7CCSMBDT

Coursework 2

1. **MongoDB queries :** Load the file championsleauge_1.json (available in KEATS) into a database cw2 and a collection cl. Write a single query, for each subtask 1 to 3 below. Solutions with more than one queries will receive a zero mark. It is ok if your query answers output "_id" too.
    a. Write a query that returns the name, number of followers, and number of friends, of each user with fewer than 25 friends, whose name starts with "A" (case insensitive) and ends with "es" (case sensitive). The results must be sorted in decreasing order of displayName.

```
db.cl.find({$and:[{"displayName":{$regex:/^A/,$options:'i'}},{"displayName":{$regex:/es$/}}],"friendsCount":{$lt:25}},{"displayName":1,"followersCount":1,"friendsCount":1}).sort({"displayName":-1})
```

```
{ "_id" : ObjectId("605b0f0d64fb43dd4b4c6fb3"), "displayName" : "angie torres", "friendsCount" : 23, "followersCount" : 33 }
```

```
{ "_id" : ObjectId("605b0f0d64fb43dd4b4c6fc3"), "displayName" : "angie torres", "friendsCount" : 23, "followersCount" : 33 }
```

```
{ "_id" : ObjectId("605b0f0d64fb43dd4b4c9f34"), "displayName" : "Arizona Companies", "friendsCount" : 0, "followersCount" : 10 }
```

```
{ "_id" : ObjectId("605b0f0d64fb43dd4b4c8f37"), "displayName" : "Adejies", "friendsCount" : 13, "followersCount" : 10 }
```

```
{ "_id" : ObjectId("605b0f0d64fb43dd4b4c9f73"), "displayName" : "Adejies", "friendsCount" : 13, "followersCount" : 10 }
```

```
{ "_id" : ObjectId("605b0f0e64fb43dd4b4ca47d"), "displayName" : "Adejies", "friendsCount" : 13, "followersCount" : 10 }
```

    b. Write a query that returns the average ratio between numbers of followers and number of friends, over all documents. Note that the number of friends (denominator in the ratio) must be >0 for the ratio to be defined.

```
db.cl.aggregate([{$group:{"_id":"_id",AverageRatio:{$avg:{$cond:{if:{$gt:["$followersCount",0]},then:{$divide:["$friendsCount","$followersCount"]},else:"$$REMOVE"}}}}}])
```

```
{ "_id" : "_id", "AverageRatio" : 2.9499109542008175 }
```

c. Write a query that returns the number of users who have at least 1000 friends, posted a tweet (the field verb has the value "post") and whose tweet (field body) contains the string “Madrid” (even as part of a word).

```
db.cl.find({"friendsCount":{$gte:1000},"verb":{$regex:".*post.*"},"body":{$rege
x:".*Madrid.*"}}).count()
```

```
185
```

2. **Apache Spark :** Download the file u.user from KEATS. This file has lines that correspond to users with the following attributes: user id, age, gender, occupation, zipcode. For example, the following part of u.user has 5 attribute values (namely 1, 24, M, technician, 85711): 1|24|M|technician|85711. Complete the given program, so that it outputs all occupations that are performed by users in the age group [40,50) and by users in the age group [50,60) and are among the 10 most frequent occupations for the users in each age group.

```
from pyspark import SparkContext
sc = SparkContext( 'local', 'pyspark')
#given age_group function
def age_group(age):
      if age < 10 :
        return '0-10'
      elif age < 20:
        return '10-20'
      elif age < 30:
        return '20-30'
      elif age < 40:
        return '30-40'
      elif age < 50:
        return '40-50'
      elif age < 60:
        return '50-60'
      elif age < 70:
        return '60-70'
      elif age < 80:
        return '70-80'
      else :
```

```
        return '80+'
#given parse_with_age_group function
def parse_with_age_group(data):
      userid,age,gender,occupation,zip = data.split("|")
      return userid, age_group(int(age)),gender,occupation,zip,int(age)
#separate lines according to groups
fs=sc.textFile('file:///home/cloudera/Desktop/Coursework2/u.user')
data_with_age_group=fs.map(parse_with_age_group)
#get the lines where age is between 40 and 50
data_with_age_40_50=data_with_age_group.filter(lambda x: '40-50' in x)
#get the lines where age is between 50 and 60
data_with_age_50_60=data_with_age_group.filter(lambda x: '50-60' in x)
#get the occupations where age is between 40 and 50
tmp=data_with_age_40_50.map(lambda x: x[3]).collect()
x = {}
c = []
#count the number of each occupation
for i in tmp:
    x[i] = x.get(i, 0) + 1
#sort the couting
b=sorted(x.items(), key=lambda kv: kv[1])
#data cleaning
for i in b:
      c.append(i[0])
#get the 10 most frequent occupations
top10_40_50 = c[-10:len(c)]
#get the occupations where age is between 50 and 60
tmp=data_with_age_50_60.map(lambda x: x[3]).collect()
x = {}
c = []
#count the number of each occupation
```

```
for i in tmp:
    x[i] = x.get(i, 0) + 1
#sort the couting
b=sorted(x.items(), key=lambda kv: kv[1])
#data cleaning
for i in b:
      c.append(i[0])
#get the 10 most frequent occupations
top10_50_60 = c[-10:len(c)]
#get the intersection of the two frequent lists
top10_40_50 = sc.parallelize(top10_40_50)
top10_50_60 = sc.parallelize(top10_50_60)
sol=top10_40_50.intersection(top10_50_60)
#output
print(sol.collect())
```

3. **HIVE :** Download the file query_logs.txt from KEATS. This file contains searches made by different users. Specifically, there are three tab-separated attributes user, time, and query. The user attribute may appear in many lines, meaning that the user has issued different queries (attribute query), at different times (attribute time). Create a database log_db and a table logs that has the attributes user, time, query, all of which are of type STRING The table must contain all records of the file query_logs.txt Note that the fields are terminated by '\t'.
    a. Write a query that returns the maximum number of visits of all users. Note that in each page visit a user may or may not issue a query.

```
select user,count(*) from logs group by user;
002BB5A52580A8ED  18
005BD9CD3AC6BB38  18
00A08A54CD03EB95  3
011ACA65C2BF70B2  5
01500FAFE317B7C0  15
0158F8ACC570947D  3
018FBF6BFB213E68  1
019E9463F6695963  10
01F6B9CA495576BA  7
```

```
027DCCE98A4B6E84   11
02A563E9CABE69F8   3
02BDAC74E8B09D09   5
02E76389CBC661F7   7
0333463FD50A0859   9
0338D63FFC24DC2F   3
035471C5E5343788   4
0375C7FE1C10CC2E   6
04699D2910C817B1   1
054340E4B8F63E34   2
0567639EB8F3751C   2
```

b. Write a query that returns all attributes of each user who issues a query that contains the string "job". Note that "job" may appear within a word (e.g.,"job" is part of "jobs").

```
select * from logs where query rlike 'job';
4077443B5801F0C3   970916182623 job openings
4077443B5801F0C3   970916182752 job openings listings
4077443B5801F0C3   970916182823 agricultural job listings
4077443B5801F0C3   970916182834 agricultural job listings employdogst
4077443B5801F0C3   970916182942 job listings
83607290B8BEAFC6   970916070440 jobs=hong kong
D5D8220D36969861   970916222708 job interview tips
D5D8220D36969861   970916224215 job interview tips
0E10DD8EB5EEB192   970916134219 jobs at the university of minnesota
567854C718273984   970916021936 part time jobs
567854C718273984   970916022012 part time jobs
567854C718273984   970916022043 part time jobs
567854C718273984   970916022117 part time jobs
567854C718273984   970916022202 part time jobs
567854C718273984   970916022313 home jobs
567854C718273984   970916022444 home jobs
```

c. Write a query that returns the number of distinct users who issue a (non-empty) query between 21:00:00 (inclusive) and 22:59:59 (inclusive). Note that the second attribute in query_logs.txt is in the form yyMMddHHmmss . For example, in 970916105432 yy=97, MM=09, dd=16, HH=10, mm=54, ss=32, which means that the query was issued in 1997, September, 16th, and at time 10:54:32.

```
select count(*) from (select user from logs where substr(time,7,2)>=21 and substr(time,7,2)<23 group by user) as user;
```

```
69
```