Big Data Indexing

Prepared by Jeong-Hun Kim (etyanue@chungbuk.ac.kr)

Practice

- Introduction to indexing
  - Let's create a collection with 1 million documents in it

```
> for (i=0; i<1000000; i++) {
...     db.users.insert(
...         {
...             "i" : i,
...             "username" : "user"+i,
...             "age" : Math.floor(Math.random()*120),
...             "created" : new Date()
...         }
...     );
... }
```

Practice

- Use the explain("executionStats") function to see what MongoDB is doing when it executes the query

```
> db.users.find({username: "user101"}).explain()
{
    "cursor" : "BasicCursor",
    "nscanned" : 1000000,
    "nscannedObjects" : 1000000,
    "n" : 1,
    "millis" : 721,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {

    }
}
```

Practice

- Solution
  - To optimize this query, we could limit it to one result so that MongoDB would stop looking after it found user101

```
> db.users.find({username: "user101"}).limit(1).explain()
{
    "cursor" : "BasicCursor",
    "nscanned" : 102,
    "nscannedObjects" : 102,
    "n" : 1,
    "millis" : 2,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {

    }
}
```

Practice

- However
  - This is an impractical solution in general
    - What if we were looking for user999999?
  - We'd still have to traverse the entire collection
  - Indexes are a great way to fix queries like this
    - They organize data by a given field to let MongoDB find it quickly
  - Try creating an index on the username field
    - db.users.createIndex({"username" : 1})

Practice

- Result with indexing
  - Once the index build is complete, try repeating the original query:

```
> db.users.find({"username" : "user101"}).explain()
{
    "cursor" : "BtreeCursor username_1",
    "nscanned" : 1,
    "nscannedObjects" : 1,
    "n" : 1,
    "millis" : 3,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {
        "username" : [
            [
                "user101",
                "user101"
            ]
        ]
    }
}
```

Practice

- Restaurant dataset


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 }
,
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

Practice

- Create an Ascending Index on a Single Field
  - createIndex() is used to create indexes on collections
    - *db.collection.createIndex(keys, options)*
      - keys are document ({<field>: value})
  - Index
    - db.restaurant.createIndex({ borough: 1 })
  - Query
    - db.restaurant.find({borough: "Bronx"})
  - Task 1
    - Create an ascending index on “restaurant_id” field and check the index efficiency

Practice

- Create an Index on an Embedded Field
  - createIndex() is used to create indexes on collections
    - *db.collection.createIndex(keys, options)*
      - keys are document ({<field>: value})
  - Index
    - db.restaurant.createIndex({"address.zipcode": 1 })
  - Query
    - db.restaurant.find({"address.zipcode": "10462"})
  - Task 2
    - Create an ascending index on "street" field and query restaurants in "Broadway" street and "3220" building

Practice

- Create a Compound Index
  - Syntax
    - *db.collection.createIndex({ <field1>: <type>, <field2>: <type2>, ... })*
  - Index
    - db.restaurant.createIndex({"address.street": 1, "address.building" : 1})
  - Query
    - db.restaurant.find({"address.street": "Broadway", "address.building": "3220"})
  - Task 3
    - Create an ascending compound index on three fields (name, borough, cuisine) and make the query to check it is efficiency

Practice

- Create a Compound Index
  - Index
    - db.restaurant.createIndex({name: 1, borough: 1, cuisine: 1})
  - Index Prefix
    - name
    - name and borough
  - Task 4
    - Check the efficiency of index on "name" field
    - Check the efficiency of index on "borough" field
    - Check the efficiency of index on "name" and "borough" fields
    - Check the efficiency of index on "borough" and "cuisine" fields
    - Check the efficiency of index on “name" and "cuisine" fields

Practice

- Create an Index on Array of Values (Multikey Index)
  - Syntax
    - db.coll.createIndex({ <field>: < 1 or -1 > })
  - Index
    - db.restaurant.createIndex({"address.coord": 1})
  - Query
    - db.restaurant.find({"address.coord": [-73.856077, 40.848447]})
  - Task 5
    - Make the query to find the coordinate of "73.9862536“ and see how many records are checked

Practice

- Create an Index on Array of Documents
  - Index
    - db.restaurant.createIndex({"grades.grade": "C"})
  - Query
    - db.restaurant.find({"grades.grade": "C"})
  - Last Task
    - Create a compound index for grade and score fields and check the effect of indexing

Practice

- Download one of the datasets provided by MongoDB
  - https://github.com/neelabalan/mongodb-sample-dataset
  - https://github.com/ozlerhakan/mongodb-json-files
  - Optional: You can also use your own dataset
- Create various indexes and check its efficiency
  - Single field index
  - Compound index
  - Multi-key index
- Submit
  - Your dataset, your index and queries, and result screen before indexing and after indexing

Questions?

See you next time!