M149/M116: Database Management Systems
University of Athens
Deprt. of Informatics & Telecommunications

Programming Project II
Today's Date: December 13^{th}, 2019
Due Date: January 16^{th}, 2020

PREAMBLE

In this project, you will design, implement and demonstrate a *NoSQL*–database solution to manage *server logs*. You will also provide access to your database through a *REST API*.

A server–log is a file that consists of a list of activities a machine has performed and recorded. For example, a web–server log comprises a list of page requests. The database you will create, termed `NoSQL-311CI`, will be populated with data already collected and available at:
`https://hive.di.uoa.gr/M149/logs.tar.gz` and
`https://my.pcloud.com/publink/show?code=XZwXH8kZHhp9paneeokwU13NIhUJFz6xnQGy`.
Pertinent data of the above data set have to be *stored* in `NoSQL-311CI` and can be used when the system becomes operational.

PROBLEM DESCRIPTION:

Your database should include information about the server logs in a *normalized form* and should provide various queries and/or aggregations on logs recorded. Moreover, the database should hold data related to the registered users of the web application and their actions. Below is a description of the general guidelines of the project. While you are working, keep in mind that these items make up a minimum set of requirements. Hence, you may extend the scope of your work as you see it appropriate and/or interesting.

Server Log Data: Different services run by the OS often maintain a history of their activities scattered in a number of files and formats. A typical web server will add an entry to its access–log for every request it services, featuring among else the IP address of the client, the request line, and a time stamp.

`NoSQL-311CI` will allow for its users to perform powerful analysis on various server logs of different services and formats. In this context, registered users can analyze the data provided either though a set of (canned) queries or via having read-access to the database (for the more sophisticated).

There are *3* sets of log files in `.txt` format:

1. access.log: This is a web–server log that features:
   - IP address of the client (remote host),
   - User ID of the person requesting the document as determined by HTTP authentication. If the document is not password protected, this entry will be '-',
   - Time stamp,
   - HTTP method (GET, POST, etc),
   - Resource requested,
   - HTTP response status (200, 400, etc),

- Response size,
- Referer,
- User agent string.

2. HDFS_DataXceiver.log:
   - Time stamp,
   - Block ID,
   - Source IP,
   - Destination IP,
   - (Optional) Size,
   - Type (receiving, received, served, etc).
3. HDFS_FS_Namesystem.log:
   - Time stamp,
   - Block ID(s),
   - Source IP,
   - Destination IP(s),
   - Type (replicate, remove).

Using `MongoDB` Community Server,[1] create the *collections* of your database, and insert *all* pieces of incident information. You have the freedom to define the collections as per your own choice/design. It is imperative that no information is ever deleted even in light of modification(s); obsolete information should be *automatically* staged to collection(s) that maintain it for the time to come.

Admin Data: should include the upvotes for each administrator, accompanied with the username and e-mail and telephone number of the administrator.

REST API: You should create an API method for each of the following queries, as they are particularly useful to the context of `NoSQL-311CI`. The responses of your API should use the JSON format:

1. Find the total logs per type that were created within a specified time range and sort them in a descending order. Please note that individual files may log actions of more than one type.
2. Find the number of total requests per day for a specific log type and time range.
3. Find the three most common logs per source IP for a specific day.
4. Find the two least common HTTP methods with regards to a given time range.
5. Find the referers (if any) that have led to more than one resources
6. Find the blocks that have been replicated the same day that they have also been served.
7. Find the fifty most upvoted logs for a specific day.

[1] `https://www.mongodb.com/download-center/community`

8. Find the fifty most active administrators, with regard to the total number of upvotes.
9. Find the top fifty administrators, with regard to the total number of source IPs for which they have upvoted logs.
10. Find all logs for which the same e-mail has been used for more than one usernames when casting an upvote.
11. Find all the block ids for which a given name has casted a vote for a log involving it.

In addition, you should provide API methods for updating the database. Updates may include *i)* inserting new logs and *ii)* casting of upvotes. In case the same administrator casts a vote for the same log a second time, the vote should be rejected.

It is particularly important that you make sure that the above queries are executed *efficiently* by creating the appropriate *collections* and adding the necessary *indices*.

IMPLEMENTATION ASPECTS:

You will use `MongoDB` as your database in this project. In addition, you may use any language/framework you want including `Spring-boot`, `Laravel`, `Django`, `Ruby On Rails`, `Flask`, `Express.js` `PHP, Java, Python, PHP`, etc. As a matter of fact, the use of the `https://spring.io/projects/spring-boot` is encouraged.

You may work in any environment you wish but at the end you should be able to demonstrate your work (with your notebook or via remote access to a machine).

OVERVIEW OF PROJECT PHASES:

There are two distinct phases in this project that you will need to work on:

◇ Database

In this phase, you are required to decide upon the database collections you will use and populate them with data. For the administrator data you are expected to generate the data yourselves. To this end you could employ an open-source library such as `faker`[2][3], `java-faker`[4], and `datafactory`.[5] You are expected to generate enough data so that at least $\frac{1}{3}$ of the logs have at least one upvote, and no administrator has more than 1000 upvotes.

◇ REST API

You will use the framework of your choice (such as `Spring-boot`, `Laravel`, `Django`, `RoR`, `Flask`, `Express.js` etc.) to provide an API that offers access to the database and enables users to query and update the database.

[2] `https://github.com/stympy/faker`
[3] `https://github.com/joke2k/faker`
[4] `https://github.com/DiUS/java-faker`
[5] `https://github.com/andygibson/datafactory`

COOPERATION:
You may either work individually or pick *at most one partner* for this project. If you pick a partner you should let us know who this person is.

REPORTING:
The final *typed* project report (brief report) must consist of:

1. A description of the schema design of the database used along with justification for your choices.
2. The `MongoDB` queries for the required functionality.
3. The code of your *REST API*, preferably through a link to a `git` repository.
4. Sample responses for each query.

Finally as mentioned earlier, you will have to demonstrate your work.

SUPPORT:
Panagiotis Liakos (`p.liakos+@-di.`) will be escorting this assignment, fill in questions, and carry out the final interviews.