MILESTONE-1

TASK COMPLETION STATUS

***Pre-Requisites:***

1. Install MongoDB: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>
2. Basic Knowledge of query processing using cmd.
3. Knowledge of Agile Methodology.

**Note**:

* We have spent time to understand use MongoDB Compass a UI from Mongo which allows users to perform the same actions from a UI perspective instead of the cmd.
* In case trying to connect via Compass

Connect to local host on Compass: mongodb://localhost:27017/admin?readPreference=primary&appname=MongoDB%20Compass&ssl=false

***Task1: Data Pre-processing***

***Subtask1: Identify the type of data that we wish to use for this project and research to verify if large volume of data is available.***

* One of the major advantages of MongoDB is its ability to handle large quantities of data and specifically unstructured data.
* To understand and verify this advantage we wanted to identify a dataset that has a substantially large amount of data.
* We began researching online to find free data sets that were primarily containing many records.
* This was done in order to see first hand the advantages of query optimization techniques such as indexing and also their disadvantages
* The results of our research online are the following 4 datasets which we could download. Link to download has been mentioned below

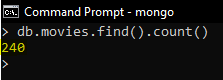
|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Link** | **Rows** | **Columns** |
| 1 | <https://data.edd.ca.gov/Industry-Information-/Current-Employment-Statistics-Mono-County/ru45-rtii> | 7062 | 9 |
| 2 | <https://data.edd.ca.gov/Employment-Projections/Short-Term-Occupational-Employment-Projections/guh4-bakw> | 781 | 18 |
| 3 | <http://eforexcel.com/wp/downloads-18-sample-csv-files-data-sets-for-testing-sales/> | 10,00,000 | 14 |
| 4 | <http://eforexcel.com/wp/downloads-18-sample-csv-files-data-sets-for-testing-sales/> | 20,00,000 | 14 |

***Subtask2: Identify a smaller sub set of chosen data which would be used to populate the Database created***

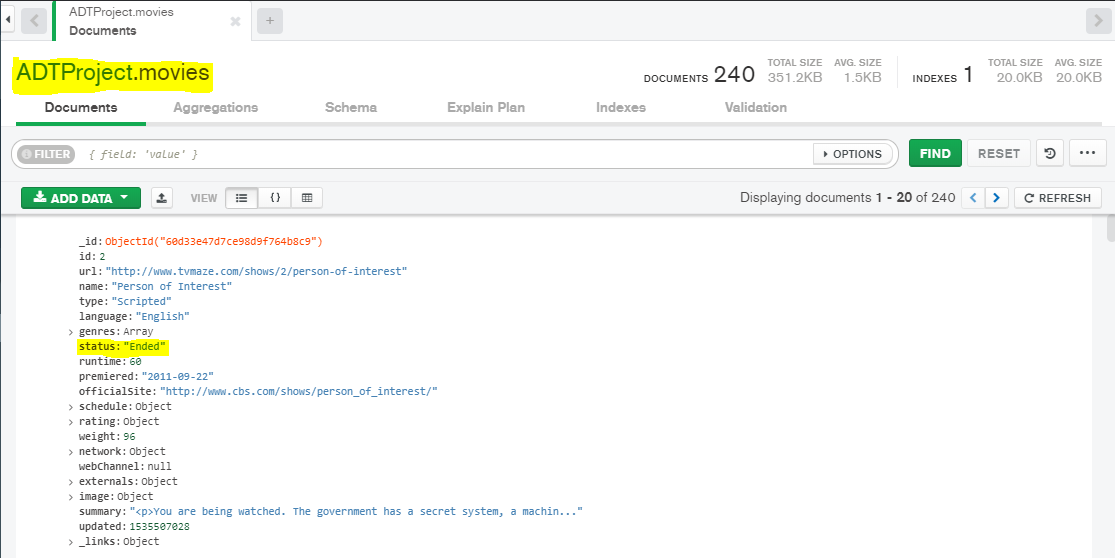
* As a proof of our understanding of the concept, we first wanted to implement a simple index on a small dataset.
* We checked online for a small sized dataset that we could download and came across this simple tv show data JSON which we fell was a good starting point.



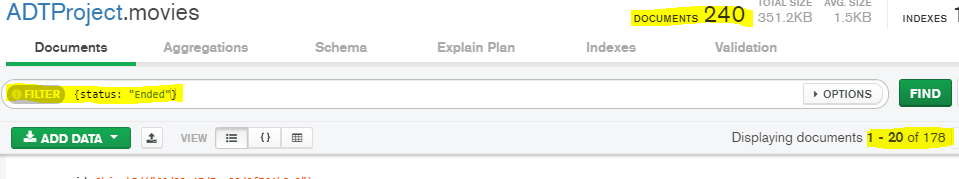
* The next step would be to import this dataset into our MongoDB
* We accomplished this using
  + Navigate to the folder containing the file and then execute the following on cmd
  + mongoimport tv-shows.json -d ADTProject -c movies –jsonArray –drop
  + A couple of key learnings here
    - By default, the import expects only one document and therefore when multiple documents are involved, we have to utilize the –jsonArray.
    - --drop drops the collection if it already exists
    - By default this command appends data to the existing
* The alternative approach to importing data would be to utilize the UI from MongoDB Compass and these steps are discussed in the next section where we import a larger dataset.
* This dataset contains only 240 documents and this can be validated from the cmd by running the following query
  + db.movies.find().count()



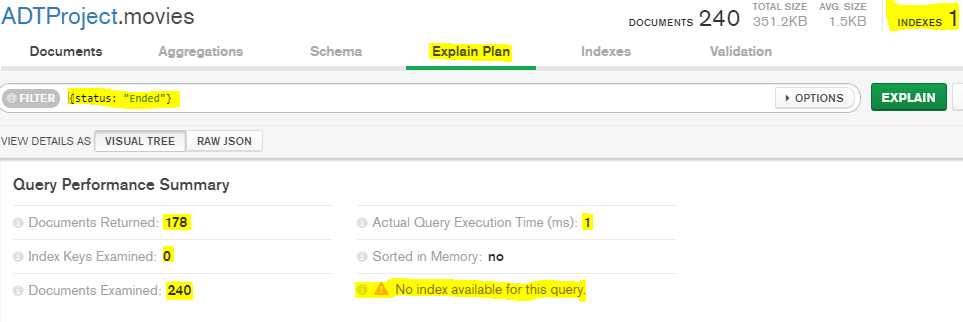
* Alternatively just opening the collection gives you the count and the first 20 records by default on MongoDB Compass



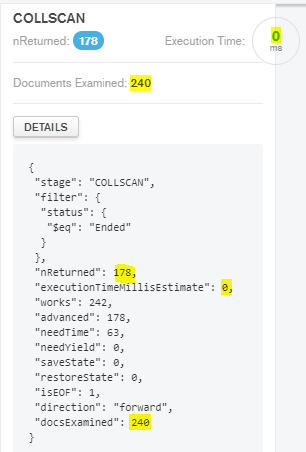
* For a implementation of proof of concept perspective we identified the key “status” as a criteria to run a query on.
* We filtered the results based on {status: “Ended”} as our filtering criteria on Compass.
* The query is as follows
  + db.movies.find(“status”: “Ended”}).pretty()
    - this would return the first 20 and ask user approval for subsequent records to be fetched
    - **Note:** We learned that find returns a cursor in MongoDB and not all records and by default the cursor contains 20 limit. This itself is a optimization in MongoDB
  + Alternatively, we can utilize the MongoDB Compass to enter filtering criteria directly.



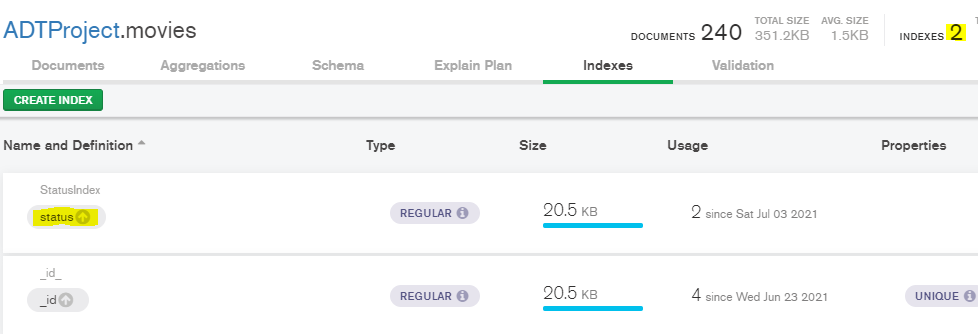
* We can see that there are 170 documents out of 240 which satisfy the criteria
  + **Note:** Note that only 20 records are returned here again due to the cursor implementation we spoke about earlier
* For a better visual representation of the query processing, let us continue to check in Compass
* Click on the “Explain Plan” button on the Compass banner and then click on the “Explain” button to see the major stats behind the query



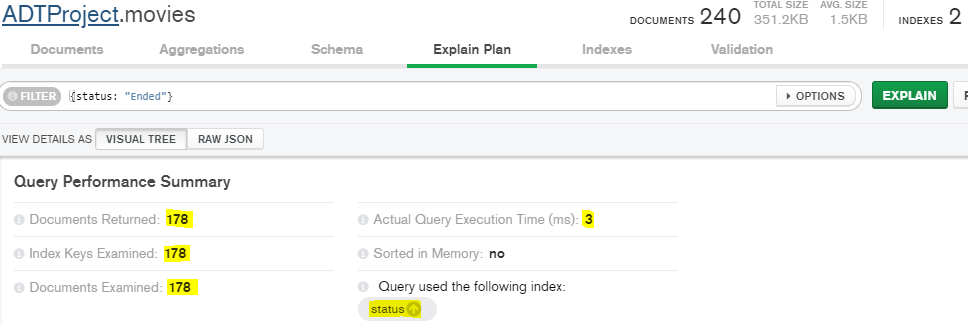
* We can see from the above screenshot that a total of 178 documents were returned out of a total 240. We also see that there is one index and this is the default “\_id” field index.
* Time taken to execute this query is “1 millisecond”.
* If we click on the “details” button under the COLLSCAN banner we get much more details regarding the query being executed



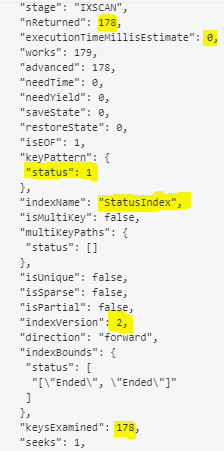
* Navigate to the Index Banner and then create an index on the field “status”



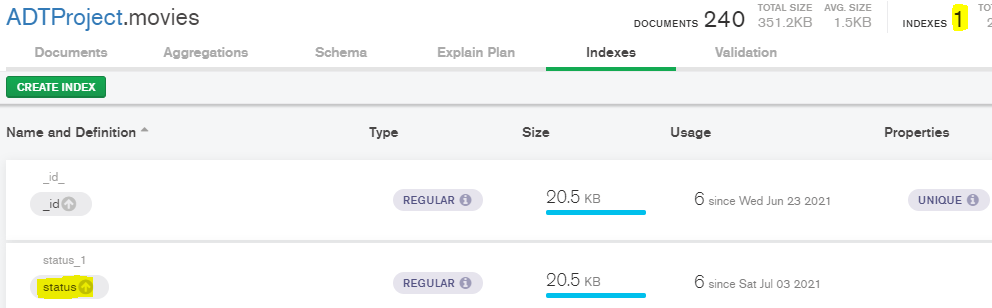
* Execute the same query explain again



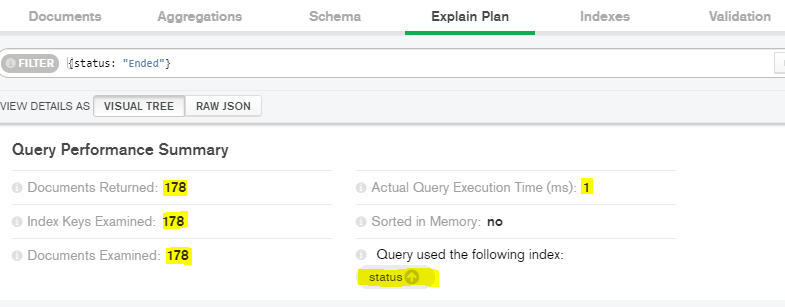
* We can observe from this that Actual query execution time actually took greater time with the index being executed.
* Now if we look at the details under COLLSPAN, we see executionTime is the same



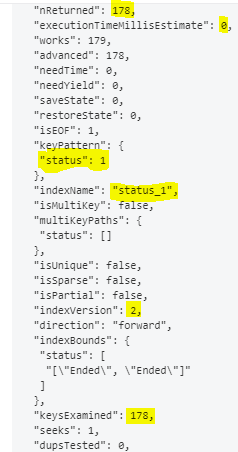
* Requires a lot more research into why actual query execution time took longer
* Let us delete the index and create the index from the cmd using
  + db.movies.createIndex({status:1})



* Now run the explain query from the Compass and see the results



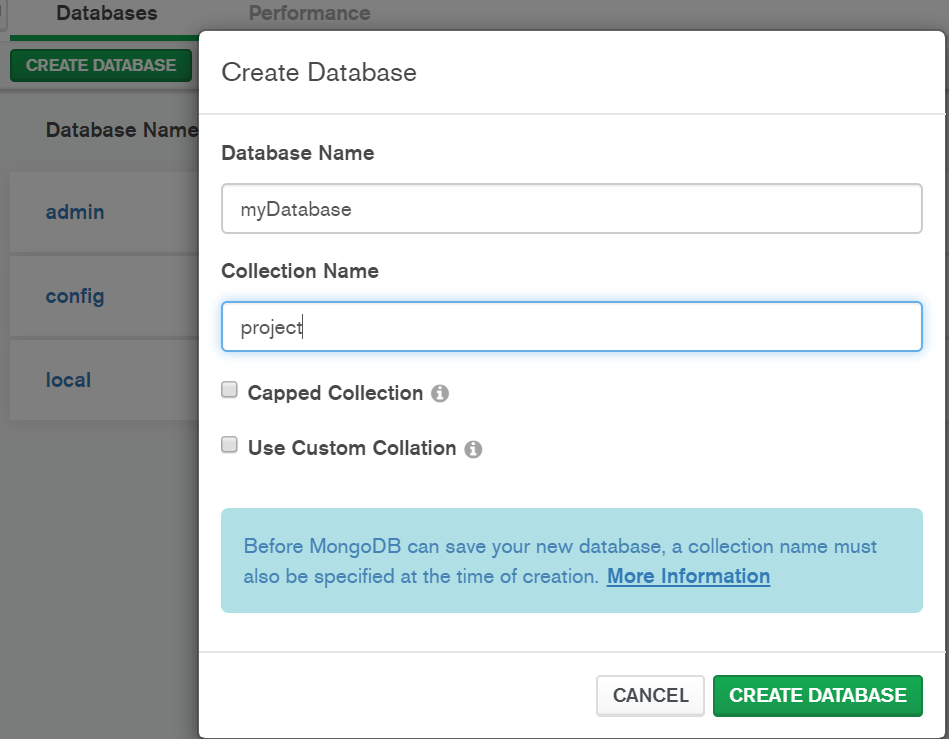
* Checking the details under COLLSPAN



* With experimenting with even a simple index on a small subset we can understand the following things
  + Use indexes only when having an index will enable performance gains
  + Indexes take up memory and should be used only when a query on a field is called so often that examining less documents will result in faster results
  + When the total number of documents is small, then there is no meaningful gains to benefit from

***Subtask3: Create a database which would be populated with data from previous steps.***

* Creating a Database in Compass:



* Importing csv file to MongoDB using cmd:

mongoimport --db myDatabase --collection movies --type csv --headerline --file Desktop\MAC\Sem-2\ADT\Project\ amazontest.csv

***Subtask4: Identify the type of query that is common on the data and finalize it for execution.***

* From our experience from the sample data set tests, we have learned that indexes are of value when they are implemented on fields which get fetched very often.
* Therefore, understanding the implementations and having first hand application knowledge is critical in determining which fields to apply indexes on.
* There is no point in creating an index on a field that gets fetched once every year but rather implementing an index on a field that gets called 10,000 times a day over 1,00,00,000 documents will ensure astronomical performance improvement.

***Subtask5: Identify the various query performance metrics used widely in the industry.***

* We need to understand how to measure performance metrics on any query in a MongoDB setup.
* Therefore, we went ahead with lot of reading of articles and resources to understand how to gauge performance in a manner that provides value.
* A couple of the critical reads and videos, we came across are the following
  + <https://docs.mongodb.com/manual/tutorial/analyze-query-plan/>
  + <https://youtu.be/0WkJKa_Nv_o>

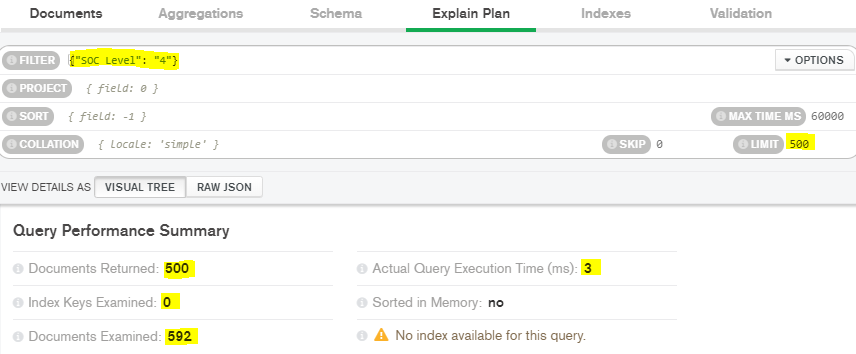
***Subtask6: Document performance stats for query identified adhering to guidelines discussed above***

* Run the find() query on all 3 datasets and note down their execution Stats.(Before Indexing)

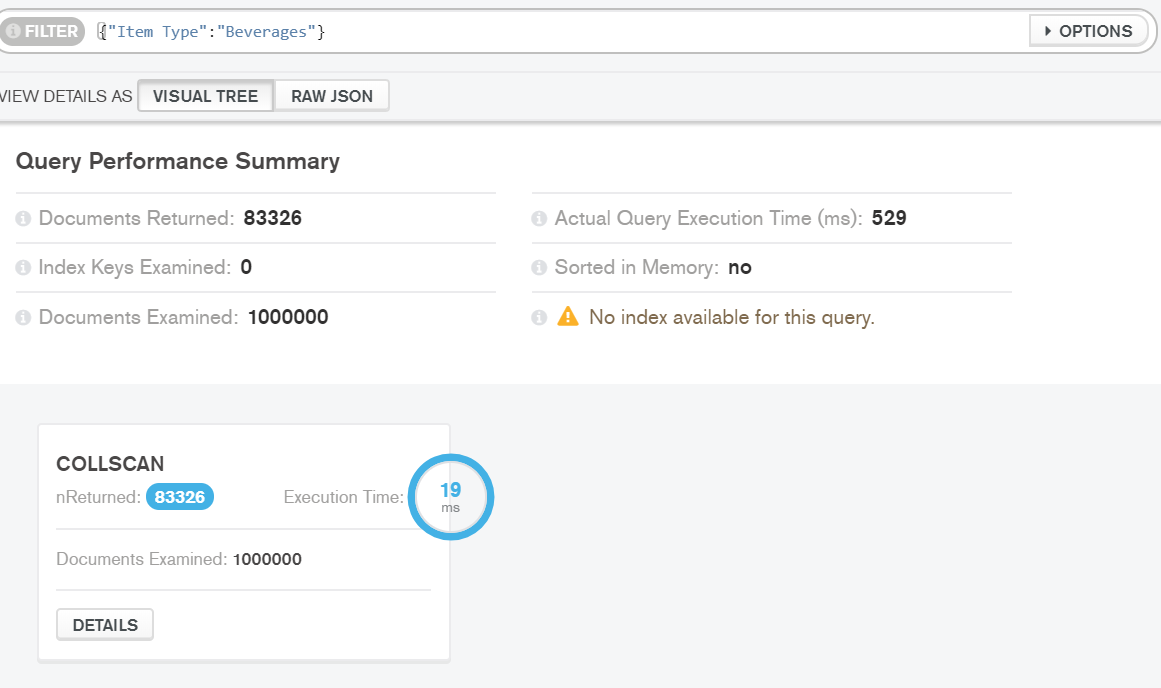
1. db.project.find( { “Current Employment”: { $gt: “500” } } )

******

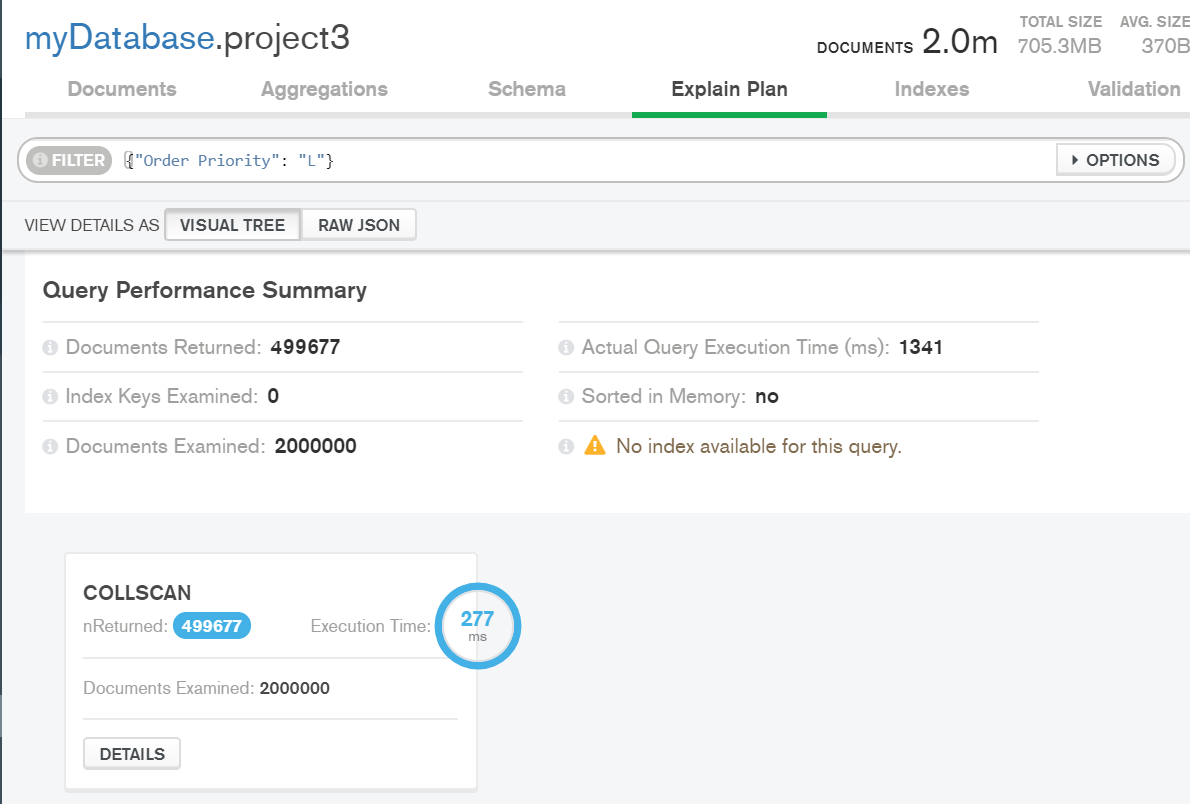
1. db.project1.find({"SOC Level":"4"}).limit(500)



1. db.project2.find({“Item Type”: “Beverages”})

******

1. db.project3.find({“Order Priority”: “L”})

******

***TASK2: INDEXING***

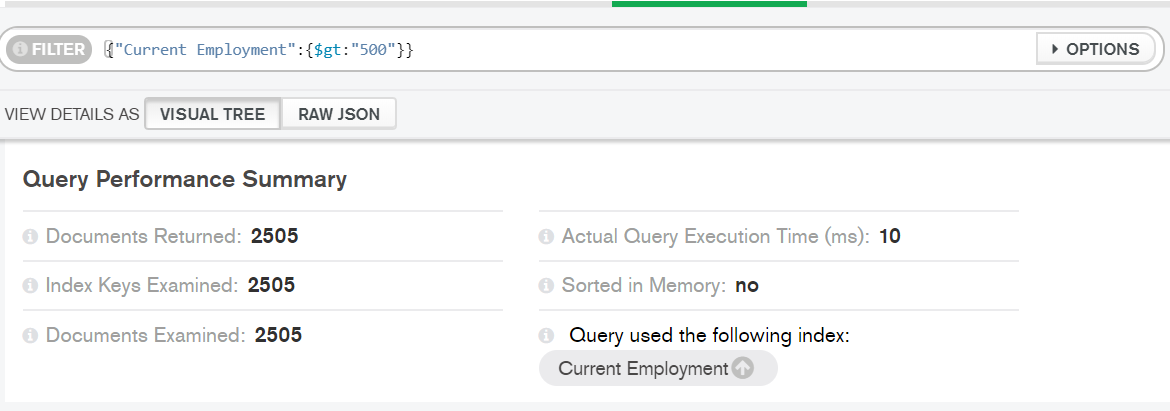
***Subtask1: Create an index.The index stores the value of a specific field or set of fields, ordered by the value of the field***

* For creating indexes we referred to the official MongoDB documentation to understand syntax and implementation. The link is given below

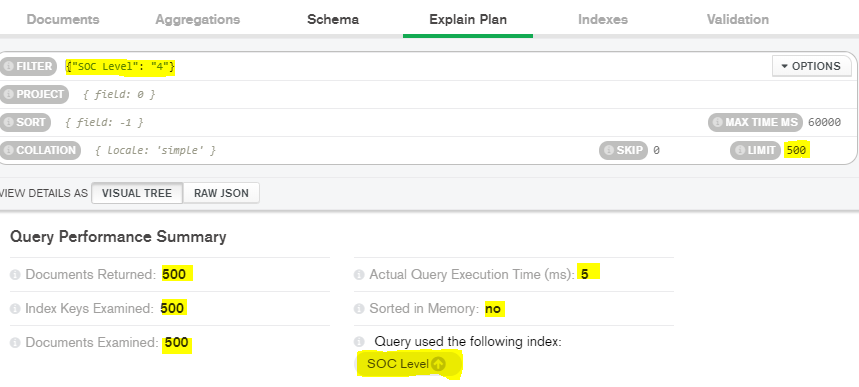
<https://docs.mongodb.com/manual/indexes/>

* Set Indexes: db.project.createIndex({Current Employment:1})
* Results after Indexing:

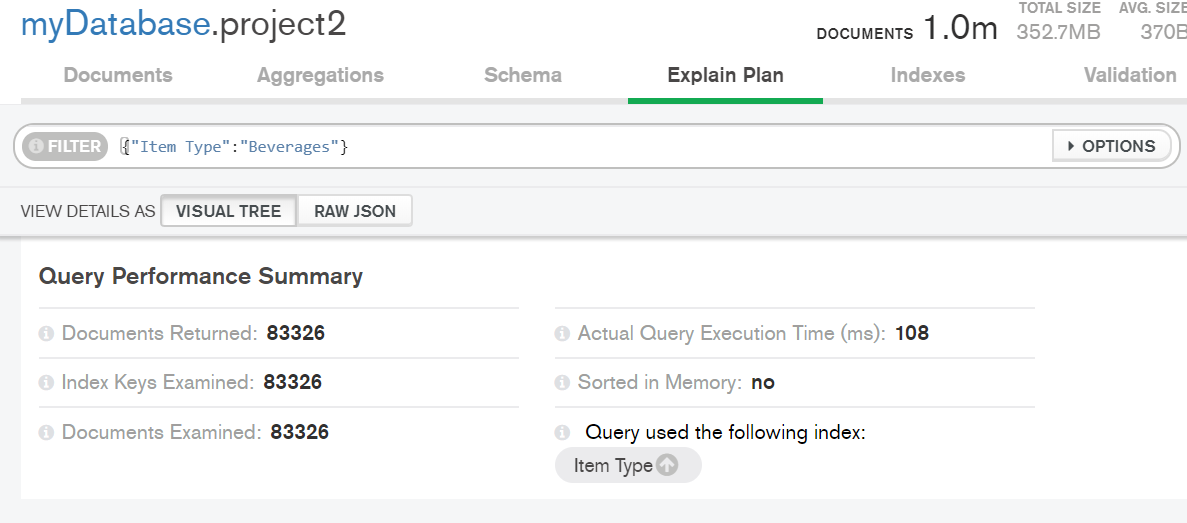
Collection: project



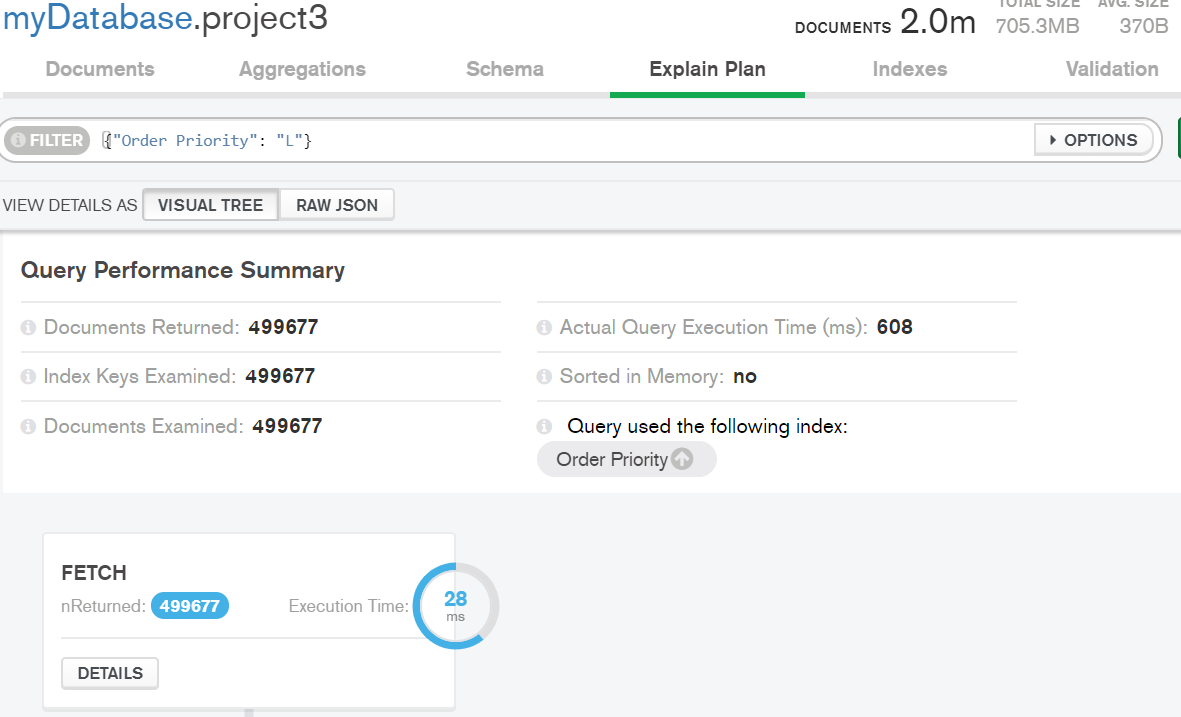
Collection: project1:



Collection: project2:



Collection: project3



**Challenges / Things which didn’t go per plan:**

* For the dataset imported as Project1 having 781 rows and 18 columns, the time taken for actual query execution was 2 milliseconds more than before implementing an index
* Further reading and understanding is needed to validate these results and to gain understanding of why indexing was detrimental here.
* This behaviour is similar to behaviour observed during the sample data set execution.