Purdue University
Fall 2016
CS 348: Information Systems
Project 4: SQL on Spark
Due: 11:59 pm on November 30, 2016

/**********
* Overview
**********/

This project will show you how to execute SQL queries on Apache Spark.

/**********
* Environment and Setup
**********/

We will use Purdue's OpenStack cluster for this assignment. The master node for this cluster is 172.18.11.251. You can SSH into the master node with credentials username:[PurdueID]_ostack, password:[PurdueID]_ostackpwd. (By PurdueID, we mean the username that you use to login to MyPurdue, Blackboard, etc.) If you are connecting from somewhere off campus, you may have to set up a VPN connection first. Or you can ssh to another node which is visible off campus (e.g. data.cs.purdue.edu) and then from there ssh to the cluster

Be sure to change your password after you log in.
```
$ passwd
```

To see a list of all the nodes in the cluster, run
```
$ cat /etc/hosts
```

CAUTION: This cluster is temporary. It will be wiped after the lab is graded. If you have any codes or results that you wish to save, move them to permanent storage on another system.

NOTE: This cluster does not mount the CS Department's NFS shared file system, so your CS home directory is not available.

You will each have your own directory in HDFS where you will have permissions to read and write data. To view the contents of your personal HDFS directory (which is initially empty), use:

```
$ hadoop fs -ls -R /user/[username] | grep -v Trash
```

You can take a look at common HDFS commands at
https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html

```
/**********
* Dataset
**********/
```

Movie Lens 1M dataset, taken from GroupLens Research:
http://grouplens.org/datasets/movielens/1m/

Tables:

```
     users.dat      ( UserID::Gender::Age::Occupation::Zip-code )
     movies.dat     ( MovieID::Title::Genres )
     ratings.dat    ( UserID::MovieID::Rating::Timestamp )
```

```
/***********************
* Uploading Files to HDFS
************************/
```

To pull the data and place it into a directory

```
$ wget http://files.grouplens.org/datasets/movielens/ml-1m.zip
$ unzip ml-1m.zip
```

This will place the above listed files into ml-1m directory

NOTE: Some of the titles in movies.dat contain characters that are not UTF-8. These characters may compromise your results. So, as is often the case in the real world, we must put our data through a "cleaning" phase before we query it. Use the following bash command to remove all non-UTF-8 characters from the movies.dat file before you load it into HDFS:

```
$ iconv -f utf-8 -t utf-8 -c movies.dat > tmp.dat; mv tmp.dat
movies.dat;
```

To upload the data into HDFS, use following steps

1. Create the directory in HDFS to place your data files, <username> is the username you use to SSH into the openstack cluster (<purdue_email_id>_ostack)

```
$ hadoop fs –mkdir /user/<username> {If doesn’t exist already}
$ hadoop fs –mkdir /user/<username>/sparksql
$ hadoop fs –mkdir /user/<username>/sparksql/input
```

2. Move your data files to the HDFS directory

```
$ cd ml-1m
$ hadoop fs -put *.dat /user/<username>/sparksql/input
```

3. Confirm that the files are moved into HDFS

```
$ hadoop fs -ls -R /user/<username>/sparksql | grep -v Trash
```

4. After executing your queries, to copy over the files back to local file system into a directory

```
$ mkdir –p outputdir
$ cd outputdir
$ hadoop fs –get /user/<username>/sparksql/output/*
```

5. To remove all created folder and  files from HDFS after everything is done

```
$ hadoop fs rm –r /user/<username>/*
```

```
/***********************
* Spark Environment
************************/
```

Spark provides APIs for Scala, Java, and Python. For this project we will use Python.

For an introduction to Spark, take a look at the following links.

http://spark.apache.org/docs/1.6.0/quick-start.html
http://spark.apache.org/docs/1.6.0/programming-guide.html
http://spark.apache.org/docs/1.6.0/sql-programming-guide.html
http://www.slideshare.net/LisaHua/spark-overview-37479609

NOTE: Be mindful of the version number whenever you are searching online for documentation. The version of Spark installed on the Openstack cluster is version 1.6.0. Spark is under heavy development, so if you are looking at documentation for another version there is a good chance that features will be changed or missing.

Spark provides a Python shell where you can submit commands interactively. This shell can be very useful when trying to learn the Spark interface. To start the shell, SSH to the master node and use:

```
$ pyspark
```

If you like, you can develop your queries in the shell first, and then copy your shell commands to a Python script to create a Python job. To distribute a Python job with Spark, use:

```
$ spark-submit --master yarn [job.py] [command_line_args]
```

```
/**********
* Example
**********/
```

We will provide you with the code to complete one query:
The code is present at
https://www.cs.purdue.edu/homes/clifton/cs348/q1.py

```
/* Query 1 (example query) */
/* Give the distinct title of each movie that received the rating 1 at
least once. */
SELECT DISTINCT m.title
FROM
     movies AS m
     JOIN ratings AS r ON m.movieid = r.movieid
WHERE r.rating = 1;
/* output */
101 Dalmatians (1996)
```

Note that this query requires us to join two tables.
To run the query, copy the file q1.py to your home directory on the master node and run the following command:

```
$ spark-submit --master yarn q1.py [input_dir] [output_dir]
e.g.
$ spark-submit —-master yarn q1.py
/user/<username>/sparksql/input /user/<username>/sparksql/output
```

```
/**********
* Task
**********/
```

We ask that you implement the following queries as Spark jobs. For each query, we demonstrate what a single line of output from your final job should look like.

```
/* Query 2 */
/* Give the distinct title and genre of each movie released in 2000
that received the rating 5 at least once. */
[SQL omitted]
/* output */
Almost Famous (2000)::Comedy|Drama
```

```
/* Query 3 */
/* For each occupation, compute the number of users with that
occupation. */
SELECT u.occupation, count(u.occupation) AS cnt
FROM users AS u
GROUP BY u.occupation;
/* output */
18::70
```

```
/* Query 4 */
/* Give the title and average rating for each movie released in 1998.
Round the average to the nearest tenth. */
[SQL omitted]
/* output */
After Life (1998)::4.1

/* Query 5 */
/* For each user age group, compute the average rating given by the
users in that age group. Round the average to the nearest tenth. */
[SQL omitted]*
/* output */
25::3.5
```

Because the Spark SQL library is well developed, some of these queries can be completed using a call to sqlContext.sql(). For other queries, you may have to do some post-query processing.

When writing your queries in Spark, take care that you do not use any Spark SQL keywordds as column names. This will give you an error.

NOTE: Queries 4 and 5 ask you to compute averages. Depending on how you compute your average, your solution may differ from our solution by ±0.1. There will not be any penalty for this.

```
/**********
* Deliverables
**********/
```

Note that it is not sufficient to simply execute your queries in the Spark shell and turn in your query results. You must collect your commands in a Python job. We should be able to launch your python job using:

```
$ spark-submit --master yarn [job.py] [path_to_input_dir] [output_dir]
*
```

To get full credit on the assignment, you must carefully follow the instructions below. Make sure to name your files as indicated. You should turn in your code and results file for Query 1 (the example query) and for Queries 2-5. (You may turn in the Query 1 code without modification.)

Create a directory with your Purdue ID as the directory name. (Here, by Purdue ID we mean your career account username. Do not use your openstack username.) In this directory, create five subdirectories, one for each query. Within each query directory, place the code for your Python job as well as a single file with your query results (retrieved from HDFS with -getmerge).

```
[PurdueID]_p4/
```

```
    q1/
        q1.py
        q1_results.txt
    q2/
      q2.py
      q2_results.txt
    ...
```

Create a zip file of the above folder, named [PurdueId]_p4.zip and submit the file to Blackboard.