Tuần 8: Index

Mục tiêu:
- *Hiểu vai trò của Index trong truy vấn và thao tác trên cơ sở dữ liệu.*
- *Tạo và sử dụng index.*

Yêu cầu:

Tạo file MSWord theo định dạng ***HoTen_MSSV_Tuan8.docx***, chụp lại các câu lệnh và kết quả chạy vào file MSWord, nộp lại vào cuối buổi thực hành.

Tóm tắt lý thuyết:

- **Cú pháp tạo index:** db.collection.createIndex(keys, options)
- **Các thao tác với Index:**
  - createIndex()
  - getindexes()
  - dropIndex()
- **Các loại Index**:
  - **Single Field**: các chỉ mục **ascending/descending** do người dùng xác định trên một trường duy nhất của document.
  - **Compound Index**: một user-defined indexes trên nhiều fields.
  - **Multikey Index**: lập chỉ mục trên nội dung được lưu trữ trong mảng. Các chỉ mục Multikey Index cho phép các truy vấn chọn các documents có chứa mảng bằng cách so khớp trên phần tử hoặc các phần tử của mảng.
  - **Unique Indexes**: đảm bảo các trường được lập chỉ mục không lưu trữ các giá trị trùng lặp. Mặc định, MongoDB tạo một chỉ mục duy nhất trên trường **_id** trong collection.
  - **Hash index**: Hash index dựa trên giải thuật Hash Function (hàm băm). Tương ứng với mỗi khối dữ liệu (index) sẽ sinh ra một bucket key(giá trị băm) để phân biệt.

Bài 1. Tìm hiểu vai trò và cách sử dụng index trong cơ sở dữ liệu

1. Tạo một cơ sở dữ liệu **highestmountains**, với một collection **peaks** gồm các document chứa các thông tin:
   - **name**: tên của đỉnh.
   - **height**: độ cao của đỉnh, tính bằng mét.
   - **location**: các quốc gia nơi ngọn núi tọa lạc. Trường này lưu trữ các giá trị dưới dạng một mảng để cho phép các ngọn núi nằm ở nhiều quốc gia.
   - **ascents**: giá trị của trường này là một document embedded. Mỗi ascents document mô tả số lần đi lên thành công của ngọn núi nhất định. Mỗi **ascents** document chứa một **total** field liệt kê tổng số lần đi lên thành công của mỗi đỉnh nhất định. Ngoài ra, mỗi nested documents này chứa hai trường có giá trị cũng là nested documents.
     - ***first***: là một nested document có chứa trường ***year***, mô tả năm của lần đi lên thành công đầu tiên.

- ***first_winter:*** là một nested document cũng chứa trường ***year***, giá trị của trường đại diện cho năm diễn ra mùa đông thành công đầu tiên đi lên ngọn núi nhất định.

2. Dùng phương thức **insertMany()** method trong MongoDB shell để đồng thời tạo một collection tên **peaks** và chèn 5 documents như sau:

```
db.peaks.insertMany([
    {
        "name": "Everest",
        "height": 8848,
        "location": ["Nepal", "China"],
        "ascents": {
            "first": {
                "year": 1953
            },
            "first_winter": {
                "year": 1980
            },
            "total": 5656
        }},
    {
        "name": "K2",
        "height": 8611,
        "location": ["Pakistan", "China"],
        "ascents": {
            "first": {
                "year": 1954
            },
            "first_winter": {
                "year": 1921
            },
            "total": 306
        }},
    {
        "name": "Kangchenjunga",
        "height": 8586,
        "location": ["Nepal", "India"],
        "ascents": {
            "first": {
                "year": 1955
            },
            "first_winter": {
                "year": 1986
            },
            "total": 283
        }},
    {
        "name": "Lhotse",
        "height": 8516,
        "location": ["Nepal", "China"],
        "ascents": {
            "first": {
                "year": 1956
            },
            "first_winter": {
                "year": 1988
            },
            "total": 461
```

```
        }},
    {
        "name": "Makalu",
        "height": 8485,
        "location": ["China", "Nepal"],
        "ascents": {
            "first": {
                "year": 1955
            },
            "first_winter": {
                "year": 2009
            },
            "total": 361
        }
    }
])
```

3. Thực hiện truy vấn tìm ngọn núi có độ cao >8700.
4. Dùng phương thức **explain(executionStats**) để xem thông tin về cách thực hiện truy vấn. Quan sát kết quả về số lần duyệt các document để tìm được kết quả.

```
db.peaks.find(
    { "height": { $gt: 8700 } }
).explain("executionStats")
```

5. Tạo một chỉ mục (**Single Field Index**) trên trường height trong peaks collection sử dụng phương thức **createIndex().** Viết câu lệnh xem có bao nhiêu chỉ mục đã được xác định trên **peaks** collection và trạng thái đó khác với trạng thái trước đó như thế nào.
6. Thực hiện lại câu truy vấn tìm ngọn núi có độ cao >8700 dựa trên chỉ mục đã tạo. Dùng phương thức **explain(executionStats**) để xem thông tin về cách thực hiện truy vấn, so sánh với cách thực thi khi chưa có index.
7. Tạo một chỉ mục (**Unique Indexes**) đảm bảo tên các đỉnh núi (trường Name) là duy nhất. Viết câu lệnh xem kết quả của Index vừa tạo.

```
db.peaks.createIndex( { "name": 1 }, { "unique": true } )
```

```
Output
{
        "createdCollectionAutomatically”: false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok": 1
}
```

8. Thực hiện câu lệnh truy vấn tìm đỉnh núi tên **Everest** kết hợp với phương thức **explain ("executionStats")** xem trạng thái thực thi của câu truy vấn trên Index vừa tạo.
9. Dùng phương thức **insertOne()** chèn thêm một document mới với tên ngọn núi là Everest. Nhận xét kết quả của câu lệnh.

```
db.peaks.insertOne({
    "name": "Everest",
    "height": 9200,
    "location": ["India"],
    "ascents": {
```

```
            "first": {
                "year": 2020
            },
            "first_winner": {
                "year": 2021
            },
            "total": 2
        }
    })
```

10. Tạo một Index trên **Embedded Field**
    a. Thực hiện truy vấn tìm những ngọn núi có **total>300**, kết quả sắp xếp theo thứ tự giảm dần.

```
db.peaks.find(
    { "ascents.total": { $gt: 300 } }
).sort({ "ascents.total": -1 }).explain("executionStats")
```

    b. Thực hiện lại truy vấn trên nhưng kết hợp với phương thức **explain("executionStats")** để xem tình trạng thực thi của câu truy vấn. (*giá trị COLLSCAN trong phần này của đầu ra cho biết, MongoDB đã sử dụng quét toàn bộ bộ sưu tập và duyệt qua tất cả các tài liệu từ bộ sưu tập đỉnh để so sánh chúng với các điều kiện truy vấn*)
    c. Tạo một index trên **total** field bên trong **ascents** document (Embedded Field). Viết câu lệnh xem index vừa tạo.
    d. Thực thi lại câu truy vấn tìm những ngọn núi có Total>300, kết quả sắp xếp theo thứ tự giảm dần. Viết câu lệnh để kiểm tra xem chỉ mục có giúp MongoDB tránh thực hiện quét toàn bộ collection hay không?

    (*IXSCAN được sử dụng đối với chỉ mục ascents.total_-1 mới được tạo và chỉ có bốn tài liệu đã được kiểm tra. Đây là cùng một số lượng tài liệu được trả lại và kiểm tra trong chỉ mục, vì vậy không có tài liệu bổ sung nào được truy xuất để hoàn thành truy vấn*)

11. **Compound Field Index**: sử dụng các chỉ mục khi thực hiện các truy vấn trên nhiều trường.
    a. Hãy tìm những ngọn núi có chiều cao dưới 8600 mét có **winter ascent** đầu tiên xảy ra sau năm 1990. (chưa tạo index)

```
db.peaks.find(
    {
        "ascents.first_winter.year": { $gt: 1990 },
        "height": { $lt: 8600 }
    }
).sort({ "height": -1 })
```

    b. Thực hiện lại truy vấn trên, thêm phương thức explaion("executionStats") để xem cách MongoDB thực hiện truy vấn.

    Nhận xét kết quả:

- ✓ *MongoDB đã sử dụng single field index trên trường height để thu hẹp kết quả từ 5 xuống 3, nhưng sau đó phải **quét các document còn lại để kiểm tra ngày đi lên (winter ascent) trong mùa đông đầu tiên**.*
- ✓ *Mặc dù không có chỉ mục nào có thể ảnh hưởng đến ngày đi lên (winter ascent) trong mùa đông đầu tiên, MongoDB đã sử dụng chỉ mục được tạo trước đó thay vì thực hiện quét toàn bộ collection.*

c. Tạo một index dựa trên 2 trường: "**height**" giảm dần và "**ascents.first_winter.year**" tăng dần. Viết câu lệnh xem kết quả index vừa tạo.

d. Thực hiện lại truy vấn tìm những ngọn núi có **height< 8600 mét** có winter ascent đầu tiên (**ascents.first_winter.year**) xảy ra sau năm 1990, kết quả sắp xếp theo chiều giảm dần của trường **height**. Dùng phương thức **explain("executionStats")** xem cách Mongodb thực hiện truy vấn, phân tích và nhận xét kết quả.

```
db.peaks.find(
    {
        "ascents.first_winter.year": { $gt: 1990 },
        "height": { $lt: 8600 }
    }
).sort({ "height": -1 }).explain("executionStats")
```

12. **Multi-key Index:** MongoDB lưu trữ giá trị của trường trực tiếp dưới dạng khóa chỉ mục (index key), giúp chỉ mục có thể duyệt nhanh chóng. Bước này phác thảo cách MongoDB hoạt động khi trường được sử dụng để tạo chỉ mục là trường lưu trữ nhiều giá trị, chẳng hạn như một mảng

a. Viết câu lệnh tìm tất cả các ngọn núi trong collection nằm ở **Nepal**.

*Nhận xét: Mỗi đỉnh bao trùm nhiều quốc gia như được biểu thị bằng các trường location, là một mảng gồm nhiều giá trị. Do không có sẵn chỉ mục mở rộng trường **location**, MongoDB hiện thực hiện quét toàn bộ bộ collection để thực hiện truy vấn.*

b. Tạo một index mới cho trường **location**, viết câu lệnh xem kết quả index vừa tạo.

c. Thực hiện lại câu truy vấn tìm các ngọn núi ở Nepal, kết hợp phương thức **explain("executionStats")** để xem cách MogoDB thực thi.

```
db.peaks.find(
    {"location": "Nepal"}
). explain("executionStats")
```

Nhận xét kết quả

- *MongoDB đã sử dụng quét chỉ mục ("stage" : "IXSCAN",) làm chiến lược, đề cập đến chỉ mục location_1 mới được tạo*
- *Thuộc tính **isMultiKey**: true. MongoDB tự động tạo một multi-key index cho trường location.*
- *Đối với document có trường **location** lưu trữ array ["China", "Nepal"], hai index entries riêng biệt sẽ xuất hiện cho cùng một tài liệu, một cho China và một cho Nepal. Bằng cách này, MongoDB có thể sử dụng chỉ mục một cách hiệu quả ngay cả khi truy vấn yêu cầu khớp một phần với nội dung mảng.*

13. Liệt kê tất cả các index trong collection Peaks, sử dụng phương thức getIndexes()

```
Output
[
        {
                "v": 2,
                "key": {
                        "_id": 1
                },
                "name": "_id_"
        },
        {
                "v": 2,
                "key": {
                        "height": 1
                },
                "name": "height_1"
        },
        {
                "v": 2,
                "unique": true,
                "key": {
                        "name": 1
                },
                "name": "name_1"
        },
        {
                "v": 2,
                "key": {
                        "ascents.total" : 1
                },
                "name": "ascents.total_1"
        },
        {
                "v": 2,
                "key": {
                        "ascents.first_winter.year" : 1,
                        "height": -1
                },
                "name": "ascents.first_winter.year_1_height_-1"
        },
        {
                "v": 2,
                "key": {
                        "location": 1
                },
                "name": "location_1"
        }
]
```

14. Viết câu lệnh xóa index height_1, sử dụng phương thức dropIndex(), dựa trên tên trường tạo chỉ mục.
15. Viết câu lệnh xóa index dựa trên tên của index **ascents.first_winter.year_1_height_-1**.
16. Viết câu lệnh liệt kê tất cả các index còn lại trong collection Peaks

Bài 2. Sử dụng collection restaurants thực hiện các thao tác sau đây:

1. Liệt kê danh sách restaurant có cuisine là “Hamburger”, có dùng phương thức explain() để xem thông tin về cách thực hiện truy vấn. Quan xát kết quả về số lần duyệt, thời gian, cách duyệt,…

```
restaurantdb> db.restaurant.find({"cuisine":"Hamburgers"}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'restaurantdb.restaurant',
    indexFilterSet: false,
    parsedQuery: { cuisine: { '$eq': 'Hamburgers' } },
    queryHash: '1940DD67',
    planCacheKey: '251FB4B1',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
```

```
  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: [ '["Hamburgers", "Hamburgers"]' ] }
    }
  },
  rejectedPlans: []
```

2. Tạo một index trên field cuisine theo thứ tự tăng dần.
3. Liệt kê danh sách restaurant có cuisine là “Hamburger”, có dùng phương thức explain() để xem thông tin về cách thực hiện truy vấn. Quan xát kết quả về số lần duyệt, thời gian, cách duyệt,… So sánh với kết quả câu 1.
4. Liệt kê danh sách các index hiện có trong restaurants, cho biết tên của index đó

   db.restaurant.getIndex()

```
restaurantdb> db.restaurant.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { cuisine: 1 }, name: 'cuisine_1' }
]
```

5. Xóa index hiện có trong collection restaurants

```
restaurantdb> db.restaurant.dropIndexes("cuisine_1")
{ nIndexesWas: 2, ok: 1 }
restaurantdb> db.restaurant.getIndexes()
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

6. Tạo lại index trên trường cuisine theo thứ tự giảm dần và đặt tên là cusine_inc. Viết câu lệnh xem lại index vừa tạo.
7. Liệt kê danh sách các restaurant có borough là "Brooklyn" khi chưa được lập chỉ mục. Sau đó tạo chỉ mục trên borough, liệt kê lại danh sách và so sánh kết quả.
8. Tạo chỉ mục compound field index trên trường cuisine (giảm dần) và name (tăng dần). Xem lại các index hiện có restaurants.
9. Thực hiện liệt kê danh sách các restaurants có cuisine là "Hamburger" và name bắt đầu là "Wil".
10. Cho biết hiện tại trong restaurants có những index nào
11. Tạo một Unique Index trên trường restaurant_id, sau đó chèn một document mới có restaurant_id trùng với restaurant_id của một restaurant đã có, cho nhận xét.
12. Thực hiện tìm các restaurants có restaurant_id là 40361521, kết hợp phương thức explain("executionStats") xem trạng thái thực thi của câu truy vấn.
13. Thực hiện ẩn Unique Index vừa tạo ở trên, sau đó chèn một document mới có restaurant_id trùng với restaurant_id của một restaurant đã có (ví dụ 40361521). Cho nhận xét.
14. Tạo một Embedded Field Index trên trường address.zipcode theo thứ tự tăng dần.
15. Thực hiện tìm các restaurants có zipcode lớn hơn 11000, kết hợp kết hợp phương thức explain("executionStats") xem trạng thái thực thi của câu truy vấn.
16. Thực hiện xóa tất cả các Index hiện có trong restaurants.
17. Tạo index trên trường Name (tăng dần) và trường address.zipcode (tăng dần).
18. Thực hiện liệt kê danh sách các restaurants có zipcode lớn hơn 11000 và xếp theo thức tự tăng dần của tên, kết hợp phương thức explain("executionStats") xem trạng thái thực thi của câu truy vấn.
19. Thực hiện tạo Multi-key Index trên trường grades tăng dần.
20. Liệt kê danh sách các restaurants có grades.grade là "A", kết hợp kết hợp phương thức explain("executionStats") xem trạng thái thực thi của câu truy vấn.
21. Thực hiện tạo Multi-key Index trên trường grades.score tăng dần.
22. Thực hiện liệt kê danh sách các restaurant có grades.score từ 5 đến 10, kết hợp kết hợp phương thức explain("executionStats") xem trạng thái thực thi của câu truy vấn.