Universidad de las Américas Puebla
Cloud Computing and Big Data
LIS4102-1

Hands-On: MongoDB

Author:
Fernando Nieto Morales (ID: 163930)

Professor:
Genoveva Vargas Solar

Introduction

MongoDB is a NoSQL document database based on storing BSON (a binary representation of data structures and maps) data structures with a dynamic schema, making data integration in certain applications easier and faster. It is characterized by its ad hoc queries, indexing, replication, sharding and aggregation.

Focusing on indexes, these support the efficiency of the execution of queries by establishing a limit number of documents that may match with a query statement. Thus, it is faster to find a query than checking each document in the database.

> Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. (MongoDB, Inc., 2021)

To create an index in MongoDB, we use the command .createIndex() in the database where we need the index. The basic structure of the command is:

db.collection.**createIndex(** *<key and index type specification>*, *<options>* **)**

Each provided key will add specified information that a query can use to find the desired documents. The created indexes will be named as the concatenation of the indexed keys with the key's direction in the index, increasing (1) or decreasing (-1), and using underscores as separator. To understand this better, the next steps are examples of how the indexes are created and works.

Methodology

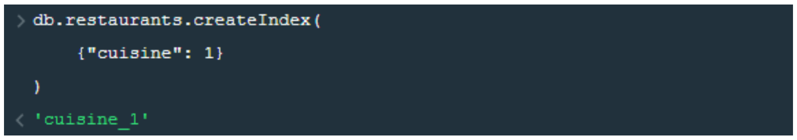
Given a database of different restaurants, I will create indexes to perform a faster query onto the collection. The process is divided into five steps/questions:

1. Create a single-field index (one key) based on the *cuisine* attribute.
2. Create a compound-field index based on the *cuisine* attribute and the restaurant's zipcode.
3. Check all the existing indexes in the database.
4. Find the query of the Italians restaurants in the collection and check the execution process of the query.
5. Check the execution statistics of the same query of the step 4.
6. Drop the created indexes and execute the same query.

Results

The results of each step are:

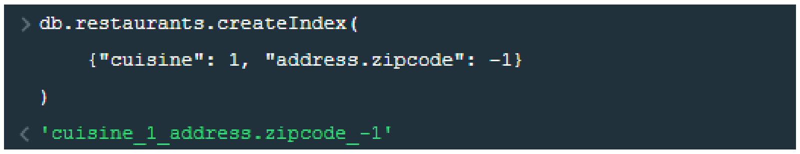
Question 20: Create an increasing index on the cuisine attribute of the restaurants collection.



```
> db.restaurants.createIndex(
      {"cuisine": 1}
  )
< 'cuisine_1'
```

Figure 1: Create a single-field index.

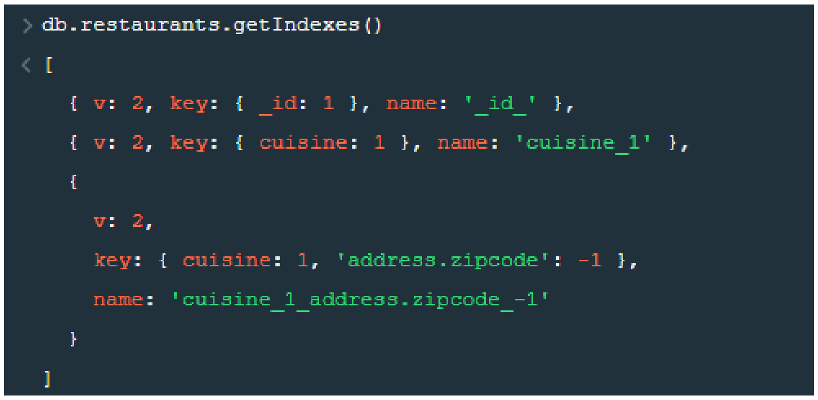
Question 21: Create another index for the restaurants collection, consisting of the cuisine attribute (increasing) and the zipcode attribute (decreasing).



```
> db.restaurants.createIndex(
      {"cuisine": 1, "address.zipcode": -1}
  )
< 'cuisine_1_address.zipcode_-1'
```

Figure 2: Create a compound-field index.

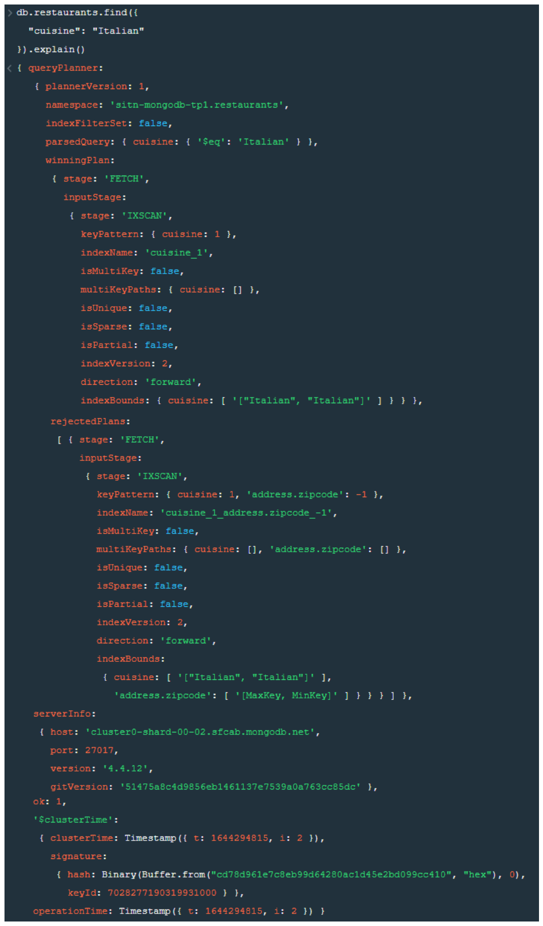
Question 22: List all indexes created on the restaurants collection.



```
> db.restaurants.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { cuisine: 1 }, name: 'cuisine_1' },
    {
      v: 2,
      key: { cuisine: 1, 'address.zipcode': -1 },
      name: 'cuisine_1_address.zipcode_-1'
    }
  ]
```

Figure 3: Check all the existing indexes in the database.

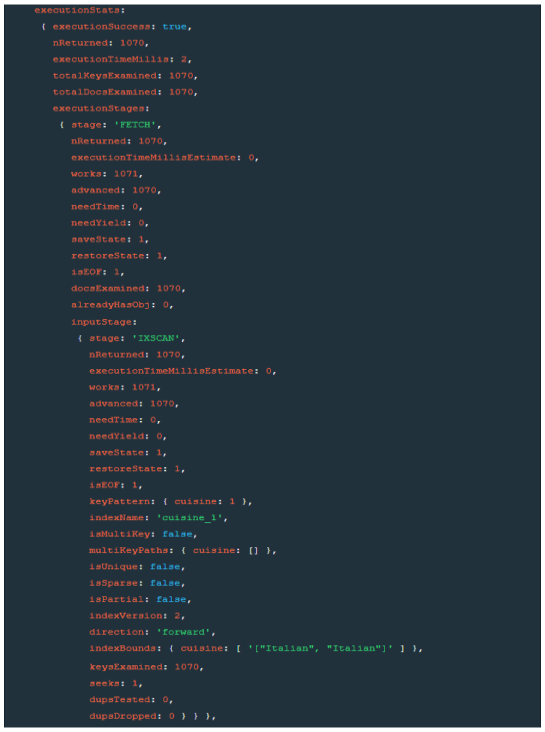
Question 23: Use the explain method to display the execution plan for the query that returns all Italian restaurants. What information is provided by the system?



```
> db.restaurants.find({
    "cuisine": "Italian"
  }).explain()
< { queryPlanner:
     { plannerVersion: 1,
       namespace: 'sitn-mongodb-tp1.restaurants',
       indexFilterSet: false,
       parsedQuery: { cuisine: { '$eq': 'Italian' } },
       winningPlan:
        { stage: 'FETCH',
          inputStage:
           { stage: 'IXSCAN',
             keyPattern: { cuisine: 1 },
             indexName: 'cuisine_1',
             isMultiKey: false,
             multiKeyPaths: { cuisine: [] },
             isUnique: false,
             isSparse: false,
             isPartial: false,
             indexVersion: 2,
             direction: 'forward',
             indexBounds: { cuisine: [ '["Italian", "Italian"]' ] } } },
        rejectedPlans:
         [ { stage: 'FETCH',
             inputStage:
              { stage: 'IXSCAN',
                keyPattern: { cuisine: 1, 'address.zipcode': -1 },
                indexName: 'cuisine_1_address.zipcode_-1',
                isMultiKey: false,
                multiKeyPaths: { cuisine: [], 'address.zipcode': [] },
                isUnique: false,
                isSparse: false,
                isPartial: false,
                indexVersion: 2,
                direction: 'forward',
                indexBounds:
                 { cuisine: [ '["Italian", "Italian"]' ],
                   'address.zipcode': [ '[MaxKey, MinKey]' ] } } } ] },
    serverInfo:
     { host: 'cluster0-shard-00-02.sfcab.mongodb.net',
       port: 27017,
       version: '4.4.12',
       gitVersion: '51475a8c4d9856eb1461137e7539a0a763cc85dc' },
    ok: 1,
    '$clusterTime':
     { clusterTime: Timestamp({ t: 1644294815, i: 2 }),
       signature:
        { hash: Binary(Buffer.from("cd78d961e7c8eb99d64280ac1d45e2bd099cc410", "hex"), 0),
          keyId: 7028277190319931000 } },
    operationTime: Timestamp({ t: 1644294815, i: 2 }) }
```

Figure 4: Find the query of the Italians restaurants in the collection and check the execution process of the query.

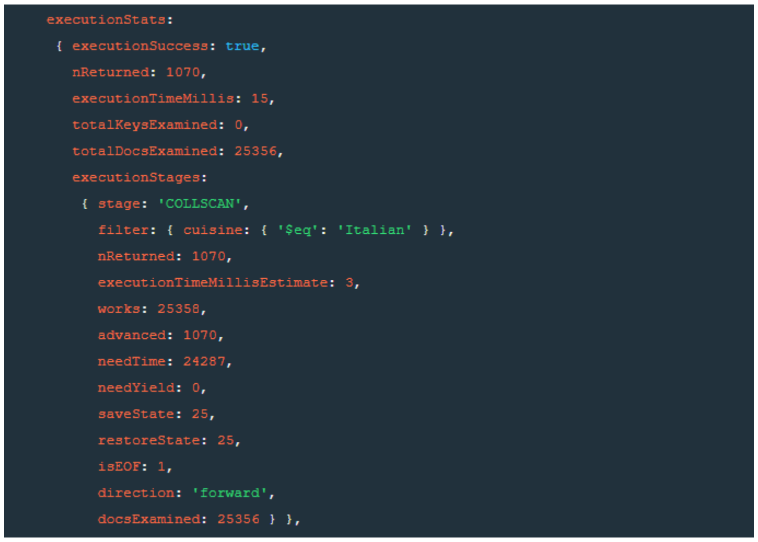
Question 24: Same question but adding the parameter "executionStats" in the explain method.



```
executionStats:
 { executionSuccess: true,
   nReturned: 1070,
   executionTimeMillis: 2,
   totalKeysExamined: 1070,
   totalDocsExamined: 1070,
   executionStages:
    { stage: 'FETCH',
      nReturned: 1070,
      executionTimeMillisEstimate: 0,
      works: 1071,
      advanced: 1070,
      needTime: 0,
      needYield: 0,
      saveState: 1,
      restoreState: 1,
      isEOF: 1,
      docsExamined: 1070,
      alreadyHasObj: 0,
      inputStage:
       { stage: 'IXSCAN',
         nReturned: 1070,
         executionTimeMillisEstimate: 0,
         works: 1071,
         advanced: 1070,
         needTime: 0,
         needYield: 0,
         saveState: 1,
         restoreState: 1,
         isEOF: 1,
         keyPattern: { cuisine: 1 },
         indexName: 'cuisine_1',
         isMultiKey: false,
         multiKeyPaths: { cuisine: [] },
         isUnique: false,
         isSparse: false,
         isPartial: false,
         indexVersion: 2,
         direction: 'forward',
         indexBounds: { cuisine: [ '["Italian", "Italian"]' ] },
         keysExamined: 1070,
         seeks: 1,
         dupsTested: 0,
         dupsDropped: 0 } } },
```

Figure 5: Check the execution statistics of the query.

Question 25: Drop the two indexes you previously created, and then re-display the statistics on the query execution plan that returns all Italian restaurants. What do you see?



```
executionStats:
 { executionSuccess: true,
   nReturned: 1070,
   executionTimeMillis: 15,
   totalKeysExamined: 0,
   totalDocsExamined: 25356,
   executionStages:
    { stage: 'COLLSCAN',
      filter: { cuisine: { '$eq': 'Italian' } },
      nReturned: 1070,
      executionTimeMillisEstimate: 3,
      works: 25358,
      advanced: 1070,
      needTime: 24287,
      needYield: 0,
      saveState: 25,
      restoreState: 25,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 25356 } },
```

Figure 6: Drop the created indexes and execute the same queries.
This figure shows the execution stats of executing the query.

Conclusion

As we can see, the Figures 1,2 & 3 demonstrate how the indexes are created and the default name that MongoDB will provide, it also shows that they are added correctly to the database. On Figure 4, the important information is shown in the *winningPlan* section, where we can see that it selected the *cuisine_1* index as the best option to do the query. The Figures 5 & 6 help us to compare the efficiency of the index with the default query index. As we can see, using *cuisine_1* only examined 1070 documents, while the default index searched into 25356 documents. It also reports that *cuisine_1* had to examine 1070 keys but compared with the difference of searched documents, the new index works faster.

To conclude, indexing is a good manner to reduce the time in which a query is found drastically. On small databases this is not clearly seen, but with big data, this type of protocols and methodologies will create a difference on time.

Bibliography

MongoDB, Inc. (2021). *Indexes*. Retrieved february 12, 2022, from MongoDB: https://docs.mongodb.com/manual/indexes/