Assignment 1: Building a cache for MongoDB

* **Due** Monday by 23:59
* **Points** 12
* **Submitting** a file upload
* **Available** Nov 30 at 0:00 - Dec 27 at 23:59 28 days

Caches are often used to improve database query performance, especially to exploit temporal and spatial locality of user queries. In this assignment, you are expected to build a front-end cache that demonstrates the benefits of caching to improve end user performance of a Cloud hosted MongoDB.

**Scenario:**

The MongoDB instance on Cloud contains an Airbnb dataset with accommodation listing. Users search the database using MongoDB queries. A cache stores the results of the user queries. A user’s MongoDB query is first analyzed by the cache lookup to figure out if it can be served from the local cached data. If the query cannot be served from cache, then it is sent to the Cloud instance which leads to higher overheads. In case of significant cache hits across several queries, the average query response time should improve.

Let’s take an example. A user query requests for all accommodation names that contain the word “beach” OR the property type is “house”. The query runs on the Cloud instance for the first time. The result of the query is cached with some meta-data to describe the cached data. For e.g. each cache entry’s meta-data captures that the record contains the “name” attribute and the filter condition that matched, i.e. whether the “name” contains “beach” OR “property\_type”=”house” OR both. This makes it easy to find subset matches in the cache, e.g. all properties of type “house” even if the name doesn’t contain “beach”. Now, when the same or a different user makes a query for properties of type “house”, the query result can be served from the cache because the cache already contains the data for matching accommodations (cache hit). Notice that the query need not be exactly the same and the meta-data helps to find out a subset match. The cache lookup logic can find out if the cached data “contains” the result for the new query. The cache lookup logic also needs to be correct to ensure that when data is not in the cache it is a cache miss. For a miss, the query is executed on the Cloud DB and result returned, as well as the cache is updated with data with the appropriate meta-data.

**Limited scenario for assignment:**

Your software implementation needs to have 3 components:

1. A **cloud instance of MongoDB** with the Airbnb data set (aka collection)
2. A **cache module** that you have to develop and run on your machine / laptop. It will connect to the Cloud MongoDB instance and run given MongoDB queries to fetch data. It provides a query interface to the end-user application (discussed next) to provide user MongoDB queries.
3. An **end-user application** that reads a set of MongoDB queries from a file and calls the cache module interface to get results back. This file based approach to send queries essentially simulates an actual user sending queries. The cache module can be integrated into this application as a library and the interface to receive a user query can be just a function call. This keeps deployment simple, but limits the cache to a single user and not a common cache that can be used across multiple end-user machines.

The types of user queries can be limited to simple filtering queries. For e.g. fetch sets of accommodation names based on various attribute filters. This simplifies your cache lookup logic to use meta-data and look for subsets of data in cache.

**Here are the specific steps and technical components to build:**

1. Setup an account on cloud.mongodb.com and load the ‘sample\_airbnb’ collection provided with your account. You will be given credentials to run queries on this data from an external program (which in this case will be the cache module you will develop).
2. Test your MongoDB instance and queries using a sample python code provided on discussion forum. You have to use your own credentials from step (1). This code can connect to the DB and run sample queries. You can use this sample code to develop your cache module's interaction with a MongoDB instance.
3. Use any free SQL compatible DB server that can be installed on your laptop, e.g. BerkeleyDB, Apache DerbyDB etc. to serve as the cache storage. You will store cache results in a DB Table and query the cache using SQL. The cache will store query results and some meta-data to help with cache lookup.
4. Query parsing and re-writing: Given a user query is in MongoDB language and cache is in SQL, your cache lookup logic (in the cache module) needs to parse the MongoDB query to create the meta-data, using which you can create an SQL to query the cache for the results. If the result is not there, then the original query goes to the Cloud instance, but the caching module should store the results with the meta-data before forwarding the results.
5. Write your end-user program to take MongoDB queries as input (from a text file) and call the cache module query interface. The interface should return results back to the end-user program but under-the-covers it will do query parsing, re-writing the query in SQL, cache lookup and if required, on a cache miss, run the original MongoDB query and store back the results in cache with meta-data for future lookup. This cache module can be implemented as a library with a method call interface for cache lookup.
6. Demonstrate the effectiveness of the cache using a sample set of queries read from a text file and executed sequentially. For each query, log whether it is a cache hit or a miss and what’s the end-to-end response time.
7. Cache replacement: Enhance your caching module to limit the cache size to a maximum number of entries and include an LRU (Least Recently Used) cache replacement algorithm. So each time the cache runs out of space and a new query result has to be inserted, the oldest ‘used’ entry has to be removed. So each entry must have a timestamp to indicate when it was last used to serve a response to the user. Demonstrate the replacement algorithm with a sample set of queries and a trace log as in step (6). So certain queries that would have been a hit earlier may now be a miss with a limited cache size.

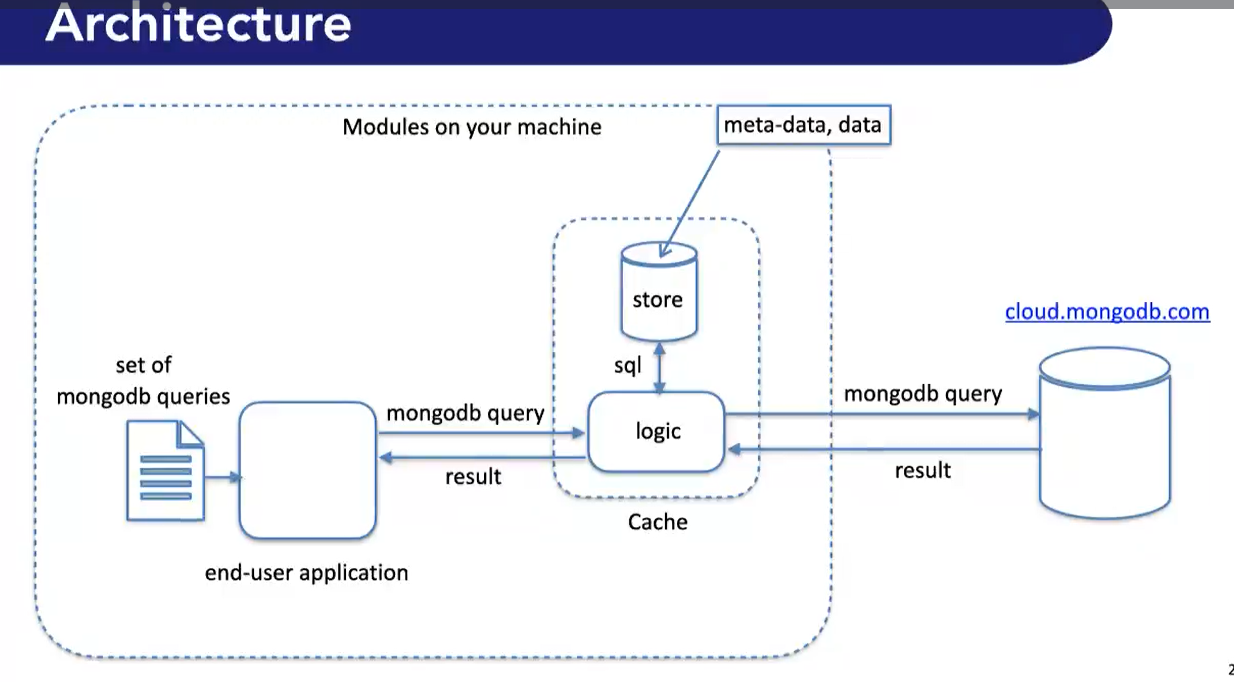
**Important note:**As a team you should focus on an end to end working system that uses the cache. You can make it work for simple queries with filtering attributes only so that the cache can be used for subsets of data and these subsets are easy to identify. It is important to provide the test set of queries you have tried in steps 6 and 7 to assess your solution.

**Mandatory submission material:**

1. Working code with clear instructions to execute using your Cloud MongoDB credentials. You (as a group) may be asked to demonstrate your code and results live as well in an online meeting.
2. Set of test queries in sequence (in a text file) used for demonstrations in steps (6) and (7) along with the performance trace log as specified in the steps.
3. A screen recording of the program execution and logs proving that you have actually tested the code.

**General Notes:**

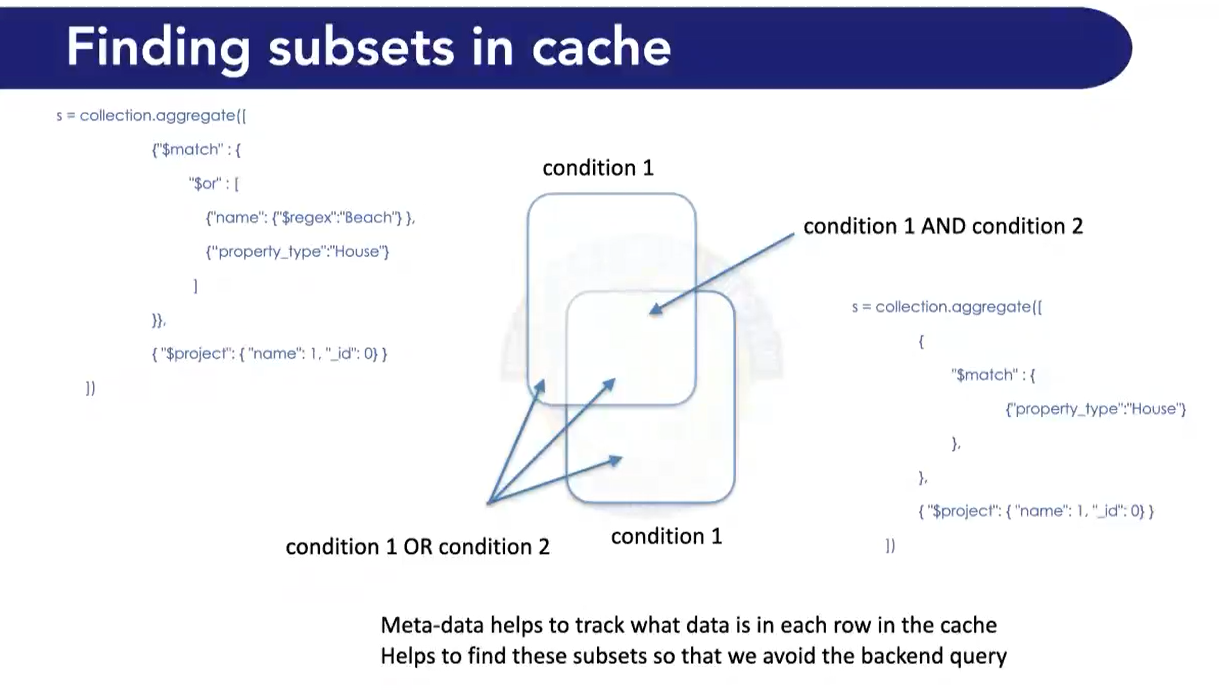
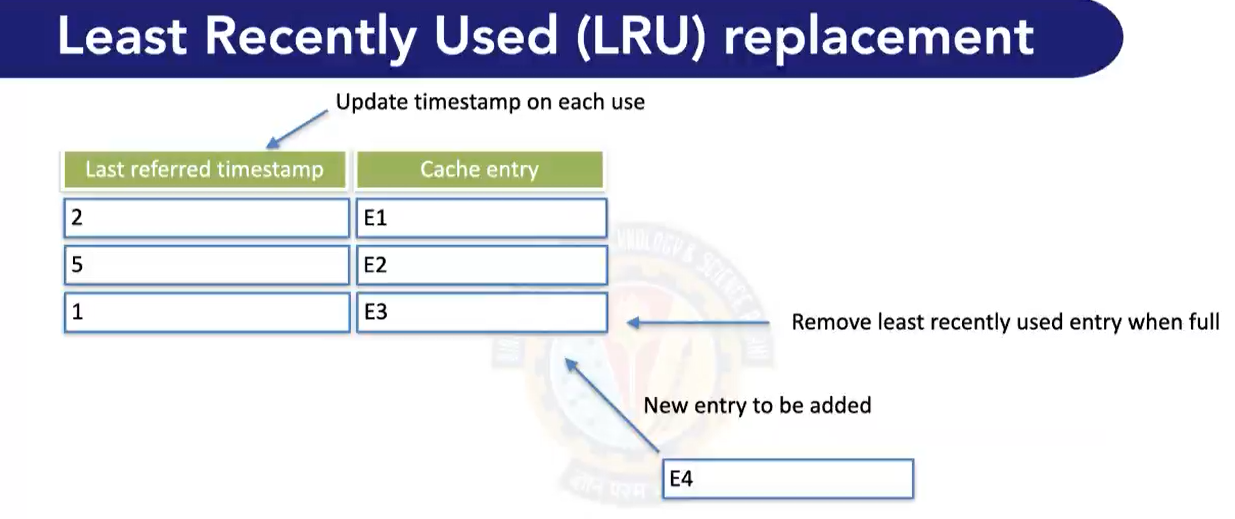
1. Any downloaded code or duplicate copies of code across teams will lead to 0 marks on the assignment.
2. Using Canvas, only the first member of group has to upload the file. No submission over email will be considered.
3. Upload a zip for multiple files and name the file in format like "Grp\_<your\_group\_number>.zip" only. Don't add anything into the file names.
4. Make sure that you upload the file well ahead of deadline. At the last moment, we have seen several groups have face issues while doing the submissions. There will be no exceptions allowed on deadline.
5. Since it’s a group assignment, only one submission is expected from each group. Unnecessarily don’t upload the solution on individual basis. If it’s observed, then a penalty (25% reduction) will be applicable.



query rewriting for local db as end user sent in form of mongodb queries

store meta-data in cache (example: rows from particular area with propertytype house, we can store area & house no need to store exact query)

local db ; RDBMS

need to show output of traces of miss/hit.