**Lab 7 : NoSQL Databases**
**CMSC 362**
**Spring 2017**

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 7. In that folder, create a file named "lab7.js". Mongo uses Javascript as its interface language, so we are going to put some basic Javascript code in this file. This code will contain both queries and data definition statements.

You can run the code by typing:

mongo lab7.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/Spring2017/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,P
001001,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 lab7.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 lab7.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 add code to the bottom of your lab7.js file that:

1. Prints “Query 2”.
2. Performs a query that prints out the record for “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 lab7.js file:

```
print("Counties with population over 30,000");
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. This list is enclosed in curly braces and passed to the “db.census.find” function.

2. We use Javascript array notation (square brackets) 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 when you test:

mongo lab7.js | less

Let’s be a little more sophisticated. Add this:

```
print('Sorted Counties with Population over 30,000');
results = db.census.find(query3, {NAME: 1, POP100: 1}).sort({POP100: -1});
while (results.hasNext())
{
        record = results.next();
        print( record['NAME'], 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, which is why we gave it the value -1).

Now add a query that lists the names and populations of all counties with population less than 30000 sorted by POP100 in ascending order.

**Step 4. Adding data**

Let's add a record to the census data. Add this to the bottom of your lab7.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.

Add counties named “Flatland” and “Perelandra” with populations of 20 and 9,000,000 respectively.

**Step 5. Deleting Records**

To remove a record, we use the “remove” function.

Let's remove our imaginary county.

Type:

```
db.census.remove( {NAME: 'Imaginary County'})
```

Then run your population queries again. Does “Imaginary County” show up? Does “Perelandra?”

**Step 6. Cassandra**

MongoDB is a document-driven NoSQL database, which is designed to allow you to use a huge cluster of servers to improve performance of simple queries. Another NoSQL option is Cassandra, a column-driven database which uses distributed computing to improve both scalability and fault tolerance.

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.

Unfortunately, the Cassandra cluster in the lab doesn’t listen on the local interface. This means that to communicate with it, we have to tell the client program to use the external IP address of your system. Here’s a quick way to set that up once so you don’t have to do it again. Run this command from the terminal:

```
export CQLSH_HOST=$(ifconfig | tr '\t' ' ' | tr -s ' ' | grep '192.168' | cut -d' ' -f3)
```

This will work as long as you remain in that terminal window. If you open a new window or log out and back in, you’ll need to run the export command again.

You can now access Cassandra through the cqlsh interface. Create a file named lab7.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 < lab7.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. Rerun the cqlsh command. You can ignore any error about “census_keys” already existing.

**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.

When you it working, 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. When you have it working, write a query that lists the name and POP100 of all counties with POP100 less than 30,000.

**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 lab7.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' WITH HEADER=true;

This will load the entire CSV file into the database.

Now add this query to your lab7.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 Lab7 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.

You can find documentation on MongoDB at https://docs.mongodb.com/ and documentation on Cassandra at http://cassandra.apache.org/doc/latest/.

**Submitting**

As usual, save your work and upload a tarball of your Lab7 folder file to
http://narnia.homeunix.com/~robert/submit.

```
cd ..
tar czvf Lab7.tar.gz Lab7/
```