


[Home](#) » [MongoDB Indexes](#) » MongoDB Compound index


MongoDB Compound index

ADVERTISEMENTS

Google Play 게임즈로 PC에서 즐겨보세요

Google Play 게임즈에서 아키텍트
플레이하고 최대 20% 할인 +
포인트 10배 적립



 Google Play Games

Summary: in this tutorial, you'll learn about MongoDB compound indexes and how to create a compound index for fields in a collection.

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  
})
```

Google Play 게임즈로 PC에서 즐겨보세요

아키텍트 PC로 즐기고 최대 70,000까지 20% 할인과 포인트 10배 적립 받으세요
Google Play Games

자세히 알아보기 >

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 })
```

Output:

```
title_1_year_1
```

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');
```

Output:

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

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');
```

Output:

```
...
  inputStage: {
    stage: 'IXSCAN',
    filter: { title: { '$regex': 'valley', '$options': 'is' } },
    nReturned: 21,
    ...j
    indexName: 'title_1_year_1',
    ...
  }
  ...
}
```

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');
```

Output:

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

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.

Was this tutorial helpful ?



ADVERTISEMENTS

UP NEXT

[MongoDB Unique Index](#)



PREVIOUSLY

[MongoDB Drop Index](#)

Search ...

ADVERTISEMENTS

GETTING STARTED

[What is MongoDB](#)

[Install MongoDB](#)

[MongoDB Basics](#)

[MongoDB Shell](#)

[MongoDB Data Types](#)

INSERTING DOCUMENTS

[insertOne](#)

[insertMany](#)

SELECTING DOCUMENTS

[findOne](#)

[find](#)

[Projection: Selecting Returned Fields](#)

COMPARISON QUERY OPERATORS

[\\$eq: Equal To Operator](#)

[\\$lt: Less Than Operator](#)

[\\$lte: Less Than or Equal To Operator](#)

[\\$gt: Greater Than Operator](#)

[\\$gte: Greater Than or Equal To Operator](#)

[\\$ne: Not Equal To Operator](#)

[\\$in: In Operator](#)

[\\$nin: Not In Operator](#)

ADVERTISEMENTS

LOGICAL QUERY OPERATORS

[\\$and](#): Logical AND Operator

[\\$or](#): Logical OR Operator

[\\$not](#): Logical NOT Operator

[\\$nor](#): Logical NOR Operator

ELEMENT QUERY OPERATORS

[\\$exists](#)

[\\$type](#)

ARRAY QUERY OPERATORS

[\\$size](#)

[\\$all](#)

[\\$elemMatch](#)

SORTING & LIMITING

[sort\(\)](#): Sorting documents

[limit\(\)](#): Limiting documents

ADVERTISEMENTS

UPDATING DOCUMENTS

[updateOne](#): Update one Document

updateMany: Update Multiple Documents

\$inc: Increase / Decrease Field Value

\$min: Update Field Value

\$max: Update Field Value

\$mul: Mutiply Field By a Number

\$unset: Remove Fields

\$rename: Rename Fields

Upsert

DELETING DOCUMENTS

deleteOne

deleteMany

AGGREGATION

Aggregation Pipeline

\$avg

\$count

\$sum

\$max

\$min

INDEXES

MongoDB Indexes

Create Index

Unique Index

Compound index

ABOUT MONGODBTUTORIAL.COM

This MongoDB Tutorial helps you master MongoDB quickly.

RECENT TUTORIALS

[MongoDB \\$min](#)

[MongoDB \\$max](#)

[MongoDB \\$avg](#)

[MongoDB \\$count](#)

[MongoDB \\$sum](#)

SITE LINKS

[Home](#)

[Contact](#)

[About](#)

[Privacy Policy](#)

Copyright © 2022 by mongodbtutorial.org Website. All Right Reserved.

This mongodbtutorial.org website has no relationship with MongoDB, Inc Also, MongoDB, Inc does not sponsor contents on this website.