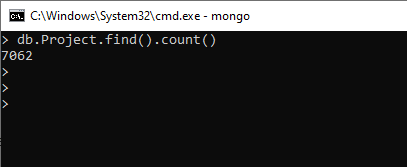
**Query Optimisation :**

As of now there are 7062 records in our database



Now let us see how normal select query will perform in production environment using **Query planner**

db.Project.find({"Year": "2014"}).explain()

We can have any number of plans but there can be only one winning plan which is going to be executed while at query executions

We can have multiple rejected plans on the basis of combination of indexes

But from above query result, it is clear that there are no rejected Plans as of now.

We will not be getting execution status using query planner. If we need to check the execution status, like how much time its going to take to execute , how much scanning needs to be done to match the records . We will need to provide execution stats.

db.Project.find({"Year": "2014"}).explain("executionStats")



As it is clear from above screenshot there are 264 matches and it took almost 264 milliseconds

totalKeysExamined is 0 as there are no indexes currently.

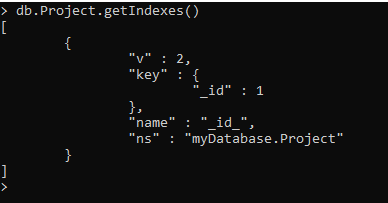
Also it had to access 7062 documents . which is total number of documents present in our DB.

Suppose Production DB has millions of documents, then it will be a major issue.

**Adding Index to improve Performance**

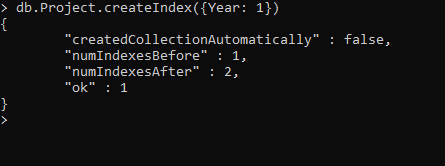
Before creating an index, let us check if we have any existing indexes.

db.Project.getIndexes()



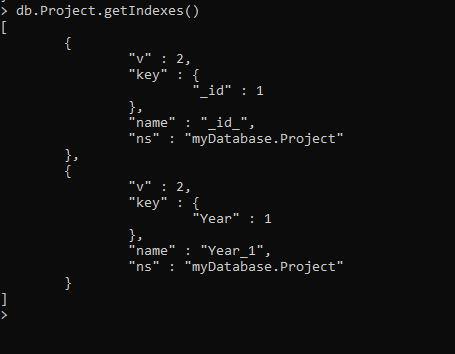
Now, Lets create an index on “Year” column using this query

db.Project.createIndex({Year: 1})



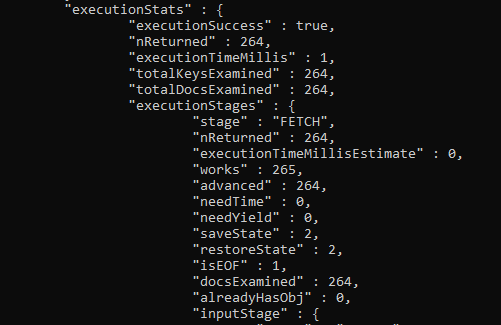
Index is successfully created

Now let us see all the available indexes:-



Now index is successfully created, let us check same query again.

db.Project.find({"Year": "2014"}).explain("executionStats")



Hence it is clear from above screenshot after creation of index , excecutionTime has gone down from 264 milliseconds to 1 millisecond and it had to access only 264 documents compared to 7062 earlier.