# Assignment 3: Database indexing and web programing

## Version 0.1

Code shown bellow might not run as is on csoracle/csgrads1, and might need some minor changes to compile and run. It is included for illustration of the NodeJS API use.

### Indexing in Databases

Majestic Million CSV (https://blog.majestic.com/development/majestic-million-csv-daily/) is an example of a million record databases. According to the website, it "a list of the top 1 million website in the world, ordered by the number of referring subnets."

As you might imagine, putting this data in a database table, and performing some database queries on it would be interesting. But as you also know, we might get a performance hit when using such data. The solution would be to indexing some of the field. In addition to class material, refer to this [A]

### Web Programing with NodeJS

We have seeing class how to use Nodejs, in addition to several modules, in order to query database tables, and return data as a webpage. More in following sections

### Measuring Execution Time.

1- In sqlplus ORACLE utility, you can measure the execution of a single query by executing "set timing in". See [B]for syntax help.
2- In *nix (Unix/Linux), you can measure execution time of single commands using the command "time", followed by the command to execute. See [C] for how to use it. In this assignment, we will use "real time" as measurement for the execution time.

**Part 1: Indexing Business**

1- Download Majestic Million CSV file, and save it in csgrads1. You will call it majestic.csv
2- Using "more" utility, take a look into the file, and build an appropriate sql schema for it. Write the schema creation file in a file called "majestic_schema.sql". The schema file should create four identical tables called "MAJESTIC", "MAJESTIC_INDEX1", "MAJESTIC_INDEX2", and "MAJESTIC_INDEX3".
3- Add to "majestic_schema.sql" sql statements to create an index on the "TLD" field of table "MAJESTIC_INDEX1", field "RefSubNet" of table "MAJESTIC_INDEX2", and both "TLD" and "RefIPs" fields of table "MAJESTIC_INDEX3",
4- Load the data from the CSV file downloaded from Majestic Million CSV to all 4 tables created in 3). Use "time" Unix command to measure execution time. Put the command with time in a script called load_data. Report in your Readme file how long it takes to load the data in each table. Comment on the values you are getting, and give a possible explanation.
5- Write the following sql queries (each one in a .sql file), and run them on each table, while recording the execution time. Comment on the execution time differences.
    a. Return domain name, global rank and RedSubNet for all record where the domain is based in India (.in): query_india_domains.sql
    b. Return all sites (domain names) that they have 100k or more RefSubNet: query_top_domains.sql
    c. A query that returns the ranking of "wikipedia.org": query_wikipedia.sql

6- Write a sql script to delete the tables, index files and data loaded part of this exercise. This is very important. Make sure that you execute it when done with this assignment. If our system admin complains that you left this huge data in the database, points will be deducted from your assignment grade: clean_data.sql

Part 2: Lets put it on the Web:

We are going to program a simple web interface for our Majestic Million database, using NodeJS.

1- Program the artifacts (both entry and view pages) to let the user query a domain name, then get its ranking. If domain does not exist, need to return an appropriate message
2- Program the artifacts (both entry and view pages) to let the user query domains belong with certain TLD. For example, if the user input India (.in), the page will return the GlobalRank and Domain name of each domain matching that TLD.
3- For each for the queries above, make a separate context for hitting each of the 4 tables created. For example, the url (cagrads1.utdallas.edu:5555/majestic/domain/no_idx) for query 1, hitting "MAJESTIC" table, and (cagrads1.utdallas.edu:5555/majestic/domain/idx1) for query 1 hitting "MAJESTIC_INDEX1" etc..

To submit: a single majestic.js, with all the views and html needed.

*Some advises:*

- With any programing job, use an iterative programing approach. Test each iteration/step separately to be able to isolate any bug/malfunction. Start small, and then make it bigger with each iteration.
- Always test first with queries that return a small amount of data. Do not start with queries for the ".com" TLD!! Make use of the right node.oracledb API.

Part 3: Testing how fast if your website?

Now that the middle tier is built, lets try to test it.

- **CURL**

"curl" [D] is a *nix utility that allows sending HTTP POST/GET to websites. For example, if you execute the following command, testing "bookapp.js":

```
curl --data "author=J. K. Rowling" http://csgrads1.utdallas.edu:5000/search
```

you will get the following reply:

```
<h4>Book Info:</h4>
<ul>
  <li>Book Title: Harry Potter and the goblet of fire</li>
  <li>Book Author: J. K. Rowling</li>
  <li>Book Price: 25.95</li>
</ul>
<form action="/author" method="get">
    <br><br>
    <input type="submit" value="Search again">
</form>
```

1- Execute "curl" with "time" to measure POST requests for each query developed in part2. Collect timing data and comment on it in the project README: query1_curl, query2_curl, query3_curl

- **Apache JMeter**

Measuring one request is something, but making the website can handle several concurrent clients is another. Imagine the load that Amazon.com handles everyday. Can your website handle that load?

Apache JMeter is a tool that allows load testing web sites. It is graphical tool that can, among other things:

- Automate the entry of HTTP POST parameters, then issue an HTTP post to a web page
- Simulate multiple users accessing the same web page.
- Collect results and response times.

We want to use JMeter to test our newly built web interface built in Part2. Some facts about the desired use of JMeter:

1- Install JMeter in your laptop/lab machine, not in any CS unix machine
2- Refer to the Internet guides on how to use JMeter. A simple example tutorial can be accessed though [E] . I am attaching a simple test plan that runs against "bookapp.js", called bookapp_testplan.jmx
3- The desired test plan will be to test the pages developed in Part2. Start with query1 (enter a domain, return its rank). Simulate only 1 user first, then 10, then 50. Run it multiple times? What do you notice? Any errors? Why do you think it is happening? Be specific.

4- In order to handle the limitation discovered in 3-, look at the new implementation of "bookapp.js", called "bookapp2.js". Take a look, and notice the difference. Google is your lifeline.
5- With the new implementation, see how many concurrent clients you can push through. Collect performance data of the max number of clients you can push, and compare it with 1 client. Compare for both part2 queries: query1_testplan.jmx, query2_testplan.jmx, query3_test_plan.jmx

**What to Submit**

In a zip file properly named, collect all the artifacts marked with a box. Do not forget a well-documented README.txt.

References:

[A] http://www.java2s.com/Tutorial/Oracle/0180__Index/CreatinganIndex.htm
[B] http://www.dba-oracle.com/t_measure_sql_response_time.htm
[C] https://en.wikipedia.org/wiki/Time_(Unix)
[D] https://en.wikipedia.org/wiki/CURL
[E] http://jmeter.apache.org/usermanual/build-web-test-plan.html