This Lab assignment will be turned in as a PDF document. Save a copy of this document, paste your answers in the marked areas, and generate a PDF when you are complete. Your PDF file name should have the following format, all lower-case:

yourname-mongodb-1.pdf

All PDFs are to be committed to the /homework/ directory in source control before the beginning of the following class.

MongoDB, Part 1

MongoDB is a non-relational (NoSQL) database system. It is document-oriented with JSON structure, and presents a flat storage space where you can put all of your records. It uses the "_id" convention for primary keys.

MongoDB uses native drivers and a console to manage the database. If you wanted to use MongoDB from PHP, for example, you'd need to find and install a PHP driver.

Setting Up A MongoDB Service

You can download the latest version of mongod, the MongoDB daemon/service, from:

http://www.mongodb.org/downloads

There should be a binary package for OSX, and either the 64-bit or 32-bit version will work. Unzip the file to your desktop, and you should find a bin folder inside. Open a Terminal window into that bin folder. You should be able to run the mongod program and get help for its command-line arguments:

```
$ ./mongod --help
Allowed options:
General options:
  -h [ --help ]         show this usage information
```

```
  --version           show version information
...
```

For what you'll do in this assignment, you only need one argument, which is where to store the databases. Make a db directory and run the mongod service:

```
$ mkdir db
$ ./mongod --dbpath db
```

You should see a bunch of debugging information, but the program should not exit. You can verify that it is running by bringing up its status page in a web browser:

http://127.0.0.1:28017/

MongoDB runs a very minimalist web interface. Its status page is on port 28017, while the native drivers will connect on port 27017 to do all of the queries and other work.

On the status page there will be lines with db version, git hash, and sys info. Copy and paste those lines here:

```
db version v2.4.7
git hash: 016173abf06c1f067b56a465b706efd6f4bf2aa
sys info: Darwin bs-osx-106-x86-64-2.10gen.cc 10.8.0 Darwin Kernel Version
10.8.0: Tue Jun  7 16:32:41 PDT 2011; root:xnu-1504.15.3~1/RELEASE_X86_64
x86_64 BOOST_LIB_VERSION=1_49
uptime: 12466 seconds
```

Using the Console

Before you get MongoDB running with a server-side language, you need to learn how it works. There is a console application called mongo that will let you talk to the service interactively. Open a second Terminal window in MongoDB's bin folder and run it:

```
$ ./mongo
```

It should output a few lines of status information. Paste them here:

```
MongoDB shell version: 2.4.7
connecting to: test
Server has startup warnings:
Sun Oct 27 21:57:45.959 [initandlisten]
Sun Oct 27 21:57:45.959 [initandlisten] ** WARNING: soft rlimits too low.
Number of files is 256, should be at least 1000
```

Unlike using MySQL from the command line, you don't need to end your MongoDB commands with semicolons. Enter help to get a list of commands:

```
> help
    db.help()                    help on db methods
    db.mycoll.help()             help on collection methods
...
```

Get a list of databases and paste them here:

```
abd1311    0.203125GB
local 0.078125GB
```

In MongoDB you don't need to create a database before you use it—the service will create it for you when you insert your first document. Tell MongoDB you'd like to use a database named itunes:

```
> use itunes
switched to db itunes
```

MongoDB has a concept of a *collection*, which is sort of like a database within a database. Like databases, MongoDB will create collections for you when you use them. You'll use a collection

named itunes, just like your database. Most of the rest of your commands will be performed on an object named db, which has members named for your collections.

For example, you can get help on the methods available to your itunes collection from that object:

```
> db.itunes.help()
DBCollection help
    db.itunes.find().help() - show DBCursor help
...
```

Try getting a count of the number of items in your itunes collection. You should get zero. Paste the command here:

```
db.itunes.find().count()
```

Creating Objects & Documents

The MongoDB console works much like the Firebug or Developer Tools console. You can create and echo variables in it:

```
> var rick = { type: 'person', name: 'Rick' }
> rick
{ "type" : "person", "name" : "Rick" }
```

This can be handy for creating objects before you save them to the database. Saving is done with the save() method of your collection:

```
> db.itunes.save(rick)
```

You can also call the save() method with a JSON object literal. Create an object of type person with your name and save it to the database in one command, using a literal instead of a variable. Paste the command here:

```
db.itunes.save({type:'person', name:'Tatiana'})
```

The find() method of the collection can be called without arguments to list all of the documents. Run a find() and paste the command and its results here:

```
{ "_id" : ObjectId("526df8498a4a7729e3d775f9"), "type" : "person", "name" :
"Rick" }

{ "_id" : ObjectId("526df8b48a4a7729e3d775fa"), "type" : "person", "name" :
"Tatiana" }
```

You can see from the results that MongoDB will give your documents a random _id field if you don't explicitly provide one. MongoDB will let you search on fields other than the key. For example, you could find the Rick object you created earlier:

```
> db.itunes.find({name: 'Rick'})
{ "_id" : ObjectId("..."), "type" : "person", "name" : "Rick" }
```

The find() function works similarly to how ORM works, by finding matches to a given object. In the case above, MongoDB searches for documents with a key name with a value Rick. Write an expression to find all person-typed documents and paste it here:

```
db.itunes.find({type:'person'})
```

If you know you will only get back a single record, MongoDB provides a findOne() function that will return a single object that can be assigned to a variable. It takes the same arguments:

```
> rick = db.itunes.findOne({name: 'Rick'})
{
    "_id" : ObjectId("..."),
    "type" : "person",
```

```
    "name" : "Rick"
}
```

Write an expression to find the document with your name and assign it to a variable:

```
tatiana = db.itunes.findOne({name:'Tatiana'})
```

You can update a document by saving over it. On the console, this is easiest done using variables:

```
> rick.heightInches = 74
74
> rick
{
    "_id" : ObjectId("..."),
    "type" : "person",
    "name" : "Rick",
    "heightInches" : 74
}
> db.itunes.save(rick)
> db.itunes.find({heightInches: 74})
{ "_id" : ObjectId("..."), "type" : "person", "name" : "Rick",
"heightInches" : 74 }
```

Update the document with your name to have your height in inches, save it, and find it again.

Paste those commands and results here:

```
>tatiana.heightInched = 48
>48
>Tatiana
{
    "_id" : ObjectId("526df8b48a4a7729e3d775fa"),
```

```
    "type" : "person",
    "name" : "Tatiana",
    "heightInched" : 48
}
>db.itunes.save(tatiana)
>db.itunes.find({heightInched:48})
{
    "_id" : ObjectId("526df8b48a4a7729e3d775fa"),
    "type" : "person",
    "name" : "Tatiana",
    "heightInched" : 48
}
```

More Advanced Queries

MongoDB also supports a number of advanced query options for its find and findOne methods. For example, you could search for person-type documents with heightInches greater than 72:

```
> db.itunes.find({type: 'person', heightInches: { $gt: 72 }})
{ "_id" : ObjectId("4d786a8b09dc882f439c0081"), "type" : "person",
"name" : "Rick", "heightInches" : 74 }
```

That may look funny, but the $gt acts as a greater-than operator. The operators go inside of an object so they can be combined:

```
> db.itunes.find({type: 'person', heightInches: { $gt: 72, $lte: 76 }})
```

These $ operators are special in MongoDB, and pretty much all of the operators you're used to are supported. You can get the full list here:

http://www.mongodb.org/display/DOCS/Advanced+Queries

Write a query that would get person-type documents with either a name of Rick or your name, but nothing else. Paste it below:

```
{ "_id" : ObjectId("526df8498a4a7729e3d775f9"), "type" : "person", "name" :
"Rick", "heightInched" : 74 }

{ "_id" : ObjectId("526df8b48a4a7729e3d775fa"), "type" : "person", "name" :
"Tatiana", "heightInched" : 48 }
```

Working With Results

To show off the next set of features you will need some more documents. Create and save a document for each of the students in the class. Make sure that each document has a type of person, a unique name, and a guess for heightInches. When you're done, find() all of them and paste the results below:

```
{ "_id" : ObjectId("526df8498a4a7729e3d775f9"), "type" : "person", "name" :
"Rick", "heightInched" : 74 }

{ "_id" : ObjectId("526df8b48a4a7729e3d775fa"), "type" : "person", "name" :
"Tatiana", "heightInched" : 48 }

{ "_id" : ObjectId("526dfcfd8a4a7729e3d775fb"), "type" : "person", "name" :
"Natasha", "heightInched" : 49 }

{ "_id" : ObjectId("526dfd148a4a7729e3d775fc"), "type" : "person", "name" :
"John", "heightInched" : 57 }

{ "_id" : ObjectId("526dfd228a4a7729e3d775fd"), "type" : "person", "name" :
"Russ", "heightInched" : 57 }

{ "_id" : ObjectId("526dfd3f8a4a7729e3d775fe"), "type" : "evil", "name" :
"Susan", "heightInched" : 49 }

{ "_id" : ObjectId("526dfd5a8a4a7729e3d775ff"), "type" : "person", "name" :
"Jacob", "heightInched" : 52 }

{ "_id" : ObjectId("526dfd6b8a4a7729e3d77600"), "type" : "person", "name" :
"Edward", "heightInched" : 55 }
```

You can sort the results of a find() by chaining the sort() method:

```
> db.itunes.find({type: 'person'}).sort({ name: 1 })
```

The sort() function takes an object with the name of the key to sort on and a direction (1 for ascending, and -1 for descending).

You can chain limit() to fetch a specific number of documents:

```
> db.itunes.find({type: 'person'}).limit(2)
```

There's also skip() to help with pagination:

```
> db.itunes.find({type: 'person'}).limit(2).skip(2)
```

Write a query to find the 3 tallest persons and paste it below:

```
db.itunes.find({type:'person',
heightInched:{$gt:50}}).sort({heightInched:1}).limit(3)
```

Aggregate Functions

Just like SQL, you can perform aggregate functions on results in MongoDB. For example, you could get an array of the distinct person heights:

```
> db.itunes.distinct('heightInches')
[ 70, 71, 72, 74 ]
```

Get an array of distinct person names and paste it below:

```
"Rick",
"Tatiana",
"Natasha",
"John",
"Russ",
"Susan",
```

```
"Jacob",
"Edward"
```

Grouping works like SQL's GROUP BY clause:

```
> db.itunes.group({cond: {heightInches: { $exists: true }}, key: {type:
true}, initial: {totalHeight: 0, count: 0}, reduce: function(obj,prev){
prev.totalHeight += obj.heightInches; prev.count++; }, finalize:
function(out){ out.avgHeight = out.totalHeight / out.count; } })
```

This looks bad, but not if you reformat it:

```
db.itunes.group({
    cond: {heightInches: { $exists: true }},
    key: {type: true},
    initial: {totalHeight: 0, count: 0},
    reduce: function(obj, prev){
        prev.totalHeight += obj.heightInches;
        prev.count++;
    },
    finalize: function(out){
        out.avgHeight = out.totalHeight / out.count;
    }
})
```

Look at each of the arguments in turn:

- cond: This is the condition, like the WHERE clause or what you've used in a find(). Here, we're just making sure that we only look for documents that have a heightInches key.
- key: This is like the GROUP BY clause and defines which fields you'll have left over for keys.

- initial: MongoDB lets you give your Reduce function a starting object.
- reduce: This is the Reduce function. But instead of having to loop over an array of values, MongoDB calls your Reduce function for each and passes in the current object (obj) and your state variables (prev) that you set up in the initial key.
- finalize: Since you can't finish your calculations in the Reduce function because it will be called many times, you can add a function that will run once after all of the objects have been reduced.

Extend the query above to find and return the minimum and maximum heights:

```
db.itunes.group({
   cond:{heightInched:{$exists:true}},
   key:{type:true},
   initial:{minHeight:100, maxHeight:0},
   reduce: function(obj, prev){
      if(obj.heightInched < prev.minHeight){
          prev.minHeight = obj.heightInched;
      }
      if(obj.heightInched > prev.maxHeight){
          prev.maxHeight = obj.heightInched;
      }
   },
   finalize: function(out){out.minHeight; out.maxHeight;}
})

[ { "type" : "person", "minHeight" : 48, "maxHeight" : 74 } ]
```

Deleting Documents

The remove() method takes the same arguments as the find() function, but deletes anything it matches. For example, to delete the Rick document:

```
> db.itunes.remove({name:'Rick'})
> db.itunes.find({name:'Rick'})
```

Write a query to remove all of the person-type documents and then a second query to show they are gone. Paste both below:

```
db.itunes.remove({type:'person'})
db.itunes.find({type:'person'})
```

Bulk Import

MongoDB comes with some import tools, but we're going to import your iTunes Library. You can download Rick's importer from:

http://cyberneticsimpleton.com/adb/itunes2db.jar (Also in your repo)

Open a third Terminal to the directory where you saved the JAR. You can run the JAR from the command line to get its usage instructions:

```
$ java -jar itunes2db.jar
```

In this case, you're importing to MongoDB, and the default options are fine:

```
$ java -jar itunes2db.jar mongodb
```

The importer will display quite of bit of status information while it imports, but the entire process shouldn't take more than a few minutes. Once it is done you can close that Terminal window and switch back to the MongoDB console.

Get a count of documets in your database. Paste the query and result below:

This importer has set up all of the documents with a type key. Get an array of distinct type values. Paste the query and result.

```
db.itunes.count()
1722
```

Write a find() query to fetch the track with the longest duration (TotalTime). Paste your query and results below:

```
db.itunes.find().sort({TotalTime:-1}).limit(1)

{
    "_id" : "track:e53948059ba9a5a0",
    "PlayCount" : 3,
    "LibraryFolderCount" : 1,
    "DateAdded" : [  2012,  4,  5,  23,  29,  5 ],
    "SampleRate" : 44100,
    "Location" :
"file://localhost/Users/tatianakerick/Music/iTunes/iTunes%20Media/M
usic/Unknown%20Artist/Unknown%20Album/Rain.mp3",
    "TrackType" : "File",
    "PersistentID" : "E53948059BA9A5A0",
    "TrackID" : 1199,
    "type" : "track",
    "Kind" : "MPEG audio file",
    "TotalTime" : 900022,
    "Name" : "Rain",
    "DateModified" : [  2012,  4,  5,  23,  28,  53 ],
    "BitRate" : 192,
    "PlayDateUTC" : [  2012,  4,  6,  9,  23,  2 ],
    "FileFolderCount" : 5,
    "PlayDate" : NumberLong("3420854582"),
    "Size" : 21575082
}
```

Write a group() query to fetch the count of documents for each type. Don't overthink this! Hint: You don't need a cond or a finalize, but you do need a key. Paste the query and results below:

```
db.itunes.group({
   key:{type:true},
   initial:{count:0},
   reduce: function(obj, prev){
      prev.count++;}
   })
```

```
[{
   "type" : "library",?
   "count" : 1
 },
 {
   "type" : "track",
   "count" : 1403
 },
 {
   "type" : "album",
   "count" : 174
 },
 {
   "type" : "artist",
   "count" : 144
 }]
```

Closing Out MongoDB

You can quit the mongo console with the exit command:

```
> exit
bye
```

You can tell the mongod service to stop by typing Control-C into its Terminal. It will take a second or two to shut down gracefully and should drop you back a command prompt.

Completing the Assignment

1. Go back through this document and ensure that you've filled in all of the answers, and that the orange "Answer" style has remained applied to all of them.

2. Save this document as a PDF with the following name:

   /homework/yourname-mongodb-1.pdf

3. Add the PDF file to the repository and commit it.

4. At the beginning of the next class, push your repository changes up to Git