

Lab 8

Karl Roth

In this lab we demonstrated fundamental concepts of MongoDB. We learned how to access, insert, and delete entries in a database. We also wrote and ran a python script that imported Chicago twitter data into our MongoDB database.

!. Access MongoDB

```
[karo4@postgres-g479:~]$ mongo
MongoDB shell version: 2.6.10
connecting to: test
[> show dbs
aaronh2      0.078GB
ab2          0.078GB
admin        (empty)
ahkatz       0.078GB
ccong2       0.078GB
dab3         0.078GB
g479         0.078GB
geog479      0.078GB
guimin2      0.078GB
ifoli2       0.203GB
jacquot3     0.078GB
jianwen2     0.078GB
kasonye2     0.078GB
local        0.078GB
mlpope2      0.078GB
momoult2     0.078GB
nj7          0.078GB
nurlank2     0.078GB
rcv3         0.078GB
sguo         0.203GB
test         0.078GB
tongg2       0.078GB
wclin2       0.078GB
wenhank2     0.078GB
xiaoluq2     0.078GB
yangkx       0.078GB
yatongz2     0.203GB
your_netID   0.203GB
yujiany2     0.078GB
[> use karo4
switched to db karo4
> _
```

Fig 1: A list of all of the databases in our courses ROGER MongoDB

```
[> use karoth4 ]
switched to db karoth4
[> db.class.insert({num:"geog479",name:"cyberGIS"}) ]
WriteResult({ "nInserted" : 1 })
[> db.class.insert({_id:"g480",num:"geog480",name:"Principles of GIS"})
2017-10-25T13:59:21.322-0500 SyntaxError: Unexpected string
[> db.class.save({_id:"g480",num:"geog480",name: "Principles of GIS"})
2017-10-25T13:59:48.594-0500 SyntaxError: Unexpected identifier
[> db.class.save({num:"geog480",name:"Principles of GIS -new"})
WriteResult({ "nInserted" : 1 })
[> show collections ]
class
system.indexes
```

Fig 2: Demonstrating various insertions into a database called “class”, and saving the results

```
[> db.class.find() ]
{ "_id" : ObjectId("59f0e18d3b48e49413238870"), "num" : "geog479", "name" : "cyberGIS" }
{ "_id" : "g480", "num" : "geog480", "name" : "Principles of GIS - Old" }
{ "_id" : ObjectId("59f0e2773b48e49413238871"), "num" : "geog480", "name" : "Principles of GIS -new" }
[> db.class.findOne() ]
{
  "_id" : ObjectId("59f0e18d3b48e49413238870"),
  "num" : "geog479",
  "name" : "cyberGIS"
}
```

Fig 3: Listing the contents of our “class” database

```
[> db.class.find({num:"geog480"}) ]
{ "_id" : "g480", "num" : "geog480", "name" : "Principles of GIS - Old" }
{ "_id" : ObjectId("59f0e2773b48e49413238871"), "num" : "geog480", "name" : "Principles of GIS -new" }
[> db.class.find({$and:[{num:"geog480"},{_id:"g480"}]}) ]
{ "_id" : "g480", "num" : "geog480", "name" : "Principles of GIS - Old" }
[> db.class.find({$and:[{num:"geog480"},{_id:{$ne: "g480"}]}) ]
{ "_id" : ObjectId("59f0e2773b48e49413238871"), "num" : "geog480", "name" : "Principles of GIS -new" }
```

Fig 4: Querying all entries where “num is geog480”, “num is geog480 and id is g480”, and where “num is geog480 and id is not g480”

```
> db.class.update({num:"geog479"},{"num":"geog479","name":"cyberGIS",classroom:"Davenport Hall 338"})
[2017-10-25T14:20:45.638-0500 SyntaxError: Unexpected token ILLEGAL
> db.class.update({num:"geog479"},{"num":"geog479","name":"cyberGIS",class:"Davenport Hall 338"
... })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.class.update({num:"geog479"},{$set:{hours:"3:30-4:450pm T Th"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.class.find({num:{$ne:"geog480"}})
{ "_id" : ObjectId("59f0e18d3b48e49413238870"), "num" : "geog479", "name" : "cyberGIS", "class" : "Davenport Hall 338", "hours" : "3:30-4:450pm T Th" }
```

Fig 5: Modifying the entries and displaying the results

```
[> db.class.remove({num:/480$/})
WriteResult({ "nRemoved" : 2 })
[> exit
bye
```

Fig 6: remove all entries with 480 in them, and then log out.

Chicago Twitter Data

```
karothe4@postgres-g479:/projects/geog479/lab8$ head 2014_07_04_chi.txt
92799208      41.88186949      -87.70267987      Thu Jul 03 23:59:55 CDT 2014      I look so beautiful on glide 🍷
1468734256    41.72373577      -88.25298795      Thu Jul 03 23:59:55 CDT 2014      @benwimer_ you followed me in ig and twitter 🍷
348179245     41.5973941       -88.1317567       Thu Jul 03 23:59:57 CDT 2014      I can live off just chocolate covered bananas , no questions asked , I can
1468788615    41.4206919       -88.2482691       Fri Jul 04 00:00:01 CDT 2014      Happy birthday bro! @cfernandez242
481767368     42.2309208       -88.44688679      Fri Jul 04 00:00:01 CDT 2014      Happy 4th!!! 🍷
```

Fig 7: Examine the structure of the data. [uid, lat, long, datetime, msg]

```
karothe4@postgres-g479:/projects/geog479/lab8$ wc -l 2014_07_04_chi.txt
71477 2014_07_04_chi.txt
karothe4@postgres-g479:/projects/geog479/lab8$ du -h 2014_07_04_chi.txt
8.6M    2014_07_04_chi.txt
```

Fig 8: The text file contains 71,477 lines and is 8.6MB in size

```
GNU nano 2.5.3      File: mongoimport.py      Modified

from pymongo import MongoClient
import datetime
import json
import os

def main():

    client = MongoClient('mongodb://localhost:27027')
    db = client.karothe4
    collection = db['chicago']

    mFile = open("/projects/geog479/lab8/2014_07_04_chi.txt", "rb")
    for mline in mFile:
        [uid, lat, lng, tm, msg] = mline.split("/")
        latt=float(lat)
        lngg = float(lng)
        geoString = {'user_id':uid, 'time':tm, 'message':msg, 'loc': {'type': 'Point', 'coordinates': [lngg, latt]}}
        print geoString
        db.chicago.insert(geoString)

if __name__ == '__main__':
    main()
```

Fig 9: A python script written to import the Chicago twitter data into MongoDB.

```

karothe4@postgres-g479:~$ mongo
MongoDB shell version: 2.6.10
connecting to: test
> use karothe4
switched to db karothe4
> show collections
chicago
class
system.indexes
> db.chicago.findOne()
{
  "_id" : ObjectId("59f235979a659f35be503167"),
  "loc" : {
    "type" : "Point",
    "coordinates" : [
      -87.70267987,
      41.88184949
    ]
  },
  "message" : "I look so beautiful on glide 🍷n",
  "user_id" : "92799200",
  "time" : "Thu Jul 03 23:59:55 CDT 2014"
}

```

Fig 10: Logging into MongoDB to demonstrate that the data imported properly. Displaying the first tweet in the database.

```

> db.chicago.createIndex({loc:"2dsphere"})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.chicago.find({loc:{$nearSphere:{$geometry:{type:"Point",coordinates:[-87.639160,41.878628]}},$minDistance:1000,$maxDistance:5000}})

```

Fig 11: To perform spatial queries we must let MongoDB which field contains the location data, otherwise an error will be displayed. (I have not included the actual query results because they are too long).

```

> db.chicago.find({loc:{$geoWithin:{$geometry:{type:"Polygon",coordinates:[[-87.639546,41.878053],[-87.639643,41.879506],[-87.638602,41.879363],[-87.638270,41.878045],[-87.639546,41.878053]]]}})

```

Fig 12: Query of tweets within a specific region (i.e. "Polygon") Again, query results not included in screen capture.

```

> db.chicago.find({loc:{$nearSphere:{$geometry:{type:"Point",coordinates:[-87.639160,41.878053]}},$minDistance:1000,$maxDistance:5000}}).sort({"loc":1})
{ "_id" : ObjectId("59f235c29a659f35be51095e"), "loc" : { "type" : "Point", "coordinates" : [ -87.69904629, 41.88192914 ] }, "message" : "Today ! http://t.co/kw2UGvHXW3\n", "user_id" : "283282734", "time" : "Fri Jul 04 10:45:09 CDT 2014" }

```

Fig 13: The query performed in figure 11, but sorted by distance. The first result was included in this screen capture