



# MongoDB | Shell & CRUD operations

SELF GUIDED

## Learning goals

After this lesson, you will be able to:

- Use the MongoShell to **C**reate, **R**ead, **U**psert and **D**estroy documents

## Intro



When working with any database, there are a few operations that we will always need to use. These operations are so popular that we have an acronym for it: **CRUD** stands for create, read, update and delete:



These are the basic operations to persist data in our database, and with them, we can do anything we want. Ready?

## Quick Recap

### Database commands

We can launch the MongoShell by simply typing `mongo` in our terminal. Remember that each mongo server can hold many **databases**.

Database Command	Explanation
<code>show dbs</code>	List all the databases inside our mongo server
<code>use &lt;dbName&gt;</code>	This will switch to the database <code>dbName</code> or create it if it doesn't exist
<code>db</code>	Show the name of the current database

### Collection commands

A Database is composed of different **collections**. This lesson will be focused on how to create, read, update, and delete documents inside a *collection*. But first of all, let's take a look at a couple of useful collection commands:

Collection Command	Explanation
<code>show collections</code>	List all the collections inside the current database

### JSON Documents

Each collection is composed of different **documents**. A document is nothing more than a JSON object, with different **keys** and its associated values. We can identify a document by its `_id` property that mongo generates automatically for us.

#### Example:

```
1  {  
2    "_id" : ObjectId("58401d8b71caea72370262c1"),  
3    "name" : "John Doe",  
4    "age" : 25,
```

[Copy](#)

```
5     "address": {  
6         "country": "Japan",  
7         "address": "#901, 74-1 Yamashita-cho, Naka-ku"  
8     }  
9 }
```

## Create | Inserting Documents

The `create` commands enable us to insert new documents into a collection. If the collection doesn't exist, the operation will create it.

Insert Command	Summary
<code>db.collection.insertOne(doc)</code>	Adds a document to the collection
<code>db.collection.insertMany([docs])</code>	Adds one or more documents to the collection

### insertOne()

The `db.collection.insertOne` command inserts a single document into the collection.

#### Example 1:

```
1  > db.enemies.insertOne({  
2      "name": "Blinky",  
3      "color": "Red"  
4  })  
5  
6  // The command returns an object with the following structure  
7  //  
8  // {  
9  //   "acknowledged" : true,  
10 //   "insertedId" : ObjectId("583c63f96fa7d619e2c574e6")  
11 // }
```

[Copy](#)

- Since we don't have a collection called `enemies`, it will be created automatically.
- As you can see, the document didn't have an `id`, so MongoDB created one for us.

## Example 2:

```
1  > db.enemies.insertOne({
2    "_id": 10,
3    "name": "Pinky",
4    "color": "Pink",
5    "status": "alive"
6  })
7
8  // The command returns an object with the following structure
9  //
10 // {
11 //   "acknowledged" : true,
12 //   "insertedId" : 10
13 // }
```

[Copy](#)

Notice how the second document we inserted (pinky) has an extra field (status). MongoDB allows us to have much flexibility with the schema, but we should be careful with it.

## insertMany()

**db.collection.insertMany()** allows us to insert more than one document at the same time. `insertMany()` receives an array of documents as a parameter.

### Example

```
1  > db.enemies.insertMany([
2    {
3      "name": "Inky",
4      "color": "Cyan",
5    },
6    {
7      "name": "Clyde",
8      "color": "Orange"
9    }
10 ])
11
```

[Copy](#)

```
12 // The command returns an object with the following structure
13 //
14 // {
15 //   "acknowledged" : true,
16 //   "insertedIds" : [
17 //     ObjectId("583c645b6fa7d619e2c574e7"),
18 //     ObjectId("583c645b6fa7d619e2c574e8")
19 //   ]
20 // }
```

The return object will have a key `insertedIds`, which contains an array with the `ids` of the inserted elements

## About `.insert()`

The `db.collection.insert()` method inserts a single document in the same way that `insertOne()` does.

### Warning Notice

The `insert` method is deprecated (should not be used because it will be removed soon) in the **NodeJS mongo driver**. You can use it in `MongoShell`, but

we strongly recommend to use the `insertOne()` and `insertMany()` methods above to get used to the syntax.

## Read | Finding/Listing Documents

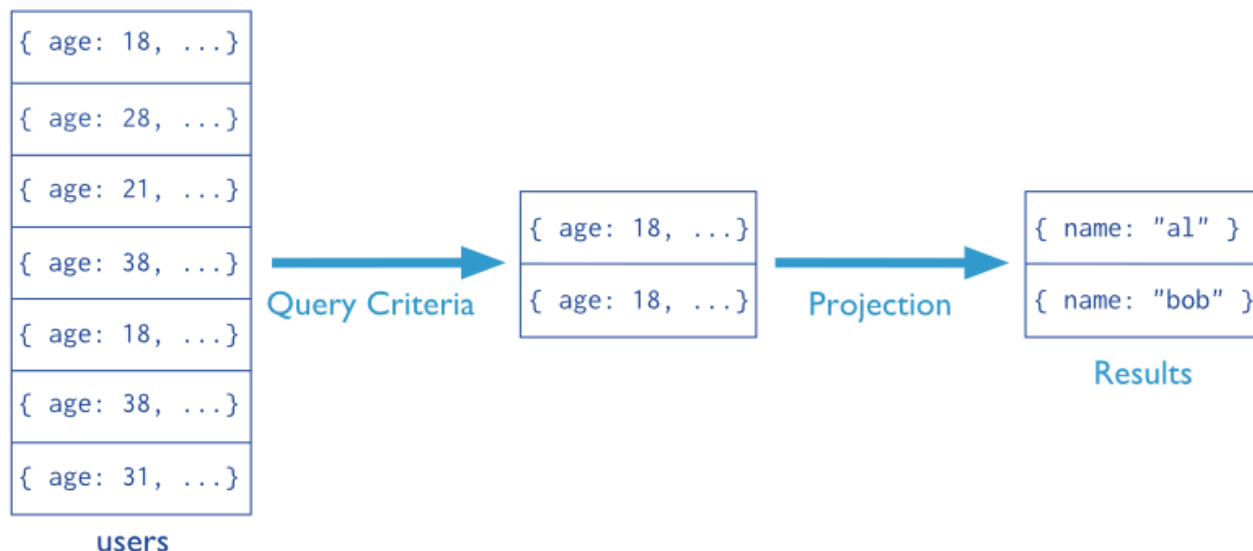
The idea behind the read operation is that we should be able to retrieve documents from the database and read their contents. The basic method to find documents is `db.collection.find()` where you can specify two optional arguments:

```
db.collection.find( query_filter, projection )
```

- `query_filter` an object that specifies which documents to return
- `projection` an object that allows us to return only certain fields from those documents

Collection                      Query Criteria                      Projection

```
db.users.find( { age: 18 }, { name: 1, _id: 0 } )
```



## Query Filters

### Select all documents

To get all documents from a collection is as simple as specifying an empty object as the first argument.

```
1 > db.enemies.find({})
```

Copy



If we only pass one argument we can omit the empty object.  
`db.enemies.find()` is the same as `db.enemies.find({})`

### Specify Equality Condition

Equality conditions will select all documents where the `<field>` matches the exact `<value>`.

```
1 > db.collection.find({<field>: <value>})
```

Copy

### Example:

```
1 > db.enemies.find({"name": "Blinky"})
```

Copy



Keep in mind that `<value>` is case sensitive, so `'Blinky'` is not the same as `'blinky'`.

## Advanced Conditions

We can specify conditions like greater than or less than, to do so we use query operators like:

```
1 > db.collection.find(  
2   { <field>: { <operator>: <value> } }  
3 )  
4
```

[Copy](#)

Mongo provides us with a comprehensive list of [query operators](#), but for now, let's look at some of the most useful ones:

Mongo Operators	Description
\$eq	equal to
\$ne	not equal to

\$gt	greater than
\$gte	greater than equal
\$lt	less than
\$lte	less than equal

### Example

Imagine that we have a collection where documents have the following structure:

```
1 [  
2   { name: 'Popeye', age: 30 },  
3   { name: 'Mickey Mouse', age: 35 },  
4   { name: 'Batman', age: 24 }  
5 ];
```

[Copy](#)

If we want to select all the users who are older than 32 years, we can do the following query.

```
1 db.user.find({"age": { $gt: 32 }})
```

[Copy](#)

## Limiting Fields | Projections

The `.find()` method's second argument, which we mentioned earlier, is called a **projection**, and it allows us to specify which fields should be returned from each document in the result.

**Using projections makes our database queries faster.** When MongoDB has to retrieve fewer fields, it's less work for the computer to do and less information to transfer over the network, so the query takes less time.

Let's see how to use the projection.

The projection must be an object with either:

- All the fields we want to include (set them to `1`)
- All the fields we want to exclude (set them to `0`)

The projection cannot combine `0` and `1` at the same time except for the `_id` that can be `0` when others are `1`.

By default, "`_id`" is set to `1`, so if we don't want to show it, we have to specify `_id: 0`.

Example:

```
1 > db.enemies.find({}, {  
2   "name": 1,  
3   "_id": 0  
4 })
```

[Copy](#)

As you can see we specified as a projection `{"name": 1, "_id": 0}`. Mongo will return the `name` property of each documents, but not their `_id`:



```
1 { "name" : "Blinky" }
2 { "name" : "Pinky" }
3 { "name" : "Inky" }
4 { "name" : "Clyde" }
```

[Copy](#)

## Independent Practice

Let's introduce the following documents to our database so that we can all start with the same data.

[Copy](#)

```
1 {
2   name: "Sue",
3   age: 19,
4   phone: {
5     personal: "555-123-123",
6     work: "555-456-456",
7     ext: "2342"
8   },
9   privileges: "user",
10  favorites: { artist: "Picasso", food: "pizza" },
11  finished: [ 17, 3 ],
12  badges: [ "blue", "black" ],
13  points: [
14    { points: 85, bonus: 20 },
15    { points: 85, bonus: 10 }
16  ],
17 },
18 {
19   name: "Bob",
20   age: 42,
21   phone: {
22     personal: "555-123-123",
```

```
23     work: "555-456-456",
24     ext: "7673"
25 },
26 privileges: "admin",
27 favorites: { artist: "Miro", food: "meringue" },
28 finished: [ 11, 25 ],
29 badges: [ "green" ],
30 points: [
31     { points: 85, bonus: 20 },
32     { points: 64, bonus: 12 }
33 ]
34 },
35 {
36     name: "Willy",
37     age: 22,
38     phone: {
39         personal: "555-123-123",
40         work: "555-456-456",
41         ext: "8263"
42     },
43     privileges: "user",
44     favorites: { artist: "Cassatt", food: "cake" },
45     finished: [ 6 ],
46     badges: [ "blue", "Picasso" ],
47     points: [
48         { points: 81, bonus: 8 },
49         { points: 55, bonus: 20 }
50     ]
51 },
52 {
53     name: "John",
54     age: 34,
55     phone: {
56         personal: "555-123-123",
57         work: "555-456-456",
58         ext: "2143"
59     },
60     privileges: "admin",
61     favorites: { artist: "Chagall", food: "chocolate" },
62     finished: [ 5, 11 ],
63     badges: [ "Picasso", "black" ],
64     points: [
65         { points: 53, bonus: 15 },
66         { points: 51, bonus: 15 }
67     ]
68 },
69 {
```

```
70   name: "Steve",
71   age: 23,
72   phone: {
73     personal: "555-123-123",
74     work: "555-456-456",
75     ext: "8253"
76   },
77   privileges: "user",
78   favorites: { artist: "Noguchi", food: "nougat" },
79   finished: [ 14, 6 ],
80   badges: [ "orange" ],
81   points: [
82     { points: 71, bonus: 20 }
83   ]
84 },
85 {
86   name: "Martin",
87   age: 43,
88   phone: {
89     personal: "555-123-123",
90     work: "555-456-456",
91     ext: "5623"
92   },
93   privileges: "user",
94   favorites: { food: "pizza", artist: "Picasso" },
95   finished: [ 18, 12 ],
96   badges: [ "black", "blue" ],
97   points: [
98     { points: 78, bonus: 8 },
99     { points: 57, bonus: 7 }
100  ]
101 }
```

Open your MongoShell, introduce the documents above in the collection `employees` and perform the following queries:

- List all the employees
- Find the employee whose name is Steve
- Find all employees whose age is greater than 30
- Find the employee whose extension is 2143

*(Should take approximately 10 minutes)*

## Update commands

The update command allows us to update some fields in the document or replace all document:


Mongo Method	Description
<code>db.collection.updateOne()</code>	Updates a single document within the collection
<code>db.collection.updateMany()</code>	Updates multiple documents within the collection
<code>db.collection.replaceOne()</code>	Replaces a single document within the collection

All these methods accept two parameters:

- `<filter>` finds what documents to update
- `<update>` specifies what fields to update in the document

## updateOne()

`db.collection.updateOne()` updates a single document that matches the specified filter.


 If we have more than one document that matches that filter, this operation will update only one.

To keep working with the collection `employees`, let's update the user "Martin" with his new extension number.

```
1 db.employees.updateOne({ name: 'Martin' }, { $set: { 'phone.ext':
```

 Copy

This function will return an object where "matchedCount" is how many documents satisfy the filter and "modifiedCount" is how many documents were updated.

 Notice that when we need to access nested objects like the `ext` field inside `phone`, we need to use dot notation like JavaScript objects. They have to be in double quotes: `"phone.ext"`:

```
1 {
2   name: "Martin",
3   age: 43,
4   phone: {
```

 Copy

```

5     personal: "555-123-123",
6     work: "555-456-456",
7     ext: "5623"
8   },
9   privileges: "user",
10  favorites: { food: "pizza", artist: "Picasso" },
11  finished: [ 18, 12 ],
12  badges: [ "black", "blue" ],
13  points: [
14    { points: 78, bonus: 8 },
15    { points: 57, bonus: 7 }
16  ]
17 }

```

```
1 { "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 } Copy
```

**Your turn!** Try to update the age of Sue to 20.

## updateMany()

**db.collection.updateMany()** updates all documents that match the specified filter.

We want to update all employees older than 30 so that their favorite writer is Cervantes.

```
1 db.employees.updateMany({ age: { $gte: 30 } }, { $set: { 'favorite' : 'Cervantes' } }) Copy
```

Result:

```
1 { "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 } Copy
```

## replaceOne()

**db.collection.replaceOne()** replaces the first document that matches the specified filter even if there are more than one.

Our employee "Martin" is not working anymore in our company so let's replace him with our new employee "Susan".

The new document for "Susan"

```
1  const susan = {  
2    name: 'Susan',  
3    age: 25,  
4    phone: { personal: '555-223-223', work: '555-421-426', ext: 4240 },  
5    privileges: 'user'  
6  };  
7  
8  db.employees.replaceOne({ name: 'Martin' }, susan);
```

[Copy](#)

Replacing a document does not change its `_id` value. The `_id` field is immutable. If we include it in the new document, it must have the same value as before.

## Delete commands

The delete commands remove documents from our collections.

### deleteOne()

```
1  db.employees.deleteOne({ _id: ObjectId('583ea82c58613b64d5df8405') })
```

[Copy](#)

### deleteMany()

If we want to remove **all** documents

```
1  db.employees.deleteMany({});
```

[Copy](#)

Removes the documents that match the filter criteria.

```
1 db.employees.deleteMany({ age: { $gte: 30 } });
```

[Copy](#)

## Independent Practice

Now that you know how to add, delete, query, and update documents, let's practice!

- Find all employees that are over 30.
- Find all employees that are less than or equal to 30.
- Find all the employees whose favorite food is pizza.
- Change Willy's personal phone number to "93-123-45-67".
- Change Bob's privilege to the regular user.
- Find all employees whose favorite artist is equal to Picasso.
- Delete the user John.
- (EXTRA) Add a bonus of 15 to all those who have a bonus less than 10.

## Summary

In this lesson, we have learned how to perform basic operations with MongoShell.

## Extra Resources

- [Mongo CRUD Documentation](#)
- [Mongo shell quick reference](#)
- [Query selectors in MongoDB](#)
- [Reference Operators](#)
- [MongoDB Shell Enhancements - give your shell some colors](#)

**Mark as completed**



PREVIOUS

MongoDB |  
Introduction

NEXT

MongoDB | MongoDB  
Compass CRUD  
Operations