**Lab 6 : NoSQL Databases**
**CMSC 362**
**Spring 2014**

**Due Monday, Apr. 21, 2014**

The purpose of this lab is to walk you through the basics of using two NoSQL databases:
MongoDB and Cassandra

## Step 1. Setting Up

Create a new folder named Lab 6. In that folder, create a file named "lab6.js".

We are going to put our queries and data definition statements in this file. Mongo uses Javascript as its interface language, so we are going to put some basic javascript code in this file. You can then run your script by typing:

mongo lab6.js

Mongo has a really nice import tool that will allow us to pull in data from a .json file or from a .csv spreadsheet.

Download the file *all_050_in_51.P1.csv* from the course web site. To save yourself some time, you can grab it with this wget command:

```
wget http://marmorstein.org/~robert/Spring2014/cs362/all_050_in_51.P1.csv
```

Then import it into a "census" collection in Mongo by typing:

```
mongoimport --type=csv --collection census all_050_in_51.P1.csv -f
GEOID,SUMLEV,STATE,COUNTY,CBSA,CSA,NECTA,CNECTA,NAME,POP100,HU100,POP100.2000,HU100
.2000,P001001,P001001.2000
```

The type field specifies that we are pulling in a CSV (comma separated values) file. The collection identifies the "collection" or "table" that we should import the data into. The "-f" flag gives names to each column of the spreadsheet.

## Step 2. A simple query

In your lab6.js file, add the following code:

```
print("Query 1");
query = {
        NAME: 'Hanover County'
}

results = db.census.find(query);
while (results.hasNext())
{
        printjson( results.next() );
}
```

Then type:

mongo lab6.js

to run it. This creates a query that matches the census information for Hanover County. It then loops through all records that match and prints out a nicely formatted Json object for each one. We only need the loop because we're running this through a script. If you were in the interactive Mongo shell, you could just run the query and it would automatically output to the screen.

Now copy and paste this code to the bottom of the file to create a new query. Change the print statement at the top of the new query to “Query 2”. Then change the query so that it prints out records for counties with the name “Prince Edward County”.

**Step 3. A more advanced query**

Instead of matching a specific county, let's find all counties with a population greater than 30000 people.

Add this query to your lab6.js file:

```
print("Query 3");
query3 = {
        POP100 : { "$gt" : 30000 }
}

results = db.census.find(query3, {NAME: 1});
while (results.hasNext())
{
        print( results.next()['NAME'] );
}
```

Notice that we use the $gt operator to match all records greater than 30000. Since this is a lot of information, we want to print only the name field. We handle that in two ways:

1. We include a list of fields to “project” as the second argument to find.
2. We use Javascript array notation to only print out the name field as a string instead of an entire JSON record

Since this produces a lot of data, you may want to pipe the output of mongo through the “less” command.

mongo lab6.sql | less

**Step 4. Adding data**

Let's add a record to the census data. Add this to the bottom of your lab6.js file:

```
db.census.insert( { NAME: 'Imaginary County', POP100: 800000 })
```

This creates a new entry with name “Imaginary County” and a population of 800,000 in the collection.

```
print('Query 4');
results = db.census.find(query3, {NAME: 1, POP100: 1}).sort({POP100: -1});
while (results.hasNext())
{
        record = results.next();
        print( record['NAME'] );
        print( record['POP100'] );
}
```

Notice that we reuse the query from step 3, but made a few changes to the printing loop so that we could project TWO fields (the name and the population. We also sorted by population (in descending order).

**Step 5. Deleting Records**

Let's remove our imaginary county. Add: db.census.remove( {NAME: 'Imaginary County'})

Then run Query 4 again (but label it “Query 5”).

**Step 6. Cassandra**

I've set up a Cassandra cluster in the lab. You can type:

```
nodetool status
```

To see all the servers that are accessible from your host. The key field is the “status” which should be “UN” (up, normal) for all nodes.

You can now access Cassandra through the cqlsh interface. Create a file named lab6.cql. In that file, add the following lines of code:

```
CREATE KEYSPACE census_keys
WITH replication=    { 'class' : 'SimpleStrategy', 'replication_factor': 2 };

USE census_keys;

CREATE TABLE counties(GEOID int, SUMLEV int, STATE int, COUNTY int, CBSA text, CSA
text, NECTA text, CNECTA text, NAME text, POP100 int, HU100 int, POP1002000 int,
HU1002000 int, P001001 int, P0010012000 int);
```

You can then run your code like this:

cqlsh < lab6.cql

This should fail, because we haven't specified a primary key. Unlike relational databases, column-based databases HAVE to have a primary key to hash on. Notice that our types MUST be lowercase, too.

Modify the CREATE TABLE line to make “name” the PRIMARY KEY.

**Step 7. Adding data**

We can now insert data into our table. Add these lines to your file:

```
INSERT INTO counties(NAME, POP100) VALUES ('Imaginary County', 800000);
INSERT INTO counties(NAME, POP100) VALUES ('Real County', 8);
```

Then add this query:

```
SELECT * FROM counties;
```

Test to see that it works. Now add this query:

```
SELECT NAME
FROM counties
WHERE POP100 = 800000;
```

The query will fail because pop100 isn't yet indexed.

Insert this line before the select:

CREATE INDEX ON counties(POP100);

Then run your query again.

**Step 8. Adding more data**

Both relational databases and Cassandra allow you to “insert” several records at once from a file. Let's import the data from our .csv file. Add this to the bottom of your lab6.cql file:

```
COPY counties(geoid, sumlev, state, county, cbsa, csa, necta, cnecta, name, pop100, hu100,
pop1002000, hu1002000, p001001, p0010012000)
FROM 'all_050_in_51.P1.csv';
```

This will load the entire CSV file into the database.

Now add this query to your lab6.cql file:

```
select * from counties WHERE name IN ('Buckingham County', 'Cumberland County', 'Prince
Edward County');
```

Test your work.

**Glitter**

For glitter, I want you to find one feature/command of either MongoDB or Cassandra that we didn't cover in the lab. Create a file named README.TXT in your Lab6 folder which contains a short (no more than one paragraph) explanation of that feature and gives a short example which applies it to the census data.

**Submitting**

As usual, save your work and upload a tarball of your Lab6 folder file to
http://narnia.homeunix.com/~robert/submit.

```
cd ..
tar czvf Lab6.tar.gz Lab6/
```