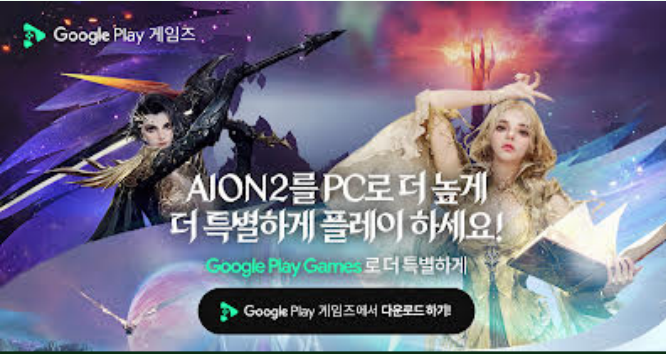


# MongoDB Unique Index

ADVERTISEMENTS

모두의 플레이에 날개를 달다  
AION2



다운로드하기 >

**Summary:** in this tutorial, you'll learn about unique indexes and how to use them to enforce the uniqueness of values for a field across documents in a collection.

## 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:



Google Play 게임즈로 즐기는 AION2

PC에서 AION2 플레이하고, 구글플레이 포인트 8배 적립까지 Google

다운로드하기 >

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"},
]);
```

Second, create a unique index for the email field:

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

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

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

MongoDB returned the following error:

```
MongoServerError: E11000 duplicate key error collection: mflix.users index: em
```

## 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"},  
])
```

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

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

MongoDB returned the following error:

```
MongoServerError: Index build failed: 95f78956-d5d0-4882-bfe0-2d856df18c61: Co
```

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'});
```

Output:

```
{ acknowledged: true, deletedCount: 1 }
```

Then, create a unique index for the email field:

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

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});
```

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  
})
```

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 });
```

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  
})
```

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  
})
```

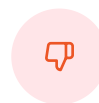
MongoDB issued the following error:

```
MongoServerError: E11000 duplicate key error collection: mflix.locations index
```

## 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.

Was this tutorial helpful ?



ADVERTISEMENTS

UP NEXT

[MongoDB Tools](#)



PREVIOUSLY

[MongoDB Compound index](#)

Search ...

ADVERTISEMENTS

## Google Play 게임즈로 즐기는 AION2

PC에서 AION2 플레이하고,  
구글플레이 포인트 8배  
적립까지



Google Play Games

### 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

Drop 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.