MongoDB CRUD Operations

Complete Guide to Create, Read, Update, and Delete Operations

Updated for MongoDB 8.0 (2024) • Beginner-Friendly Guide with Latest Best Practices

Introduction to CRUD Operations

What is CRUD?

CRUD represents the four fundamental operations for managing data in MongoDB. These operations form the backbone of database interactions and allow you to manipulate documents within collections effectively.

- **Create:** Insert new documents into collections
- **Read:** Query and retrieve documents from collections
- **Update:** Modify existing documents with new values
- **Delete:** Remove documents from collections permanently

Essential Key Concepts

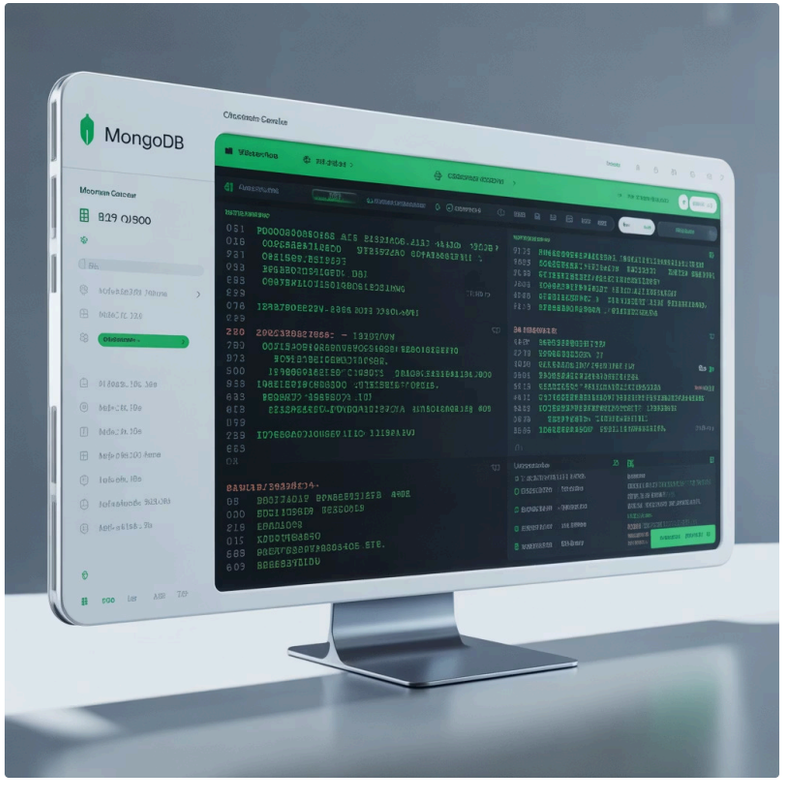
Understanding these foundational principles will help you work more effectively with MongoDB.

- All write operations are **atomic at the document level**, ensuring data consistency
- Collections are created **automatically** when the first document is inserted
- MongoDB stores data in **BSON** (Binary JSON) format for efficient storage and retrieval
- Extended JSON format provides human-readable representation of BSON data types

Exploring Databases and Collections

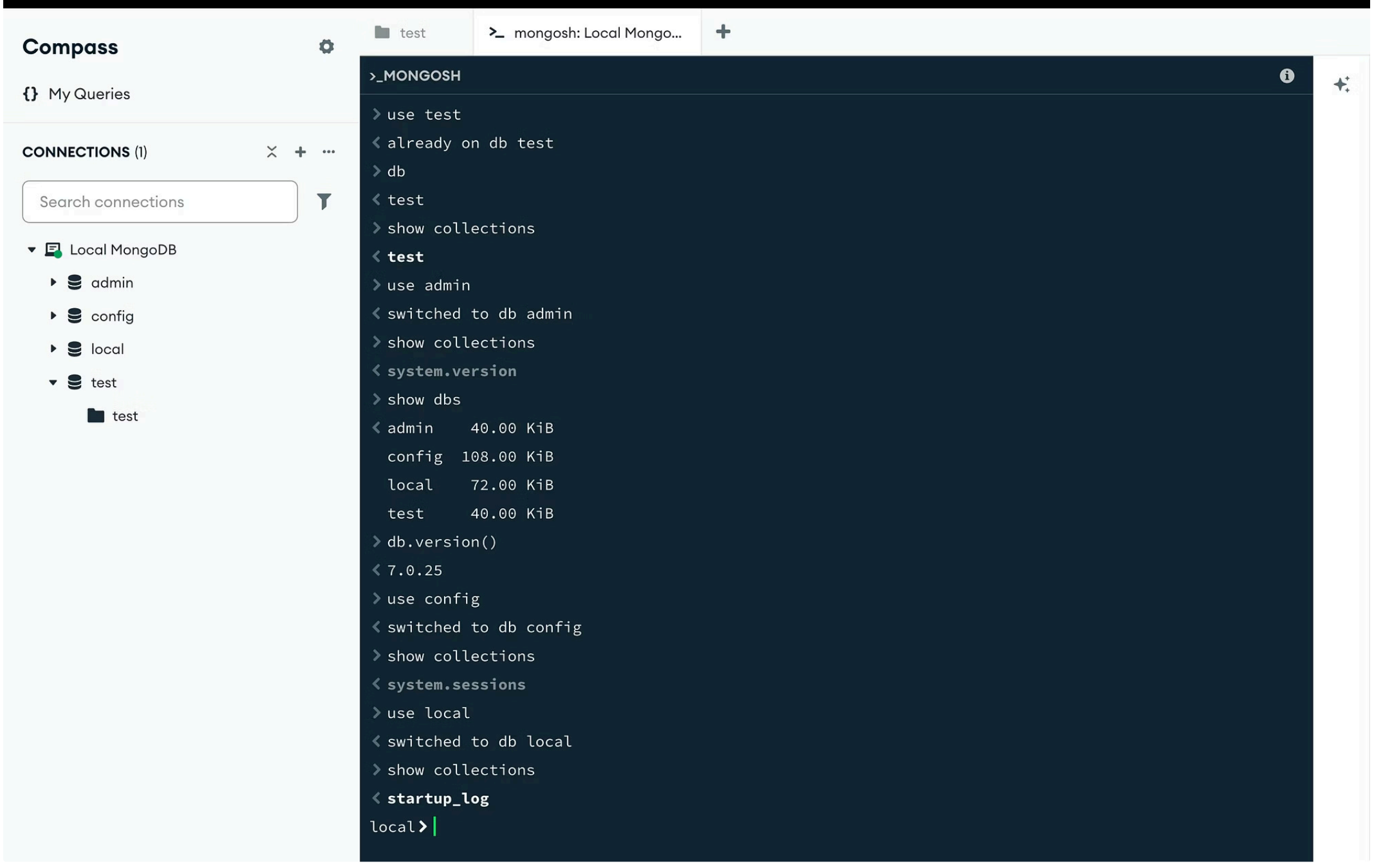
Essential Database Commands

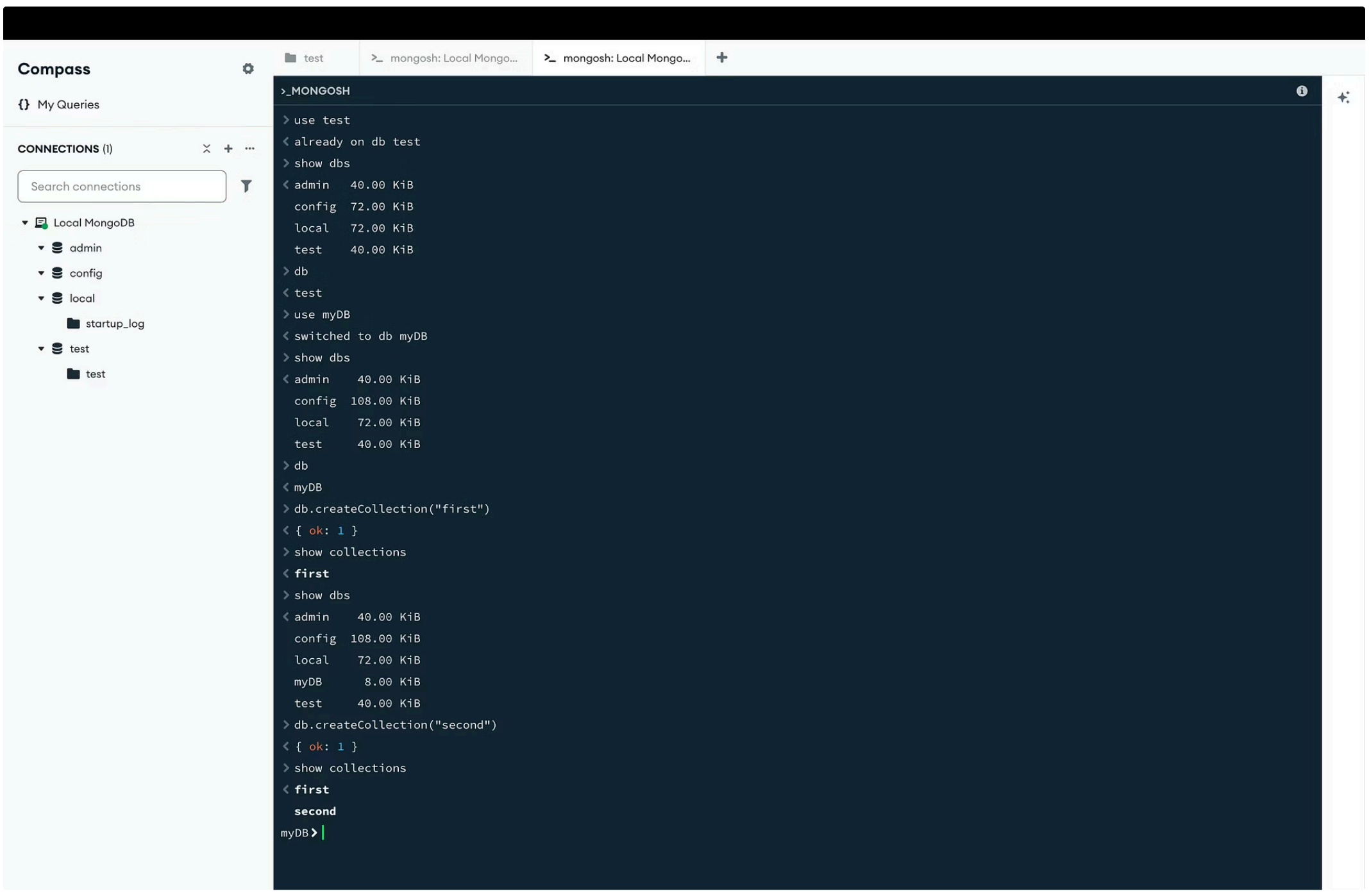
These fundamental commands help you navigate and manage your MongoDB environment. Mastering these will make working with databases significantly easier.




```
// Show the currently active database
db

// Display all available databases
show dbs

// Switch to or create a database
use <database_name>

// List all collections in active database
show collections
```

Important: When using `use <database_name>`, the database is only created when you add the first collection. Until then, it exists only in your session.






```
   The server generated these startup warnings when booting
   2025-10-19T09:38:51.156+05:30: Access control is not enabled for the database. Read and write access to
data and configuration is unrestricted
------

test> show dbs
admin    40.00 KiB
config  108.00 KiB
local    72.00 KiB
myDB     16.00 KiB
test     40.00 KiB
test> use myDB
switched to db myDB
myDB> show collections
first
second
myDB> db.create
db.createUser                 db.createCollection
db.createEncryptedCollection  db.createView
db.createRole

myDB> db.create
db.createUser                 db.createCollection
db.createEncryptedCollection  db.createView
db.createRole

myDB> db.createCollection("third")
{ ok: 1 }
myDB> show collections
first
second
third
myDB>
```

Creating and Deleting Databases & Collections

Creating Collections

MongoDB offers two approaches for collection creation:

```
// Method 1: Explicit creation
db.createCollection("collection_name")

// Method 2: Implicit (preferred)
// Created automatically when
// inserting first document
```

Deleting Collections

Remove collections using either method:

```
// Method 1
db.collection_name.drop()

// Method 2
db.getCollection("collection_name")
  .drop()

// Returns true if successful,
// false if collection doesn't exist
```

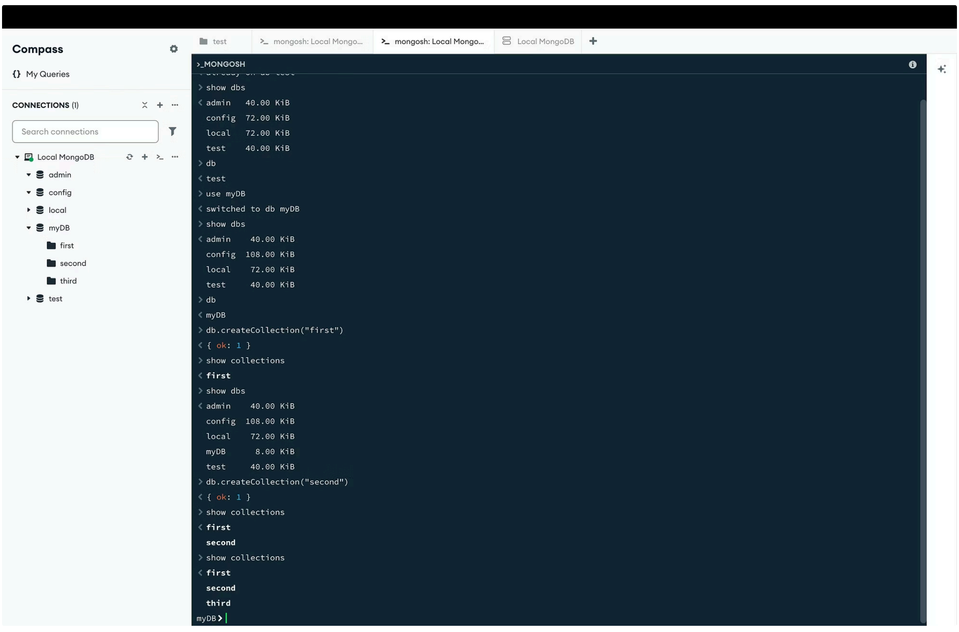
Deleting Database

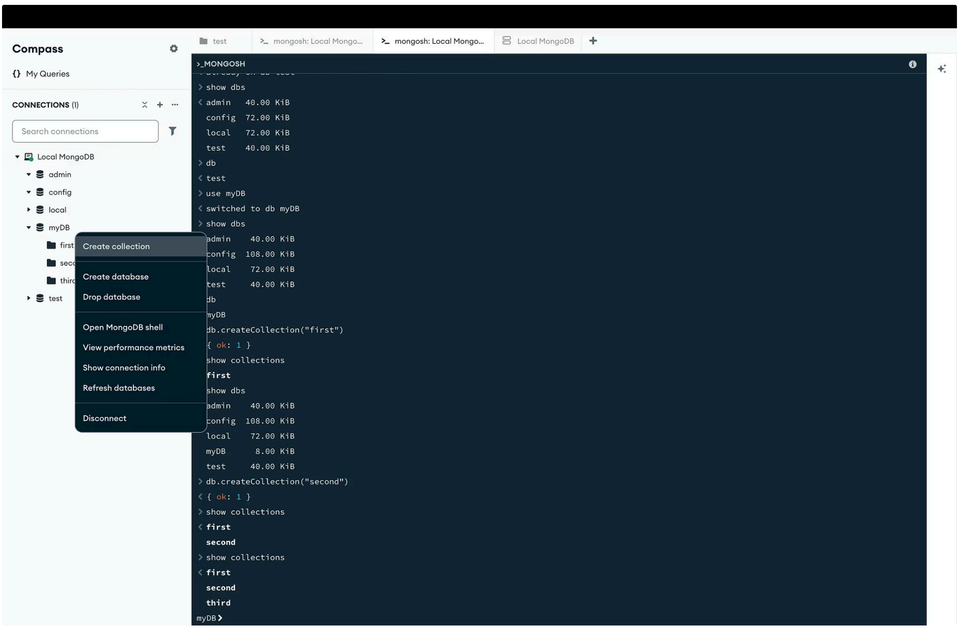
Permanently remove the active database:

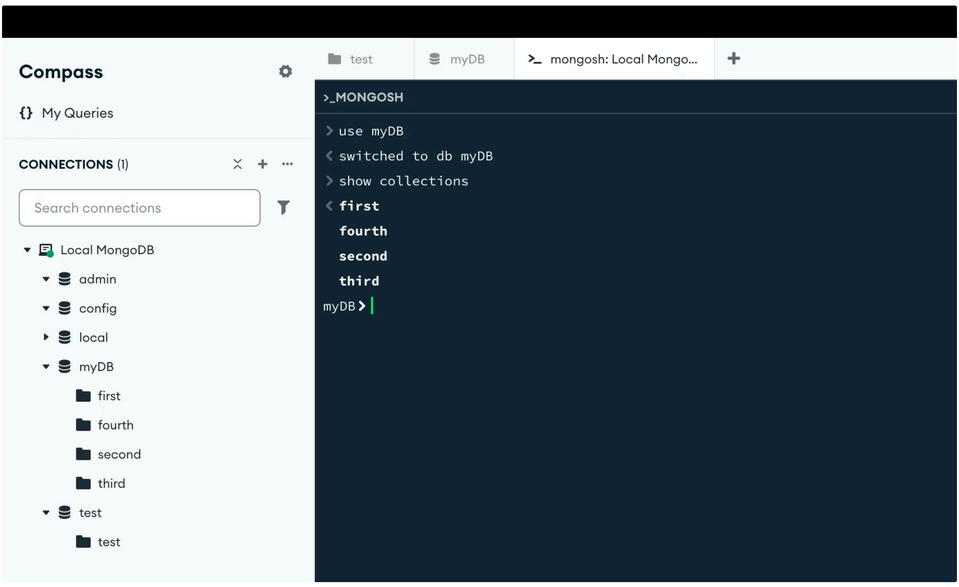
```
db.dropDatabase()
```

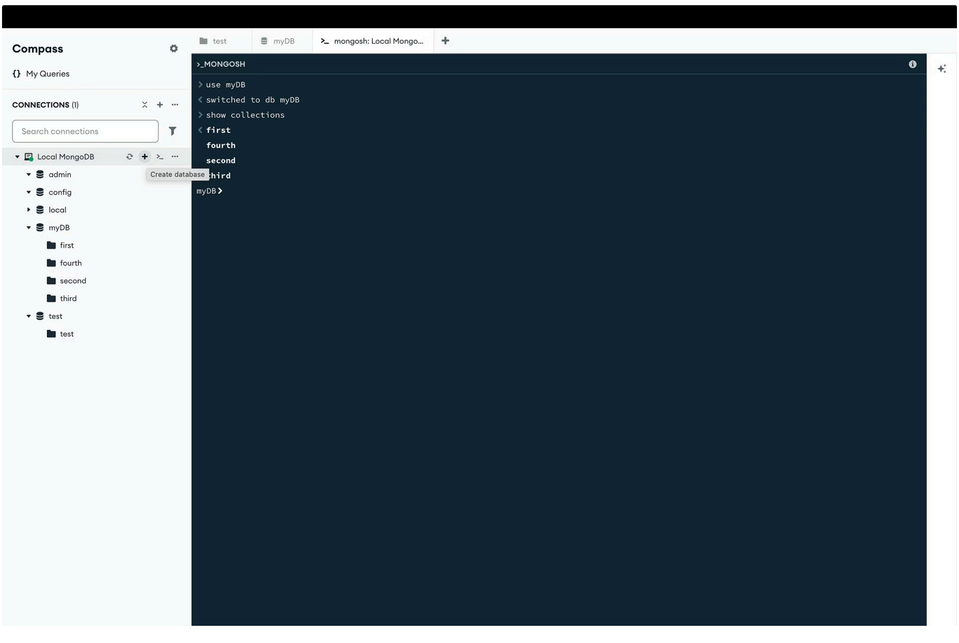
⚠️ Warning: This permanently deletes the active database and all its collections. Use with extreme caution!

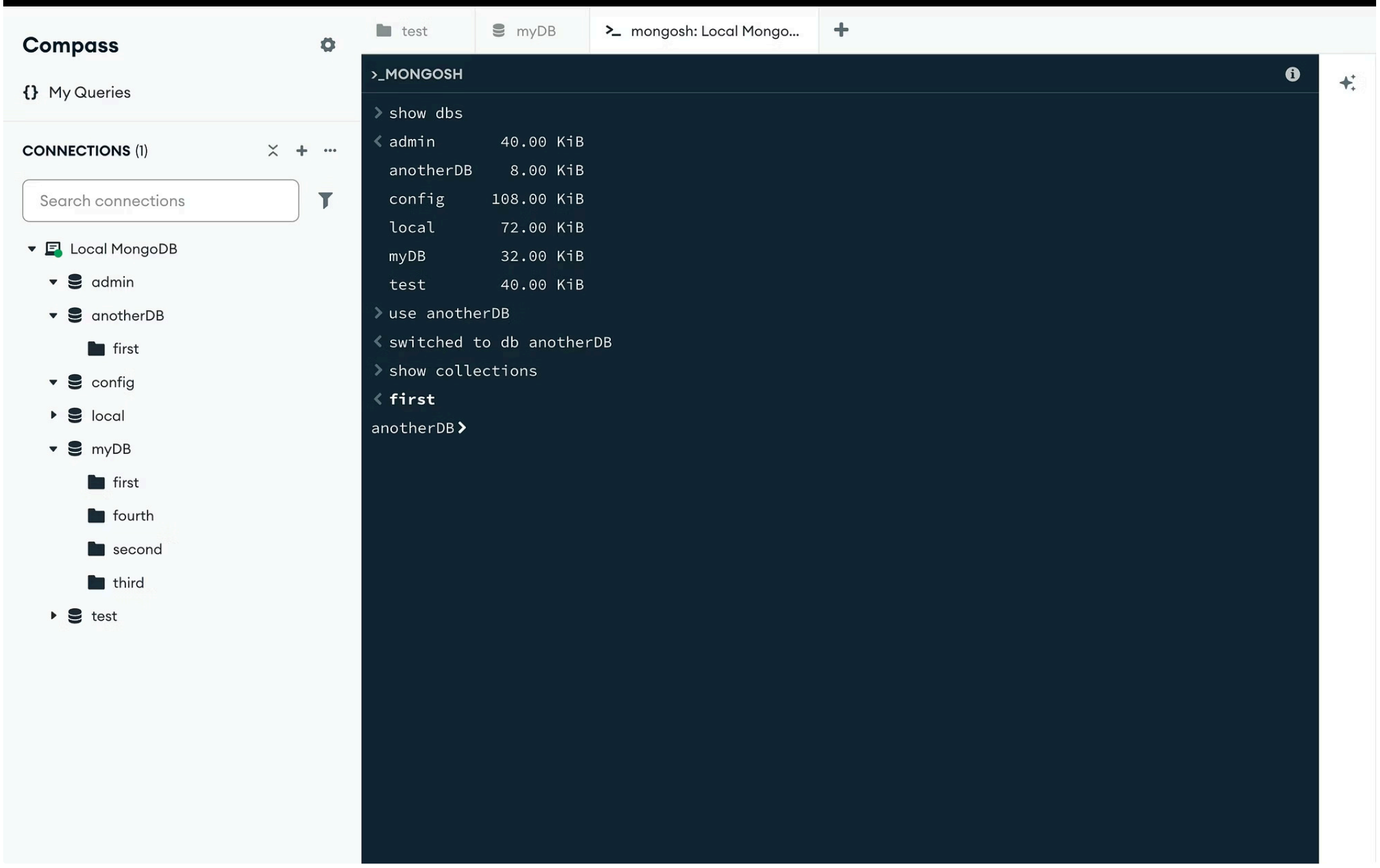
Key Point: There is no explicit command to create a database. Databases are created automatically when you insert the first document into a collection.


Compass
My Queries
CONNECTIONS (1)
Search connections
Local MongoDB
admin
config
local
myDB
first
second
third
test
test
mongosh: Local Mongo...
mongosh: Local Mongo...
Local MongoDB
>_MONGOSH
> show dbs
< admin 40.00 KiB
config 72.00 KiB
local 72.00 KiB
test 40.00 KiB
> db
< test
> use myDB
< switched to db myDB
> show dbs
< admin 40.00 KiB
config 108.00 KiB
local 72.00 KiB
test 40.00 KiB
> db
< myDB
> db.createCollection("first")
< { ok: 1 }
> show collections
< first
> show dbs
< admin 40.00 KiB
config 108.00 KiB
local 72.00 KiB
myDB 8.00 KiB
test 40.00 KiB
> db.createCollection("second")
< { ok: 1 }
> show collections
< first
second
> show collections
< first
second
third
myDB>


Compass
My Queries
CONNECTIONS (1)
Search connections
Local MongoDB
admin
config
local
myDB
test
Create collection
Create database
Drop database
Open MongoDB shell
View performance metrics
Show connection info
Refresh databases
Disconnect
mongosh: Local Mongo...
mongosh: Local Mongo...
Local MongoDB
>_MONGOSH
> show dbs
< admin 40.00 KiB
config 72.00 KiB
local 72.00 KiB
test 40.00 KiB
> db
< test
> use myDB
< switched to db myDB
> show dbs
< admin 40.00 KiB
config 108.00 KiB
local 72.00 KiB
test 40.00 KiB
> db.createCollection("first")
< { ok: 1 }
> show collections
< first
> show dbs
< admin 40.00 KiB
config 108.00 KiB
local 72.00 KiB
myDB 8.00 KiB
test 40.00 KiB
> db.createCollection("second")
< { ok: 1 }
> show collections
< first
second
> show collections
< first
second
third
myDB>


Compass
My Queries
CONNECTIONS (1)
Search connections
Local MongoDB
admin
config
local
myDB
first
fourth
second
third
test
test
test
myDB
mongosh: Local Mongo...
>_MONGOSH
> use myDB
< switched to db myDB
> show collections
< first
fourth
second
third
myDB>


Compass
My Queries
CONNECTIONS (1)
Search connections
Local MongoDB
admin
config
local
myDB
first
fourth
second
third
test
test
Create database
test
myDB
mongosh: Local Mongo...
>_MONGOSH
> use myDB
< switched to db myDB
> show collections
< first
fourth
second
third
myDB>




```
db.getMongo             db.getName              db.getCollectionNames  db.getCollectionInfos
db.getSiblingDB         db.getCollection        db.getUser             db.getUsers
db.getRole              db.getRoles             db.getProfilingStatus  db.getLogComponents

anotherDB> db.getCollection("first").drop()
true
anotherDB> show collections

anotherDB> show dbs
admin    40.00 KiB
config  108.00 KiB
local    72.00 KiB
myDB     32.00 KiB
test     40.00 KiB
anotherDB> use myDB
switched to db myDB
myDB> show collections
first
fourth
second
third
myDB> db.getCollection("third").drop()
true
myDB> show collections
first
fourth
second
myDB> db.fourth.drop()
true
myDB> show collections
first
second
myDB>
```

CREATE Operations Overview

Available Insert Methods

MongoDB provides several methods for inserting documents. Understanding which methods are current and which are deprecated is crucial for writing maintainable code that follows best practices.

Method	Purpose	Return Type	Status
insertOne()	Insert single document	Object with insertedId	✅ Current
insertMany()	Insert multiple documents	Object with insertedIds array	✅ Current
bulkWrite()	Multiple mixed operations	BulkWriteResult	✅ Current
insert()	Insert one or many	WriteResult/BulkWriteResult	⚠️ DEPRECATED

⚠️ **DEPRECATED:** `db.collection.insert()`

✅ **USE INSTEAD:**

- `insertOne()` for single documents
- `insertMany()` for multiple documents
- `bulkWrite()` for mixed operations

Why Deprecated? The insert() method was deprecated to align with the MongoDB CRUD specification and provide more explicit, predictable behaviour.

insertOne() Method

Syntax and Parameters

```
db.collection_name.insertOne(
  <document>,
  {
    writeConcern: <document>
  }
)
```

Parameters Explained

- **document:** The BSON document object to insert into the collection
- **writeConcern (optional):** Level of acknowledgement requested from MongoDB for the write operation

Return Value

```
{
  "acknowledged": true,
  "insertedId": ObjectId("...")
}
```

Practical Example

```
db.students.insertOne({
  name: "Alice Johnson",
  age: 22,
  grade: "A",
  subjects: [
    "Maths",
    "Physics",
    "Computer Science"
  ]
})
```

Essential Points

- If `_id` is not specified, MongoDB automatically generates an ObjectId
- Operation is atomic at the document level, ensuring consistency
- Returns acknowledgement status and the inserted document's ID for reference

Compass

My Queries

CONNECTIONS (1)

Search connections

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongosh: Local Mongo...

>_MONGOSH

```
> use myDB
< switched to db myDB
> show collections
< first
  second
> db.getCollection("first").insert({})
< DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68f76c5c83e212d2e7c10348')
    }
  }
> db.getCollection("first").insertone({})
✖ ▸ TypeError: db.getCollection("first").insertone is not a function
> db.getCollection("first").insertOne({})
< {
    acknowledged: true,
    insertedId: ObjectId('68f76c9783e212d2e7c10349')
  }
> db.first.find()
< {
    _id: ObjectId('68f76c5c83e212d2e7c10348')
  }
  {
    _id: ObjectId('68f76c9783e212d2e7c10349')
  }
```

Compass

My Queries

CONNECTIONS (1)

Search connections

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongosh: Local Mongo... | mongosh: Local Mongo...

>_MONGOSH

```
< switched to db myDB
> db.first.insertMany([{} , {} , {}])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68f76e024b4e6973452bef20'),
      '1': ObjectId('68f76e024b4e6973452bef21'),
      '2': ObjectId('68f76e024b4e6973452bef22')
    }
  }
> db.first.find()
< {
    _id: ObjectId('68f76c5c83e212d2e7c10348')
  }
  {
    _id: ObjectId('68f76c9783e212d2e7c10349')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef20')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef21')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef22')
  }
myDB >
```

Compass

My Queries

CONNECTIONS (1)

Search connections

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongosh: Local Mongo... | mongosh: Local Mongo...

>_MONGOSH

```
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef21')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef22')
  }
> db.first.find()
< {
    _id: ObjectId('68f76c5c83e212d2e7c10348')
  }
  {
    _id: ObjectId('68f76c9783e212d2e7c10349')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef20')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef21')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef22')
  }
  {
    _id: ObjectId('68f76eec584125e9a8b4307c')
  }
myDB >
```

insertMany() Method

Syntax and Configuration

```
db.collection_name.insertMany(
  [<document1>, <document2>, ...],
  {
    writeConcern: <document>,
    ordered: <boolean>  // Optional parameters
  }
)
```

Parameters Explained

- **documents:** Array of document objects to insert
- **ordered (optional, default true):** If true, stops on first error; if false, continues with remaining documents
- **writeConcern (optional):** Write acknowledgement level for the operation

Return Value Structure

```
{
  "acknowledged": true,
  "insertedIds": [
    ObjectId("507f...39011"),
    ObjectId("507f...39012"),
    ObjectId("507f...39013")
  ]
}
```

Comprehensive Example

```
db.students.insertMany([
  {
    name: "Bob Smith",
    age: 23,
    grade: "B"
  },
  {
    name: "Carol White",
    age: 21,
    grade: "A"
  },
  {
    name: "David Brown",
    age: 24,
    grade: "C"
  }
])
```

🚀 **Performance Note:** In MongoDB 8.0, bulk inserts are 56% faster than previous versions, making insertMany() significantly more efficient for large-scale operations!

Compass

My Queries

CONNECTIONS (1)

Search connections

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongosh: Local Mongo... | mongosh: Local Mongo...

>_MONGOSH

```
  }
  {
    _id: ObjectId('68f76c9783e212d2e7c10349')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef20')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef21')
  }
  {
    _id: ObjectId('68f76e024b4e6973452bef22')
  }
  {
    _id: ObjectId('68f76eec584125e9a8b4307c')
  }
> db.first.remove({})
< DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete
< {
    acknowledged: true,
    deletedCount: 6
  }
> df.first.find()
✖ ▸ ReferenceError: df is not defined
> db.first.find()
<
myDB >
```

Compass

My Queries

CONNECTIONS (1)

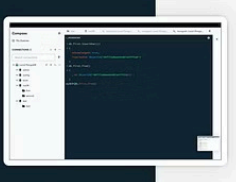
Search connections

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongosh: Local Mongo... | mongosh: Local Mongo... | mongosh: Local Mongo...

>_MONGOSH

```
> db.first.insertOne({})
< {
    acknowledged: true,
    insertedId: ObjectId('68f771ebeee01d67a47ff2a9')
  }
> db.first.find()
< {
    _id: ObjectId('68f771ebeee01d67a47ff2a9')
  }
> db.first.find()
< {
    _id: ObjectId('68f771ebeee01d67a47ff2a9')
  }
  {
    _id: ObjectId('68f77238584125e9a8b4307d')
  }
myDB >
```



Compass

My Queries

CONNECTIONS (1)

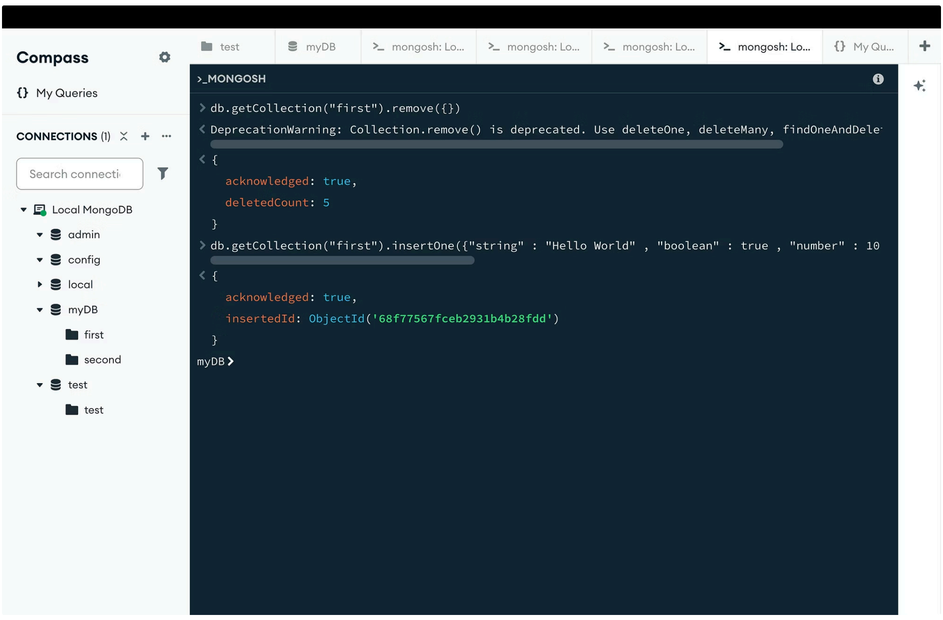
Search connections

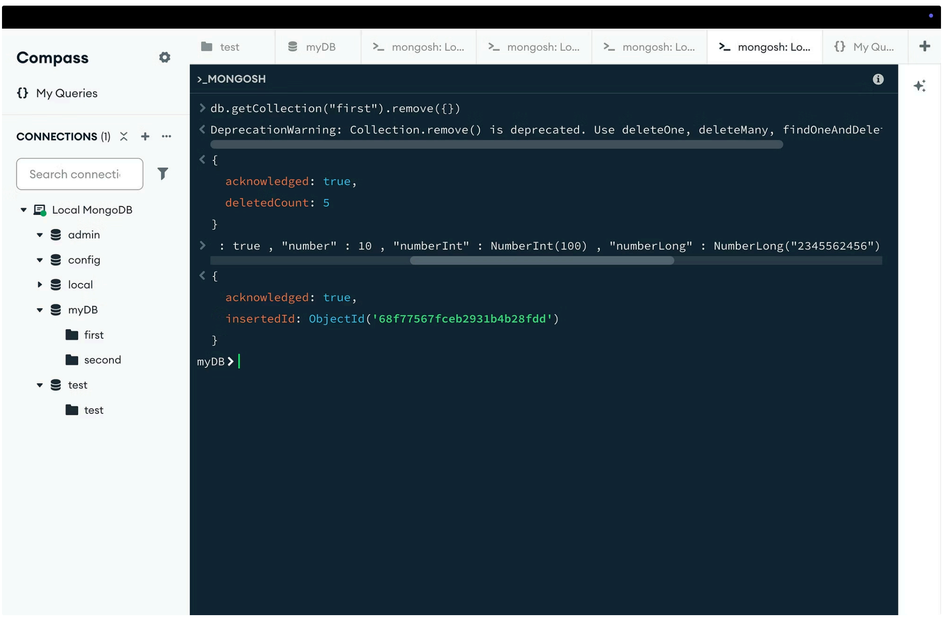
- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

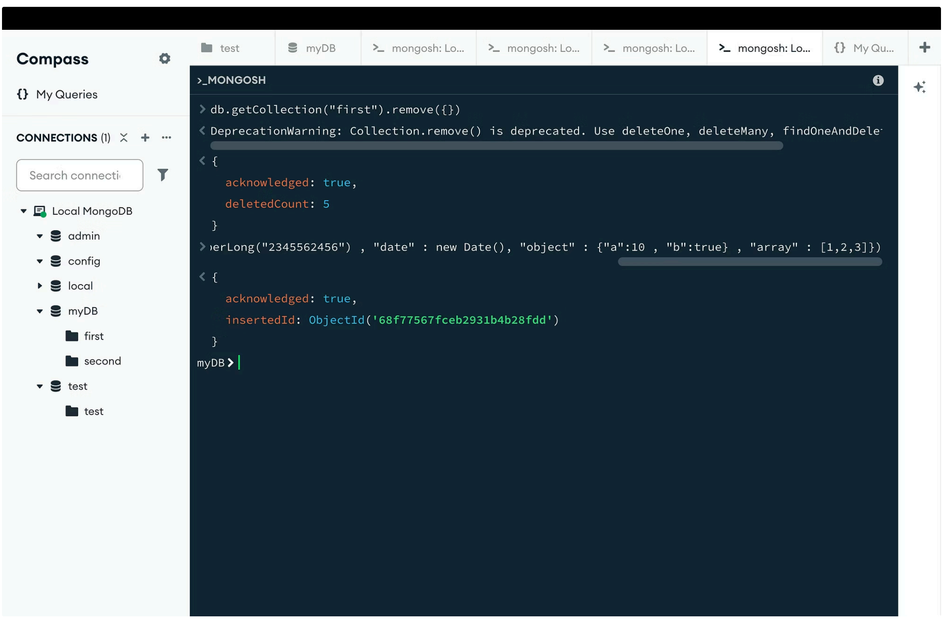
test | myDB | mongosh: Local ... | mongosh: Local ... | mongosh: Local ... | mongosh: Local ...

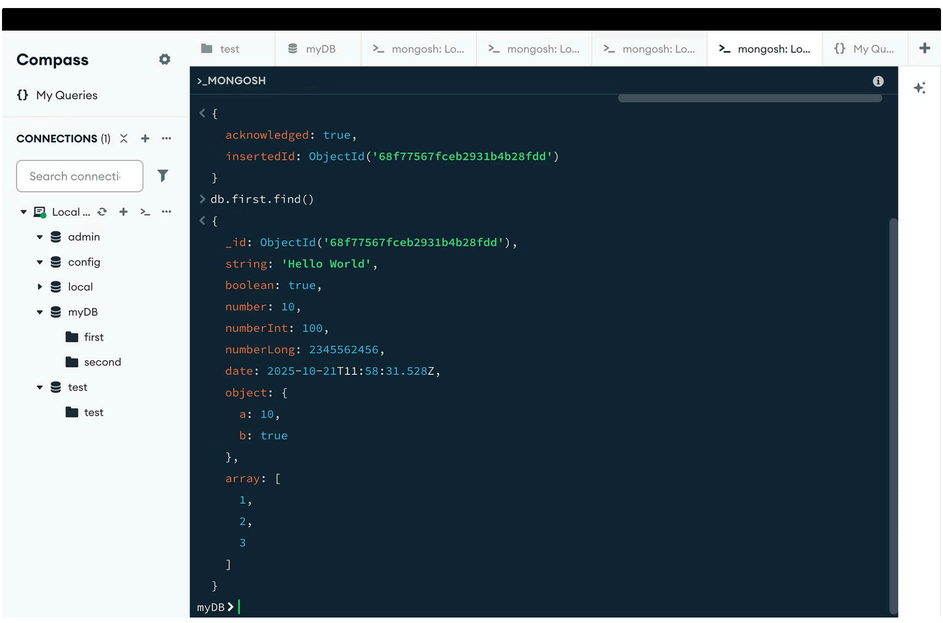
>_MONGOSH

```
> db.first.insertMany([{} , {} , {} , {} , {}])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68f77327fceb2931b4b28fd8'),
      '1': ObjectId('68f77327fceb2931b4b28fd9'),
      '2': ObjectId('68f77327fceb2931b4b28fda'),
      '3': ObjectId('68f77327fceb2931b4b28fdb'),
      '4': ObjectId('68f77327fceb2931b4b28fdc')
    }
  }
> db.first.find()
< {
    _id: ObjectId('68f77327fceb2931b4b28fd8')
  }
  {
    _id: ObjectId('68f77327fceb2931b4b28fd9')
  }
  {
    _id: ObjectId('68f77327fceb2931b4b28fda')
  }
  {
    _id: ObjectId('68f77327fceb2931b4b28fdb')
  }
  {
    _id: ObjectId('68f77327fceb2931b4b28fdc')
  }
```


Compass
My Queries
CONNECTIONS (1)
Search connecti
Local MongoDB
admin
config
local
myDB
first
second
test
test
test
myDB
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
My Qu...
>_MONGOSH
db.getCollection("first").remove({})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDele
{
acknowledged: true,
deletedCount: 5
}
db.getCollection("first").insertOne({"string" : "Hello World" , "boolean" : true , "number" : 10
{
acknowledged: true,
insertedId: ObjectId('68f77567fceb2931b4b28fdd')
}
myDB


Compass
My Queries
CONNECTIONS (1)
Search connecti
Local MongoDB
admin
config
local
myDB
first
second
test
test
test
myDB
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
My Qu...
>_MONGOSH
db.getCollection("first").remove({})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDele
{
acknowledged: true,
deletedCount: 5
}
: true , "number" : 10 , "numberInt" : NumberInt(100) , "numberLong" : NumberLong("2345562456")
{
acknowledged: true,
insertedId: ObjectId('68f77567fceb2931b4b28fdd')
}
myDB


Compass
My Queries
CONNECTIONS (1)
Search connecti
Local MongoDB
admin
config
local
myDB
first
second
test
test
test
myDB
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
My Qu...
>_MONGOSH
db.getCollection("first").remove({})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDele
{
acknowledged: true,
deletedCount: 5
}
erLong("2345562456") , "date" : new Date(), "object" : {"a":10 , "b":true} , "array" : [1,2,3]})
{
acknowledged: true,
insertedId: ObjectId('68f77567fceb2931b4b28fdd')
}
myDB


Compass
My Queries
CONNECTIONS (1)
Search connecti
Local ...
admin
config
local
myDB
first
second
test
test
test
myDB
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
mongosh: Lo...
My Qu...
>_MONGOSH
{
acknowledged: true,
insertedId: ObjectId('68f77567fceb2931b4b28fdd')
}
db.first.find()
{
_id: ObjectId('68f77567fceb2931b4b28fdd'),
string: 'Hello World',
boolean: true,
number: 10,
numberInt: 100,
numberLong: 2345562456,
date: 2025-10-21T11:58:31.528Z,
object: {
a: 10,
b: true
},
array: [
1,
2,
3
]
}
myDB

Compass

My Queries

CONNECTIONS (1)

Search connecti

Local MongoDB
- admin
- config
- local
- myDB
  - first
  - second
- test
  - test

test | myDB | mongos... | mongos... | mongos... | mongos... | first | mongos...

>_MONGOSH

```
test> db.first.insertOne({
        "_id": ObjectId('68f77567fceb2931b4b28fdd'),
        "string": 'Hello World',
        "boolean": true,
        "number": 10,
        "numberInt": 100,
        "numberLong": 2345562456,
        "date": 2025-10-21T11:58:31.528Z,
        "object": {
          "a": 10,
          "b": true
        },
        "array": [
          1,
          2,
          3
        ]
      })
```

Compass

My Queries

CONNECTIONS (1)

Search connecti

Local MongoDB
- admin
- config
- local
- myDB
  - first
  - second
- test
  - test

test | myDB | mongos... | mongos... | mongos... | mongos... | first | mongos...

>_MONGOSH

```
      2,
      3
    ]
  }
> db.first.insertOne({
    "_id": ObjectId('68f77567fceb2931b4b28fdd'),
    "string": 'Hello World',
    "boolean": true,
    "number": 10,
    "numberInt": 100,
    "numberLong": 2345562456,
    "date": new Date(),
    "object": {
      "a": 10,
      "b": true
    },
    "array": [
      1,
      2,
      3
    ]
  })
MongoServerError: E11000 duplicate key error collection: test.first index: _id_ dup key: { _id:
test>
```

Compass

My Queries

CONNECTIONS (1)

Search connecti

Local MongoDB
- admin
- config
- local
- myDB
  - first
  - second
- test
  - test

test | myDB | mongosh: L... | mongosh: L... | mongosh: L... | mongosh: L... | first | mongosh: L...

>_MONGOSH

```
> db.getCollection("first").insertOne({"string" : "Hello World" , "boolean" : true , "number" : 10 , "numberInt
< {
    acknowledged: true,
    insertedId: ObjectId('68f77b013a36c1f96101fa74')
  }
> db.first.find()
< {
    _id: ObjectId('68f77b013a36c1f96101fa74'),
    string: 'Hello World',
    boolean: true,
    number: 10,
    numberInt: 100,
    numberLong: 2345562456,
    date: 2025-10-21T12:22:25.422Z,
    object: {
      a: 10,
      b: true
    },
    array: [
      1,
      2,
      3
    ]
  }
> db.getCollection("first").insertOne({"_id": ObjectId('68f77b013a36c1f96101fa74'),"string" : "Hello World" , "
MongoServerError: E11000 duplicate key error collection: test.first index: _id_ dup key: { _id: ObjectId('6
```

Compass

My Queries

CONNECTIONS (1)

Search connecti

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongosh: L... | mongosh: L... | mongosh: L... | mongosh: L... | first | mongosh: L...

>_MONGOSH

```
    insertedId: ObjectId('68f77b013a36c1f96101fa74')
  }
> db.first.find()
< {
    _id: ObjectId('68f77b013a36c1f96101fa74'),
    string: 'Hello World',
    boolean: true,
    number: 10,
    numberInt: 100,
    numberLong: 2345562456,
    date: 2025-10-21T12:22:25.422Z,
    object: {
      a: 10,
      b: true
    },
    array: [
      1,
      2,
      3
    ]
  }
> db.getCollection("first").insertOne({"_id": ObjectId('68f77b013a36c1f96101fa74'),"string" : "Hello World" , "
✖ MongoServerError: E11000 duplicate key error collection: test.first index: _id_ dup key: { _id: ObjectId('6
> db.getCollection("first").insertOne({"_id": ObjectId('68f77b013a36c1f96101fa74')})
✖ MongoServerError: E11000 duplicate key error collection: test.first index: _id_ dup key: { _id: ObjectId('6
test>
```

Compass

My Queries

CONNECTIONS (1)

Search connecti

- Local MongoDB
  - admin
  - config
  - local
  - myDB
    - first
    - second
  - test
    - test

test | myDB | mongos... | mongos... | mongos... | mongos... | first | mongos... | mongos...

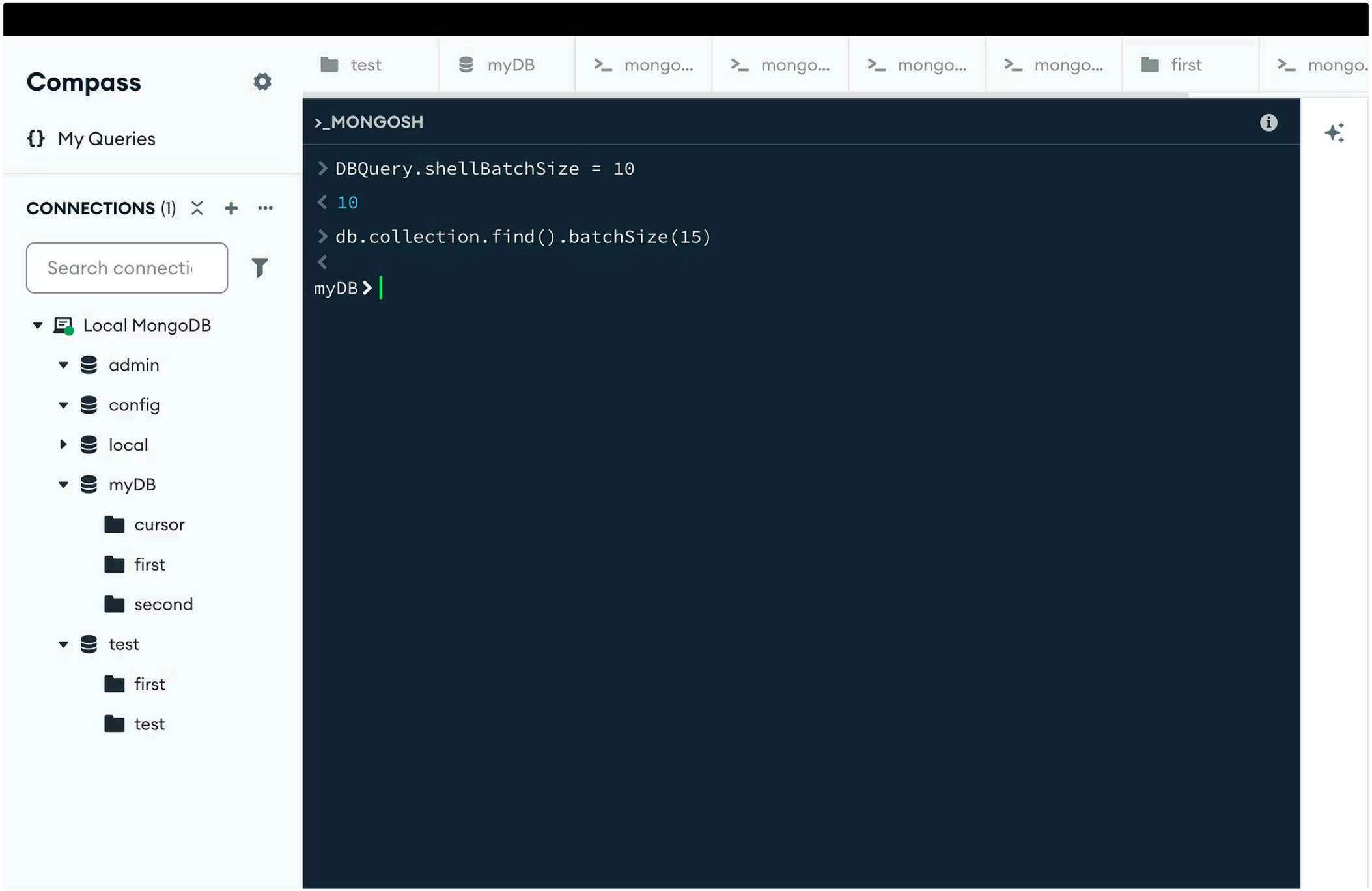
>_MONGOSH

```
      3
    ]
  })
✖ MongoServerError: E11000 duplicate key error collection: myDB.first index: _id_ dup key: { _id: ObjectId('6
> db.second.insertOne({
    "_id": ObjectId('68f77567fceb2931b4b28fdd'),
    "string": 'Hello World',
    "boolean": true,
    "number": 10,
    "numberInt": 100,
    "numberLong": 2345562456,
    "date": new Date(),
    "object": {
      "a": 10,
      "b": true
    },
    "array": [
      1,
      2,
      3
    ]
  })
< {
    acknowledged: true,
    insertedId: ObjectId('68f77567fceb2931b4b28fdd')
  }
myDB>
```

```
>_MONGOSH
> var docs = [];
> for (let i = 1; i <= 100; i++)
      {
          obj = {
              index : NumberInt(i)
              };
          docs.push(obj)
      }
< 100
> db.cursor.insertMany(docs)
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68f85f07093b5b90d59f48ed'),
      '1': ObjectId('68f85f07093b5b90d59f48ee'),
      '2': ObjectId('68f85f07093b5b90d59f48ef'),
      '3': ObjectId('68f85f07093b5b90d59f48f0'),
      '4': ObjectId('68f85f07093b5b90d59f48f1'),
      '5': ObjectId('68f85f07093b5b90d59f48f2'),
```

```
>_MONGOSH
> db.getCollection('cursor').find({})
< {
    _id: ObjectId('68f85f07093b5b90d59f48ed'),
    index: 1
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f48ee'),
    index: 2
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f48ef'),
    index: 3
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f48f0'),
    index: 4
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f48f1'),
```

```
>_MONGOSH
    index: 18
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f48ff'),
    index: 19
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f4900'),
    index: 20
  }
  Type "it" for more
> it
< {
    _id: ObjectId('68f85f07093b5b90d59f4901'),
    index: 21
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f4902'),
    index: 22
```


Compass
My Queries
CONNECTIONS (1)
Search connecti
Local MongoDB
admin
config
local
myDB
cursor
first
second
test
first
test
test
myDB
mongo...
mongo...
mongo...
mongo...
first
mongo.
>_MONGOSH
DBQuery.shellBatchSize = 10
10
db.collection.find().batchSize(15)
myDB

test | myDB | mongo... | mongo... | mongo... | mongo... | first

>_MONGOSH

```
> var cursor = db.getCollection('cursor').find({});
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48ed'), index: 1 }
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48ee'), index: 2 }
myDB >
```

test | myDB | mongo... | mongo... | mongo... | mongo... | first

>_MONGOSH

```
> var cursor = db.getCollection('cursor').find({});
> cursor.objsLeftInBatch()
< 0
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48ed'), index: 1 }
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48ee'), index: 2 }
> cursor.hasNext()
< true
> cursor.objsLeftInBatch()
< 98
myDB >
```

test | myDB | mongo... | mongo... | mongo... | mongo... | first

>_MONGOSH

```
> cursor.objsLeftInBatch()
< 98
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48ef'), index: 3 }
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48f0'), index: 4 }
> cursor.next()
< { _id: ObjectId('68f85f07093b5b90d59f48f1'), index: 5 }
> cursor
< {
    _id: ObjectId('68f85f07093b5b90d59f48f2'),
    index: 6
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f48f3'),
    index: 7
  }
```

test | myDB | mongo... | mongo... | mongo... | mongo... | first

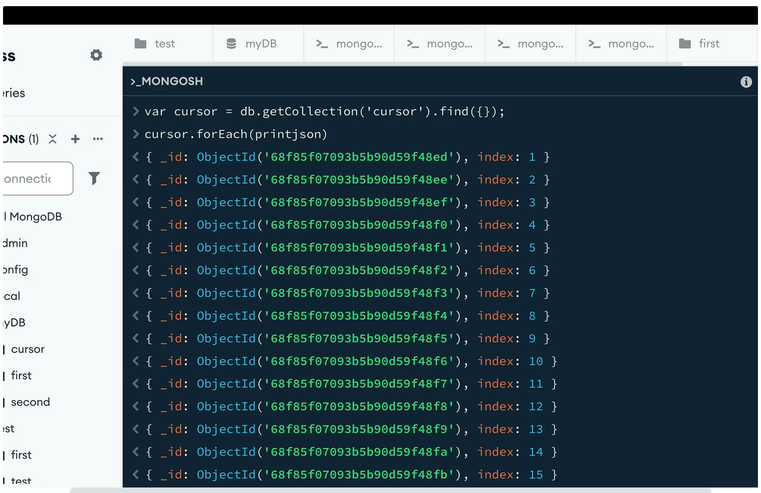
>_MONGOSH

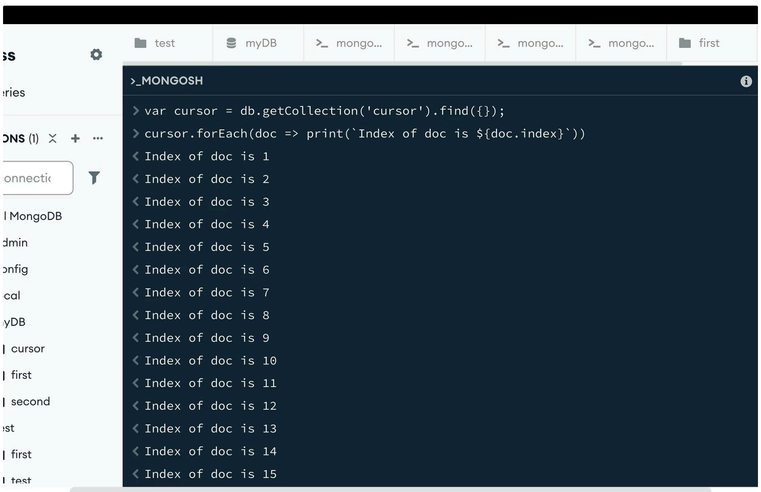
```
    index: 98
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f494f'),
    index: 99
  }
  {
    _id: ObjectId('68f85f07093b5b90d59f4950'),
    index: 100
  }
> cursor.objsLeftInBatch()
< 0
> cursor.hasNext()
< false
> cursor.next()
< null
myDB >
```

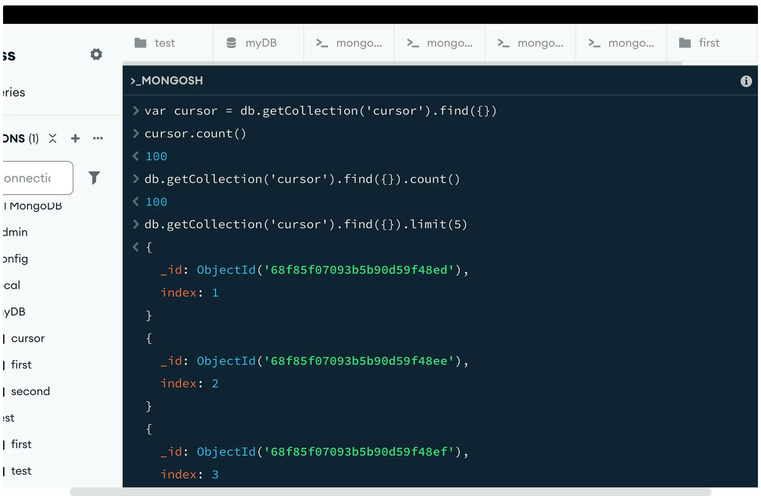
test | myDB | mongo... | mongo... | mongo... | mongo... | first

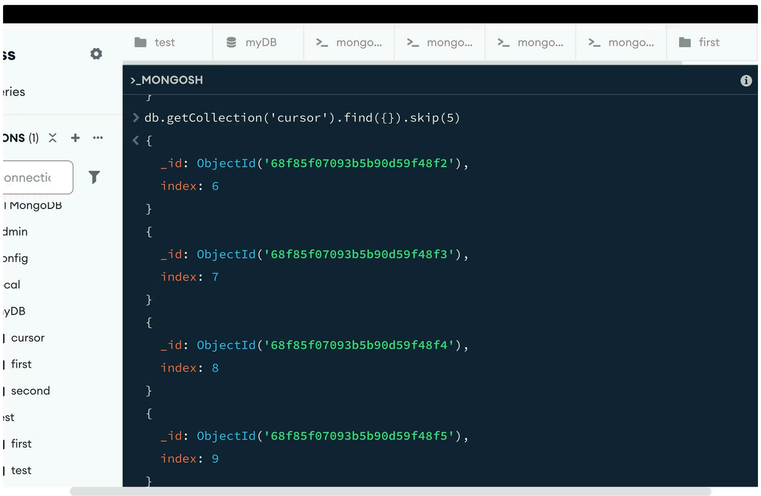
>_MONGOSH

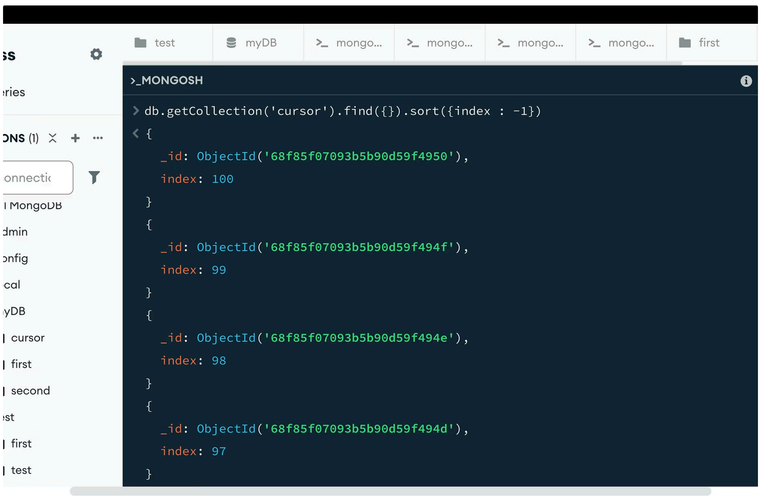
```
> var cursor = db.getCollection('cursor').find({});
> cursor.toArray()
< [
    { _id: ObjectId('68f85f07093b5b90d59f48ed'), index: 1 },
    { _id: ObjectId('68f85f07093b5b90d59f48ee'), index: 2 },
    { _id: ObjectId('68f85f07093b5b90d59f48ef'), index: 3 },
    { _id: ObjectId('68f85f07093b5b90d59f48f0'), index: 4 },
    { _id: ObjectId('68f85f07093b5b90d59f48f1'), index: 5 },
    { _id: ObjectId('68f85f07093b5b90d59f48f2'), index: 6 },
    { _id: ObjectId('68f85f07093b5b90d59f48f3'), index: 7 },
    { _id: ObjectId('68f85f07093b5b90d59f48f4'), index: 8 },
    { _id: ObjectId('68f85f07093b5b90d59f48f5'), index: 9 },
    { _id: ObjectId('68f85f07093b5b90d59f48f6'), index: 10 },
    { _id: ObjectId('68f85f07093b5b90d59f48f7'), index: 11 },
    { _id: ObjectId('68f85f07093b5b90d59f48f8'), index: 12 },
    { _id: ObjectId('68f85f07093b5b90d59f48f9'), index: 13 },
    { _id: ObjectId('68f85f07093b5b90d59f48fa'), index: 14 },
```

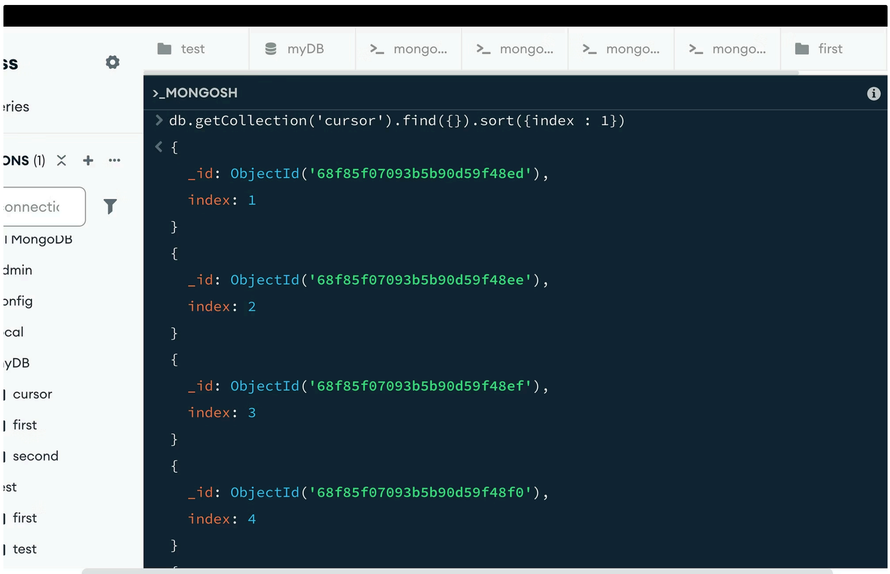

test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> var cursor = db.getCollection('cursor').find({});
> cursor.forEach(printjson)
< { _id: ObjectId('68f85f07093b5b90d59f48ed'), index: 1 }
< { _id: ObjectId('68f85f07093b5b90d59f48ee'), index: 2 }
< { _id: ObjectId('68f85f07093b5b90d59f48ef'), index: 3 }
< { _id: ObjectId('68f85f07093b5b90d59f48f0'), index: 4 }
< { _id: ObjectId('68f85f07093b5b90d59f48f1'), index: 5 }
< { _id: ObjectId('68f85f07093b5b90d59f48f2'), index: 6 }
< { _id: ObjectId('68f85f07093b5b90d59f48f3'), index: 7 }
< { _id: ObjectId('68f85f07093b5b90d59f48f4'), index: 8 }
< { _id: ObjectId('68f85f07093b5b90d59f48f5'), index: 9 }
< { _id: ObjectId('68f85f07093b5b90d59f48f6'), index: 10 }
< { _id: ObjectId('68f85f07093b5b90d59f48f7'), index: 11 }
< { _id: ObjectId('68f85f07093b5b90d59f48f8'), index: 12 }
< { _id: ObjectId('68f85f07093b5b90d59f48f9'), index: 13 }
< { _id: ObjectId('68f85f07093b5b90d59f48fa'), index: 14 }
< { _id: ObjectId('68f85f07093b5b90d59f48fb'), index: 15 }

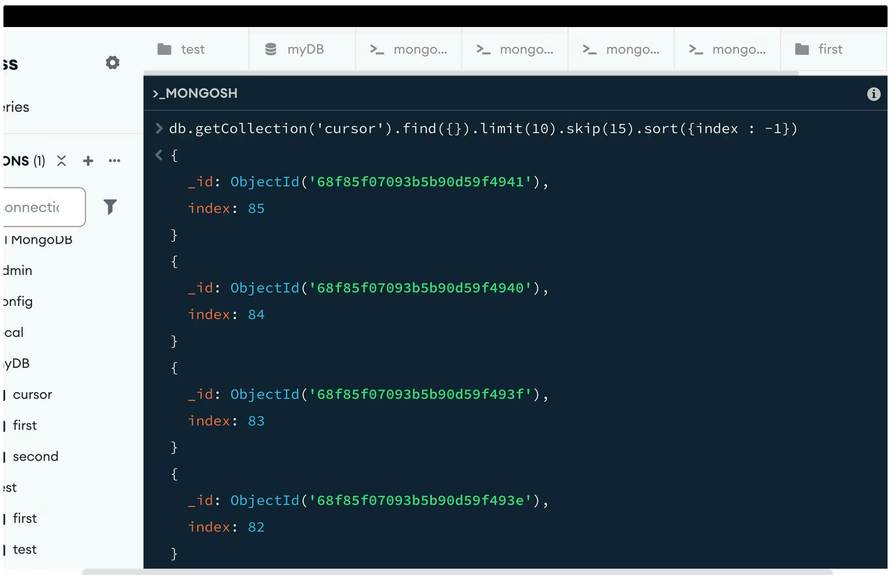

test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> var cursor = db.getCollection('cursor').find({});
> cursor.forEach(doc => print(`Index of doc is ${doc.index}`))
< Index of doc is 1
< Index of doc is 2
< Index of doc is 3
< Index of doc is 4
< Index of doc is 5
< Index of doc is 6
< Index of doc is 7
< Index of doc is 8
< Index of doc is 9
< Index of doc is 10
< Index of doc is 11
< Index of doc is 12
< Index of doc is 13
< Index of doc is 14
< Index of doc is 15

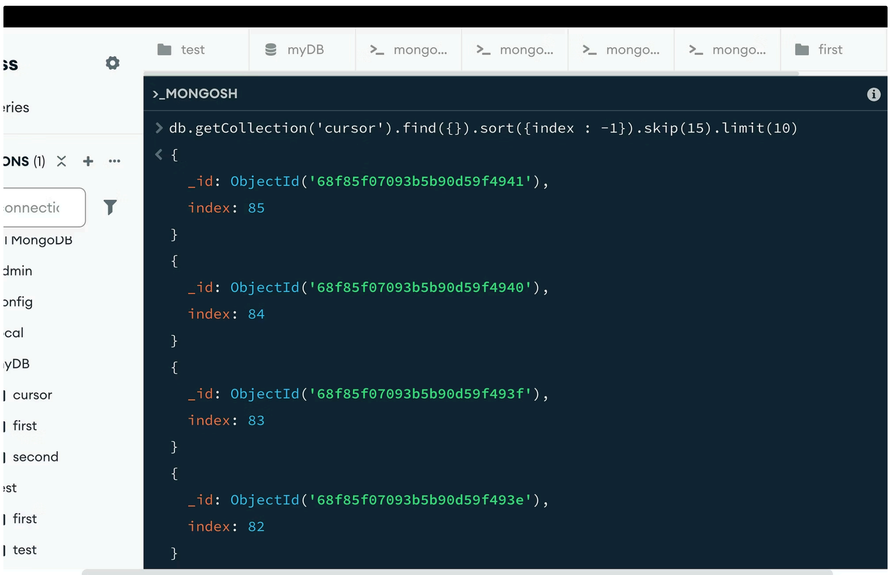

test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> var cursor = db.getCollection('cursor').find({})
> cursor.count()
< 100
> db.getCollection('cursor').find({}).count()
< 100
> db.getCollection('cursor').find({}).limit(5)
< {
_id: ObjectId('68f85f07093b5b90d59f48ed'),
index: 1
}
{
_id: ObjectId('68f85f07093b5b90d59f48ee'),
index: 2
}
{
_id: ObjectId('68f85f07093b5b90d59f48ef'),
index: 3

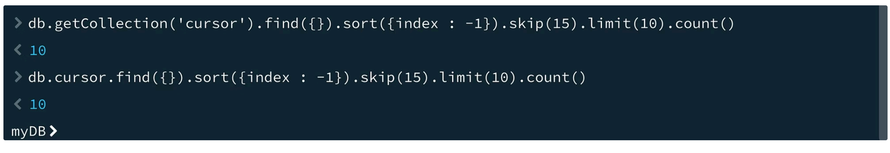

test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> db.getCollection('cursor').find({}).skip(5)
< {
_id: ObjectId('68f85f07093b5b90d59f48f2'),
index: 6
}
{
_id: ObjectId('68f85f07093b5b90d59f48f3'),
index: 7
}
{
_id: ObjectId('68f85f07093b5b90d59f48f4'),
index: 8
}
{
_id: ObjectId('68f85f07093b5b90d59f48f5'),
index: 9
}

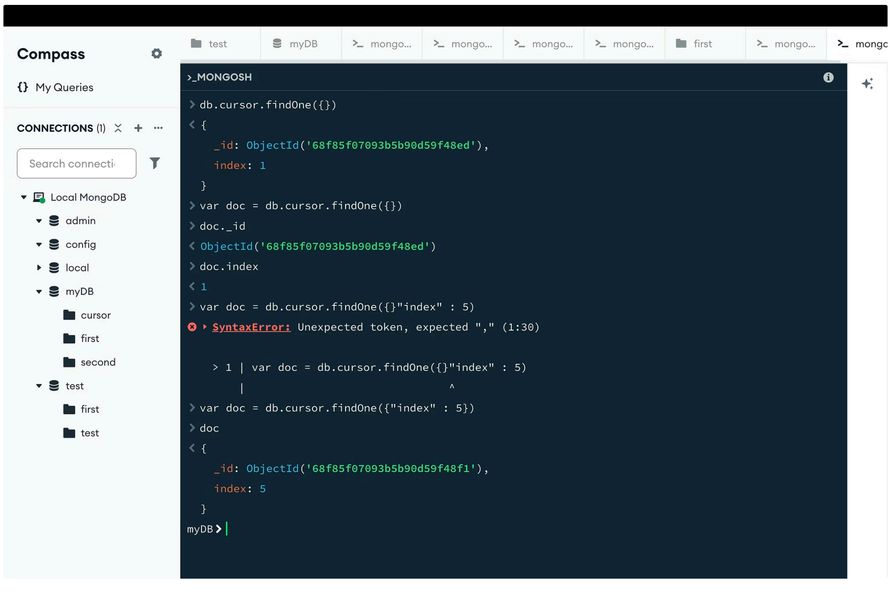

test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> db.getCollection('cursor').find({}).sort({index : -1})
< {
_id: ObjectId('68f85f07093b5b90d59f4950'),
index: 100
}
{
_id: ObjectId('68f85f07093b5b90d59f494f'),
index: 99
}
{
_id: ObjectId('68f85f07093b5b90d59f494e'),
index: 98
}
{
_id: ObjectId('68f85f07093b5b90d59f494d'),
index: 97
}


test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> db.getCollection('cursor').find({}).sort({index : 1})
{
_id: ObjectId('68f85f07093b5b90d59f48ed'),
index: 1
}
{
_id: ObjectId('68f85f07093b5b90d59f48ee'),
index: 2
}
{
_id: ObjectId('68f85f07093b5b90d59f48ef'),
index: 3
}
{
_id: ObjectId('68f85f07093b5b90d59f48f0'),
index: 4
}
cursor
first
second
first
test


test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> db.getCollection('cursor').find({}).limit(10).skip(15).sort({index : -1})
{
_id: ObjectId('68f85f07093b5b90d59f4941'),
index: 85
}
{
_id: ObjectId('68f85f07093b5b90d59f4940'),
index: 84
}
{
_id: ObjectId('68f85f07093b5b90d59f493f'),
index: 83
}
{
_id: ObjectId('68f85f07093b5b90d59f493e'),
index: 82
}
cursor
first
second
first
test


test
myDB
mongo...
mongo...
mongo...
mongo...
first
>_MONGOSH
> db.getCollection('cursor').find({}).sort({index : -1}).skip(15).limit(10)
{
_id: ObjectId('68f85f07093b5b90d59f4941'),
index: 85
}
{
_id: ObjectId('68f85f07093b5b90d59f4940'),
index: 84
}
{
_id: ObjectId('68f85f07093b5b90d59f493f'),
index: 83
}
{
_id: ObjectId('68f85f07093b5b90d59f493e'),
index: 82
}
cursor
first
second
first
test


> db.getCollection('cursor').find({}).sort({index : -1}).skip(15).limit(10).count()
10
> db.cursor.find({}).sort({index : -1}).skip(15).limit(10).count()
10
myDB


Compass
My Queries
CONNECTIONS (1)
Search connecti
Local MongoDB
admin
config
local
myDB
cursor
first
second
test
first
test
test
myDB
mongo...
mongo...
mongo...
mongo...
first
mongo...
mongc
>_MONGOSH
> db.cursor.findOne({})
{
_id: ObjectId('68f85f07093b5b90d59f48ed'),
index: 1
}
> var doc = db.cursor.findOne({})
> doc._id
ObjectId('68f85f07093b5b90d59f48ed')
> doc.index
1
> var doc = db.cursor.findOne({}"index" : 5)
SyntaxError: Unexpected token, expected "," (1:30)
> 1 | var doc = db.cursor.findOne({}"index" : 5)
| ^
> var doc = db.cursor.findOne({"index" : 5})
> doc
{
_id: ObjectId('68f85f07093b5b90d59f48f1'),
index: 5
}
myDB

insert() Method – DEPRECATED

> ⚠️ **DEPRECATED METHOD – For Reference Only**
>
> This method should not be used in new code. It is included here for reference when maintaining legacy applications.

Old Syntax (DO NOT USE)

```
db.collection_name.insert(
  <document or array>
)
```

What it Returned

For single document:

```
WriteResult({
  "nInserted": 1
})
```

For multiple documents:

```
BulkWriteResult({
  "writeErrors": [],
  "writeConcernErrors": [],
  "nInserted": 3,
  "nUpserted": 0,
  "nMatched": 0,
  "nModified": 0,
  "nRemoved": 0,
  "upserted": []
})
```

Migration Guide

```
//
```

bulkWrite() Method – Advanced Operations

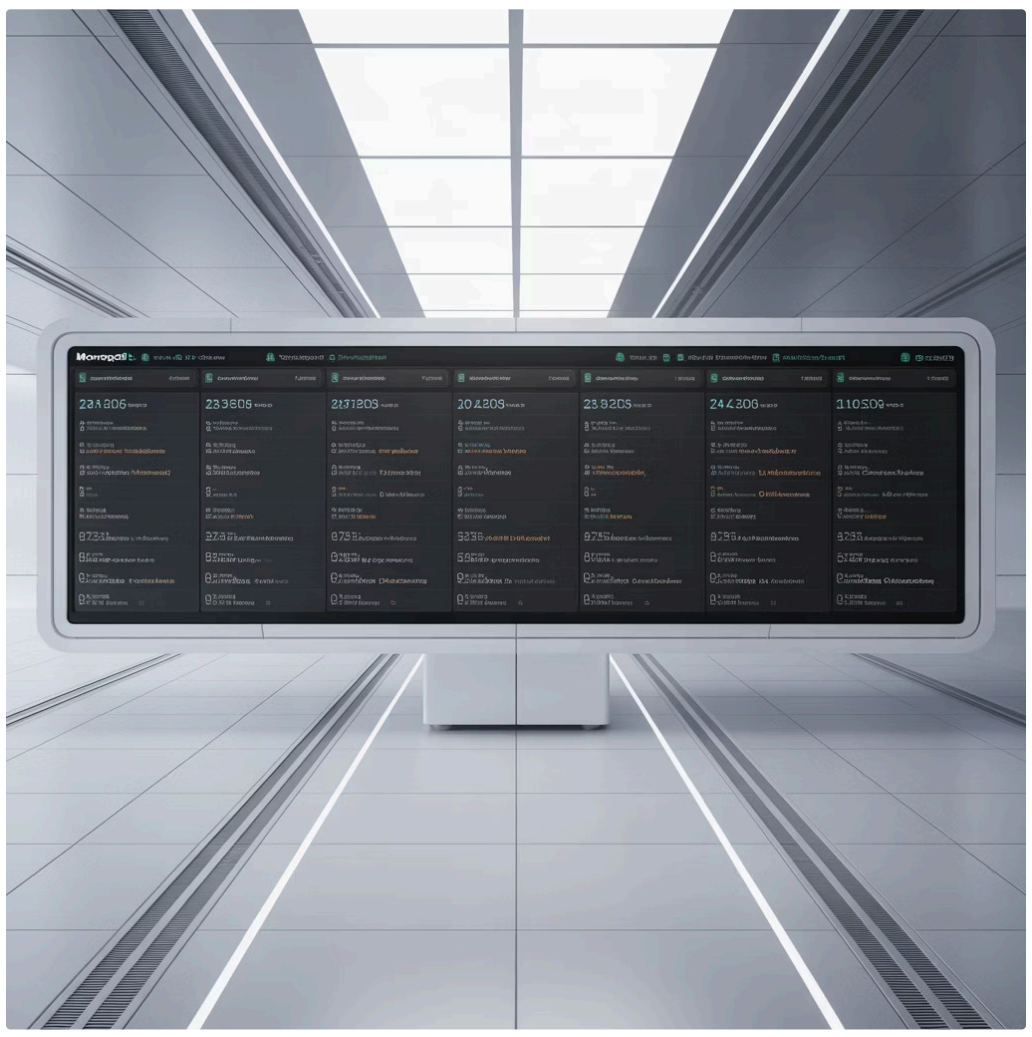
The bulkWrite() method allows you to perform multiple operations (insert, update, delete) in a single command, significantly improving performance and reducing network overhead.

Syntax Structure

```
db.collection_name.bulkWrite([
 {
  insertOne: {
   document: <doc>
  }
 },
 {
  updateOne: {
   filter: <query>,
   update: <update>
  }
 },
 {
  deleteOne: {
   filter: <query>
  }
 },
 {
  replaceOne: {
   filter: <query>,
   replacement: <doc>
  }
 }
], {
 ordered: <boolean>,
 writeConcern: <document>
})
```

Practical Example

```
db.students.bulkWrite([
 {
  insertOne: {
   document: {
    name: "Eve",
    age: 20
   }
  }
 },
 {
  updateOne: {
   filter: { name: "Alice" },
   update: {
    $set: { age: 23 }
   }
  }
 },
 {
  deleteOne: {
   filter: { name: "Bob" }
  }
 }
])
```



56% Faster Performance

MongoDB 8.0 delivers dramatic performance improvements for bulk operations

Reduced Network Trips

Combining operations reduces round trips to the server significantly

Mixed Operations

Combine different operation types in a single efficient command

WriteConcern Explained

WriteConcern describes the level of acknowledgement requested from MongoDB for write operations. It's crucial for balancing performance with data durability and consistency across your deployment.

Syntax Structure

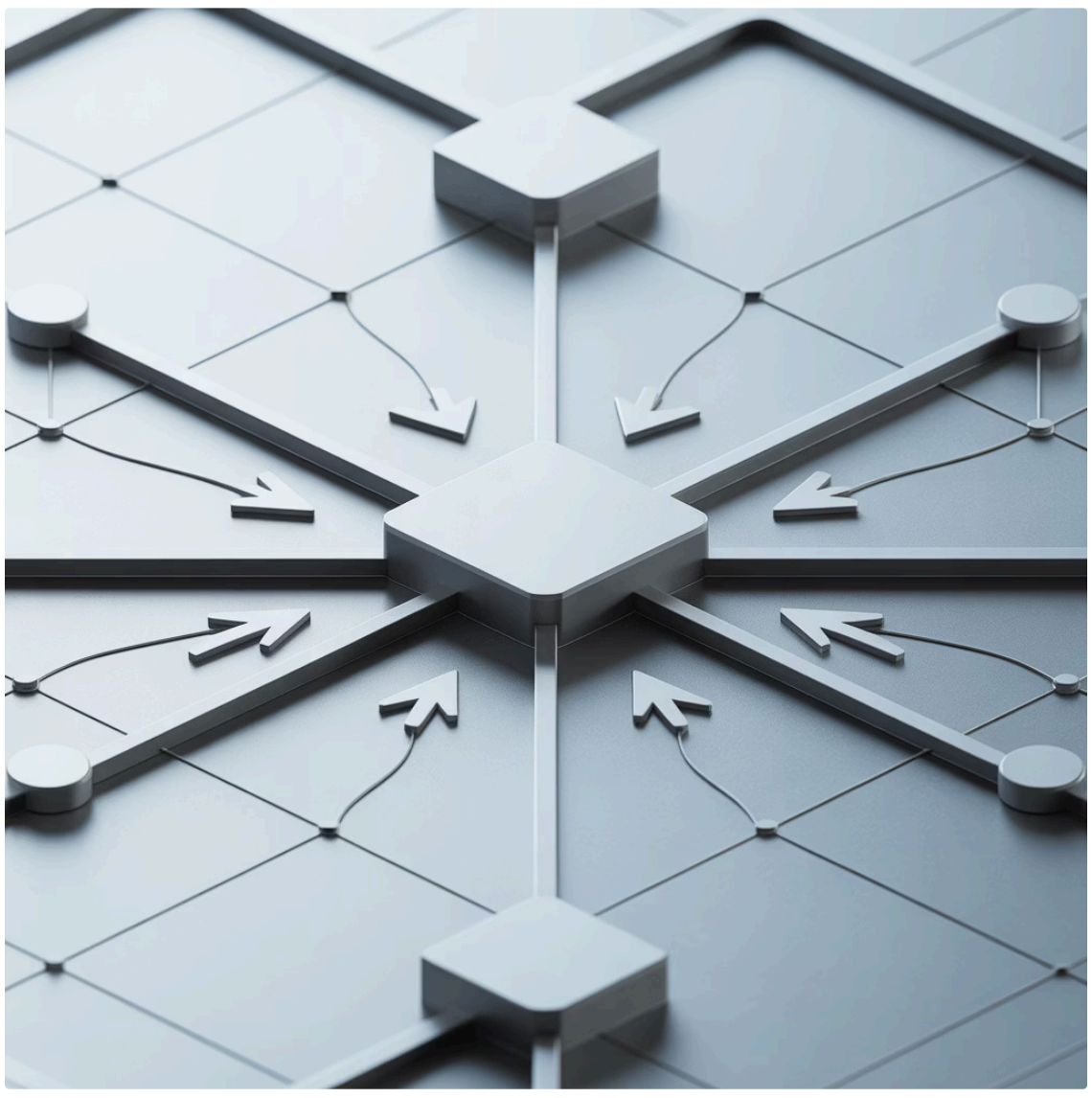
```
{
  w: <value>,
  j: <boolean>,
  wtimeout: <number>
}
```

Parameters Detailed

Parameter	Description
w	Number of nodes to acknowledge (0, 1, "majority", <number>)
j	Request journal acknowledgement (true or false)
wtimeout	Timeout in milliseconds (e.g., 5000)

Common Configurations

```
// Default - Primary acknowledgement only
{ w: 1 }

// Majority (recommended for critical)
{ w: "majority", j: true }

// Fire-and-forget (fastest, least safe)
{ w: 0 }

// Custom - 2 nodes with journal + timeout
{ w: 2, j: true, wtimeout: 5000 }
```



Best Practices

- Use `w: "majority"` for critical data to ensure durability across replica sets
- Avoid `w: 0` for important operations as it provides no acknowledgement
- Enable journaling (`j: true`) for added durability and crash recovery
- Set appropriate `wtimeout` to avoid indefinite waits in case of network issues

MongoDB 5.0+ Default: { w: "majority" }

READ Operations Overview

Available Read Methods

MongoDB provides several methods for querying and retrieving documents. Understanding the current methods versus deprecated ones ensures you write code that follows best practices and performs optimally.

Method	Purpose	Return Type	Status
find()	Retrieve multiple documents	Cursor	✅ Current
findOne()	Retrieve single document	Document (Extended JSON)	✅ Current
countDocuments()	Count matching documents	Number	✅ Current
estimatedDocumentCount()	Estimate total documents	Number	✅ Current
count()	Count documents	Number	⚠️ DEPRECATED

⚠️ **DEPRECATED:** `cursor.count()` and `collection.count()`

✅ **USE INSTEAD:**

- `countDocuments(filter)` for filtered counts with accurate results
- `estimatedDocumentCount()` for total collection count estimates

Why Deprecated? The old count() methods could return inaccurate results in sharded clusters during chunk migrations and couldn't work properly within transactions.

find() Method

Syntax and Parameters

```
db.collection_name.find(
  <filter>,
  <projection>
)
```

Parameters Explained

- **filter (optional):** Query criteria to match documents (empty `{}` returns all documents)
- **projection (optional):** Specifies which fields to include or exclude in returned documents

> ⚠️ **Important:** Use `projection`, not `fields` (deprecated parameter name)

Return Value

Returns a **Cursor object**, not the documents directly. The cursor must be iterated to access documents.

Comprehensive Examples

```
// Find all documents
db.students.find()

// Find with filter criteria
db.students.find({ grade: "A" })

// Find with projection (include fields)
db.students.find(
  { grade: "A" },
  { name: 1, age: 1, _id: 0 }
)

// Find with projection (exclude fields)
db.students.find(
  {},
  { password: 0, secretKey: 0 }
)
```

Projection Rules

Inclusion vs Exclusion

Use `1` to include specific fields, `0` to exclude fields from results

Cannot Mix Both

Cannot mix inclusion and exclusion in the same projection (except for `_id`)

Default _id Behaviour

The `_id` field is included by default unless explicitly excluded with `_id: 0`

Understanding Cursors

A cursor is a server-side pointer to query results. Instead of returning all documents at once (which could overwhelm memory), MongoDB returns results in manageable batches, making it efficient to work with large datasets.

Why Cursors Matter

Efficient Memory Usage

Process large result sets without loading everything into memory at once, preventing system overload

Reduced Network Traffic

Data is transferred in optimised batches rather than all at once, improving performance

Stream Processing

Process documents as they arrive, enabling real-time data handling for large queries

Cursor Lifecycle

Query Execution

`find()` creates cursor on server

First Batch

Default 101 docs or 1MB sent to client

Client Iteration

Client processes documents in batch

Automatic getMore

When batch exhausted, request next batch

Subsequent Batches

Up to 16MB per batch delivered

Cursor Batch Sizes

Default Batch Sizes

Batch Type	Default Size	Maximum
First batch (find/aggregate)	101 documents or 1 MB	16 MB
Subsequent batches (getMore)	No default limit	16 MB per batch
MongoDB Shell iterator	20 documents	-

Setting Custom Batch Size

```
// Method 1: Assign to variable
var cursor = db.students
  .find()
  .batchSize(50)

// Method 2: Inline chaining
db.students
  .find()
  .batchSize(100)
```

Performance Considerations

Batch Size of 1

❌ Very inefficient – Creates excessive network round trips

Larger Batches

✅ Fewer round trips – Better for throughput-intensive operations

Smaller Batches

✅ Less memory – Better for memory-constrained environments

Practical Example

```
// Efficient large collection processing
var cursor = db.largeCollection
  .find({ status: "active" })
  .batchSize(500)

while (cursor.hasNext()) {
  var doc = cursor.next()
  // Process document
}
```

🚀 **MongoDB 8.0:** Batch processing is significantly optimised for better performance!

Cursor Iteration Methods

Method 1: Manual Iteration with Variables

Assign the cursor to a variable for fine-grained control over iteration. This approach gives you maximum flexibility when processing documents.

```
var cursor = db.students.find({ grade: "A" })
```

Basic Iteration Methods

Method	Description	Returns
next()	Get next document from cursor	Single document
hasNext()	Check if more documents exist	Boolean
objsLeftInBatch()	Documents remaining in current batch	Number

Practical Example

```
var cursor = db.students.find()

// Manual iteration with control
while (cursor.hasNext()) {
  var doc = cursor.next()
  print(doc.name)
  print("Docs left in batch: " +
    cursor.objsLeftInBatch())
}
```

Best Practice: Use manual iteration when you need fine-grained control over processing, error handling, or batch monitoring during document processing.

Advanced Cursor Methods

toArray() Method

Convert entire cursor to an array in memory

```
var allDocs = db.students
  .find({ grade: "A" })
  .toArray()

print(allDocs.length)
```

⚠️ Warning: Loads all documents into memory at once – use cautiously with large result sets to avoid memory exhaustion!

forEach() Method

Iterate with callback function for processing

```
// Traditional function syntax
db.students
  .find({ grade: "A" })
  .forEach(function(doc) {
    print("Student: " + doc.name +
      ", Age: " + doc.age)
  })

// ES6 arrow function syntax
db.students
  .find({ grade: "A" })
  .forEach(doc => {
    print(`${doc.name}: ${doc.grade}`)
  })
```

Choosing the Right Method

next() / hasNext()

Use for fine-grained control and processing documents one at a time with custom logic

toArray()

Use when you need all results in memory at once (only for small result sets)

forEach()

Use for simple iteration with consistent processing logic for each document

Cursor Helper Methods

1

countDocuments() – Count Matching Docs

```
//
```

Method Execution Order

Understanding how MongoDB processes cursor methods is crucial for writing efficient queries and avoiding unexpected results.

Critical Concept: Regardless of how you write the query chain, MongoDB **ALWAYS** executes methods in this fixed order:

1 2 3

1. SORT

Organises documents by specified fields

2. SKIP

Skips specified number of documents

3. LIMIT

Returns only specified number of documents

Examples – All Execute Identically

```
// Example 1
db.students.find()
  .limit(10)
  .skip(20)
  .sort({ name: 1 })

// Example 2
db.students.find()
  .sort({ name: 1 })
  .skip(20)
  .limit(10)

// Example 3
db.students.find()
  .skip(20)
  .limit(10)
  .sort({ name: 1 })

// All execute as: SORT → SKIP → LIMIT
```

Visual Execution Flow

All Documents

Start with 1,000 documents in collection

After SORT

1,000 documents sorted by name alphabetically

After SKIP

980 documents remain (skipped first 20)

After LIMIT

10 documents returned as final result

Performance Implications

Understanding execution order helps optimise queries for large datasets and index usage

Pagination Logic

Essential for implementing correct page calculations in applications

Query Optimisation

Enables better index design and query planning for improved performance

findOne() Method

Syntax and Parameters

```
db.collection_name.findOne(
  <filter>,
  <projection>
)
```

Key Differences from find()

Feature	find()	findOne()
Returns	Cursor	Single document (Extended JSON)
Result count	Multiple	Maximum 1
Iteration needed	Yes	No
Use case	Multiple results	Single specific document

Comprehensive Examples

```
// Find first document
var doc = db.students.findOne()

// Find specific document
var student = db.students.findOne({
  name: "Alice Johnson"
})

// With projection
var student = db.students.findOne(
  { name: "Alice Johnson" },
  { name: 1, grade: 1, _id: 0 }
)

// Handle not found case
var result = db.students.findOne({
  name: "NonExistent"
})

if (result === null) {
  print("Student not found")
}
```

Return Format (Extended JSON)

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "name": "Alice Johnson",
  "age": 22,
  "grade": "A"
}
```

When to Use findOne()

Retrieving documents by unique identifier (`_id`) for quick lookups

Existence Checks

Checking if a specific document exists (returns `null` if not found)

Configuration Documents

Getting single configuration or settings documents from collections

Extended JSON Format Explained

Extended JSON (EJSON) is a JSON-compatible format that preserves BSON type information. It bridges the gap between MongoDB's binary format and human-readable JSON representation.

Why Extended JSON is Needed

JSON Limitations

Standard JSON supports only basic types: string, number, boolean, array, object, and null

BSON Richness

BSON supports additional types like Date, ObjectId, Binary, Decimal128, and more

The Bridge

Extended JSON preserves BSON type information whilst remaining JSON-compatible

Common BSON Types in Extended JSON

BSON Type	Strict Mode	Shell Mode
ObjectId	{"$oid": "..."}	ObjectId("...")
Date	{"$date": "..."}	ISODate("...")
Decimal128	{"$numberDecimal": "..."}	NumberDecimal("...")
Binary	{"$binary": {...}}	BinData(...)
Long	{"$numberLong": "..."}	NumberLong("...")

Format Types

1. Canonical Mode (Strict)

```
{
  "_id": {
    "$oid": "507f1f77bcf86cd799439011"
  },
  "createdAt": {
    "$date": "2024-10-22T10:30:00.000Z"
  },
  "price": {
    "$numberDecimal": "99.99"
  }
}
```

2. Relaxed Mode (Shell)

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "createdAt": ISODate("2024-10-22T10:30:00Z"),
  "price": NumberDecimal("99.99")
}
```

REST API Responses

Extended JSON ensures type information is preserved in HTTP responses

Data Export

mongoexport uses Extended JSON to maintain type accuracy

Tool Displays

MongoDB Compass and drivers use Extended JSON for data visualisation