (price) per item (Use SQL):
+-------+------+-----+------------------+
|   item|   max|  min|           average|
+-------+------+-----+------------------+
|   xbox|501.77| 31.0|144.27594109195397|
|   palm| 290.0|175.0| 231.1302670272107|
|cartier|5400.0| 26.0| 925.0479057859695|
+-------+------+-----+------------------+
```

```
Maximum, minimum and average number of bids per item (Use SQL):
+-------+---+---+------------------+
|   item|max|min|           average|
+-------+---+---+------------------+
|   xbox| 75|  2| 18.81081081081081|
|   palm| 54|  1|17.250728862973762|
|cartier| 46|  2|14.360294117647058|
+-------+---+---+------------------+


Show 10 bids with price > 100 (Use SQL):
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
| auctionid|  bid|        bidder|bidderrate| bidtime|daystolive|item|openbid|price|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
|8213034705| 95.0|      jake7870|       0.0|2.927373|         3|xbox|   95.0|117.5|
|8213034705|115.0| davidbresler2|       1.0|2.943484|         3|xbox|   95.0|117.5|
|8213034705|100.0|gladimacowgirl|      58.0|2.951285|         3|xbox|   95.0|117.5|
|8213034705|117.5|       daysrus|      10.0|2.998947|         3|xbox|   95.0|117.5|
|8213060420|  2.0|    donnie4814|       5.0|0.065266|         3|xbox|    1.0|120.0|
|8213060420|15.25|   myreeceyboy|      52.0|0.123218|         3|xbox|    1.0|120.0|
|8213060420|  3.0|  parakeet2004|       5.0|0.186539|         3|xbox|    1.0|120.0|
|8213060420| 10.0|  parakeet2004|       5.0| 0.18669|         3|xbox|    1.0|120.0|
|8213060420|24.99|  parakeet2004|       5.0|0.187049|         3|xbox|    1.0|120.0|
|8213060420| 20.0| bluebubbles_1|      25.0|0.249491|         3|xbox|    1.0|120.0|
+----------+-----+--------------+----------+--------+----------+----+-------+-----+
only showing top 10 rows
```

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.

Solution:

1. I first do the query from MySQL and Hive to get the answer (This part of code will not be in the source code file). Then I will do the query using Spark DataFrame API and Spark Temporary Tables to do a cross check of the answer.

```
[cloudera@quickstart ~]$ sudo mysql --user=retail_dba -p

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| retail_db          |
+--------------------+
2 rows in set (0.01 sec)

mysql> use retail_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------+
| Tables_in_retail_db |
+---------------------+
| categories          |
| customers           |
| departments         |
| order_items         |
| orders              |
| products            |
+---------------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM order_items LIMIT 20;
```

Here I do both queries to find orders with the largest number of order quantities per order and order items per order. I use "sum(order_item_quantity)" to calculate total order quantities. I use "count(order_item_id)" to count total order items.

```
mysql> SELECT order_item_order_id, sum(order_item_quantity) AS
total_order_item_quantity FROM order_items GROUP BY order_item_order_id ORDER
BY total_order_item_quantity DESC LIMIT 10;
+---------------------+--------------------------+
| order_item_order_id | total_order_item_quantity |
+---------------------+--------------------------+
```

```
|                24967 |                         24 |
|                21920 |                         24 |
|                49429 |                         24 |
|                49405 |                         24 |
|                43848 |                         23 |
|                33283 |                         23 |
|                17455 |                         22 |
|                56414 |                         22 |
|                56559 |                         22 |
|                29272 |                         22 |
+---------------------+----------------------------+
10 rows in set (0.12 sec)
```

This is just a validation about the above results.

```
mysql> SELECT order_item_order_id, order_item_id, order_item_quantity FROM
order_items WHERE order_item_order_id = 24967;
+---------------------+---------------+---------------------+
| order_item_order_id | order_item_id | order_item_quantity |
+---------------------+---------------+---------------------+
|                24967 |         62545 |                   5 |
|                24967 |         62546 |                   5 |
|                24967 |         62547 |                   4 |
|                24967 |         62548 |                   5 |
|                24967 |         62549 |                   5 |
+---------------------+---------------+---------------------+
5 rows in set (0.03 sec)

mysql> SELECT order_item_order_id, count(order_item_id) AS total_order_items
FROM order_items GROUP BY order_item_order_id ORDER BY total_order_items DESC
LIMIT 10;
+---------------------+-------------------+
| order_item_order_id | total_order_items |
+---------------------+-------------------+
|                45172 |                 5 |
|                55149 |                 5 |
|                65119 |                 5 |
|                40306 |                 5 |
|                55168 |                 5 |
|                40323 |                 5 |
|                60223 |                 5 |
|                30299 |                 5 |
|                40340 |                 5 |
|                55205 |                 5 |
+---------------------+-------------------+
10 rows in set (0.09 sec)
```

This is just a validation about the above results.

```
mysql> SELECT order_item_order_id, order_item_id FROM order_items WHERE
order_item_order_id = 45172;
+---------------------+---------------+
| order_item_order_id | order_item_id |
```

```
+---------------------+---------------+
|               45172 |        112828 |
|               45172 |        112829 |
|               45172 |        112830 |
|               45172 |        112831 |
|               45172 |        112832 |
+---------------------+---------------+
5 rows in set (0.03 sec)
```

```
mysql> SELECT order_item_order_id, order_item_id FROM order_items WHERE order_item_order_id
 = 45172;
+--------------------+--------------+
| order_item_order_id | order_item_id |
+--------------------+--------------+
|              45172 |        112828 |
|              45172 |        112829 |
|              45172 |        112830 |
|              45172 |        112831 |
|              45172 |        112832 |
+--------------------+--------------+
5 rows in set (0.03 sec)

mysql> SELECT order_item_order_id, order_item_id FROM order_items WHERE order_item_order_id
 = 55149;
+--------------------+--------------+
| order_item_order_id | order_item_id |
+--------------------+--------------+
|              55149 |        137903 |
|              55149 |        137904 |
|              55149 |        137905 |
|              55149 |        137906 |
|              55149 |        137907 |
+--------------------+--------------+
5 rows in set (0.03 sec)
```

If we want to select the orders with exact the largest number of order items. We need to use subquery here. First get the maximum count of order items. Then select out the orders having the maximum count of order items.

```
mysql> SELECT order_item_order_id, count(order_item_id) AS total_order_items
FROM order_items GROUP BY order_item_order_id HAVING count(order_item_id) =
(SELECT count(order_item_id) AS total_order_items FROM order_items GROUP BY
order_item_order_id ORDER BY total_order_items DESC LIMIT 1) LIMIT 10;
+---------------------+-------------------+
| order_item_order_id | total_order_items |
+---------------------+-------------------+
|                   5 |                 5 |
|                  10 |                 5 |
|                  11 |                 5 |
|                  12 |                 5 |
|                  15 |                 5 |
|                  17 |                 5 |
|                  24 |                 5 |
|                  28 |                 5 |
|                  29 |                 5 |
|                  58 |                 5 |
+---------------------+-------------------+
10 rows in set (0.15 sec)
```

```
mysql> SELECT count(order_item_id) AS total_order_items FROM order_items GROUP
BY order_item_order_id ORDER BY total_order_items DESC LIMIT 1;
+-------------------+
| total_order_items |
+-------------------+
|                 5 |
+-------------------+
1 row in set (0.08 sec)
```

```
mysql> SELECT order_item_order_id, count(order_item_id) AS total_order_items FROM order_ite
ms GROUP BY order_item_order_id HAVING count(order_item_id) = (SELECT count(order_item_id)
AS total_order_items FROM order_items GROUP BY order_item_order_id ORDER BY total_order_ite
ms DESC LIMIT 1) LIMIT 10;
+--------------------+-------------------+
| order_item_order_id | total_order_items |
+--------------------+-------------------+
|                  5 |                 5 |
|                 10 |                 5 |
|                 11 |                 5 |
|                 12 |                 5 |
|                 15 |                 5 |
|                 17 |                 5 |
|                 24 |                 5 |
|                 28 |                 5 |
|                 29 |                 5 |
|                 58 |                 5 |
+--------------------+-------------------+
10 rows in set (0.15 sec)

mysql> SELECT count(order_item_id) AS total_order_items FROM order_items GROUP BY order_ite
m_order_id ORDER BY total_order_items DESC LIMIT 1;
+-------------------+
| total_order_items |
+-------------------+
|                 5 |
+-------------------+
1 row in set (0.08 sec)
```

2. Use "Sqoop" to import all tables from MySQL to Hive. Check HDFS that all the tables have been imported.

```
[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.
16/03/09 18:02:13 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.5.0
....................
       File Input Format Counters
              Bytes Read=0
       File Output Format Counters
              Bytes Written=0
16/03/09 18:06:00 INFO mapreduce.ImportJobBase: Transferred 46.1318 KB in
31.3821 seconds (1.47 KB/sec)
16/03/09 18:06:00 INFO mapreduce.ImportJobBase: Retrieved 1345 records.
```

```
[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.
16/03/09 18:02:13 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.5.0
16/03/09 18:02:13 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Conside
r using -P instead.
16/03/09 18:02:13 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
16/03/09 18:02:13 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
16/03/09 18:02:13 WARN tool.BaseSqoopTool: It seems that you're doing hive import directly into default
16/03/09 18:02:13 WARN tool.BaseSqoopTool: hive warehouse directory which is not supported. Sqoop is
16/03/09 18:02:13 WARN tool.BaseSqoopTool: firstly importing data into separate directory and then
16/03/09 18:02:13 WARN tool.BaseSqoopTool: inserting data into hive. Please consider removing
16/03/09 18:02:13 WARN tool.BaseSqoopTool: --target-dir or --warehouse-dir into /user/hive/warehouse in
16/03/09 18:02:13 WARN tool.BaseSqoopTool: case that you will detect any issues.
16/03/09 18:02:13 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
16/03/09 18:02:13 INFO tool.CodeGenTool: Beginning code generation
16/03/09 18:02:13 INFO tool.CodeGenTool: Will generate java class as codegen_categories
16/03/09 18:02:14 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `categories` AS t LIM
IT 1
```

```
        Map-Reduce Framework
                Map input records=1345
                Map output records=1345
                Input split bytes=87
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=236
                CPU time spent (ms)=3140
                Physical memory (bytes) snapshot=163160064
                Virtual memory (bytes) snapshot=1522110464
                Total committed heap usage (bytes)=60882944
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=0
16/03/09 18:06:00 INFO mapreduce.ImportJobBase: Transferred 46.1318 KB in 31.3821 seconds (1.47 KB/sec)
16/03/09 18:06:00 INFO mapreduce.ImportJobBase: Retrieved 1345 records.
```

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse
```

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse
Found 10 items
drwxrwxrwx   - cloudera supergroup          0 2016-03-03 22:17 /user/hive/warehouse/apachelog
drwxrwxrwx   - cloudera supergroup          0 2016-03-09 18:02 /user/hive/warehouse/categories
drwxrwxrwx   - cloudera supergroup          0 2016-03-09 18:03 /user/hive/warehouse/customers
drwxrwxrwx   - cloudera supergroup          0 2016-03-09 18:04 /user/hive/warehouse/departments
drwxrwxrwx   - cloudera supergroup          0 2016-03-04 08:48 /user/hive/warehouse/kingjames
drwxrwxrwx   - impala   supergroup          0 2016-03-04 08:39 /user/hive/warehouse/merged
drwxrwxrwx   - cloudera supergroup          0 2016-03-09 18:04 /user/hive/warehouse/order_items
drwxrwxrwx   - cloudera supergroup          0 2016-03-09 18:05 /user/hive/warehouse/orders
drwxrwxrwx   - cloudera supergroup          0 2016-03-09 18:05 /user/hive/warehouse/products
drwxrwxrwx   - cloudera supergroup          0 2016-03-04 08:48 /user/hive/warehouse/shake
```

3. Check from the Hive that it can see all the tables.

```
[cloudera@quickstart ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-
log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> SHOW TABLES;
OK
apachelog
categories
```

```
customers
departments
kingjames
merged
order_items
orders
products
shake

hive> SELECT order_item_order_id, sum(order_item_quantity) AS
total_order_item_quantity FROM order_items GROUP BY order_item_order_id ORDER
BY total_order_item_quantity DESC LIMIT 10;
Query ID = cloudera_20160309181414_a8ca926c-c6a1-469f-bd61-e023b2f30ede
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
....................
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.97 sec   HDFS Read: 513123
HDFS Write: 1220606 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.33 sec   HDFS Read:
1225407 HDFS Write: 90 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 300 msec
OK
49405  24
49429  24
21920  24
24967  24
33283  23
43848  23
29272  22
56414  22
17455  22
56559  22
Time taken: 59.8 seconds, Fetched: 10 row(s)
```


```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.97 sec   HDFS Read: 513123 HDFS Write: 1220606 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.33 sec   HDFS Read: 1225407 HDFS Write: 90 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 300 msec
OK
49405   24
49429   24
21920   24
24967   24
33283   23
43848   23
29272   22
56414   22
17455   22
56559   22
Time taken: 59.8 seconds, Fetched: 10 row(s)
```


```
hive> SELECT order_item_order_id, count(order_item_id) AS total_order_items
FROM order_items GROUP BY order_item_order_id ORDER BY total_order_items DESC
LIMIT 10;
```

```
Query ID = cloudera_20160310194949_71bd8c1a-9261-4b30-b208-7d3cc32b5218
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
....................
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.72 sec   HDFS Read:
1137142 HDFS Write: 1220606 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.53 sec   HDFS Read:
1225383 HDFS Write: 80 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 250 msec
OK
41224  5
41226  5
41228  5
55262  5
41230  5
55259  5
55258  5
41232  5
55256  5
41241  5
Time taken: 59.784 seconds, Fetched: 10 row(s)
```


```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.72 sec   HDFS Read: 1137142 HDFS Write
: 1220606 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.53 sec   HDFS Read: 1225383 HDFS Write
: 80 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 250 msec
OK
41224   5
41226   5
41228   5
55262   5
41230   5
55259   5
55258   5
41232   5
55256   5
41241   5
Time taken: 59.784 seconds, Fetched: 10 row(s)
```


4. Start Hive metadata server and copy the "hive-site.xml" file for Spark to locate Hive tables.

```
[cloudera@quickstart ~]$ hiveserver2 &
[1] 2182

[cloudera@quickstart]$ sudo cp /etc/hive/conf/hive-site.xml /etc/spark/conf/
```

5. Do the query again from pyspark shell, using Spark DataFrame API and Spark Temporary Tables.

```
Using Python version 2.6.6 (r266:84292, Feb 22 2013 00:00:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>> from pyspark.sql import SQLContext, HiveContext
>>> sqlContext = HiveContext(sc)
>>> sqlContext.sql("show tables").show()
+-----------+-----------+
|  tableName|isTemporary|
+-----------+-----------+
|  apachelog|      false|
| categories|      false|
|  customers|      false|
|departments|      false|
|  kingjames|      false|
|     merged|      false|
|order_items|      false|
|     orders|      false|
|   products|      false|
|      shake|      false|
+-----------+-----------+

>>> sqlContext.sql("SELECT order_item_order_id, sum(order_item_quantity) AS
total_order_item_quantity FROM order_items GROUP BY order_item_order_id ORDER
BY total_order_item_quantity DESC LIMIT 10").show()
+-------------------+-------------------------+
|order_item_order_id|total_order_item_quantity|
+-------------------+-------------------------+
|              49405|                       24|
|              21920|                       24|
|              24967|                       24|
|              49429|                       24|
|              33283|                       23|
|              43848|                       23|
|              17455|                       22|
|              20277|                       22|
|              40757|                       22|
|              38523|                       22|
+-------------------+-------------------------+
```

```
>>> sqlContext.sql("SELECT order_item_order_id, sum(order_item_quantity) AS total_order_ite
m_quantity FROM order_items GROUP BY order_item_order_id ORDER BY total_order_item_quantity
 DESC LIMIT 10").show()
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-cdh5.5.0.jar!/shaded/parqu
et/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-pig-bundle-1.5.0-cdh5.5.0.jar!/shaded/p
arquet/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-hadoop-bundle-1.5.0-cdh5.5.0.jar!/shade
d/parquet/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-exec-1.1.0-cdh5.5.0.jar!/shaded/parquet/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-jdbc-1.1.0-cdh5.5.0-standalone.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]
+-------------------+------------------------+
|order_item_order_id|total_order_item_quantity|
+-------------------+------------------------+
|              49405|                      24|
|              21920|                      24|
|              24967|                      24|
|              49429|                      24|
|              33283|                      23|
|              43848|                      23|
|              17455|                      22|
|              20277|                      22|
|              40757|                      22|
|              38523|                      22|
+-------------------+------------------------+
```

```
>>> sqlContext.sql("SELECT order_item_order_id, count(order_item_id) AS
total_order_items FROM order_items GROUP BY order_item_order_id ORDER BY
total_order_items DESC LIMIT 10").show()
+-------------------+-----------------+
|order_item_order_id|total_order_items|
+-------------------+-----------------+
|               4831|                5|
|              10831|                5|
|               6031|                5|
|               3231|                5|
|               6231|                5|
|               3831|                5|
|               6831|                5|
|               4431|                5|
|               8031|                5|
|               8231|                5|
+-------------------+-----------------+
```

```
>>> sqlContext.sql("SELECT order_item_order_id, count(order_item_id) AS total_order_items FROM
order_items GROUP BY order_item_order_id ORDER BY total_order_items DESC LIMIT 10").show()
+-------------------+-----------------+
|order_item_order_id|total_order_items|
+-------------------+-----------------+
|               4831|                5|
|              10831|                5|
|               6031|                5|
|               3231|                5|
|               6231|                5|
|               3831|                5|
|               6831|                5|
|               4431|                5|
|               8031|                5|
|               8231|                5|
+-------------------+-----------------+
```

Transform to the DataFrame and do the query again.

```
>>> orderItems = sqlContext.sql("select * from order_items")
>>> orderItems.show(5)
```

```
>>> orderItems = sqlContext.sql("select * from order_items")
>>> orderItems.show(5)
+-------------+-------------------+--------------------+-------------------+-------------------+-----------------------+
|order_item_id|order_item_order_id|order_item_product_id|order_item_quantity|order_item_subtotal|order_item_product_price|
+-------------+-------------------+--------------------+-------------------+-------------------+-----------------------+
|            1|                  1|                 957|                  1|             299.98|                 299.98|
|            2|                  2|                1073|                  1|             199.99|                 199.99|
|            3|                  2|                 502|                  5|              250.0|                   50.0|
|            4|                  2|                 403|                  1|             129.99|                 129.99|
|            5|                  4|                 897|                  2|              49.98|                  24.99|
+-------------+-------------------+--------------------+-------------------+-------------------+-----------------------+
only showing top 5 rows
```

```
>>> orderItems.printSchema()
root
 |-- order_item_id: integer (nullable = true)
 |-- order_item_order_id: integer (nullable = true)
 |-- order_item_product_id: integer (nullable = true)
 |-- order_item_quantity: integer (nullable = true)
 |-- order_item_subtotal: float (nullable = true)
 |-- order_item_product_price: float (nullable = true)
```

```
>>>
orderItems.groupBy("order_item_order_id").sum("order_item_quantity").orderBy(de
sc("sum(order_item_quantity)")).show(10)
+-------------------+------------------------+
|order_item_order_id|sum(order_item_quantity)|
+-------------------+------------------------+
|              49405|                      24|
|              24967|                      24|
|              21920|                      24|
|              49429|                      24|
|              43848|                      23|
|              33283|                      23|
|              17455|                      22|
|              38523|                      22|
|              56559|                      22|
|              29272|                      22|
+-------------------+------------------------+
```

only showing top 10 rows

```
>>> orderItems.groupBy("order_item_order_id").sum("order_item_quantity").orderBy(desc("sum(order_it
em_quantity)")).show(10)
+-------------------+-----------------------+
|order_item_order_id|sum(order_item_quantity)|
+-------------------+-----------------------+
|              49405|                     24|
|              24967|                     24|
|              21920|                     24|
|              49429|                     24|
|              43848|                     23|
|              33283|                     23|
|              17455|                     22|
|              38523|                     22|
|              56559|                     22|
|              29272|                     22|
+-------------------+-----------------------+
only showing top 10 rows
```

```
>>>
orderItems.groupBy("order_item_order_id").count().orderBy(desc("count")).show(1
0)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               6031|    5|
|              14231|    5|
|               6231|    5|
|               2431|    5|
|               6831|    5|
|               3431|    5|
|               8031|    5|
|               4231|    5|
|               8231|    5|
|               9031|    5|
+-------------------+-----+
only showing top 10 rows
```

```
>>> orderItems.groupBy("order_item_order_id").count().orderBy(desc("count")).show(10)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               6031|    5|
|              14231|    5|
|               6231|    5|
|               2431|    5|
|               6831|    5|
|               3431|    5|
|               8031|    5|
|               4231|    5|
|               8231|    5|
|               9031|    5|
+-------------------+-----+
only showing top 10 rows
```

```
>>> orderItems.registerTempTable("OrderItems")
```

```
>>> sqlContext.sql("SELECT order_item_order_id, count(order_item_id) AS
total_order_items FROM order_items GROUP BY order_item_order_id ORDER BY
total_order_items DESC LIMIT 10").show()
+-------------------+-----------------+
|order_item_order_id|total_order_items|
+-------------------+-----------------+
|               4831|                5|
|              10831|                5|
|               6031|                5|
|               3231|                5|
|               6231|                5|
|               3831|                5|
|               6831|                5|
|               4431|                5|
|               8031|                5|
|               8231|                5|
+-------------------+-----------------+
```

```
>>> orderItems.registerTempTable("OrderItems")
>>> sqlContext.sql("SELECT order_item_order_id, count(order_item_id) AS total_order_items FROM order
_items GROUP BY order_item_order_id ORDER BY total_order_items DESC LIMIT 10").show()
+-------------------+-----------------+
|order_item_order_id|total_order_items|
+-------------------+-----------------+
|               4831|                5|
|              10831|                5|
|               6031|                5|
|               3231|                5|
|               6231|                5|
|               3831|                5|
|               6831|                5|
|               4431|                5|
|               8031|                5|
|               8231|                5|
+-------------------+-----------------+
```

To select the orders having exact the largest number of order items (instead of listing them in descending order), we have to do two queries. The first query will get the largest number. The second query will select the orders having this number of order items.

```
>>> orderItemsSorted =
orderItems.groupBy("order_item_order_id").count().orderBy(desc("count"))
>>> orderItemsSorted.show(10)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               6031|    5|
|              14231|    5|
|               6231|    5|
|               2431|    5|
|               6831|    5|
|               3431|    5|
|               8031|    5|
|               4231|    5|
|               8231|    5|
|               9031|    5|
+-------------------+-----+
```

```
only showing top 10 rows

>>> orderItemsSorted.filter("count = 5").show(10)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               2431|    5|
|               3231|    5|
|               3431|    5|
|               3831|    5|
|               4231|    5|
|               4431|    5|
|               4831|    5|
|               6031|    5|
|               6231|    5|
|               6831|    5|
+-------------------+-----+
only showing top 10 rows
```


```
>>> orderItemsSorted.filter("count = 5").show(10)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               2431|    5|
|               3231|    5|
|               3431|    5|
|               3831|    5|
|               4231|    5|
|               4431|    5|
|               4831|    5|
|               6031|    5|
|               6231|    5|
|               6831|    5|
+-------------------+-----+
only showing top 10 rows
```


```
>>> orderItemsSorted.filter("count = 5").count()
11353
```

DataFrame can register to table and do the query again.

```
>>> orderItemsSorted.registerTempTable("OrderItemsSortedTbl")
>>> sqlContext.sql("SELECT * FROM OrderItemsSortedTbl WHERE count =
5").show(10)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               2431|    5|
|               3231|    5|
|               3431|    5|
|               3831|    5|
|               4231|    5|
|               4431|    5|
|               4831|    5|
|               6031|    5|
|               6231|    5|
|               6831|    5|
```

```
+-------------------+-----+
only showing top 10 rows
```

```
>>> orderItemsSorted.registerTempTable("OrderItemsSortedTbl")
>>> sqlContext.sql("SELECT * FROM OrderItemsSortedTbl WHERE count = 5").show(10)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               2431|    5|
|               3231|    5|
|               3431|    5|
|               3831|    5|
|               4231|    5|
|               4431|    5|
|               4831|    5|
|               6031|    5|
|               6231|    5|
|               6831|    5|
+-------------------+-----+
only showing top 10 rows
```

```
>>> orderItemsSorted.count()
57431
>>> sqlContext.sql("SELECT count(*) AS count FROM OrderItemsSortedTbl WHERE
count = 5").show(10)
+-----+
|count|
+-----+
|11353|
+-----+
```

We have 57431 records in total. Only 11353 of them have 5 order items.

```
>>> orderItemsSorted.count()
57431
>>> sqlContext.sql("SELECT count(*) AS count FROM OrderItemsSortedTbl WHERE count = 5").show(10)
+-----+
|count|
+-----+
|11353|
+-----+
```

6. Final results for running “Assign06_P4.py”.

```
[cloudera@quickstart hw06]$ spark-submit Assign06_P4.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/Static
LoggerBinder.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]

Orders with the largest number of order item quantities (Directly query from Hive table)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/jars/parquet-format-2.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-pig-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/org/s
lf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/parquet-hadoop-bundle-1.5.0-cdh5.5.0.jar!/shaded/parquet/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-exec-1.1.0-cdh5.5.0.jar!/shaded/parquet/org/slf4j/impl
/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/hive-jdbc-1.1.0-cdh5.5.0-standalone.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]
+-------------------+------------------------+
|order_item_order_id|total_order_item_quantity|
+-------------------+------------------------+
|              49405|                      24|
|              21920|                      24|
|              24967|                      24|
|              49429|                      24|
|              33283|                      23|
|              43848|                      23|
|              17455|                      22|
|              20277|                      22|
|              40757|                      22|
|              38523|                      22|
+-------------------+------------------------+
```

```
Orders with the largest number of order items per order (Directly query from Hive table)
+-------------------+-----------------+
|order_item_order_id|total_order_items|
+-------------------+-----------------+
|               4831|                5|
|              10831|                5|
|               6031|                5|
|               3231|                5|
|               6231|                5|
|               3831|                5|
|               6831|                5|
|               4431|                5|
|               8031|                5|
|               8231|                5|
+-------------------+-----------------+
```

```
Orders with the largest number of order item quantities (use Spark DataFrame API)
+-------------------+-----------------------+
|order_item_order_id|sum(order_item_quantity)|
+-------------------+-----------------------+
|              49405|                     24|
|              24967|                     24|
|              21920|                     24|
|              49429|                     24|
|              43848|                     23|
|              33283|                     23|
|              17455|                     22|
|              38523|                     22|
|              56559|                     22|
|              29272|                     22|
+-------------------+-----------------------+
only showing top 10 rows


Orders with the largest number of order items per order (use Spark DataFrame API)
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               6031|    5|
|              14231|    5|
|               6231|    5|
|               2431|    5|
|               6831|    5|
|               3431|    5|
|               8031|    5|
|               4231|    5|
|               8231|    5|
|               9031|    5|
+-------------------+-----+
only showing top 10 rows
```

```
+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               2431|    5|
|               3231|    5|
|               3431|    5|
|               3831|    5|
|               4231|    5|
|               4431|    5|
|               4831|    5|
|               6031|    5|
|               6231|    5|
|               6831|    5|
+-------------------+-----+
only showing top 10 rows
```

```
Orders with the largest number of order items per order (Use Spark Temporary Tables)
+-------------------+-----------------+
|order_item_order_id|total_order_items|
+-------------------+-----------------+
|               4831|                5|
|              10831|                5|
|               6031|                5|
|               3231|                5|
|               6231|                5|
|               3831|                5|
|               6831|                5|
|               4431|                5|
|               8031|                5|
|               8231|                5|
+-------------------+-----------------+

+-------------------+-----+
|order_item_order_id|count|
+-------------------+-----+
|               2431|    5|
|               3231|    5|
|               3431|    5|
|               3831|    5|
|               4231|    5|
|               4431|    5|
|               4831|    5|
|               6031|    5|
|               6231|    5|
|               6831|    5|
+-------------------+-----+
only showing top 10 rows
```