# Report

## Introduction:

This report will introduce how MongoDB executes queries and aggregations, particularly how indexes may help to improve execution performance by analyzing 6 questions in the assignment. All the queries and aggregations will import two data sets “tweets\_hurricane.json” and “users\_hurricane.json” as collection name tweets and users respectively. Performing on Robo 3T and eventually put all queries and aggregations together in a JavaScript file.

## Performance analysis of query implementations

**Q1:**

Find the number of general tweets with at least one reply and one retweet in the data set. Note that a general tweet is a tweet with neither a replyto\_id field, nor a retweet\_id field; a reply is a tweet with the replyto\_id field; a retweet is a tweet with the retweet\_id field.

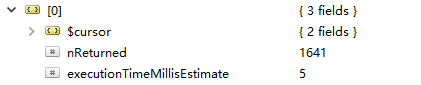


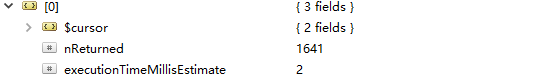
Firstly I used MongoDB aggregation, because Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. This is very suitable for this problem.



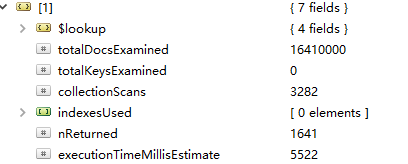
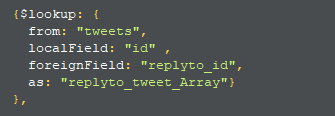
I created 2 indices on replyto\_id and retweet\_id objects and will analyze the difference between using index and not using below.

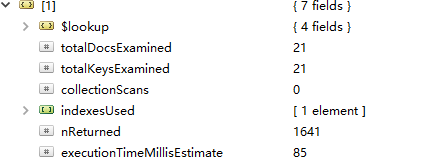




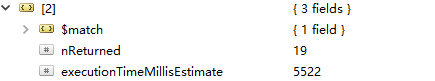
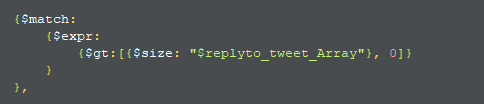


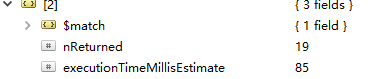
About the $match stage, even though I added index on replyto\_id and retweet\_id, the execution time only 3 millis away. This is because whether index is added or not, this stage here will need to traverse all data. Also the explanation did not show the indexsUsed.



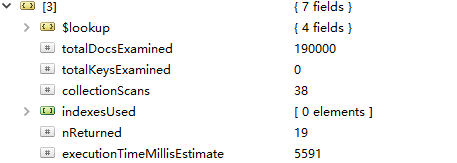
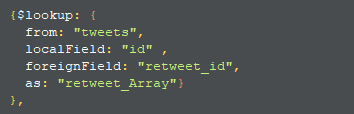


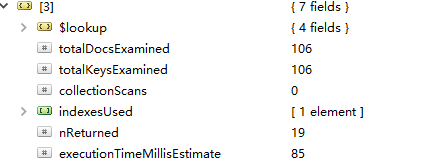
About the $lookup stage, the excuting time before adding index and after has almost 5400 milliseconds difference. This is because Firstly, before adding index, the time complexity of append each replyto\_id to id can be assessed as O(n^2), but after adding index, the data is stored in a B-tree, and the total time complexity of this step can be assessed as O(nlog(n)). So as the data become larger the excuting time gap between the two methods will rapidly increase. Secondly, we can see the totalDocsExamied before creating index is 16410000, this is because for each “id”, it will need to go through almost whole data set, to find the matching “replyto\_id” in average. After adding index, the data in tweets collection are put into entries, so in the secend image, the totalKeyExamied incresed from 0 to 1, it just need to go through the sorted key in a B-tree data structure, this resulted in a significant reduction in time.



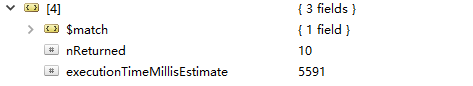
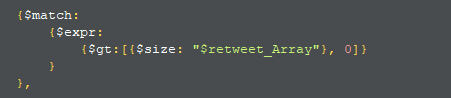


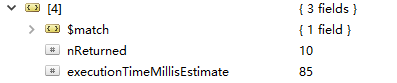
I have no idea why the execute time of both before and after adding index data will equal to the



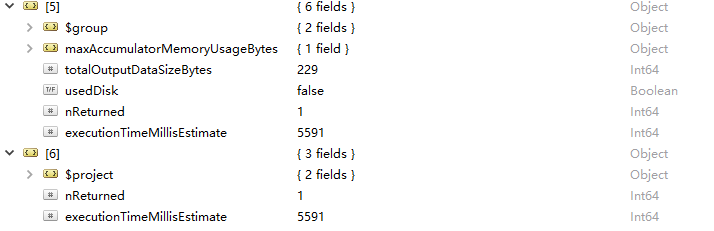


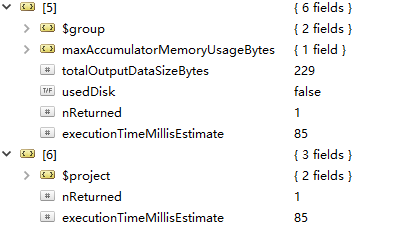
This $lookup stage after creating retweet\_id index, also saved 5500 milliseconds, the reason is same as I explained at the first $lookup stage in this question.





Same reason as the second stage.



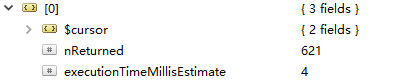


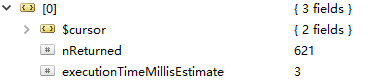
**Q2**

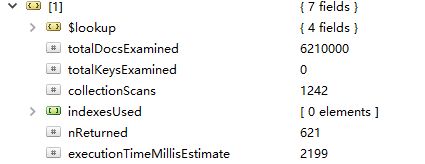
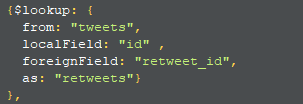
Find the reply tweet that has the most retweets in the data set.

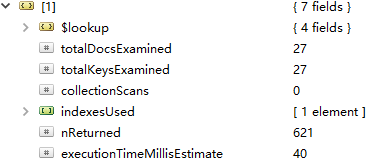


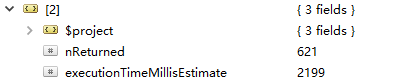
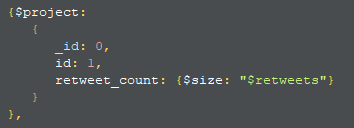
As Q1 said, aggregate is more suitable in this case. And below is the explain() of my using aggregation stages, each three

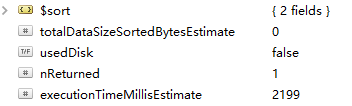
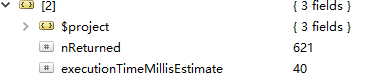


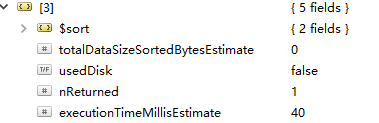














mongo G:/COMP5338/ass1/a1sample.js --eval "args='Ida'"