**MongoDB Index Demo**

Indexes support efficient execution of queries in MongoDB. Without indexes, MongoDB must scan every document in a collection to return query results. If an appropriate index exists for a query, MongoDB uses the index to limit the number of documents it must scan. Although indexes improve query performance, adding an index has negative performance impact for write operations. For collections with a high write-to-read ratio, indexes are expensive because each insert must also update any indexes.

db.collection.createIndex( { name: -1 } )

# To get all index names for a collection

db.collection.getIndexes()

# To drop a specific index

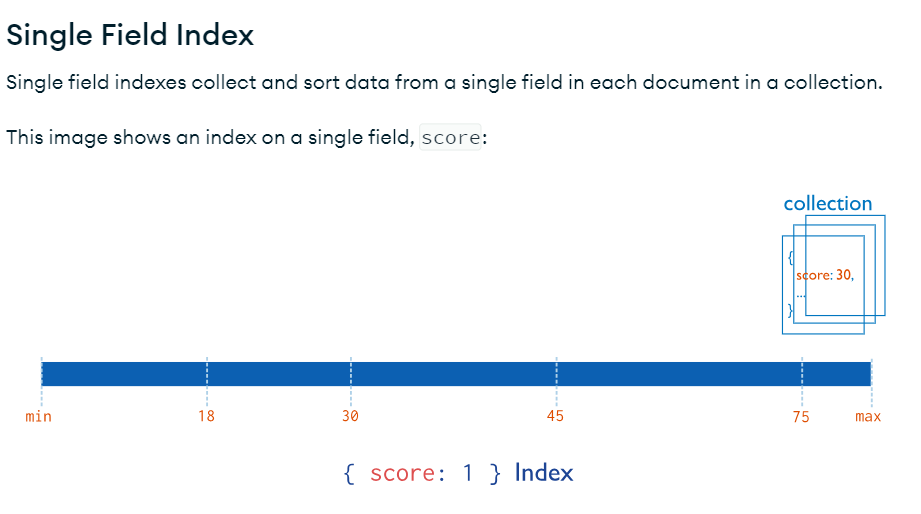
db.<collection>.dropIndex("<indexName>")

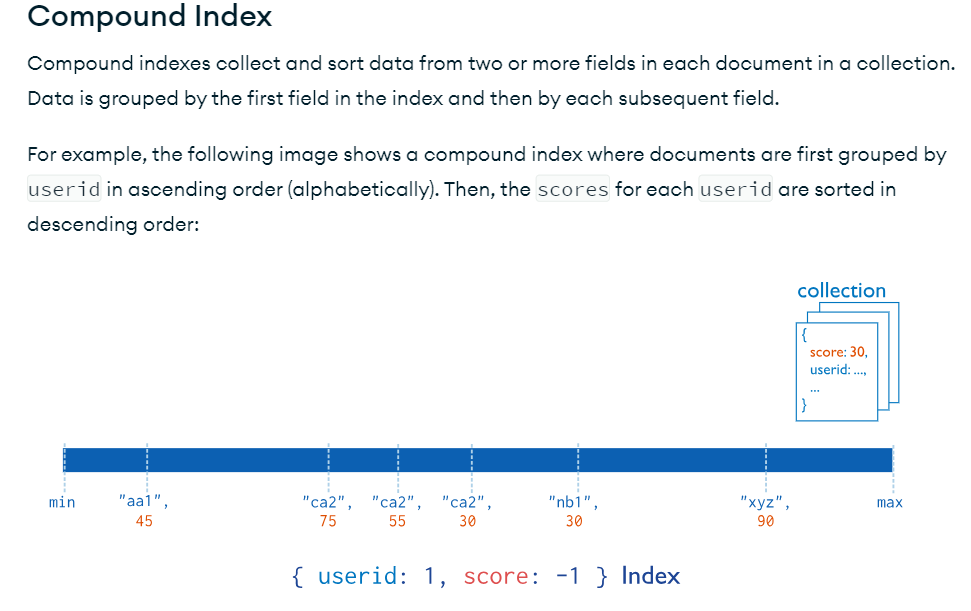
# To drop multiple indexes

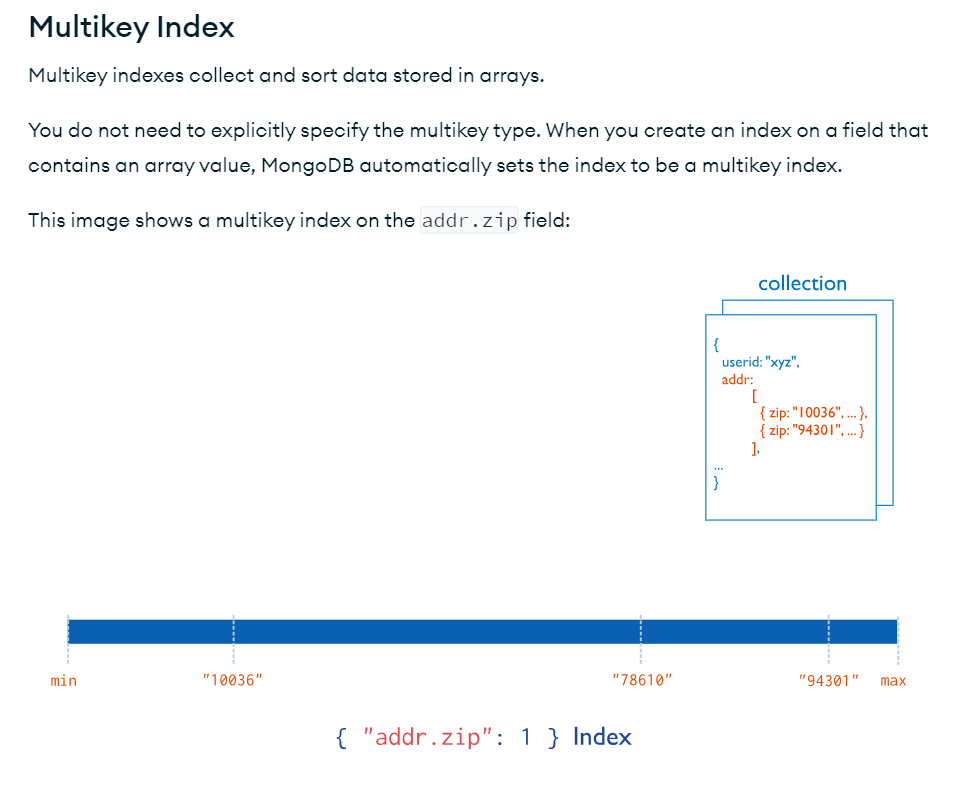
db.<collection>.dropIndexes("<index1>", "<index2>", "<index3>")

db.<collection>.dropIndexes() # To drop all indexes except the \_id index

**Index Types**







**Geospatial Index**

Geospatial indexes improve performance for queries on geospatial coordinate data. To learn more, see Geospatial Indexes.

**Text Index**

Text indexes support text search queries on fields containing string content.

**Hashed Index**

Hashed indexes support hashed sharding. Hashed indexes index the hash of the value of a field.

**Clustered Index**

Clustered indexes specify the order in which clustered collections store data. Collections created with a clustered index are called clustered collections.

**Single Field:**

**Create an Index on a Single Field**

**db.students.insertMany( [**

**{**

**"name": "Alice",**

**"gpa": 3.6,**

**"location": { city: "Sacramento", state: "California" }**

**},**

**{**

**"name": "Bob",**

**"gpa": 3.2,**

**"location": { city: "Albany", state: "New York" }**

**}**

**] )**

**Create an Index on a Single Field**

Consider a school administrator who frequently looks up students by their GPA. You can create an index on the gpa field to improve performance for those queries:

**db.students.createIndex( { gpa: 1 } )**

Results

The index supports queries that select on the field gpa, such as the following:

**db.students.find( { gpa: 3.6 } )**

**db.students.find( { gpa: { $lt: 3.4 } } )**

**Create an Index on an Embedded Document**

You can create indexes on embedded documents as a whole.

Consider a social networking application where students can search for one another by location. Student location is stored in an embedded document called location. The location document contains the fields city and state.

You can create an index on the location field to improve performance for queries on the location document:

**db.students.createIndex( { location: 1 } )**

Results

The following query uses the index on the location field:

**db.students.find( { location: { city: "Sacramento", state: "California" } } )**

For example, the following queries do not use the index on the location field because they query on specific fields within the embedded document:

**db.students.find( { "location.city": "Sacramento" } )**

**db.students.find( { "location.state": "New York" } )**

In order for a dot notation query to use an index, you must create an index on the specific embedded field you are querying, not the entire embedded object.

**Create an Index on an Embedded Field**

You can create indexes on fields within embedded documents. Indexes on embedded fields can fulfill queries that use dot notation.The location field is an embedded document that contains the embedded fields city and state. Create an index on the location.state field:

**db.students.createIndex( { "location.state": 1 } )**

Results

The index supports queries on the field location.state, such as the following:

**db.students.find( { "location.state": "California" } )**

**db.students.find( { "location.city": "Albany", "location.state": "New York" } )**

**Compound Indexes**

Compound indexes collect and sort data from two or more fields in each document in a collection. Data is grouped by the first field in the index and then by each subsequent field.

**db.students.insertMany([**

**{**

**"name": "Alice",**

**"gpa": 3.6,**

**"location": { city: "Sacramento", state: "California" }**

**},**

**{**

**"name": "Bob",**

**"gpa": 3.2,**

**"location": { city: "Albany", state: "New York" }**

**}**

**])**

**The following operation creates a compound index containing the name and gpa fields:**

**db.students.createIndex( {**

**name: 1,**

**gpa: -1**

**} )**

In this example:

The index on name is ascending (1).

The index on gpa is descending (-1).

Results

The created index supports queries that select on:

Both name and gpa fields.

Only the name field, because name is a prefix of the compound index.

For example, the index supports these queries:

**db.students.find( { name: "Alice", gpa: 3.6 } )**

**db.students.find( { name: "Bob" } )**

The index does not support queries on only the gpa field, **because gpa is not part of the index prefix**. For example, the index does not support this query:

**db.students.find( { gpa: { $gt: 3.5 } } )**

**db.leaderboard.insertMany( [**

**{**

**"score": 50,**

**"username": "Alex Martin",**

**"date": ISODate("2022-03-01T00:00:00Z")**

**},**

**{**

**"score": 55,**

**"username": "Laura Garcia",**

**"date": ISODate("2022-03-02T00:00:00Z")**

**},**

**{**

**"score": 60,**

**"username": "Alex Martin",**

**"date": ISODate("2022-03-03T00:00:00Z")**

**},**

**{**

**"score": 60,**

**"username": "Riya Patel",**

**"date": ISODate("2022-03-04T00:00:00Z")**

**},**

**{**

**"score": 50,**

**"username": "Laura Garcia",**

**"date": ISODate("2022-03-05T00:00:00Z")**

**}**

**] )**

**db.leaderboard.find().sort( { score: -1, username: 1 } )**

The results are sorted first by score in descending order, then by username in ascending order (alphabetically).

**db.leaderboard.createIndex( { score: -1, username: 1 } )**

This compound index stores:

score values in descending order.

**username values in ascending order (alphabetically).**

**Reverse Results:**

**db.leaderboard.find().sort( { score: 1, username: -1 } )**

The following query returns the leaderboard in reverse order, where results are sorted first by ascending score values and then by descending username values (reverse alphabetically).

**Multikey Indexes**

Multikey indexes collect and sort data from fields containing array values. Multikey indexes improve performance for queries on array fields.

You do not need to explicitly specify the multikey type. When you create an index on a field that contains an array value, MongoDB automatically sets that index to be a multikey index.

db.students.insertMany( [

{

"name": "Andre Robinson",

"test\_scores": [ 88, 97 ]

},

{

"name": "Wei Zhang",

"test\_scores": [ 62, 73 ]

},

{

"name": "Jacob Meyer",

"test\_scores": [ 92, 89 ]

}

] )

The following operation creates an ascending multikey index on the test\_scores field of the students collection:

**db.students.createIndex( { test\_scores: 1 } )**

Because test\_scores contains an array value, MongoDB stores this index as a multikey index.

Results

The index contains a key for each individual value that appears in the test\_scores field. The index is ascending, meaning the keys are stored in this order: [ 62, 73, 88, 89, 92, 97 ].

The index supports queries that select on the test\_scores field. For example, the following query returns documents where at least one element in the test\_scores array is greater than 90:

**db.students.find(**

**{**

**test\_scores: { $elemMatch: { $gt: 90 } }**

**}**

**)**

**Create an Index on an Embedded Field in an Array**

You can create indexes on embedded document fields within arrays. These indexes improve performance for queries on specific embedded fields that appear in arrays. When you create an index on a field inside an array, MongoDB stores that index as a multikey index.

db.inventory.insertMany( [

{

"item": "t-shirt",

"stock": [

{

"size": "small",

"quantity": 8

},

{

"size": "large",

"quantity": 10

},

]

},

{

"item": "sweater",

"stock": [

{

"size": "small",

"quantity": 4

},

{

"size": "large",

"quantity": 7

},

]

},

{

"item": "vest",

"stock": [

{

"size": "small",

"quantity": 6

},

{

"size": "large",

"quantity": 1

}

]

}

] )

The following operation creates an ascending multikey index on the stock.quantity field of the inventory collection:

**db.inventory.createIndex( { "stock.quantity": 1 } )**

Because stock contains an array value, MongoDB stores this index as a multikey index.

The index supports queries that select on the stock.quantity field. For example, the following query returns documents where at least one element in the stock array has a quantity less than 5:

**db.inventory.find(**

**{**

**"stock.quantity": { $lt: 5 }**

**}**

**)**

**Sort Results**

**The index also supports sort operations on the stock.quantity field, such as this query:**

**db.inventory.find().sort( { "stock.quantity": -1 } )**

**Text Indexes**

Text indexes support text search queries on fields containing string content. Text indexes improve performance when searching for specific words or phrases within string content.A collection can only have one text index, but that index can cover multiple fields.

db.blog.insertMany( [

{

\_id: 1,

content: "This morning I had a cup of coffee.",

about: "beverage",

keywords: [ "coffee" ]

},

{

\_id: 2,

content: "Who likes chocolate ice cream for dessert?",

about: "food",

keywords: [ "poll" ]

},

{

\_id: 3,

content: "My favorite flavors are strawberry and coffee",

about: "ice cream",

keywords: [ "food", "dessert" ]

}

] )

Create a Single-Field Text Index

Create a text index on the content field:

**db.blog.createIndex( { "content": "text" } )**

The index supports text search queries on the content field. For example, the following query returns documents where the content field contains the string coffee:

**db.blog.find(**

**{**

**$text: { $search: "coffee" }**

**}**

**)**

**Create a compound text index on the about and keywords fields in the blog collection:**

**db.blog.createIndex(**

**{**

**"about": "text",**

**"keywords": "text"**

**}**

**)**

**The index supports text search queries on the about and keywords fields. For example, the following query returns documents where the string food appears in either the about or keywords field:**

**db.blog.find(**

**{**

**$text: { $search: "food" }**

**}**

**)**

**Limit Number of Text Index Entries Scanned**

**db.inventory.insertMany( [**

**{ \_id: 1, department: "tech", description: "lime green computer" },**

**{ \_id: 2, department: "tech", description: "wireless red mouse" },**

**{ \_id: 3, department: "kitchen", description: "green placemat" },**

**{ \_id: 4, department: "kitchen", description: "red peeler" },**

**{ \_id: 5, department: "food", description: "green apple" },**

**{ \_id: 6, department: "food", description: "red potato" }**

**] )**

**db.inventory.createIndex(**

**{**

**department: 1,**

**description: "text"**

**}**

**)**

**db.inventory.find( { department: "kitchen", $text: { $search: "green" } } )**

how many documents were scanned to return the query, view the query's executionStats:

**db.inventory.find(**

**{**

**department: "kitchen", $text: { $search: "green" }**

**}**

**).explain("executionStats")**