

# Coursework for Big Data Technologies

Minyu Luo

April 2020

## 1 Task 1. mongoDB queries [total marks 30]

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 4 below. Solutions with more than one queries will receive a zero mark. It is ok if your query answers output "\_id" too.

1. 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. [8]

---

```
db.cl.find(
{
  $and: [
    { displayName: {
      $regex: /^A/i}
    },
    {friendsCount: {
      $lt:25}
    },
    {displayName: {
      $regex: "es$"}
    }
  ],
  {
    _id: 0, displayName: 1, followersCount: 1, friendsCount:1
  }
}).sort({displayName:-1});
```

---

```
{ "displayName" : "angie torres", "friendsCount" : 23, "followersCount" : 33 }
{ "displayName" : "angie torres", "friendsCount" : 23, "followersCount" : 33 }
{ "displayName" : "Arizona Companies", "friendsCount" : 0, "followersCount" : 10 }
{ "displayName" : "Adejies", "friendsCount" : 13, "followersCount" : 10 }
{ "displayName" : "Adejies", "friendsCount" : 13, "followersCount" : 10 }
{ "displayName" : "Adejies", "friendsCount" : 13, "followersCount" : 10 }
```

Figure 1: output1\_1

2. Write a query that returns the average ratio between numbers of followers and number of friends, over all documents. Note that the number of followers must be  $\neq 0$  for the ratio to be defined. [10]

---

```
db.cl.aggregate([
{
  $match: {
    friendsCount: { $gt: 0 }
  }
},
{
  $project: {
    _id : 0,
    ratio: { $divide: [ "$followersCount", "$friendsCount" ] }
  }
},
{
  $group: {
    _id: null,
    average_ratio: {
      $avg: "$ratio"
    }
  }
}
]);
```

---

```
{ "_id" : null, "average_ratio" : 156.2146935903003 }
```

Figure 2: output1\_2

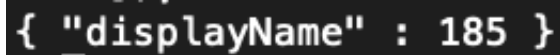
3. Write a query that returns the number of users who have at least 1000 friends, posted a tweet (the field verb has the value "post") and whose tweet (field body) contains the string "Madrid" (even as part of a

word). [7]

---

```
db.cl.aggregate([
{
  $match: {
    friendsCount: { $gte: 1000 },
    verb: { $regex: /post/ },
    body: { $regex: /Madrid/ }
  },
{
  $count: 'displayName'
}
]);
```

---



```
{ "displayName" : 185 }
```

Figure 3: output1\_3

4. Write a query that displays the number of followers of users who have more than 200 and fewer than 203 statuses. [5]

---

```
db.cl.aggregate([
{
  $match: {
    statusesCount: { $gt: 200, $lt: 203}
  },
{
  $project: {
    _id: 0,
    displayName: 1,
    followersCount: 1
  }
}
]);
```

---

```
{ "displayName" : "jUnE 6 MiNe bAbY", "followersCount" : 139 }
{ "displayName" : "Keshav Raghav", "followersCount" : 16 }
{ "displayName" : "emanuele lombardi", "followersCount" : 7 }
{ "displayName" : "DigitalAnniversaries", "followersCount" : 1056 }
{ "displayName" : "Indra J.P. Senaen", "followersCount" : 23 }
{ "displayName" : "Indra J.P. Senaen", "followersCount" : 23 }
{ "displayName" : "Lex van Houten", "followersCount" : 79 }
{ "displayName" : "Prince-Vejita", "followersCount" : 25 }
{ "displayName" : "Silvia Alonso", "followersCount" : 322 }
{ "displayName" : "juan pablo suazo", "followersCount" : 77 }
{ "displayName" : "Bendita Cocina", "followersCount" : 344 }
```

Figure 4: output1\_4

## 2 Task 2. Apache Spark [total marks 40]

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 following 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
from operator import add

sc = SparkContext('local', 'pyspark')

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+'

def parse_with_age_group(data):
    userid,age,gender,occupation,zip = data.split("|")
    return userid,age_group(int(age)),gender,occupation,zip,int(age)

#----- write my code below -----#

# Import the u.user file to fs variable
fs=sc.textFile("file:///Users/luochacha/Desktop/BigData_CW2/u.user")

# apply function parse_with_age_group to fs
fs_new = fs.map(parse_with_age_group)

# filter 40-50 and 50-60 age group respectively
fs_40_50 = fs_new.filter(lambda x: '40-50' in x)
fs_50_60 = fs_new.filter(lambda x: '50-60' in x)

# extract occupation in these two age groups
fs_40_50_s = fs_40_50.map(lambda x: x[3]).countByValue().items()
fs_50_60_s = fs_50_60.map(lambda x: x[3]).countByValue().items()

# extract 10 most common occupations for the users in each age group
fs_40_50_ss = sorted(fs_40_50_s, reverse = True, key = lambda x :
    x[1])[:10]
fs_50_60_ss = sorted(fs_50_60_s, reverse = True, key = lambda x :
    x[1])[:10]
# store data into list
fs_40_50_lst = [item[0] for item in fs_40_50_ss]
fs_50_60_lst = [item[0] for item in fs_50_60_ss]

# convert list into rdd:
fs_40_50_lst = sc.parallelize(fs_40_50_lst)
fs_50_60_lst = sc.parallelize(fs_50_60_lst)

# intersection 40-50,50-60 and 10 occupations in all age groups
common=(fs_40_50_lst.distinct()).intersection(fs_50_60_lst.distinct())

# print the output
print (sorted(common.collect()))

```

---

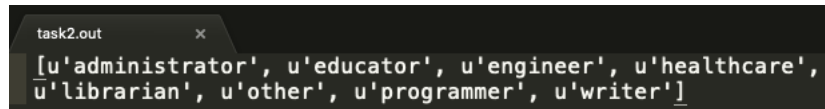


Figure 5: output2

### 3 Task 3. HIVE [total marks 30]

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'.

---

```
# Create database names log_db
CREATE DATABASE [IF NOT EXISTS] log_db;

# change to log_db database
use log_db;

# Create table logs
create table logs (user String, time String, query String) COMMENT
  'Logs details' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
  LINES TERMINATED BY '\n' STORED AS TEXTFILE;

# Load data to table logs
load data local inpath 'home/cloudera/Desktop/query_logs.txt'
  overwrite into table logs;
```

---

1. 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. [7]
-

```
# 1) returns max(num of visits of all users) /
128315306CE647F6 78

SELECT user , count(*) as cnt from logs group by user order by
cnt desc limit 1;
```

---

OUTPUT:

```
Total MapReduce CPU Time Spent: 5 seconds 830 msec
OK
128315306CE647F6      78
Time taken: 47.355 seconds, Fetched: 1 row(s)
```

Figure 6: output1

2. Write a query that returns, for each user, the number of queries that the user performed and end with the string "business". [7]

```
# 2) returns for each user, the number of queries and end with
'business' / 02E76389CBC661F7 4 / 0B294E3062F036C3 11 /
74165896F654D30 2/

SELECT user, count(query) FROM logs where substr(query,-8) ==
'business' group by user;
```

---

OUTPUT:

```
Total MapReduce CPU Time Spent: 3 seconds 480 msec
OK
02E76389CBC661F7      4
0B294E3062F036C3     11
74165896F4654D30      2
Time taken: 22.083 seconds, Fetched: 3 row(s)
```

Figure 7: output2

3. 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"). [7]

---

```
# 3) returns all attributes who issues a query that contains
the string 'job' maybe inside/ 16 rows
```

```
SELECT user, time, query FROM logs WHERE query RLIKE 'job';
```

---

OUTPUT:

```
-----
Total MapReduce CPU Time Spent: 1 seconds 820 msec
OK
4077443B5801F0C3      970916182623      job openings
4077443B5801F0C3      970916182752      job openings listings
4077443B5801F0C3      970916182823      agricultural job listings
4077443B5801F0C3      970916182834      agricultural job listings employdogst
4077443B5801F0C3      970916182942      job listings
8360729088BEAFC6      970916070440      jobs=hong kong
D5D8220D36969861      970916222708      job interview tips
D5D8220D36969861      970916224215      job interview tips
0E10D08EB5EEB192      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
Time taken: 14.156 seconds, Fetched: 16 row(s)
```

Figure 8: output3

4. 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. [9]

---

```
# 4) returns the number of distinct users who issue a
non-empty query between [21:00:00 22:59:59] yyMMddHHmmss /
69
```

```
SELECT count(distinct(user)) FROM logs WHERE query != ' ' AND
from_unixtime(unix_timestamp(time,'yyMMddHHmmss'),'HH:mm:ss')
between '21:00:00' and '22:59:59';
```

---

OUTPUT:



```
Total MapReduce CPU Time Spent: 4 seconds 210 msec  
OK  
69  
Time taken: 23.396 seconds, Fetched: 1 row(s)
```

Figure 9: output4