HW1 for W4121

Name: Haiqi Li

UNI:hl3115

All codes can be find in Appendix and there is link on each.

Also, I use Xshell instead of Putty to launch AWS.

Step1

vi twitter.JSON

{"delete":{"status":{"id":334987056411471873,"user_id":909597110,"id_str":"334987056411471873","user_id_str":"909597110"}}}
{"delete":{"status":{"id":373064189616455682,"user_id":84829030,"id_str":"373064189616455682","user_id_str":"84829030"}}}
{"delete":{"status":{"id":325194283521036288,"user_id":704204053,"id_str":"325194283521036288","user_id_str":"704204053"}}}
{"delete":{"status":{"id":365769036673454081,"user_id":231040894,"id_str":"365769036673454081","user_id_str":"231040894"}}}

Step2

I wrote a python script to do this task. It's name is lab1ProtocolBuffer.py.

Let's see what happened.

```
ubuntu@ip-172-31-18-165:~/csds-material$ python lab1protocolbuffer.py
1554 messages are deleted.
2531 tweets are replies to another tweet
Find the five user IDs (field name: uid) that have tweeted the most.
1269521828: 5
392695315: 4
424808364: 3
1706901902: 3
98305691: 2
```

I add print line in python so the picture is clear enough.

Note that the last person show in this page is not only one, which means there may be other people tweets twice.

Step 3

In SQLite3 terminal, I do the following job:

```
sqlite> select count(*) from tweets where is_delete=1;
1554
sqlite> select count(*) from tweets where reply_to > 0;
2531
sqlite> select uid,count(*) from tweets where is_delete =0 group by uid order by count(*) desc limit 5;
1269521828|5
392695315|4
424808364|3
1706901902|3
23991910|2
```

Step4

```
ubuntu@ip-172-31-18-165:~/csds-material$ mongo lab2
MongoDB shell version: 3.2.18
connecting to: lab2
Server has startup warnings:
2018-02-02T18:09:03.913+0000 I CONTROL  [initandlisten]
2018-02-02T18:09:03.913+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hug
epage/enabled is 'always'.
2018-02-02T18:09:03.914+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2018-02-02T18:09:03.914+0000 I CONTROL  [initandlisten]
2018-02-02T18:09:03.914+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hug
epage/defrag is 'always'.
2018-02-02T18:09:03.914+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2018-02-02T18:09:03.914+0000 I CONTROL  [initandlisten]
> db.tweets.find({"delete":{$exists:true}}).count()
1554
> db.tweets.find({"in_reply_to_status_id":{$ne:null}}).count()
2531
> db.tweets.ensureIndex({"user.id_str":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 3,
        "note" : "all indexes already exist",
        "ok" : 1
}
> db.tweets.aggregate([{$group:{_id:"$user.id_str",num:{$sum:1}}},{$sort:{num:-1}},{$limit:6}])
{ "_id" : null, "num" : 1554 }
{ "_id" : "1269521828", "num" : 5 }
{ "_id" : "392695315", "num" : 4 }
{ "_id" : "424808364", "num" : 3 }
{ "_id" : "1706901902", "num" : 3 }
{ "_id" : "1369322330", "num" : 2 }
>
```

I have to note that “uid” null messages are counted, so actually I have to search the most tweeted six person.

Step5

Something to note first

I am a little confused about joining rules of this part. I asked in piazza and TA answers me. I decide to use the following rule:

Example:

If I have tweets dates like this:

```
Sat Aug 31 06:57:23 +0000 2013

Sat Aug 31 16:57:23 +0000 2013
```

And if the records csv like this:

```
08/31/2013 Clear 18:36 US

08/31/2013 Overcast 6:51 US
```

Then I attach to all matched. This means that if there is a record on 08/31/2013 shows that it is clear, then we regard this day as “clear”. The above tweets are supposed to be in “clear” condition. I obey this rule both on part 1 and part 2.

Another difficulty of this step is that the date are not write in the same form. I use python **datetime** package to make them machable.

Part1

I first redesigned the proto file and get my proto1.py.Using this file to get myproto1_pb2.

Then I copied encode.py in the server and build myEncode1.py.Using this file I can get my.pb as my database.

Finally, I build myProtocol1.py as my Protocol Buffer.

Let’s see what happened this time:

(due to the fact the output is too long, I just screen shot part of it.)

```
ubuntu@ip-172-31-18-165:~/csds-material$ python myProtocol1.py
US
Sat Aug 31 06:57:30 +0000 2013
782380406
clear
US
Sat Aug 31 06:57:34 +0000 2013
196009863
clear
US
Sat Aug 31 06:57:34 +0000 2013
523639438
clear
US
Sat Aug 31 06:57:38 +0000 2013
834848905
clear
US
Sat Aug 31 06:57:51 +0000 2013
171692460
clear
US
Sat Aug 31 06:57:55 +0000 2013
356565799
clear
US
Sat Aug 31 06:58:11 +0000 2013
391053040
clear
US
Sat Aug 31 06:58:11 +0000 2013
1432092518
clear
US
Sat Aug 31 06:58:12 +0000 2013
```

Part2

Before this part, due to the fact that I have to make changes to database itself, I first make up a database named mytwitter.db.

First of all, I import csv into database, which is really a simple job.

sqlite> .mode csv

sqlite> .table

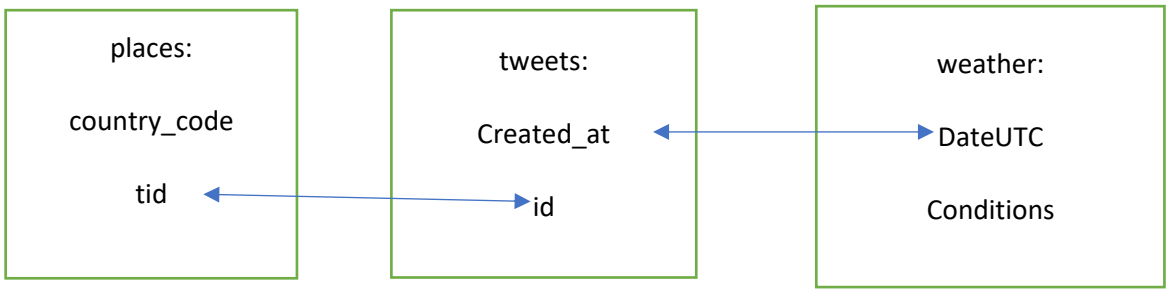
sqlite> .import /home/ubuntu/csds-material/WeatherDataset.csv weather

```
sqlite> .table
coords        place_coords  places        tweets        weather
```

As you can see, weather table has been put in our database.

The next step is to makes a join key of table weather and table tweets. The key is weather.DateUTC and tweets.created_at.

Like in part 1, I use python datetime package to update two key values to make a join. The script is [pythonsql.py](). The update process would takes about 10 minutes.



Relationship of three table.

After this, we can make join operation in SQLite.

```
sqlite> select tweets.id,weather.Conditions,weather.DateUTC from tweets inner join places on tweets.id=
places.tid inner join weather on tweets.created_at=weather.DateUTC where weather.Conditions='Clear' and
 places.country_code='US';
373701061132836864|Clear|Aug-31-2013
373701061132836864|Clear|Aug-31-2013
373701077893644288|Clear|Aug-31-2013
373701077893644288|Clear|Aug-31-2013
373701077918429184|Clear|Aug-31-2013
373701077918429184|Clear|Aug-31-2013
373701094691446785|Clear|Aug-31-2013
373701094691446785|Clear|Aug-31-2013
373701149221986304|Clear|Aug-31-2013
373701149221986304|Clear|Aug-31-2013
373701165999194112|Clear|Aug-31-2013
373701165999194112|Clear|Aug-31-2013
373701233087107072|Clear|Aug-31-2013
373701233087107072|Clear|Aug-31-2013
373701233111859201|Clear|Aug-31-2013
373701233111859201|Clear|Aug-31-2013
373701237310382081|Clear|Aug-31-2013
373701237310382081|Clear|Aug-31-2013
373701249880702976|Clear|Aug-31-2013
373701249880702976|Clear|Aug-31-2013
373701279241211904|Clear|Aug-31-2013
373701279241211904|Clear|Aug-31-2013
373701304419221504|Clear|Aug-31-2013
373701304419221504|Clear|Aug-31-2013
373701321162891265|Clear|Aug-31-2013
373701321162891265|Clear|Aug-31-2013
373701342163767296|Clear|Aug-31-2013
373701342163767296|Clear|Aug-31-2013
```

The above picture shows my consequence. The CLI here is a selection that join tweets, places and weather as my above relationship and filtered by weather.Conditions=Clear and places.countrycode=US.

1. Read the schema and Protocol Buffer definition files. What are the main differences between the two? Are there any similarities?

   The difference is that the structure is redesigned in .proto file(like the JSON is begin with insert or delete, but the Protocol Buffer add a is_delete). So there will not be any undefined type show up in Protocol Buffer.

   As for similarity, they are both tree structure(hierarchical model) .

2. Describe one question that would be easier to answer with Protocol Buffers than via a SQL query.

   If I want to do a query that not sure a certain attribute exist or not, I can write a script to check. But it is difficult for SQL to do check before hand.
   Besides, I can do a single query in Protocol Buffer to get all results. But we have to do one query for one specific target.

3. Describe one question that would be easier to answer with MongoDB than via a SQL query.

   If we try a join operation of different tables, SQL have to join two or more tables first and then query. But data stored in MongoDB is easy to extend and we do not need such operation. This is due to the fact that SQL cannot always match new record and have to define left join or right join first.

4. Describe one question that would be easier to answer via a SQL query than using MongoDB.

   Say in SQL students information stores in one table and their matched headmaster stores in another table. If the headmaster changes, we can just change the table of headmaster to update in SQL. But in MongoDB we have to update all records.

5. What fields in the original JSON structure would be difficult to convert to relational database schemas?

I.Some "optional" parts that exist in certain record in JSON. It is difficult for relational model to record the attributes that is optional(not sure exist or not).
II. Some long strings. Relational model is not appropriate to store data that length is not certain. Just like what people tweets in twitter.

6. In terms of lines of code, when did various approaches shine? Think about the challenges of defining schemas, loading and storing the data, and running queries.

   As for code, I think SQLite and MongoDB is simple enough but we have to write lines of code in Protocol Buffer.
   Actually, I think it is not fruitful to us MongoDB if the relation of data can be defined beforehand. In sophisticate query, like if the query target is changing, MongDB is not a good case.
   If we can be sure of the schema and the structure of data, SQL is perfect. The query is easy to learn and to read. But say if we want real-time sequence based query, SQL is not a good choice.
   Protocol Buffer really takes time to learn to be proficient. However, if one can get proficient with it, he or she can to whatever job he or she wants. Those dirty job can only be done in a programming language API like python.

7. What other metrics (e.g., time to implement, code redundancy, etc.) can we use to compare these different approaches? Which system is better by those measures?

   Least time to implement: MongoDB

   Code redundancy: Protocol buffer
   Write lines of codes to achieve what we want

   Storing unstructured data: MongoDB
   Tree model is always best structure for unstructured data

   Real-time processing: MongoDB
   Can store data based on real time

   Update a structured attribute: SQL
   If data is defined properly, then SQL query is simple.

Easy to learn: SQL
Readable.

To finish sophisticated job: Protocol Buffer
Under Programming language API we can finish real hard job.

8. How long did this lab take you? We want to make sure to target future labs to not take too much of your time.

   I spent about 24 hours to finished my homework.

   Setup:2 hours
   Protocol Buffer:10 hours(explanation in Feedback)
   SQLite: 1 hour
   MongoDB:1 hour
   Step 5 :7 hours (I feel really upset when I learnt that no extra credit for it…)
   Answer reflection question and write report:3 hours

Feedback

As for lab itself, here are my suggestions:

I think our lab is a little **difficult** for students who are not CS background. I think to spend 24 hours to do a homework is too long.
One of the reason is that I try to install local virtual machine but encounter many unpleasant issues, which waste me tons of time. I think **AWS should be listed first** because people would like to try the first approach he or she meet at first time.
Also, we do not know anything about Google Protocol Buffer before lab. The instruction only shows we should read official tutorial. But the tutorial gives a way to load file by input command and I do not really know what kind of files do they really open. I thought they load JSON and I read nearly all tutorials and spend so much time to learn google.protobuf.JSON_format, which has nothing to do with this lab.
If official tutorial shows me what kind of file I should load or if lab give me more specific instructions, I think that would be much more better. **I think just leave an official tutorial as instruction is not enough.**
Besides, the **content of this lab is very good**. I only learned some concept of data storing

issues in class. In this lab, I have the chance to deal with real code myself which is really fascinating!

For course

I think professor Wu is really nice. He gives us good lecture and answers us questions. The only suggestion is that I want to **see more code in class**.
Example: when we come to noSQL, after introducing us basic concept, it would be much more better if professor gives us a simple slide to show some codes about how to query and gives a short explanation of the code. After all, we are a CS course and I think to show more code is necessary. If that would spend too much time, I think to upload some pieces of code in Git or Piazza for us to follow is also a good choice.

Appendix

I do know that I shall not attach code that is provided by lab. But to achieve my target, I do have some redesign of the original code. I think it is necessary to attach it to this PDF, though some parts of them is from the original code.

lab1ProtocolBuffer.py

Protocol Buffer to achieve all requirement of Step2.

```
1.  import twitter_pb2
2.
3.
4.  tweets=twitter_pb2.Tweets()
5.
6.  f=open("twitter.pb","rb")
7.  tweets.ParseFromString(f.read())
8.  f.close()
9.
10. #Find the number of deleted messages in the dataset.
11. deleteCount=0
12. for delete in tweets.tweets:
13.     if delete.is_delete==True:
```

```
            deleteCount+=1
print "%d messages are deleted." %deleteCount

#Find the number of tweets that are replies to another tweet.
repliesToOther=0
for reply in tweets.tweets:
    if reply.HasField('insert'):
        if reply.insert.HasField('reply_to'):
            repliesToOther+=1
print "%d tweets are replies to another tweet" %repliesToOther

#Find the five user IDs (field name: uid) that have tweeted the most.
tweetsCountDict={}
for uidTweetCount in tweets.tweets:
    if uidTweetCount.HasField('insert'):
        Tempuid=str(uidTweetCount.insert.uid)
        if tweetsCountDict.has_key(Tempuid)==False:
            tweetsCountDict[Tempuid]=1
        else:
            tweetsCountDict[Tempuid]+=1

print "Find the five user IDs (field name: uid) that have tweeted the most."

i=0
for key,value in sorted(tweetsCountDict.iteritems(),key=lambda (k,v):
(v,k),reverse=True):
    if i<5:
        print "%s: %s" % (key,value)
        i+=1
```

myproto1.py

My proto file to instruct my own encode process. Most of part is from twitter.proto.

```
package my1;


```

```
4.  message Tweet {
5.      message Coord {
6.          required float lat = 1;
7.          required float lon = 2;
8.      }
9.      message Place {
10.         required string url = 1;
11.         required string country = 2;
12.         required string country_code = 3;
13.         required string place_type = 4;
14.         repeated Coord bounding_box = 5;
15.         required string id = 6;
16.         required string name = 7;
17.         optional string condition=8;
18.
19.     }
20.     message Delete {
21.         required int32 uid = 1;
22.         required int64 id = 2;
23.     }
24.     message Insert {
25.         required bool truncated    = 1;
26.         optional string text = 2;
27.         optional int64 reply_to = 3;
28.         optional string reply_to_name = 4;
```

```
29.         required int64 id = 5;
30.         required int64 uid = 6;
31.         optional int32 favorite_count = 7;
32.         optional string source = 8;
33.         required bool retweeted = 9;
34.         optional bool possibly_sensitive = 10;
35.         optional string lang = 11;
36.         required string created_at = 12;
37.         optional Coord coord = 13;
38.         required string filter_level = 14;
39.         optional Place place = 15;
40.
41.     }
42.
43.     optional bool is_delete = 1;
44.     optional Delete delete = 2;
45.     optional Insert insert = 3;
46. }
47.
48. message Tweets {
49.     repeated Tweet tweets = 1;
50. }
51.
52.
```

myEncode1.py

My encode file to get my.pb . Most of part is from encode.py.

```
1. import csv
2. from JSON import loads
3. from itertools import imap
4. from myproto1_pb2 import *
5. import datetime
6. #My target is to creat a my.pb with only record of tweets created in 2013 && US && clear Condition(define in csv)
7.
8.
9. import google.protobuf.JSON_format
10.
11. #row[12] is condition row[14] is date row 15 is countrycode
12.
13. def dealDateProtocol(string):
14. #input date from Protocol,output datatime in string
15. templist=str(string).split(' ')
16. stringv1=templist[1]+'-'+templist[2]+'-'+templist[5]
17. fdate=datetime.datetime.strptime(stringv1,'%b-%d-%Y')
18.
19. return fdate.strftime('%b-%d-%Y')
20.
21. def dealDateCsv(string):
22. #Input date from csv,output datetime in string
```

```
23. fdate=datetime.datetime.strptime(string,'%m/%d/%Y %H:%M')
24. return fdate.strftime('%b-%d-%Y')
25.
26. def TestClear(string,dateList):
27. outstring=dealDateProtocol(string)
28. Logic=False
29. for date in dateList:
30. if outstring==date:
31. Logic=True
32. return Logic
33.
34. with open('WeatherDataset.csv') as f:
35. f_csv=csv.reader(f)
36. DateInClear=[]
37. for row in f_csv:
38. if row[11]=='Clear':
39. DateRight=dealDateCsv(row[13])
40. DateInClear.append(DateRight)
41. #get a date list that is claer
42. f.close()
43.
44. tweets = Tweets()
45. with file('twitter.JSON', 'r') as f:
46. for line in imap(loads, f):
47. tweet = tweets.tweets.add()
```

```
48. if line.get('place',None):
49. place = line['place']
50. if place['country_code']=='US':
51. if TestClear(line['created_at'],DateInClear):
52. insert = tweet.insert
53. insert.uid = line['user']['id']
54. insert.truncated = line['truncated']
55. insert.text = line['text']
56. if line.get('in_reply_to_status_id', None):
57. insert.reply_to = line['in_reply_to_status_id']
58. insert.reply_to_name = line['in_reply_to_screen_name']
59. insert.id = line['id']
60. insert.favorite_count = line['favorite_count']
61. insert.source = line['source']
62. insert.retweeted = line['retweeted']
63. if line.get('possibly_sensitive', None):
64. insert.possibly_sensitive = line['possibly_sensitive']
65. insert.lang = line['lang']
66. insert.created_at = line['created_at']
67. if line.get('coordinates', None):
68. coords = line['coordinates']
69. insert.coord.lat = coords['coordinates'][0]
70. insert.coord.lon = coords['coordinates'][1]
71. insert.filter_level = line['filter_level']
72.
```

```
73. if line.get('place', None):
74. place = line['place']
75. insert.place.url = place['url']
76. insert.place.country = place['country']
77. insert.place.country_code = place['country_code']
78. insert.place.place_type = place['place_type']
79. insert.place.id = place['id']
80. insert.place.name = place['name']
81. insert.place.condition='clear'
82. if place.get('bounding_box', None):
83. def add(pair):
84. coord = insert.place.bounding_box.add()
85. coord.lat = pair[0]
86. coord.lon = pair[1]
87. map(add, place['bounding_box']['coordinates'][0])
88.
89.
90.
91. with file('my.pb', 'w') as f:
92. f.write(tweets.SerializeToString())
```

myProtocol1.py

Another Protocol Buffer to achieve Step 5 target 1.

```
1. import myproto1_pb2
2.
```

```
3. tweets=myproto1_pb2 .Tweets()
4.
5. f=open("my.pb","rb")
6. tweets.ParseFromString(f.read())
7. f.close()
8.
9. for x in tweets.tweets:
10. if x.HasField("insert"):
11. print x.insert.place.country_code
12. print x.insert.created_at
13. print x.insert.uid
14. print x.insert.place.condition
```

pythonsql.py

After put csv data into database, I use this script to make their time same format.

```
1. import sqlite3 as lite
2. import sys
3. import datetime
4.
5.
6. def dealDateSQLite(string):
7. #input date from SQLite,output datatime in string
8.     if string==None:
9.         return
10.    templist=str(string).split(' ')
```

```
11.     stringv1=templist[1]+'-'+templist[2]+'-'+templist[5]
12.     fdate=datetime.datetime.strptime(stringv1,'%b-%d-%Y')
13.
14.     return fdate.strftime('%b-%d-%Y')
15.
16. def dealDateCsv(string):
17. #Input date from csv,output datetime in string
18.     fdate=datetime.datetime.strptime(string,'%m/%d/%Y %H:%M')
19.     return fdate.strftime('%b-%d-%Y')
20.
21. con = lite.connect('mytwitter.db')
22.
23. with con:
24.
25.     cur = con.cursor()
26.
27.     cur.execute("SELECT created_at,id FROM tweets;")
28.     sqldate = cur.fetchall()
29.
30.
31.     for row in sqldate:
32.         tempdate=row[0]
33.         tempid = row[1]
34.         if tempdate:
35.
```

```
36.             datenew = dealDateSQLite(tempdate)
37.             cur.execute("update tweets set created_at = " + '\''
38.                                     + datenew + '\'' + " where id = " + str(tempid) + ";")
39.
40.     print "sql end"
41. # update date in tweet to standardize type
42.
43.     cur.execute("select DateUTC from weather;")
44.     csvdate=cur.fetchall()
45.
46.     for row in csvdate:
47.
48.         datenew=dealDateCsv(row[0])
49.
50.         cur.execute("update weather set DateUTC = "+ "\'"+datenew+"\'"+" where
    DateUTC ="+"\'"+row[0]+"\';")
51.
52.     print "csv end"
53. #update date in weather to standardized type
54.
```