**HU Extension Assignment 04 E63 Big Data Analytics**

Issued on: Feb. 18, 2017 Due on Saturday by 9:30AM EST, Feb. 25, 2017

You can do these problems in any of 4 languages: Python, Scala, Java or R.

**Problem 1.** Go to an online newspaper and select two articles in English on two new movies featured in theaters this spring 2017.

Save those articles as .txt files and then import them into two Spark RDD objects, movieA and movieB.

All my comments and code will be in blue color to make a difference with the original document code. It is stated in Piazza to add to this document snippets of code, not the complete code, but my scripts are not that large, and seem quite easy to go through them, so I have included the complete source code

I have selected 2 recent films. Those films are:

The Lego Batman Movie as movieA (the-lego-batman-movie.txt)

Fifty Shades Darker as movieB (fifty-shades-darker.txt)

I upload the 2 files to Jupyter IDE

Use Spark transformation and action functions to transform those initial RDD-s into RDD-s that contain words and numbers of occurrence of those words in respective article.

First of all, note that I’m executing python scripts through Jupyter notebooks

|  |
| --- |
| from pyspark import SparkContext, SparkConf  conf = SparkConf().setAppName("Spark Count")  sc = SparkContext(conf=conf)  lines = sc.textFile("file:///home/jovyan/work/the-lego-batman-movie.txt")  lines2 = sc.textFile("file:///home/jovyan/work/fifty-shades-darker.txt")  words = lines.flatMap(lambda line: line.split(" "))  words2 = lines.flatMap(lambda line: line.split(" "))  wordCount = words.map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2)  wordCount2 = words2.map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2) |

Eliminate from both lists so called “stop words”. Take the list of stopwords from this Web page: <http://www.lextek.com/manuals/onix/stopwords1.html>.

I create stopwords.txt file to remove the so called “stop words”. I upload the file to Jupyter

|  |
| --- |
| lines3= sc.textFile("file:///home/jovyan/work/stopwords.txt")  words3 = lines3.flatMap(lambda line: line.split(" "))  wordCount3 = words3.map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2)  list3 = wordCount3.collect() |

Next, combining the previous scripts will remove “stop words” from the two articles, to obtain 2 RDDs with number of words without considering those words included in the stopwords list.

|  |
| --- |
| #movies and stopword files  movie = sc.textFile("file:///home/jovyan/work/the-lego-batman-movie.txt")  movie2 = sc.textFile("file:///home/jovyan/work/fifty-shades-darker.txt")  stop= sc.textFile("file:///home/jovyan/work/stopwords.txt")  # movie rdds by words  movieWords = movie.flatMap(lambda line: line.split(" "))  movieWords2 = movie2.flatMap(lambda line: line.split(" "))  #stopwords list  stopWordsList = stop.flatMap(lambda line: line.split(" ")).collect()  # movie word count considering:  # "The" and "the" as same word, same for any others  # Considered only words that are not included in the stopwords list  movieWordsCount = movieWords.filter(lambda word: word.lower() not in stopWordsList).\  map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2)  movieWordsCount2 = movieWords2.filter(lambda word: word.lower() not in stopWordsList).\  map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2) |

List for us 10 most frequent words in each RDD

To get this, we need to sort the RDDs, using the below code and then take the first ten element of the RDD

|  |
| --- |
| movieWordsCount = movieWords.filter(lambda word: word.lower() not in stopWordsList).\  map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2).sortBy(lambda x: x[1],ascending=False)  movieWordsCount2 = movieWords2.filter(lambda word: word.lower() not in stopWordsList).\  map(lambda word: (word, 1)).reduceByKey(lambda v1,v2:v1 +v2).sortBy(lambda x: x[1],ascending=False)  print movieWordsCount.take(10)  print movieWordsCount2.take(10) |

And the result:

[(u'batman', 57), (u'joker', 30), (u'(voice', 21), (u'city', 12), (u'robin', 11), (u'barbara', 10), (u'alfred', 9), (u'phantom', 9), (u'villains', 8), (u'dick', 6)]

------------------------------------------------------------------------------------------------------------------------

[(u'ana', 45), (u'christian', 33), (u'jack', 11), (u"christian's", 11), (u'leila', 8), (u'elena', 7), (u'woman', 5), (u"ana's", 5), (u'him.', 4), (u'takes', 4)]

Subsequently create RDD-s that contain only words unique for each of text.

Similarly, we have to modify our source code according to what requested to consider to new lists for each film, and remove from each of the 2 movies what in the other movie’s list

|  |
| --- |
| # movie unique word count considering:  # "The" and "the" as same word, same for any others  # Considered only words that are not included in the stopwords list  MovieList = movieWords.collect()  MovieList2 = movieWords2.collect()  movieWordsUnique = movieWords.filter(lambda word: word.lower() not in stopWordsList and word.lower() not in MovieList2)  movieWordsUnique2 = movieWords2.filter(lambda word: word.lower() not in stopWordsList and word.lower() not in MovieList) |

Finally create an RDD that contains only the words common to both texts.

This is quite more the same, just needed to change the filter accordingly

|  |
| --- |
| # movie word count considering words that are in both movies  # "The" and "the" as same word, same for any others  # Considered only words that are not included in the stopwords list  movieWordsBoth = movieWords.filter(lambda word: word.lower() not in stopWordsList  and word.lower() in MovieList2)  print movieWordsBoth.collect() |

The result for the above code:

[u'friend', u'set', u'Man,', u'tries', u'watch', u'reminds', u'him,', u'looks', u'returns', u'time', u'again.', u'family.', u'reminds', u'goes', u'watches', u'news', u'sent', u'watching', u'approached', u'family.', u'him,', u'care', u'goes', u'terms.', u'up,', u'suspicious', u'suspicious', u'tells', u'it.', u'tries', u'tells', u'time', u'join', u'it.', u'goes', u'arrive', u'goes', u'party', u'leaves', u'it.', u'head', u'confronts', u'met', u'meet', u'tells', u'streets', u'agrees', u'join', u'help.', u'romantic', u'start', u'leading', u'outside', u'leave', u'join', u'him.', u'takes', u'him.', u'comes', u'meaning', u'takes', u'home,', u'it.', u'takes', u'sends', u'care', u'him.', u'again.', u'sends', u'sent', u'them.', u'sent', u'goes', u'break', u"It's", u'help.', u'together.', u'tells', u'cares', u'decides', u'head', u'start', u'them.', u'family.', u'family', u'written']

Ine latest RDD preserve numbers of occurrences in two articles. In other words a row in your RDD will look like (actor 45 32).

|  |
| --- |
| # words in the 2 movies with word count on each of them in same RDD  # "The" and "the" as same word, same for any others  # Considered only words that are not included in the stopwords list  movieWordsCount = movieWords.filter(lambda word: word.lower() not in stopWordsList).\  map(lambda word: (word.lower(), 1)).reduceByKey(lambda v1,v2:v1 +v2)  movieWordsCount2 = movieWords2.filter(lambda word: word.lower() not in stopWordsList).\  map(lambda word: (word.lower(), 1)).reduceByKey(lambda v1,v2:v1 +v2)  #put together the 2 movies RDDs  movieWordsCountBoth = movieWordsCount.join(movieWordsCount2)  print movieWordsCountBoth.collect() |

The result for the above code:

[(u'kate', (1, 1)), (u'up,', (1, 1)), (u'sent', (3, 1)), (u'streets', (1, 1)), (u'approached', (1, 1)), (u'terms.', (1, 1)), (u'him,', (2, 1)), (u'goes', (5, 2)), (u'met', (1, 1)), (u'meet', (1, 1)), (u'leaves', (1, 1)), (u'family', (1, 2)), (u'join', (3, 1)), (u'arrive', (1, 1)), (u'tries', (2, 2)), (u'looks', (1, 1)), (u'outside', (1, 1)), (u'it.', (4, 1)), (u'leading', (1, 1)), (u'care', (2, 1)), (u'family.', (3, 1)), (u'returns', (1, 2)), (u'friend', (1, 1)), (u'meanwhile,', (1, 3)), (u'later,', (1, 1)), (u'suspicious', (2, 1)), (u'tells', (4, 2)), (u'agrees', (1, 1)), (u'written', (1, 1)), (u'watching', (1, 1)), (u'watch', (1, 1)), (u'set', (1, 1)), (u'decides', (1, 1)), (u'man,', (1, 1)), (u'news', (1, 2)), (u'comes', (1, 1)), (u'together.', (1, 1)), (u"it's", (1, 1)), (u'home,', (1, 4)), (u'watches', (1, 1)), (u'leave', (1, 3)), (u'help.', (2, 1)), (u'party', (1, 1)), (u'break', (1, 1)), (u'them.', (2, 1)), (u'him.', (3, 4)), (u'sends', (2, 1)), (u'confronts', (1, 1)), (u'takes', (3, 4)), (u'cares', (1, 1)), (u'start', (2, 2)), (u'head', (2, 1)), (u'reminds', (2, 1)), (u'romantic', (1, 1)), (u'again.', (2, 1)), (u'time', (2, 1)), (u'meaning', (1, 1))]

List for us a random samples containing 5% of words in the final RDD. We are just practicing RDD transformations and actions. You could implement this problem in a command shell or as a standalone program.

To obtaing 5% of total works, we have to do the calculation based on RDD count. If number is decimal we round it, which means if 2.85, it will be rounded to 3, and so 3 values will be returned

|  |
| --- |
| movieWordsCountBoth.takeSample(False, int(round(movieWordsCountBoth.count()\*5.0/100.0))) |

**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.

Create a RDD witth file’s content

|  |
| --- |
| from pyspark import SparkContext, SparkConf  conf = SparkConf().setAppName("Spark Count")  sc = SparkContext(conf=conf)  lines = sc.textFile("file:///home/jovyan/work/emps-1.txt") |

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.

|  |
| --- |
| Emps\_fields = lines.map(lambda x: (x.split(", ")[0], x.split(", ")[1], x.split(", ")[2]) ) |

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.

|  |
| --- |
| from pyspark.sql import Row  employees = emps\_fields.map(lambda e: Row(name = e[0], age = int(e[1]), salary = float(e[2])))  print employees.collect() |

This is the resulting RDD:

[Row(age=29, name=u'Michael', salary=3000.3), Row(age=30, name=u'Andy', salary=2500.25), Row(age=19, name=u'Justin', salary=4000.99)]

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 schema of your new data frame.

|  |
| --- |
| from pyspark.sql import SQLContext  sqlContext = SQLContext(sc)  empsDF = sqlContext.createDataFrame(employees)  empsDF.show() |

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

|age| name| salary|

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

| 29|Michael| 3000.3|

| 30| Andy|2500.25|

| 19| Justin|4000.99|

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

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

|  |
| --- |
| empsDF.registerTempTable("employees")  tempTable = sqlContext.sql("SELECT \* FROM EMPLOYEES WHERE SALARY > 3500")  tempTable.show() |

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

|age| name| salary|

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

| 19|Justin|4000.99|

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

Persist your DataFrame as a Parquet file and show that you could exit your pyspark or spark-shell shell, come back into the shell and are able to read the data from the parquet file and recreate the same DataFrame you had originally.

|  |
| --- |
| empsDF.select("age","name", "salary").write.save("employees.parquet",format="parquet") |

Afterwards, since I am using Jupyter, I restart the kernel, to make all objects previously instantiated are removed from memory I try to show empsDF

|  |
| --- |
| empsDF.show() |

When doing it, I get the below error:

-------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-2-81796baa5d16> in <module>()----> 1 empsDF.show()

NameError: name 'empsDF' is not defined

Now I try to get it from the parquet file I saved before:

|  |
| --- |
| from pyspark import SparkContext, SparkConf  conf = SparkConf().setAppName("Spark Count")  sc = SparkContext(conf=conf)  from pyspark.sql import SQLContext  sqlContext = SQLContext(sc)  empsDF = sqlContext.read.parquet("employees.parquet")  parquetFile.show() |

And get the below result, corresponding to the previous DF:

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

|age| name| salary|

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

| 29|Michael| 3000.3|

| 30| Andy|2500.25|

| 19| Justin|4000.99|

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

So everything worked fine pulling out the data from the parquet file.

Implement this problem in pyspark of spark-shell or as a standalone program that will be submitted to spark-submit utility.

**Problem 3.** Make sure that two Hive services are up and running. You will find them in the same /etc/init.d directory you were looking for Hadoop services.

From CentOS /etc/init.d, we see the 2 hive services are running

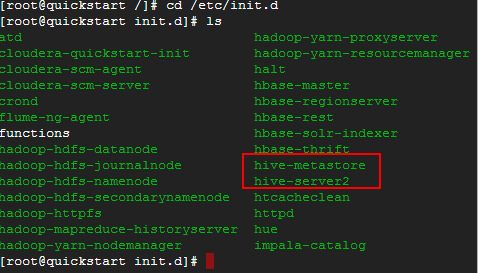


Figure - Running Linux Hive services

Click on the browser in your Cloudera QuickStart VM. The welcome screen will open. On the top navigation bar hit Hue. Hue is Cloudera’s browser for Hadoop products. Hue might complain that its configuration is not right. I believe that some user names are missing for some products we do not care right now.

I sign in with cloudera /cloudera and get into the Hue’s console

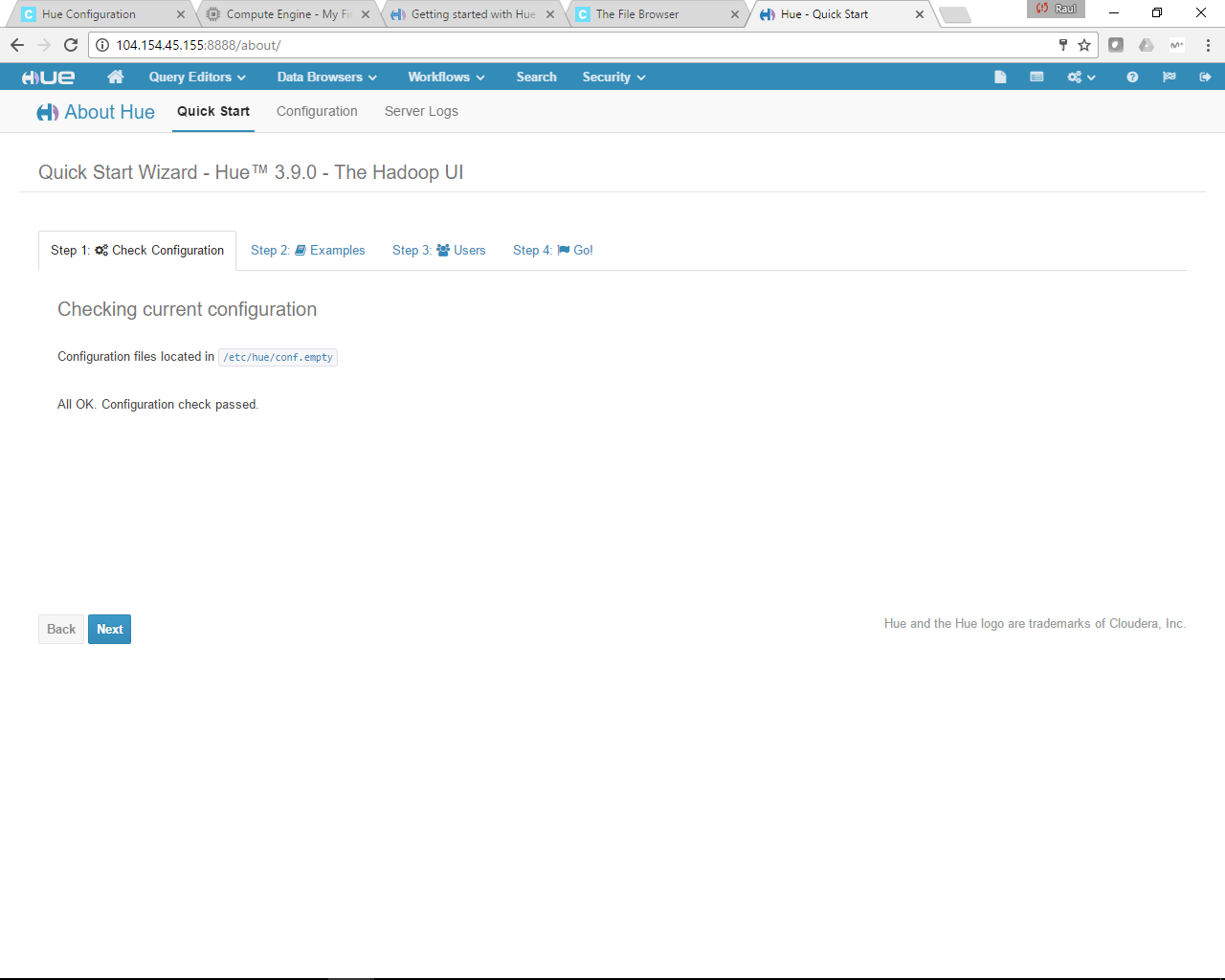


Figure - Launching Hue's front-end

On the new navigation bar that will appear select Query Editors and then select Hive.

To have the tables stated in the exercise Hive Examples have to be installed, otherwise default database remains empty

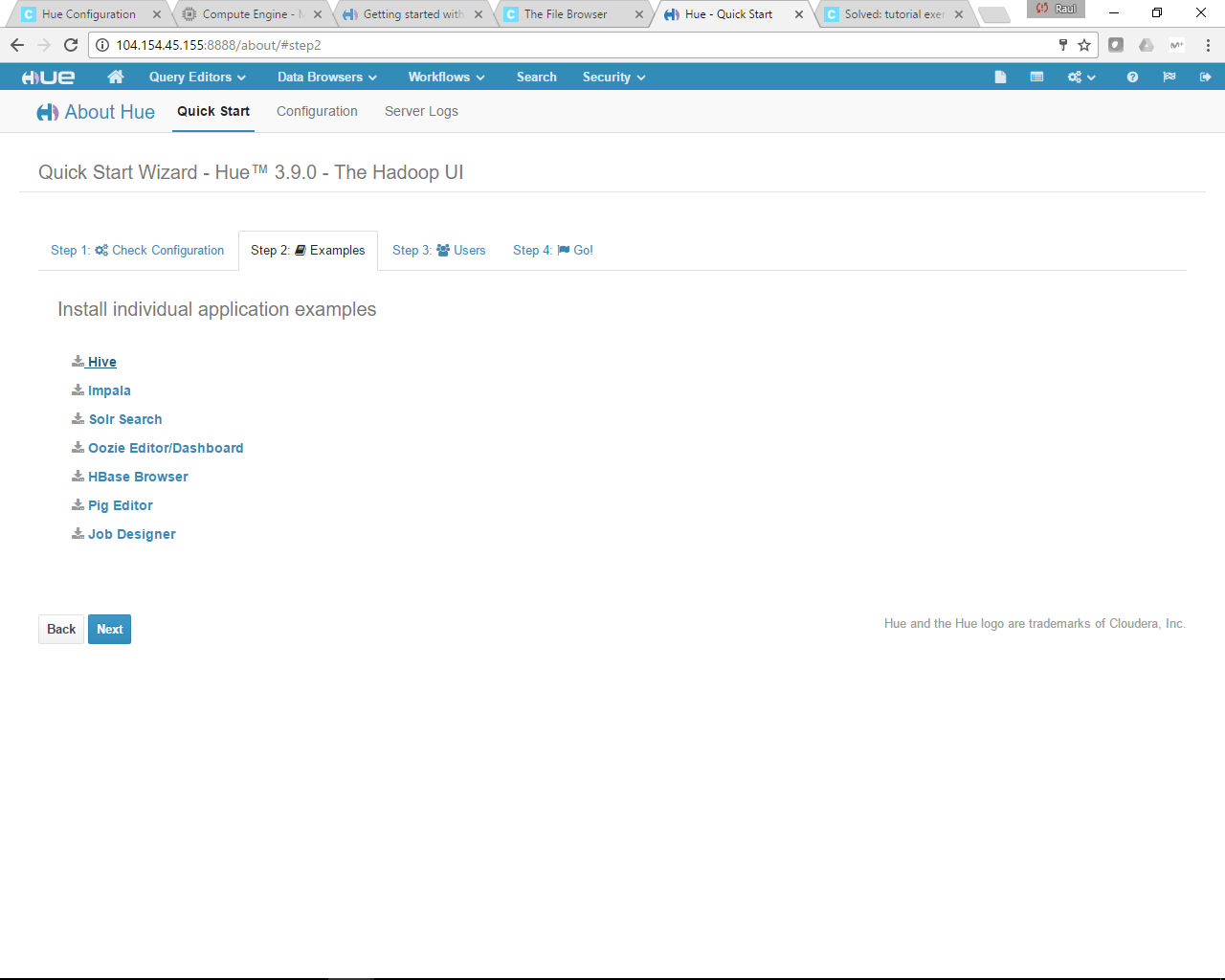


Figure - Install Hive samples to populate defautl DWH with tables

On the left navigation bar you will see a “default” database and several demo tables. In the query window to the right, type “select \* from customers” and then hit the green triangle left of the query window.

On the below image, the executed SQL and the result within Hue

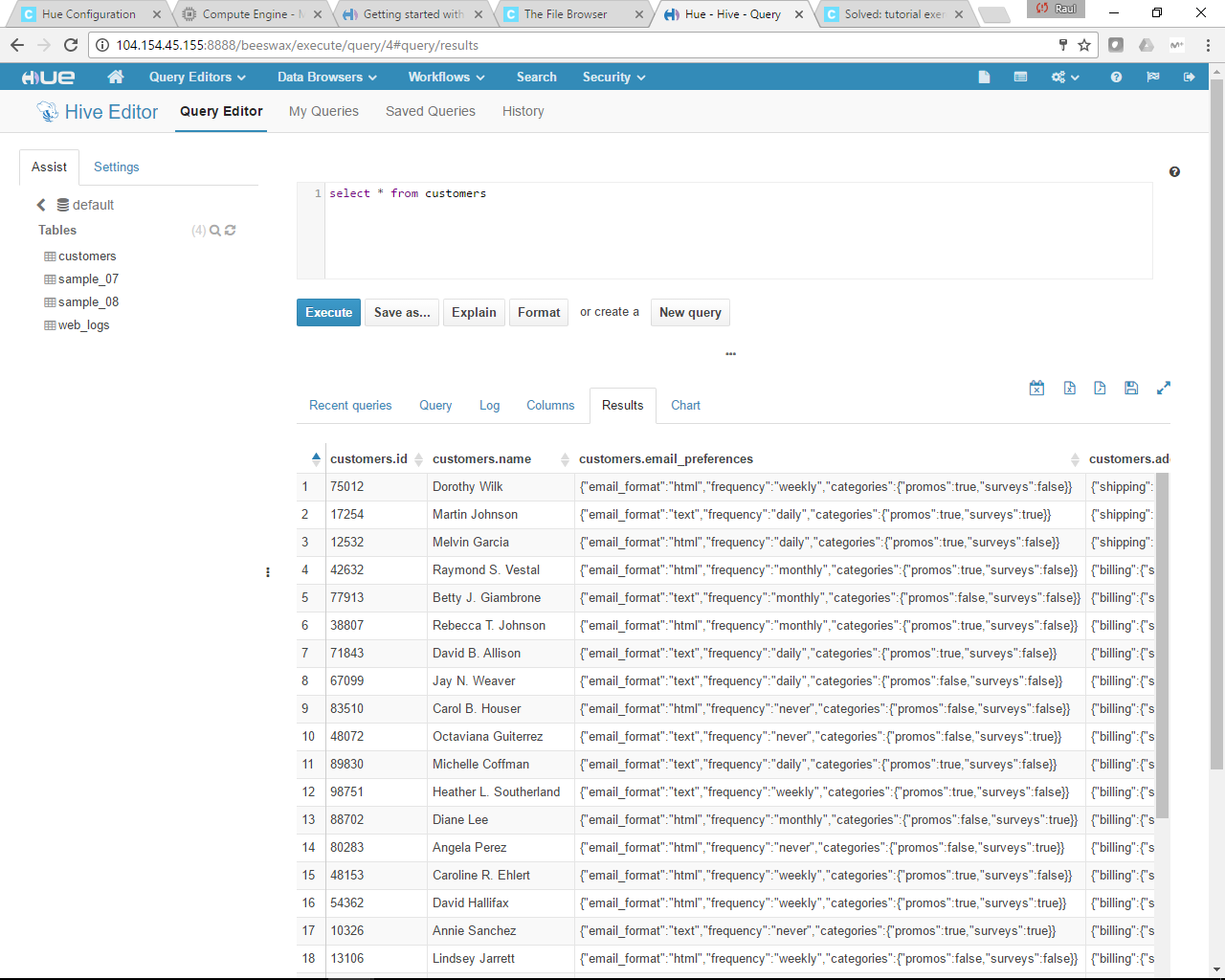


Figure - Select SQL form customers table

Next, type “select addresses from customers”. You will see that the customers have shipping and billing addresses.

Below image shows what requested:

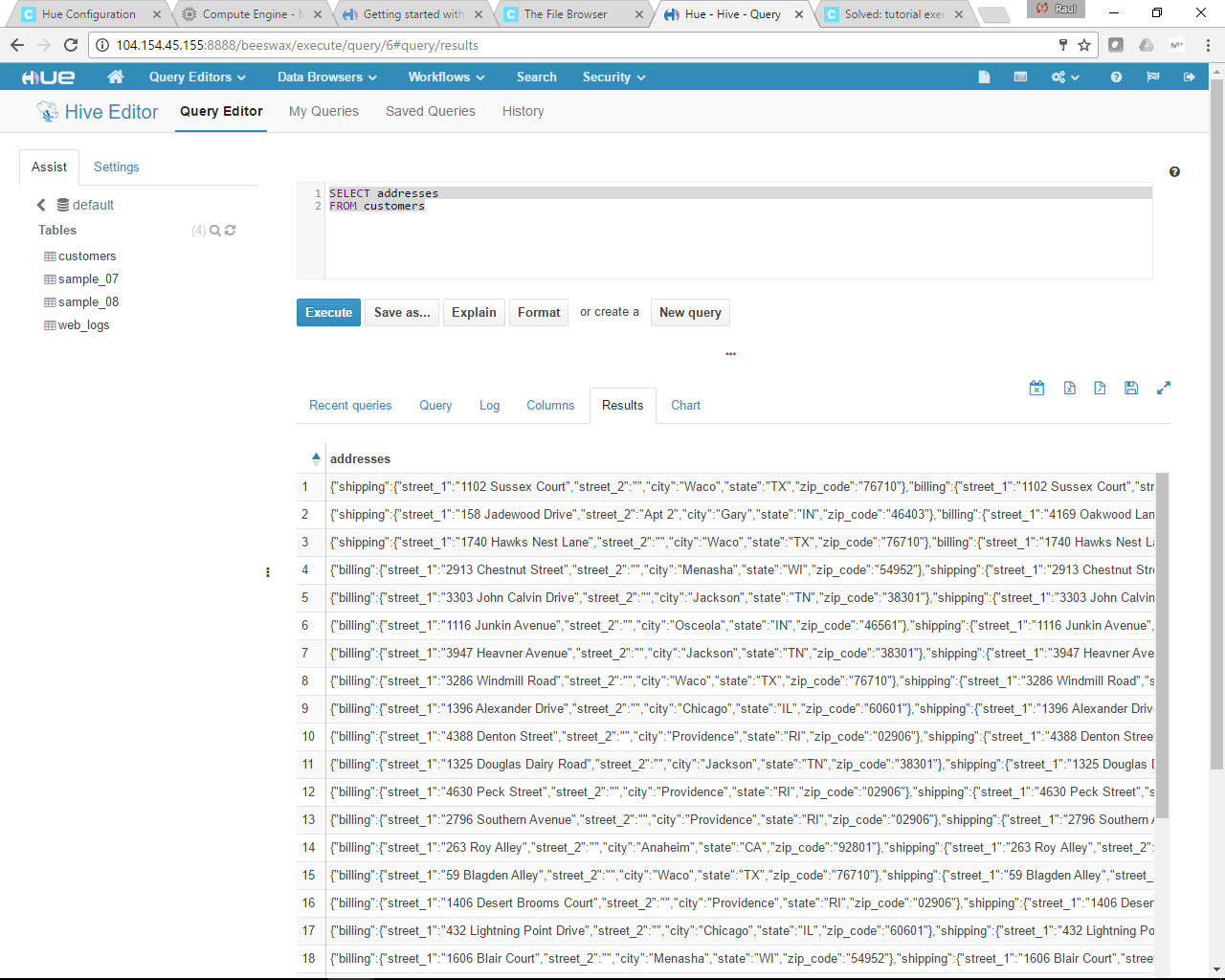


Figure - SQL select sentece from customers table for address field

Hive stores its table as Parquet files in HDFS. Use command:

$ sudo –u hdfs hadoop fs –ls /usr/hive/warehouse

to compare the content of Hive’s warehouse HDFS directory and the table list you see in Hue.

This is what returned using the previous command:

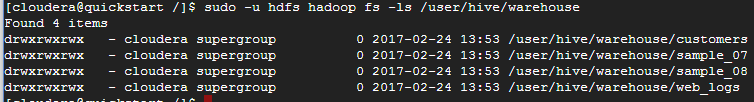


Figure - Default Hive DWH content from HDFS

Those files are exactly same names as those in Hue as in next image:

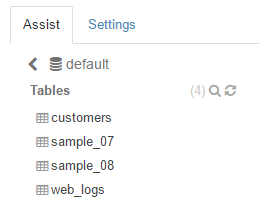


Figure - Default Hive datawarehouse content from Hive’s console

Next use the command:

$ sudo –u hdfs Hadoop fs –cat

to expose the actual content of the file stored for the customer table. The content is half readable. That is fine. Capture a portion of it and present in your solution

The complete sentence to run what requested is:

|  |
| --- |
| sudo -u hdfs hadoop fs -cat /user/hive/warehouse/customers/customers |

The content returned is as follows:

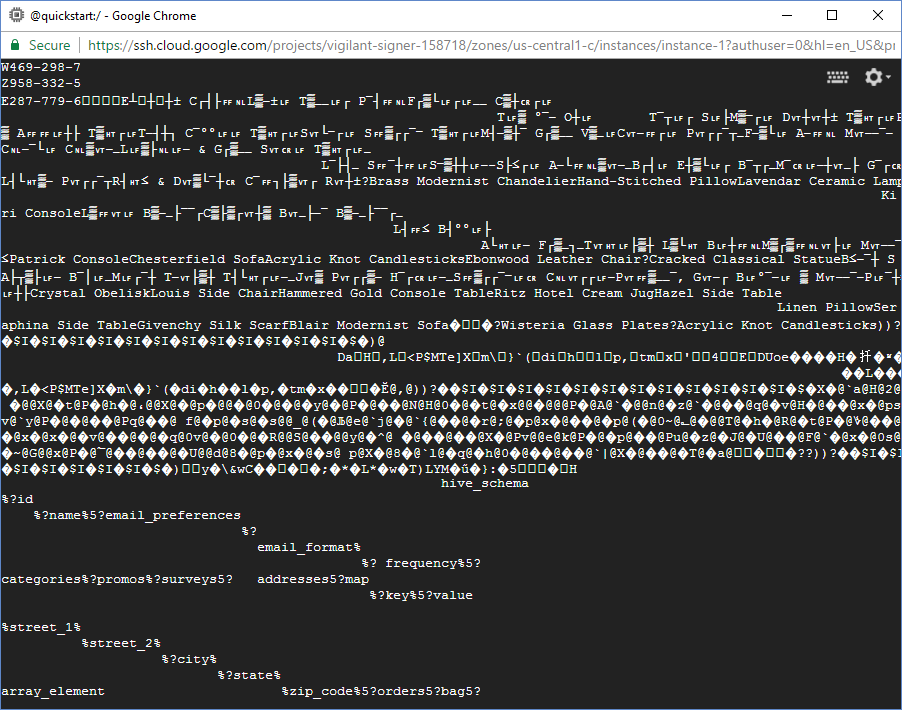


Figure - Customers table Parquet file content

|  |
| --- |
| 287-779-6�E┴␊┼␋┼± C┌┤├␌␤L▒⎼±␊ T▒⎽⎽␊┌ P⎺┤␌␤F┌▒└␊┌␊⎽⎽ C▒┼␍┌␊ T␊▒ °⎺⎼ O┼␊ T⎺┬␊┌ S␊├M▒⎻┌␊ D␋┼␋┼± T▒␉┌␊P▒┌⎺└▒ A␌␌␊┼├ T▒␉┌␊T⎼┤┼┐ C⎺°°␊␊ T▒␉┌␊S␋└⎻┌␊ S␌▒┌┌⎺⎻ TC␤⎼⎺└␊ C␤▒␋⎼⎽L␊▒├␤␊⎼ & G┌▒⎽⎽ S␋␍␊ T▒␉┌␊⎽ A⎼␌␤ M␋⎼⎼⎺⎼ T▒⎽┐ L▒└⎻D▒┼␋⎽␤ T␊▒┐ D␊⎽┐W␋⎽├␊⎼␋▒ G┌▒⎽⎽ P┌▒├␊⎽L┤└␉▒⎼ P␋┌┌⎺┬R┤␉≤ & D␋▒└⎺┼␍ C⎺␌┐├▒␋┌ R␋┼±?Brass Modernist ChandelierHand-Stitched PillowLavendar Ceramic Lamps⎼⎺⎼O▒┐ F⎼▒└␊ M␋⎼⎼⎺⎼ Kiri ConsoleL▒␌␋␊ B▒⎼⎽├⎺⎺┌C▒├▒┌␋┼▒ B␋⎽├⎼⎺ B▒⎼⎽├⎺⎺┌⎽ L┤␌≤ B┤°°␊├ A└␉␊⎼ F┌▒⎽┐⎽T␋␉␊├▒┼ L▒└␉ B␊┼␌␤M▒┌▒␌␤␋├␊ M␋⎼⎼⎺⎼␊␍ T⎼▒≤Patrick ConsoleChesterfield SofaAcrylic Knot CandlesticksEbonwood Leather Chair?Cracked ClassicA├┬▒├␊⎼ B⎺│␊⎽M␊┌⎺┼ T⎼␋├▒┼ T┤└␉┌␊⎼⎽J␋▒ P␋┌┌▒⎼ H⎺┌␍␊⎼⎽S␌▒┌┌⎺⎻␊␍ C␤␋┌┌␊⎼P␋␌▒⎽⎽⎺, G␋⎼┌ B␊°⎺⎼␊ ▒ M␋⎼⎼⎺⎼P␊⎺┼≤ A⎼⎼▒┼±␊└␊┼├Crystal ObeliskLouis Side ChairHammered Gold Console TableRitz Hotel Cream JugHazel Side Table Linen PillowSeraphina Side TableGivenchy Silk ScarfBlair Modernist Sofa��?Wisteria Glass Plates?Acrylic Knot Candlesticks))?��$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$�)@ DaH�,L�<P$MTe]X�m\�}`(�di�h��l�p,�tm�x�'��4�E�DUօe����H�扦�ʶ� ��L���z�,L�<P$MTe]X�m\�}`(�di�h��l�p,�tm�x���Ĕ@,@))?��$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$�X�@`a@H@2@1@Ĕ@�h@��@�@`~@ �@@X@�t@P�@h�@،@@X@�@p�@@�@0�@�@�y@�@P�@��@N@H@0�@�t@�x@@�@@@P�@A@`�@@n@�z@`�@��@q@�v@H�@��@x�@ps@��@�}@�y@�\@�v@`y@P�@�@��@Pq@��@ f@�p@�s@�s@@\_@(�@Љ@e@`j@�@`{@��@�r@;@�p@x�@��@�p@(�@0~@؎@�@@T@�h�@R@�t@P�@؇@��@p@,@0~@H��@?��@x�@x�@�v@��@�@�q@0v@�@0�@�R@@S@��@@y@�^@ �@��@��@X�@Pv@@e@k@P�@�p@��@Pu@�z@�J@�U@��@F@`�@x�@0s@pu@@q@�z@�Q@�k@�~@G@@x@P�@؅@��@��@�U@@d@8�@p�@x�@�s@ p@X�@8�@`l@�q@�h@0�@��@��@`|@X�@��@�T@�a@��??))?��$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$I�$�)y�\&wC����;�\*�L\*�w�T)LYM�ű�}:�5�­H hive\_schema%?id |

What type of storage format is used for that file? You can see that “meta information in the file itself.

Default storage format is parquet. It can be seen in the file that table content is stored as plain text, since we can read customer names, shipping addresses, etc. Additionally other characters seem to be kind of binary format not human readable.

**Problem 4.** We will explore that customers Hive table in Spark. Follow instructions on slide 62 of lecture notes and create a HiveContext object. Use that object to transfer the data in Hive’s customers table into a Spark DataFrame object. Use that dataFrame to tell us how many shipping addresses there are in each US state. Implement this problem in pyspark of spark-shell or as a standalone program that will be submitted to spark-submit utility.

First of all, we need to make spark aware of Hive

|  |
| --- |
| hiveserver2 &  cd /etc/hive/conf  cp hive-site.xml /etc/spark/conf |

Then we transfer customers table to a DataFrame. We launch py-sparkwithin and then we type the following python commands:

|  |
| --- |
| hivecontext = HiveContext(sc)  dfs = hivecontext.sql("select \* from customers") |

Show() command list the first 20 dataframe’s rows

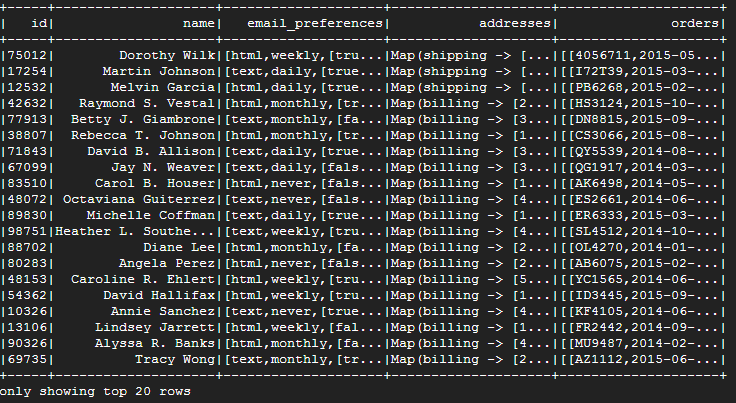


Figure -Top 20 DF rows for customers table

Finally, we can query the dataframe to get what requested:

|  |
| --- |
| dfs.groupBy("addresses.shipping.state").count().show() |

This returns the number of shipping addresses grouped by state:

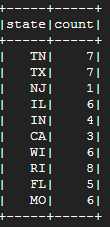


Figure - Shpping locations count grouped by State

Each problem carries 25% of the grade.

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 separate copy of the 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 include in your MS Word document only relevant portion of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. 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.