MongoDB Indexes

This section introduces you to MongoDB indexes that help perform queries faster and more efficiently. After completing the tutorials in this section, you'll have a good understanding of indexes and how to use them more effectively. You'll understand the following key topics:

- What indexes are and why you need to use them.
- How to select fields to index.
- Understand various types of indexes in MongoDB.
- How to evaluate the index usage.
- How to manage indexes such as creating and removing them.

Indexes are an important part of your application. And you'll see that selecting the right indexes for your collections is crucial to performance.

- Create indexes – introduce you to the index concept and show you how to use the createIndex() method to create an index for a field of a collection.
- Drop indexes – learn how to drop an index from a collection.
- Compound indexes – learn the compound indexes that contain references to multiple fields of a collection.
- Unique indexes – show you how to use unique indexes to enforce the uniqueness of values in a field across documents of a collection.

A quick introduction to indexes

Suppose you have a book that contains a list of movies:

To find a movie with the title Pirates of Silicon Valley, you need to scan every page of the book until you find the match. This is not efficient.

If the book has an index that maps titles with page numbers, you can look up the movie title in the index to find the page number:

Pimpernel' Smith 1

...

Pirates of Silicon Valley 201

...

Twas the Night 300

In this example, the movie with the title Pirates of Silicon Valley is located on page 201. Therefore, you can open page 201 to get detailed information about the movie:

In this analogy, the index speeds up the search and makes it more efficient.

The MongoDB index works in a similar way. To speed up a query, you can create an index for a field of a collection.

However, when you insert, update, or delete the documents from the collection, MongoDB needs to update the index accordingly.

In other words, an index improves the speed of document retrieval at the cost of additional write and storage space to maintain the index data structure. Internally, MongoDB uses the B-tree structure to store the index.

Load sample data

We'll use the movies collection from the mflix sample database to demonstrate how the indexes work in MongoDB.

First, download the movies.json file and place it in a folder on your computer
e.g., c:\data\movies.json

Second, import the movies.json file into the mflix database using the mongoimport tool:

mongoimport c:\data\movies.json -d mflix -c movies

Code language: CSS (css)

List indexes of a collection

By default, all collections have an index on the _id field. To list the indexes of a collection, you use the getIndexes() method with the following syntax:

db.collection.getIndexes()

Code language: CSS (css)

In this syntax, the collection is the name of the collection that you want to get the indexes. For example, the following shows the indexes of the movies collection in the mflix database:

db.sales.getIndexes()

Code language: CSS (css)

Output:

[{ v: 2, key: { _id: 1 }, name: '_id_' }]

Code language: CSS (css)

The output shows the index name '_id_' and index key _id. The value 1 in the key : { _id : 1 } indicates the ascending order of the _id values in the index.

When an index contains one field, it's called a single field index. However, if an index holds references to multiple fields, it is called a compound index. This tutorial focuses on a single field index.

Explain a query plan

The following query finds the movie with the title Pirates of Silicon Valley :

```
db.movies.find({
  title: 'Pirates of Silicon Valley
})
```

To find the movie, MongoDB has to scan the movies collection to find the match.

Before executing a query, the MongoDB query optimizer comes up with one or more query execution plans and selects the most efficient one.

To get the information and execution statistics of query plans, you can use the explain() method:

```
db.collection.explain()
```

Code language: CSS (css)

For example, the following returns the query plans and execution statistics for the query that finds the movies with the title Pirates of Silicon Valley:

```
db.movies.find({
  title: 'Pirates of Silicon Valley'
}).explain('executionStats')
```

Code language: JavaScript (javascript)

The explain() method returns a lot of information. And you should pay attention to the following winningPlan:

```
...
   winningPlan: {
```

```
    stage: 'COLLSCAN',
    filter: { title: { '$eq': 'Pirates of Silicon Valley' } },
    direction: 'forward'
  },
...
```

Code language: JavaScript (javascript)

The winningPlan returns the information on the plan that the query optimizer came up with. In this example, the query planner comes up with the COLLSCAN that stands for the collection scan.

Also, the executionStats shows that the result contains one document and the execution time is 9 milliseconds:

```
...
 executionStats: {
   executionSuccess: true,
   nReturned: 1,
   executionTimeMillis: 9,
   totalKeysExamined: 0,
   totalDocsExamined: 23539,
...
```

Code language: JavaScript (javascript)

Create an index for a field in a collection

To create an index for the title field, you use the createIndex() method as follows:

```
db.movies.createIndex({title:1})
```

Code language: CSS (css)

Output:

```
title_1
```

In this example, we pass a document to the createIndex() method. The { title: 1} document contains the field and value pair where:

- The field is the index key (year).

- The value describes the type of index for the year field. The value 1 for descending index and -1 for ascending index.

The createIndex() method returns the index name. In this example, it returns the title_1 which is the concatenation of the field and value.

The following query shows the indexes of the movies collection:

```
db.movies.getIndexes()
```

Code language: CSS (css)

Output:

```
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { title: 1 }, name: 'title_1' }
]
```

Code language: JavaScript (javascript)

The output shows two indexes, one is the default index and another is the year_1 index that we have created.

By default, MongoDB names an index by concatenating the indexed keys and each key's direction in the index (i.e. 1 or -1) using underscores as a separator. For example, an index created on { title: 1 } has the name title_1.

The following returns the query plans and execution statistics for the query that finds the movie with the title Pirates of Silicon Valley:

```
db.movies.find({
   title: 'Pirates of Silicon Valley'
})
```

Code language: CSS (css)

This time the query optimizer uses the index scan (IXSCAN) instead of the collection scan (COLLSCAN):

```
...
winningPlan: {
    stage: 'FETCH',
    inputStage: {
```

```
        stage: 'IXSCAN',
        keyPattern: { title: 1 },
        indexName: 'title_1',
        isMultiKey: false,
        multiKeyPaths: { title: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          title: [
            '["Pirates of Silicon Valley", "Pirates of Silicon Valley"]'
          ]
        }
      }
    },
...
```

Code language: JavaScript (javascript)

Also, the execution time (executionTimeMillis) was down to almost zero from 2 milliseconds:

```
...
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 0,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
...
```

Code language: JavaScript (javascript)

Summary

- An index improves the speed of document retrieval at the cost of additional write and storage space to maintain its data structure.
- Use the createIndex() method to create an index for a field in a collection.
- Use the getIndexes() method to list the indexes of a collection.
- Use the explain() method to get the information and execution statistics of query plans.

Introduction to the MongoDB unique index

Often, you want to ensure that the values of a field are unique across documents in a collection, such as an email or username.

A unique index can help you to enforce this rule. In fact, MongoDB uses a unique index to ensure that the _id is a unique primary key.

To create a unique index, you use the createIndex() method with the option {unique: true} like this:

db.collection.createIndex({ field: 1}, {unique: true});

Code language: CSS (css)

MongoDB unique index examples

Let's take some examples of using a unique index.

1) Create a unique index for a field

First, inserts two documents into the users collection:

```
db.users.insertMany([
  { email:  "john@test.com", name: "john"},
  { email:  "jane@test.com", name: "jane"},
]);
```

Code language: JavaScript (javascript)

Second, create a unique index for the email field:

db.users.createIndex({email:1},{unique:true});

Code language: CSS (css)

Third, attempt to insert a new document with the email that already exists:

```
db.users.insertOne(
  { email:  "john@test.com", name: "johny"}
);
```

Code language: CSS (css)

MongoDB returned the following error:

```
MongoServerError: E11000 duplicate key error collection: mflix.users index: email_1 dup key: { email: "john@test.com" }
```

Code language: CSS (css)

2) Create a unique index for collection with duplicate data

First, drop the users collection and recreate it by inserting three entries:

```
db.users.drop()


db.users.insertMany([
  { email:  "john@test.com", name: "john"},
  { email:  "john@test.com", name: "johny"},
  { email:  "jane@test.com", name: "jane"},
])
```

Code language: JavaScript (javascript)

Second, attempt to create a unique index for the email field of the users collection:

```
db.users.createIndex({email: 1},{unique:true})
```

Code language: CSS (css)

MongoDB returned the following error:

```
MongoServerError: Index build failed: 95f78956-d5d0-4882-bfe0-2d856df18c61: Collection mflix.users ( 6da472db-2884-4608-98b6-95a003b4f29c ) :: caused by :: E11000 duplicate key error collection: mflix.users index: email_1 dup key: { email: "john@test.com" }
```

Code language: CSS (css)

The reason is that the email has duplicate entries john@test.com.

Typically, you create a unique index on a collection before inserting any data. By doing this, you ensure uniqueness constraints from the start.

If you create a unique index on a collection that contains data, you run the risk of failure because the collection may have duplicate values. When duplicate values exist, the unique index creation fails as shown in the previous example.

To fix this, you need to review data and remove the duplicate entries manually before creating the unique index. For example:

First, delete the duplicate user:

db.users.deleteOne({name:'johny', email: 'john@test.com'});

Code language: CSS (css)

Output:

{ acknowledged: true, deletedCount: 1 }

Code language: CSS (css)

Then, create a unique index for the email field:

db.users.createIndex({email: 1},{unique:true})

Code language: CSS (css)

Output:

email_1

Unique compound index

When a unique index contains more than one field, it is called a unique compound index. A unique compound index ensures the uniqueness of the combination of fields.

For example, if you create a unique compound index for the field1 and field2, the following values are unique:

field1	**field2**	**Combination**
1	1	(1,1)
1	2	(1,2)
2	1	(2,1)
2	2	(2,2)

However, the following values are duplicate:

field1	field2	Combination
1	1	(1,1)
1	1	(1,1) -> duplicate
2	1	(2,1)
2	1	(2,1) -> duplicate

To create a unique compound index, you specify fields in the index specification like this:

```
db.collection.createIndex({field1: 1, field2: 1}, {unique: true});
```

Code language: JavaScript (javascript)

Let's take the example of using the unique compound index.

First, create a locations collection by adding one location to it:

```
db.locations.insertOne({
        address: "Downtown San Jose, CA, USA",
        lat: 37.335480,
        long: -121.893028
})
```

Code language: CSS (css)

Second, create a unique compound index for the lat and long fields of the locations collection:

```
db.locations.createIndex({
        lat: 1,
        long: 1
},{ unique: true });
```

Code language: CSS (css)

Output:

```
lat_1_long_1
```

Third, insert a location with the lat value that already exists:

```
db.locations.insertOne({
        address: "Dev Bootcamp, San Jose, CA, USA",
```

```
    lat: 37.335480,
    long: -122.893028
})
```

Code language: CSS (css)

It works because the lat_1_long_1 index only checks the duplicate of the combination of lat and long values.

Finally, attempt to insert a location with the lat and long that already exists:

```
db.locations.insertOne({
    address: "Central San Jose, CA, USA",
    lat: 37.335480,
    long: -121.893028
})
```

Code language: CSS (css)

MongoDB issued the following error:

```
MongoServerError: E11000 duplicate key error collection: mflix.locations index:
lat_1_long_1 dup key: { lat: 37.33548, long: -121.893028 }
```

Code language: CSS (css)

Summary

- A unique index enforces the uniqueness of values for a field in a collection.
- A unique compound index enforces the uniqueness of combination of values of multiple fields in a collection.
- Use the createIndex() method with the option { unique: true } to create a unique index and compound unique index.

Introduction to the MongoDB compound indexes

A compound index is an index that holds a reference to multiple fields of a collection. In general, a compound index can speed up the queries that match on multiple fields.

To create a compound index, you use the createIndex() method with the following syntax:

```
db.collection.createIndex({
   field1: type,
   field2: type,
   field3: type,
   ...
});
```

Code language: JavaScript (javascript)

In this syntax, you specify a document that contains the index keys (field1, field2, field3…) and index types.

The type describes the kind of index for each index key. For example, type 1 specifies an index that sorts items in ascending order while -1 specifies an index that sorts items in descending order.

MongoDB allows you to create a compound index that contains a maximum of 32 fields.

It's important to understand that the order of the fields specified in a compound index matters.

If a compound index has two fields: field1 and field2, it contains the references to documents sorted by field1 first. And within each value of field1, it has values sorted by field2.

Besides supporting queries that match all the index keys, a compound index can support queries that match the prefix of the index fields. For example, if a compound index contains two fields: field1 and field2, it will support the queries on:

- field1
- field1 and field2

However, it doesn't support the query that matches the field2 only.

MongoDB compound index example

Let's take the example of using compound indexes.

First, create a compound index on the title and year fields of the movies collection:

```
db.movies.createIndex({ title: 1, year: 1 })
```

Code language: JavaScript (javascript)

Output:

title_1_year_1

Code language: JavaScript (javascript)

Second, find the movies whose titles contain the word valley and were released in the year 2014:

```
db.movies.find({title: /valley/gi, year: 2014}).explain('executionStats');
```

Code language: JavaScript (javascript)

Output:

```
...
    inputStage: {
      stage: 'IXSCAN',
      filter: { title: { '$regex': 'valley', '$options': 'is' } },
      nReturned: 3,
      ...
      indexName: 'title_1_year_1',
      ...
...
```

Code language: JavaScript (javascript)

The query uses the index title_1_year_1 instead of scanning the whole collection to find the result.

Third, find the movies whose titles contain the word valley:

```
db.movies.find({title:/valley/gi}).explain('executionStats');
```

Code language: JavaScript (javascript)

Output:

```
...
    inputStage: {
      stage: 'IXSCAN',
      filter: { title: { '$regex': 'valley', '$options': 'is' } },
      nReturned: 21,
      ...j
```

```
    indexName: 'title_1_year_1',
    ...
...
```

Code language: JavaScript (javascript)

This query matches the title only, not the year. However, the query optimizer still makes use of the title_1_year_1 index.

Finally, find the movies that were released in the year 2014:

```
db.movies.find({year: 2014}).explain('executionStats');
```

Code language: JavaScript (javascript)

Output:

```
...
  executionStages: {
    stage: 'COLLSCAN',
    filter: { year: { '$eq': 2014 } },
    nReturned: 1147,
...
```

Code language: JavaScript (javascript)

In this example, the query optimizer doesn't use the title_1_year_1 index but scan the whole collection (COLLSCAN) to find the matches.

Summary

- A compound index contains the references to multiple fields of a collection.
- Use the db.collection.createIndex() method to create a compound index.
- The order of fields in the index is important.

Introduction to the MongoDB drop index

To drop an index from a collection, you use the db.collection.dropIndex() method:

```
db.collection.dropIndex(index)
```

Code language: CSS (css)

In this syntax, the index specifies an index to be dropped. The index can be either a string that specifies the name of an index or a document that describes the specification of the index that you want to drop.

Note that you cannot drop the default index on the _id field.

Starting in MongoDB 4.2, you cannot use the db.collection.dropIndex('*') to drop all non-_id indexes. Instead, you use the dropIndexes() method:

db.collection.dropIndexes()

Code language: CSS (css)

MongoDB drop index examples

Let's take some examples of dropping one or more indexes from a collection.

1) Drop an index example

First, create a new index for the year field in the movies collection:

db.movies.createIndex({year: 1})

Code language: CSS (css)

Output:

year_1

In this example, the createIndex() method creates an index with the name year_1.

Second, show all the indexes of the movies collection by using the getIndexes() method:

db.movies.getIndexes()

Code language: CSS (css)

Output:

```
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { year: 1 }, name: 'year_1' }
]
```

Code language: JavaScript (javascript)

Third, drop the year_1 index using the dropIndex() method:

db.movies.dropIndex('year_1')

Code language: JavaScript (javascript)

Output:

```
{ nIndexesWas: 2, ok: 1 }
```

Code language: CSS (css)

The output shows that one index has been dropped.

Finally, verify the index deletion using the getIndexes() method:

```
db.movies.getIndexes()
```

Code language: CSS (css)

Output:

```
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

Code language: CSS (css)

The output shows that the year_1 index was deleted and is not in the index list of the movies collection.

2) Drop an index by the index specification

First, create an index for runtime field in the movies collection:

```
db.movies.createIndex({runtime: -1})
```

Code language: CSS (css)

Second, list the indexes of the movies collection:

```
db.movies.getIndexes()
```

Code language: CSS (css)

Output:

```
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { runtime: -1 }, name: 'runtime_-1' }
]
```

Code language: JavaScript (javascript)

Third, drop the index runtime_-1 but use the index specification instead of the index name:

```
db.movies.dropIndex({runtime: -1});
```

Code language: CSS (css)

Output:

```
{ nIndexesWas: 2, ok: 1 }
```

Code language: CSS (css)

3) Drop all non-_id indexes example

First, create two indexes for the year and runtime fields of the movies collection:

```
db.movies.createIndex({year: 1})
db.movies.createIndex({runtime: 1})
```

Code language: CSS (css)

Second, show the indexes of the movies collection:

```
db.movies.getIndexes()
```

Code language: CSS (css)

Output:

```
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { year: 1 }, name: 'year_1' },
  { v: 2, key: { runtime: 1 }, name: 'runtime_1' }
]
```

Code language: JavaScript (javascript)

Third, drop all non-_id indexes of movies collection using the dropIndexes() method:

```
db.movies.dropIndexes()
```

Code language: CSS (css)

Output:

```
{
  nIndexesWas: 3,
  msg: 'non-_id indexes dropped for collection',
  ok: 1
}
```

Code language: CSS (css)

Summary

- Use the dropIndex() to drop an index specified by name or specification from a collection.
- The default index for the _id field cannot be dropped.
- Use the dropIndexes() to drop all non-_id indexes of a collection.