uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Database

Lourdes Moreno López
Paloma Martínez Fernández
José Luis Martínez Fernández
Rodrigo Alarcón García

Lab demo 1 (Topic MongoDB (3.3))

OpenCourseWare 	uc3m Universidad Carlos III de Madrid
Data Base **Bachelor in Data Science and Engineering** **Subject**: Lab demo 1. Installation and, use and deployment (Topic 3.3: mongoDB)	

In this lab demo, you will use two ways to manage mongoDB database:

- mongoDB (shell, command line)
- Robo3T tool (client with a graphical user interface)

Contenido

Part 1: mongoDB and Robo 3T installation

In the computer you must have installed the mongoDB and Robo 3T.

1. Run the mongoDB installer.
2. Once the installation is finished:
   2.1. you need create a "data" folder in C:
   2.2. Also, inside this "data" folder, you need create a folder called "db"
   In this folder all data and configurations are stored.
3. You need access to the “bin” folder within C:/archivos de programa/Server/4.0/bin (see Figure1)
   3.1. You need open two executables: mongo y mongod. (both of you must be open)
   - **First, you run mongod. mongod** is the "Mongo Daemon" it's basically the host process for the database. When you start mongod you're basically saying "start the MongoDB process and run it in the background". It must always be working in the background.
   - **Second, you run mongo. mongo** is the command-line shell that connects to a specific instance of mongod. In this command line we will work in mongoDB: we will create documents, collections, consult, …

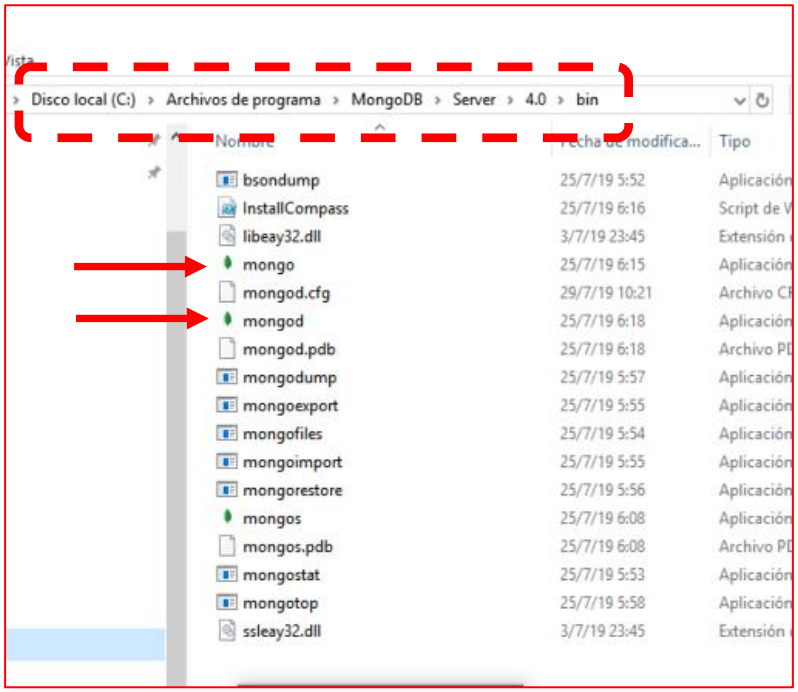

Figure 1

4. Once MongoDB is installed, you will install a client tool called (Robo 3T). It is a more usable graphical tool than the mongoDB shell.
5. mongod must be running, after, we can already run Robo 3T
6. Once the installation is finished
7. Open Robo 3T. We will make a configuration of a connection to MongoDB. Create a new connection. (see Figure 2). Select “Save” button, then “Connect” (see Figure 3).

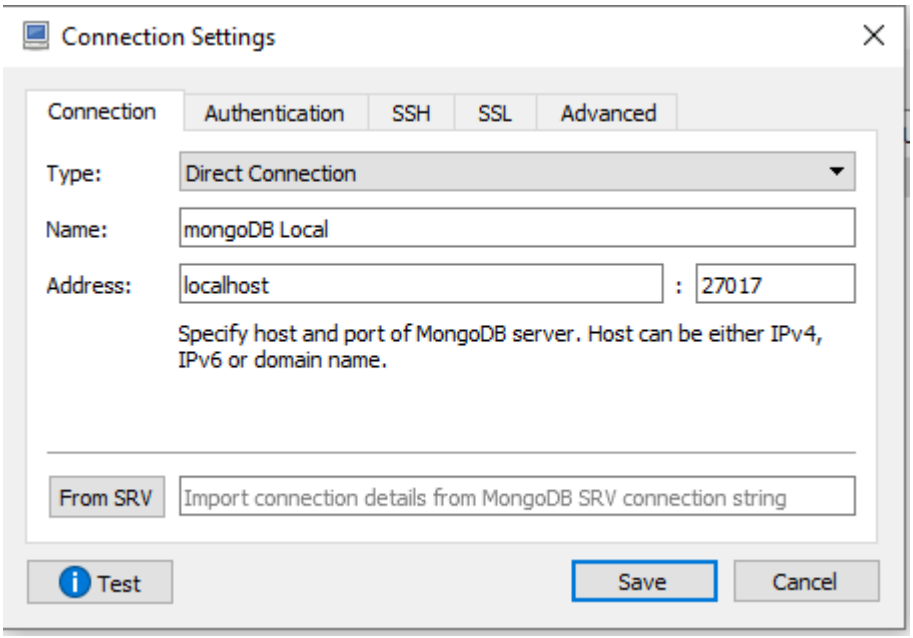

Figure 2

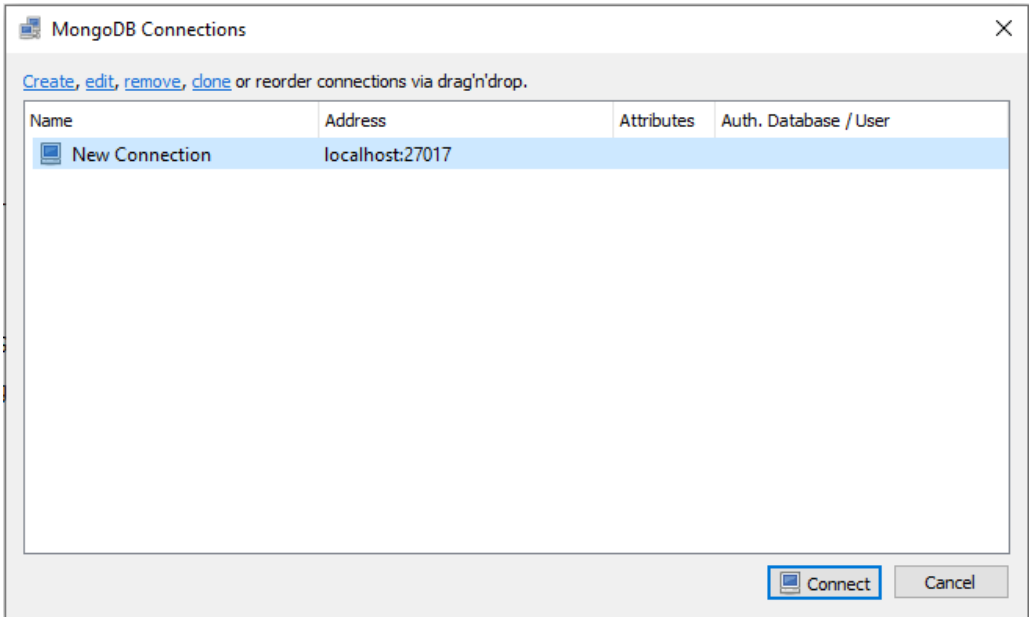

Figure 3

Part 2: using mongoDB (Shell)

(follow the step 3.1 of Part1 to use mongoDB shell)

1. Insert collection and documents

In MongoDB there is no command to create a DB as it happens in other databases, for example in ORACLE "CREATE" is used. In order to create a DB the following is done: 1) use a DB, 2) insert documents into it.

```
> use course_mongodb;
switched to db course_mongodb
```

So far, MongoDB tells us only that we have changed to the "course_mongodb" DB, that we are inside it, but if we show the DBs ('show dbs'), we can see that the new DB (course_mongodb) does not appear yet.

```
> show dbs;
admin   0.000GB
config  0.000GB
local   0.000GB
```

The new DB will appear when we insert a document or a collection of documents into it. So, we are going to create a collection called "books", and insert a document in this collection.

```
> db.books.insert({isbn: "9781593275846",title: "Eloquent JavaScript,
Second Edition", author: "Marijn Haverbeke",pages: 472});
WriteResult({ "nInserted" : 1 })
```

We have already inserted a document (in BSON format, which is a JSON in binary), if we now show the DBs, we can see that the new DB created "course_mongodb" already appears.

```
> show dbs;
admin           0.000GB
config          0.000GB
course_mongodb  0.000GB
local           0.000GB
```

If we want to retrieve the documents of a collection, it can be done with "find" command along with a condition or not. When it is without condition, all documents are recovered.

```
> db.getCollection('books').find();
```

```
{ "_id" : ObjectId("5d4169e6141e80eebd56a494"), "isbn" : "9781593275846",
"title" : "Eloquent JavaScript, Second Edition", "author" : "Marijn
Haverbeke", "pages" : 472 }
```

-Another alternative form is:

```
> db.books.find();
{ "_id" : ObjectId("5d4169e6141e80eebd56a494"), "isbn" : "9781593275846",
"title" : "Eloquent JavaScript, Second Edition", "author" : "Marijn
Haverbeke", "pages" : 472 }
```

In order to recover the data in a nicer format, we can use the "pretty" command.

```
> db.books.find().pretty();
{
        "_id" : ObjectId("5dcc04cd75503c817648860d"),
        "isbn" : "9781593275846",
        "title" : "Eloquent JavaScript, Second Edition",
        "author" : "Marijn Haverbeke",
        "pages" : 472
}
```

2. Show collections

Show the collections

```
> show collections;
books
```

3. Retrieve and search documents

First, we are going to do some inserts.

```
> db.books.insert({isbn: "9781449331818",title: "Learning JavaScript
Design Patterns", author: "Addy Osmani",pages: 254});
WriteResult({ "nInserted" : 1 })
```

The following insertion, the document has one more field than the previous documents. mongoDB allows storing documents with a flexible scheme.

```
> db.books.insert({isbn: "9781449365035",title: "Speaking JavaScript",
subtitle: "An In-Depth Guide for Programmers", author: "Axel
Rauschmayer", publisher: "O'Reilly Media", pages: 460});
WriteResult({ "nInserted" : 1 })
```

```
> db.books.insert({isbn: "9781491950296",title: "Programming JavaScript
Applications", author: "Eric Elliott", publisher: "O'Reilly Media",
pages: 254});
WriteResult({ "nInserted" : 1 })
```

We want to retrieve those documents with the publisher: "O'Reilly Media" Therefore, the "find" command includes a JSON with the condition of what we want to recover.

```
> db.books.find({publisher: "O'Reilly Media"});
{ "_id" : ObjectId("5d4171e5caa5c949a0c3eb96"), "isbn" : "9781449365035",
"title" : "Speaking JavaScript", "subtitle" : "An In-Depth Guide for
Programmers", "author" : "Axel Rauschmayer", "publisher" : "O'Reilly
Media", "pages" : 460 }
{ "_id" : ObjectId("5d417446d568b96cdd259465"), "isbn" : "9781491950296",
"title" : "Programming JavaScript Applications", "author" : "Eric
Elliott", "publisher" : "O'Reilly Media", "pages" : 254 }
```

Anotherr search:.

```
> db.books.find({author: "Addy Osmani"});
{ "_id" : ObjectId("5d417193caa5c949a0c3eb95"), "isbn" : "9781449331818",
"title" : "Learning JavaScript Design Patterns", "author" : "Addy
Osmani", "pages" : 254 }
```

4. Delete documents

We retrieve all documents

```
> db.books.find();
{ "_id" : ObjectId("5d4169e6141e80eebd56a494"), "isbn" : "9781593275846",
"title" : "Eloquent JavaScript, Second Edition", "author" : "Marijn
Haverbeke", "pages" : 472 }
{ "_id" : ObjectId("5d417193caa5c949a0c3eb95"), "isbn" : "9781449331818",
"title" : "Learning JavaScript Design Patterns", "author" : "Addy
Osmani", "pages" : 254 }
{ "_id" : ObjectId("5d4171e5caa5c949a0c3eb96"), "isbn" : "9781449365035",
"title" : "Speaking JavaScript", "subtitle" : "An In-Depth Guide for
Programmers", "author" : "Axel Rauschmayer", "publisher" : "O'Reilly
Media", "pages" : 460 }
{ "_id" : ObjectId("5d417446d568b96cdd259465"), "isbn" : "9781491950296",
"title" : "Programming JavaScript Applications", "author" : "Eric
Elliott", "publisher" : "O'Reilly Media", "pages" : 254 }
```

If we want to delete the documents that meet the criteria that the title is "Eloquent JavaScript", it would be:

```
> db.books.remove ({"title" : "Eloquent JavaScript, Second Edition"});
WriteResult({ "nRemoved" : 1 })
```

If we retrieve the documents, there must be one less.

```
> db.books.find();
```

```
{ "_id" : ObjectId("5d417193caa5c949a0c3eb95"), "isbn" : "9781449331818",
"title" : "Learning JavaScript Design Patterns", "author" : "Addy
Osmani", "pages" : 254 }
{ "_id" : ObjectId("5d4171e5caa5c949a0c3eb96"), "isbn" : "9781449365035",
"title" : "Speaking JavaScript", "subtitle" : "An In-Depth Guide for
Programmers", "author" : "Axel Rauschmayer", "publisher" : "O'Reilly
Media", "pages" : 460 }
{ "_id" : ObjectId("5d417446d568b96cdd259465"), "isbn" : "9781491950296",
"title" : "Programming JavaScript Applications", "author" : "Eric
Elliott", "publisher" : "O'Reilly Media", "pages" : 254 }
```

5. Update the fields

We are going to modify the document whose title is "Learning JavaScript Design Patterns" by "Learning JavaScript Design Patterns, Second edition".

```
> db.books.update({title: "Learning JavaScript Design Patterns"},{title:
"Learning JavaScript Design Patterns, Second Edition"});

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

We will retrieve all the documents to confirm that the update has been made.

```
> db.books.find();
{ "_id" : ObjectId("5d417193caa5c949a0c3eb95"), "title" : "Learning
JavaScript Design Patterns, Second Edition" }
{ "_id" : ObjectId("5d4171e5caa5c949a0c3eb96"), "isbn" : "9781449365035",
"title" : "Speaking JavaScript", "subtitle" : "An In-Depth Guide for
Programmers", "author" : "Axel Rauschmayer", "publisher" : "O'Reilly
Media", "pages" : 460 }
{ "_id" : ObjectId("5d417446d568b96cdd259465"), "isbn" : "9781491950296",
"title" : "Programming JavaScript Applications", "author" : "Eric
Elliott", "publisher" : "O'Reilly Media", "pages" : 254 }
```

We see that the title in the document has been updated, but has eliminated the other values such as:"isbn": "9781491950296", "autor": "Eric Elliott", "editor": "O'Reilly Media", "páginas": 254. If we want to recover the deleted values, we can use the "$set" command.

```
> db.books.update({title: "Learning JavaScript Design Patterns, Second
Edition"},{$set: {"isbn" : "9781491950296", "author" : "Eric Elliott",
"publisher" : "O'Reilly Media", "pages" : 254}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Therefore, if we retrieve the documents, we can confirm that the values have been added to the document.

```
> db.books.find();
{ "_id" : ObjectId("5d417193caa5c949a0c3eb95"), "title" : "Learning
JavaScript Design Patterns, Second Edition", "author" : "Eric Elliott",
"isbn" : "9781491950296", "pages" : 254, "publisher" : "O'Reilly Media" }
{ "_id" : ObjectId("5d4171e5caa5c949a0c3eb96"), "isbn" : "9781449365035",
"title" : "Speaking JavaScript", "subtitle" : "An In-Depth Guide for
```

```
Programmers", "author" : "Axel Rauschmayer", "publisher" : "O'Reilly
Media", "pages" : 460 }
{ "_id" : ObjectId("5d417446d568b96cdd259465"), "isbn" : "9781491950296",
"title" : "Programming JavaScript Applications", "author" : "Eric
Elliott", "publisher" : "O'Reilly Media", "pages" : 254 }
```

6. Delete a database

We are going to create a new database named 'database_proof'.

```
> use database_proof;
switched to db database_proof
```

```
> db.proof.insert({hello:"hello"});
WriteResult({ "nInserted" : 1 })
```

```
> show dbs;
admin           0.000GB
config          0.000GB
course_mongodb  0.000GB
database_proof  0.000GB
local           0.000GB
```

We will delete the "database_proof".

```
> db.dropDatabase();
{ "dropped" : "database_proof", "ok" : 1 }
```

So, if we show the DBs, we can confirm that the DB has not been deleted.

```
> show dbs;
admin           0.000GB
config          0.000GB
course_mongodb  0.000GB
local           0.000GB
>
```

we switch to our course_mongodb BD

```
> use course_mongodb;
switched to db course_mongodb
```

7 Embedded documents

We will study more content in a mongo client tool (using Robo 3T) in part3

Part 3: Using and starting with Robo 3T

We are going to study the use of mongoDB thought the Robo3T tool.

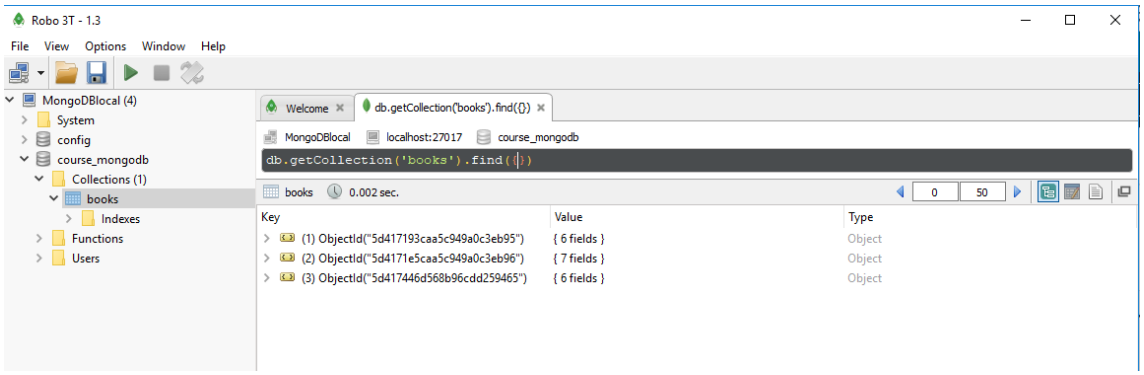

Figure 4

(see Figure 1) In the menu on the left we can see the DB "course_mongoDB", and within it, we have its collections, which in this case we have only one called "books". If we access the “books” collection, we can see in the central box that there are three documents.

1. Embedded documents

We will insert a document into another document, or embedded documents.

We are going to insert a new document, “books” (right button: Insert Document option) (see Figure 2)

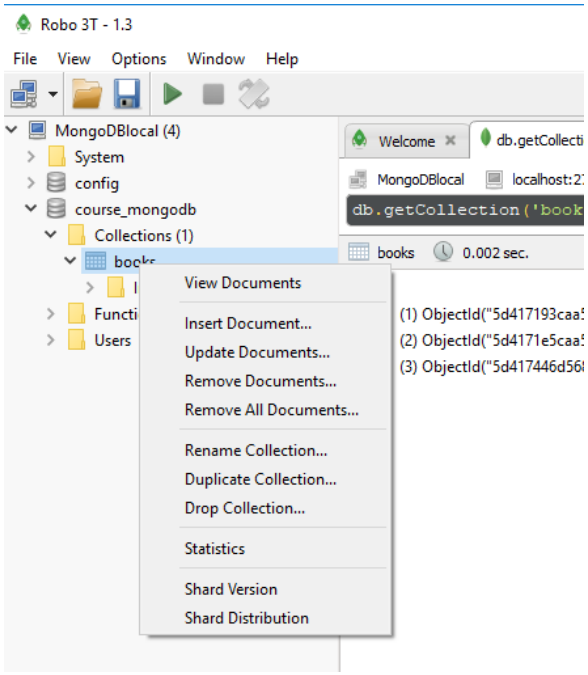

Figure 2

We will insert a new document (see Figure3 and Figure 4):

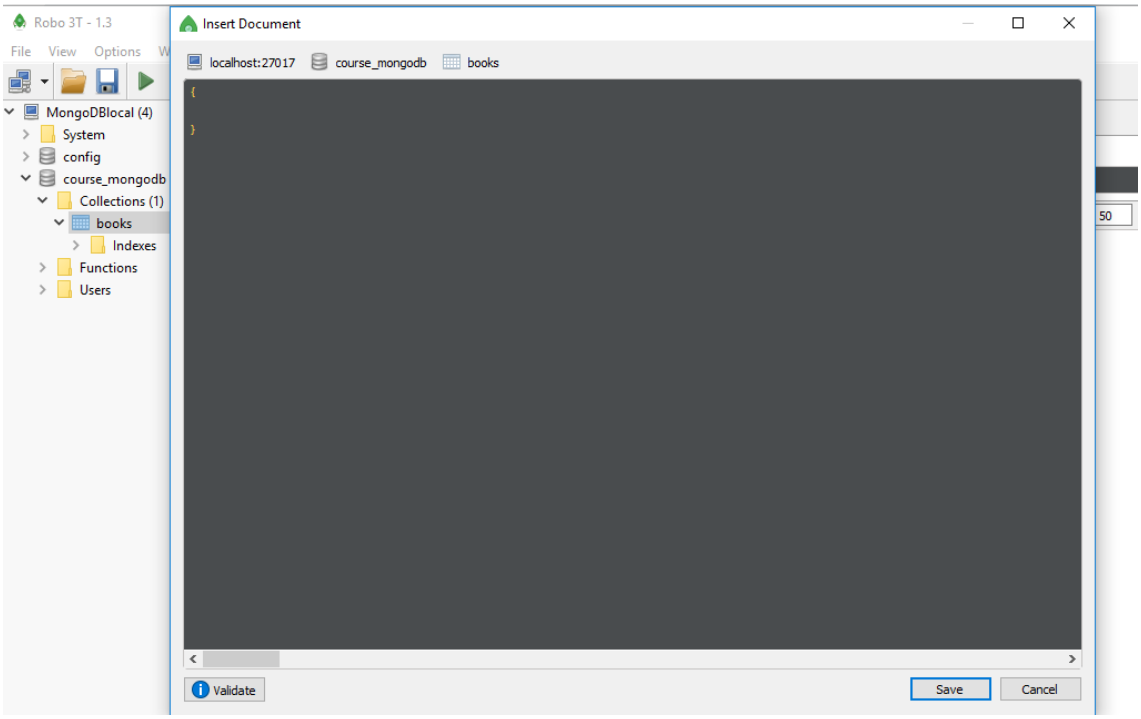

Figure 3


```
{
    "_id" : ObjectId("5d42b3b9fad1eecc6c17049e"),
    "isbn" : "9781593277574",
    "title" : "Understanding ECMAScript 6",
    "subtitle" : "The Definitive Guide for JavaScript Developers",
    "author" : "Nicholas C. Zakas",
    "published" : "2016-09-03T00:00:00.000Z",
    "publisher" : "No Starch Press",
    "pages" : 352,
    "date" : 2016,
    "description" : "ECMAScript 6 represents the biggest update to the core
of JavaScript in the history of the language. In Understanding ECMAScript 6,
expert developer Nicholas C. Zakas provides a complete guide to the object
types, syntax, and other exciting changes that ECMAScript 6 brings to
JavaScript.",
    "website" : "https://leanpub.com/understandinges6/read",
    "label" : [
        {
            "nombre" : "JavaScript"
        },
        {
            "nombre" : "Guide"
        }
    ]
}
```

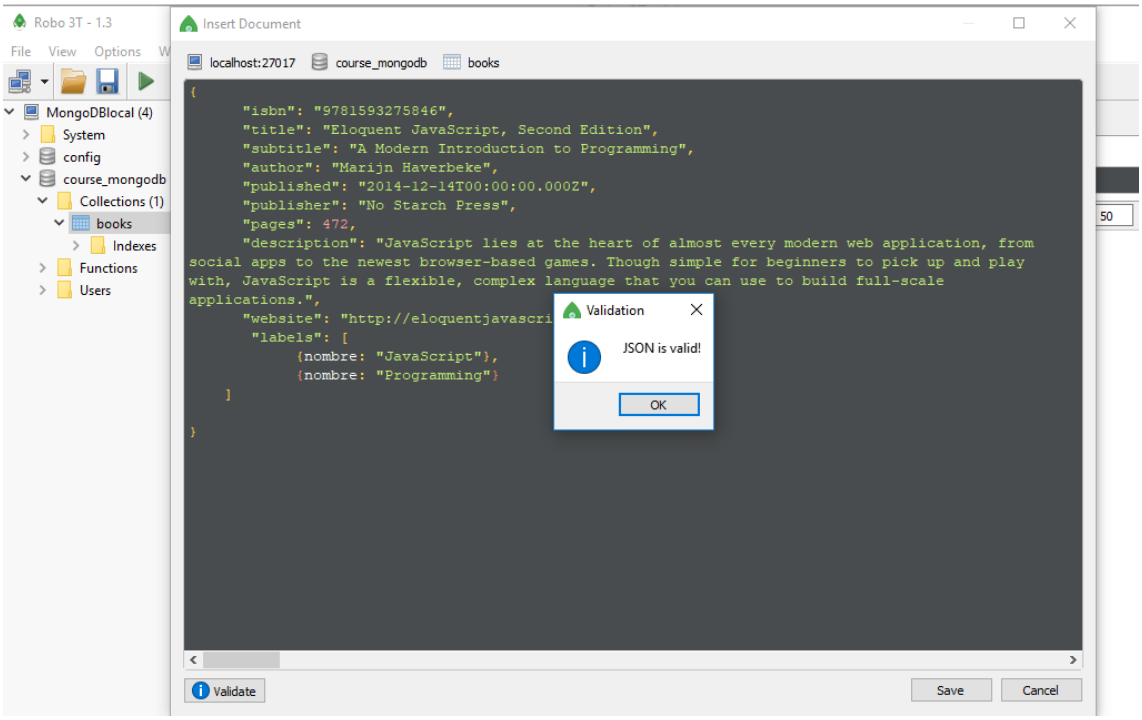

Figure 4

We can press the 'validate' button to verify that the JSON is valid. If it is valid, press the 'save' button to insert.

The document has the structure of the previous ones, but we have incorporated more fields and values, they are the labels. In a database, one or more labels can be assigned to a book.

If we show the documents in the books collection, we can see that it has one more document, and if we display document 4, we can see that it has another collection of documents called Labels. (see Figure 5)

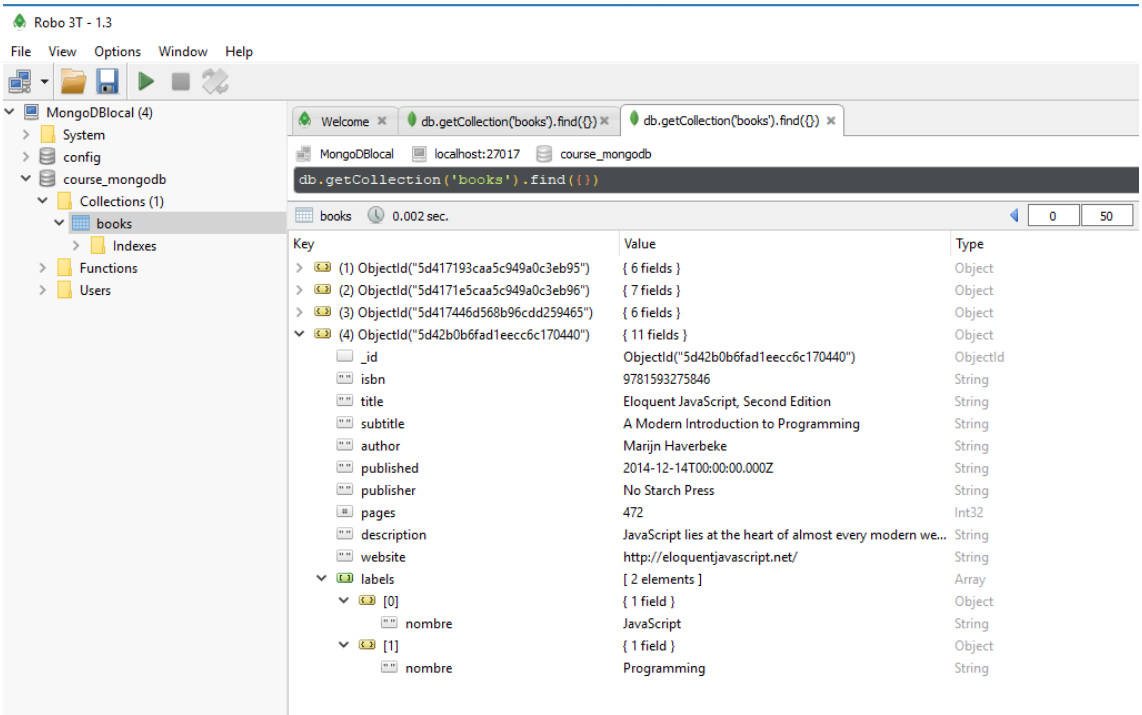

Figure 5

From the current view (view results in tree model) (see Figure 5), we can go to a view with the code, clicking on the icon in the upper right corner (view results in text mode) (see Figure 6).

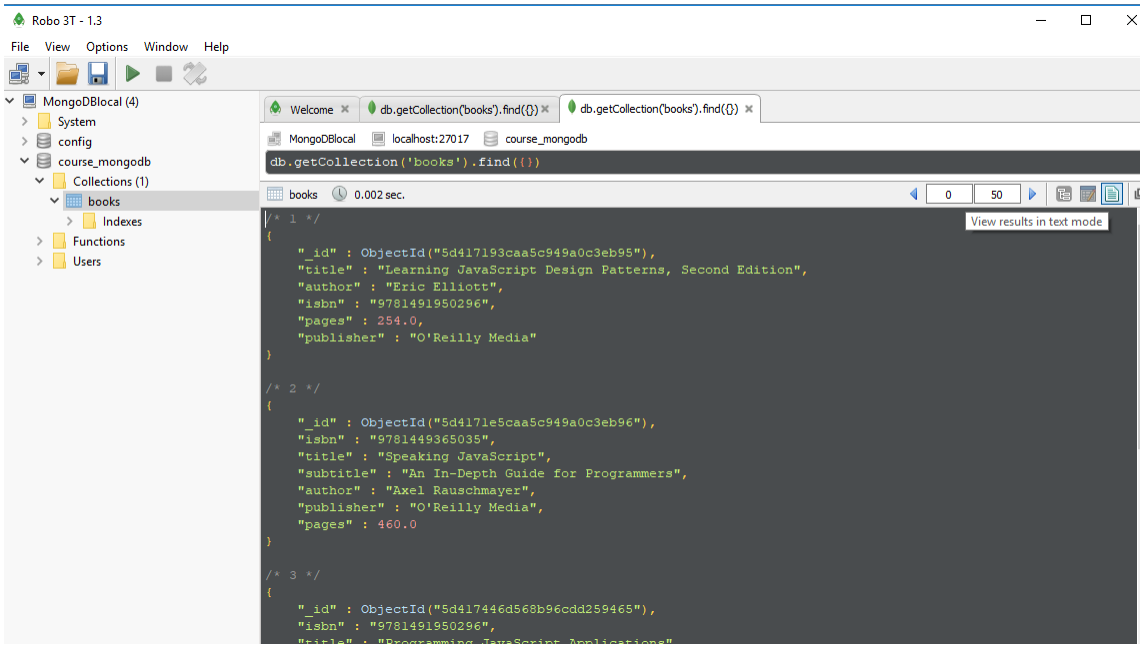

Figure 6

We are going to insert another document. This new document has a different structure because, in addition to labels, it will have comments, since in a DB a book may have comments (see Figure 7).

```
{
    "_id" : ObjectId("5d42b3b9fad1eecc6c17048e"),
    "isbn" : "9781593277574",
    "title" : "Understanding ECMAScript 6",
    "subtitle" : "The Definitive Guide for JavaScript Developers",
    "author" : "Nicholas C. Zakas",
    "published" : "2016-09-03T00:00:00.000Z",
    "publisher" : "No Starch Press",
    "pages" : 352,
    "date" : 2016,
    "description" : "ECMAScript 6 represents the biggest update to the core
of JavaScript in the history of the language. In Understanding ECMAScript 6,
expert developer Nicholas C. Zakas provides a complete guide to the object
types, syntax, and other exciting changes that ECMAScript 6 brings to
JavaScript.",
    "website" : "https://leanpub.com/understandinges6/read",
    "label" : [
        {
            "nombre" : "JavaScript"
        },
        {
            "nombre" : "Guide"
        }
    ],
```

```
    "comments" : {
        "user_id" : 120,
        "text" : "good",
        "score_book" : 5,
        "language" : "en"
    }
}
```

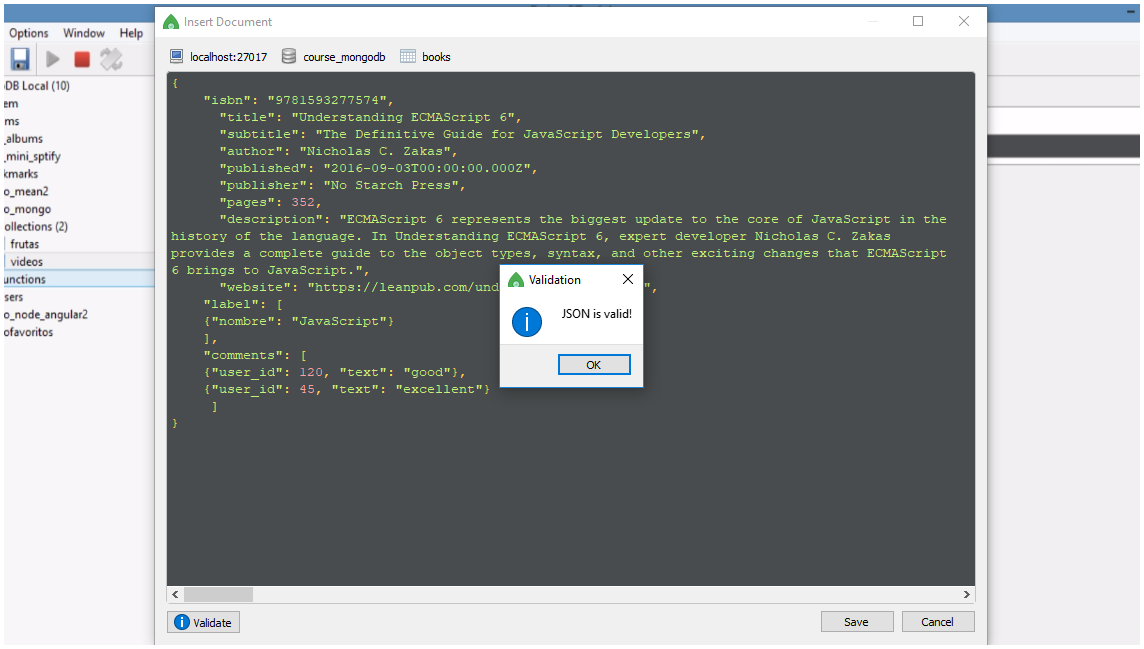

Figure 7

If we view the documents (see Figure 8)

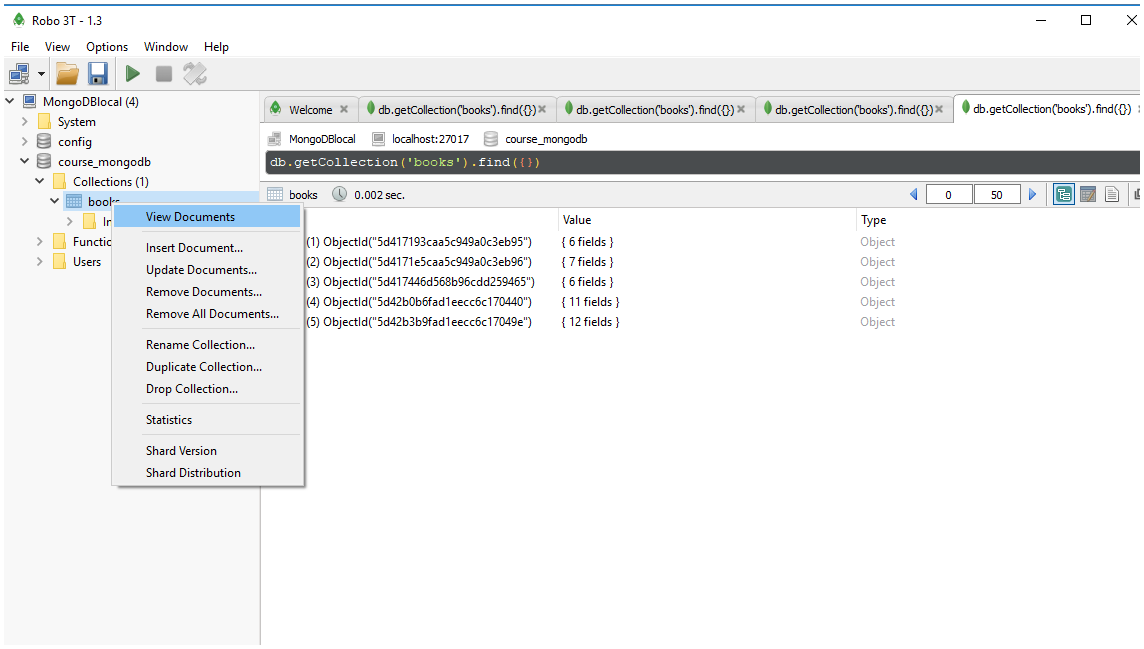

Figure 8

We can see the 5 documents, if document 5 is displayed, we can see the document Comments into the document 5 (see Figures 9 and 10)

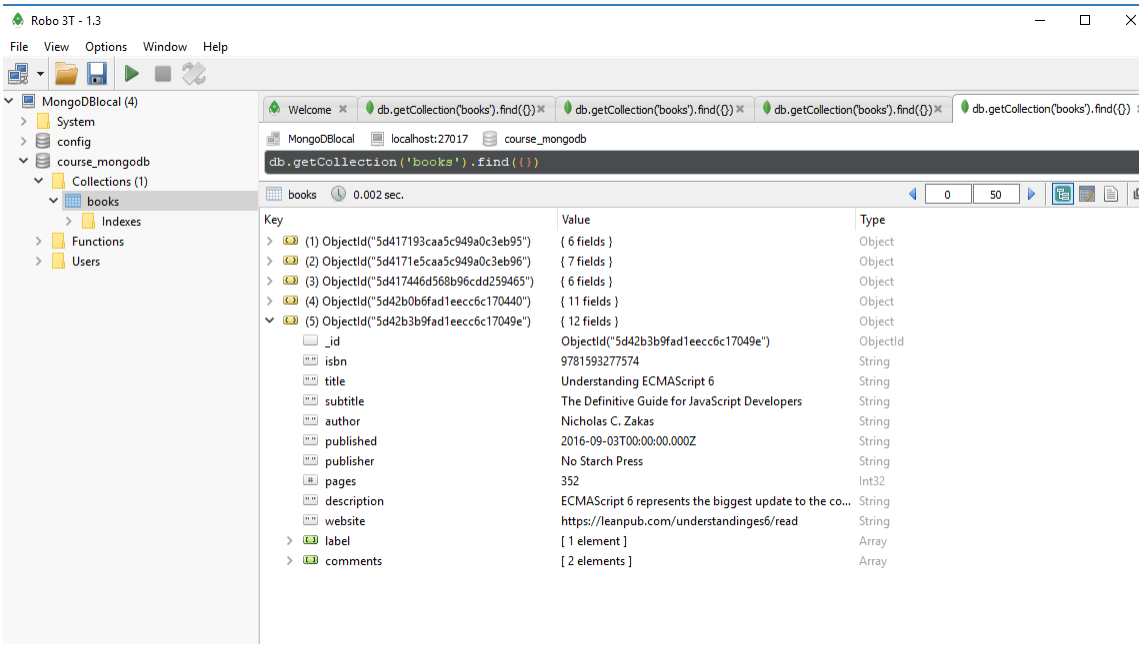

Figure 9

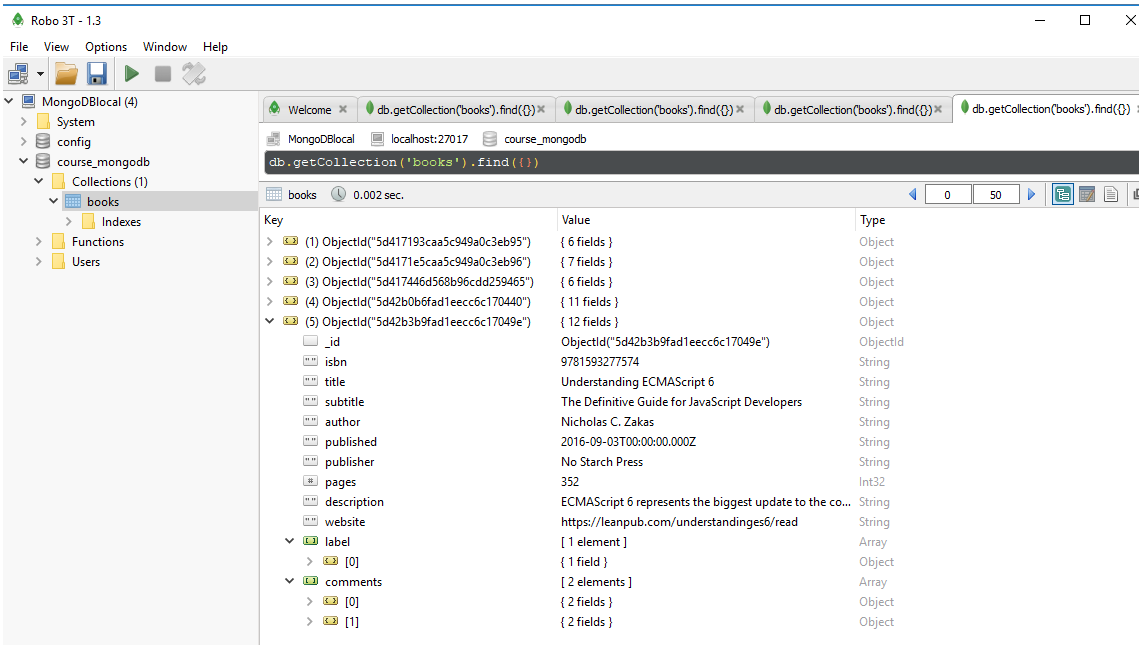

Figure 10

As it happened when we wanted to update or edit documents, we saw that using "update" all the values were completely updated, and if we wanted to add a single field, we could do it with "$ set" within the "update". In collections it is the same, you can add a document with "$push" (See Figure 11).

Figure 11

To execute the query, we click on the play icon. Robo 3T indicates that 1 record has been updated (see Figure 12).

Figure 12

We can see the results in tree mode view too (see Figure 13) or text mode (see Figure 14).

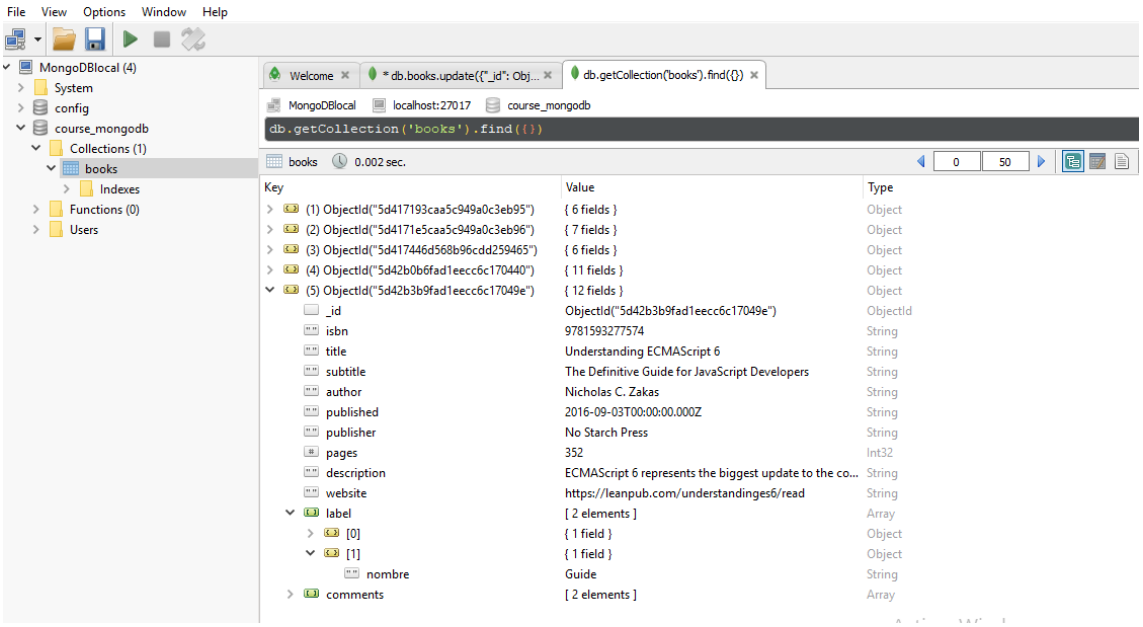

Figure 13

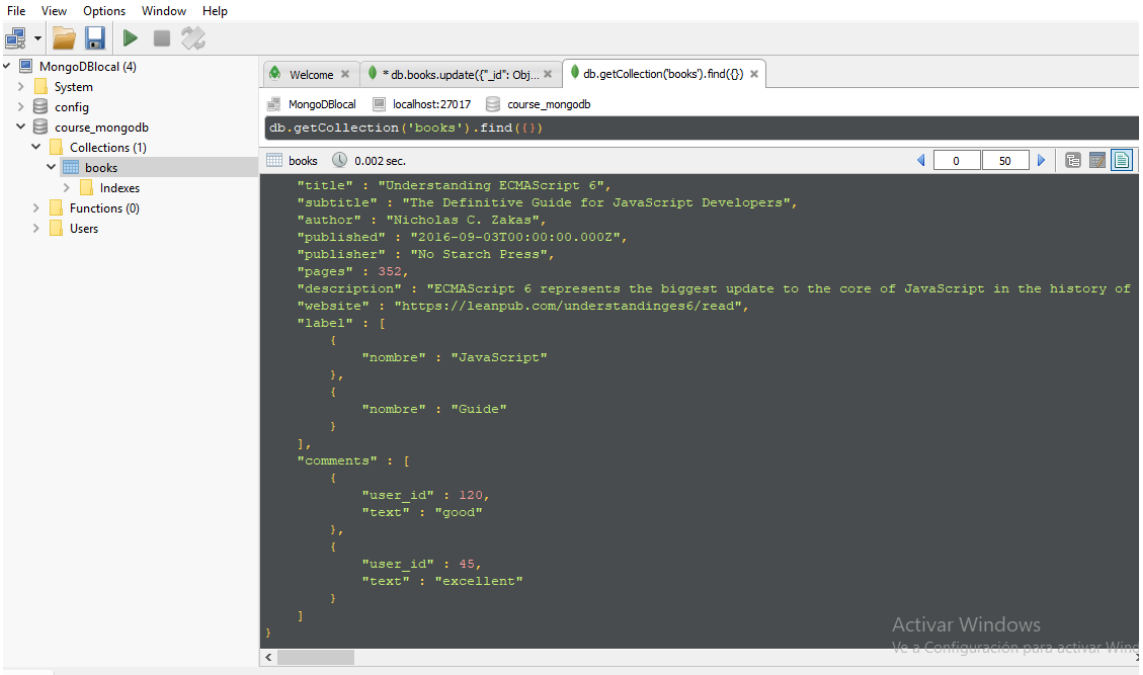

Figure 14

Also, we can update directly in the tree mode (see Figure 15)

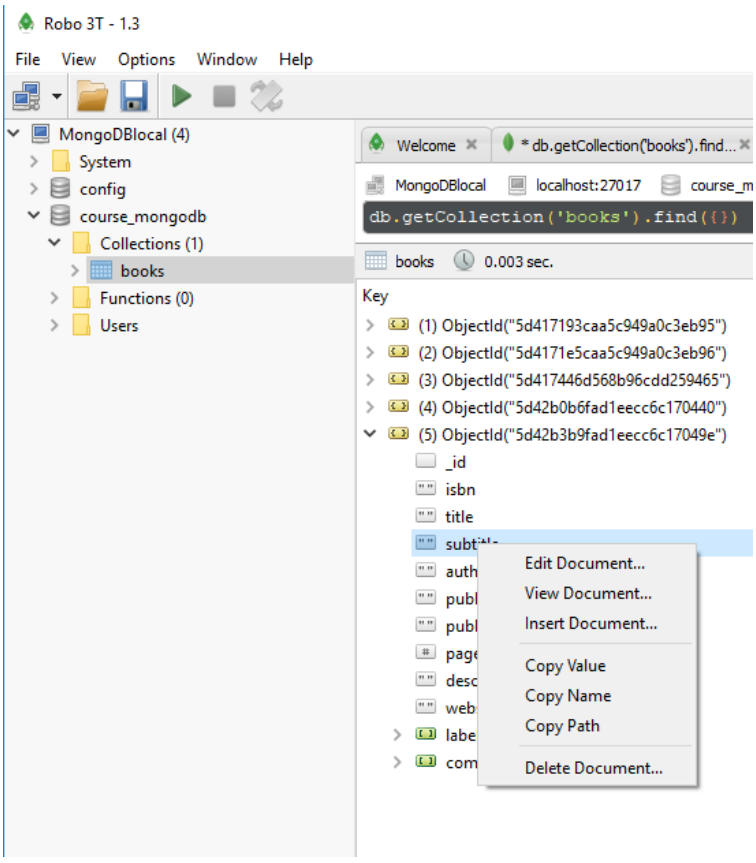

Figure 15

2. Sort documents

Ascending (see Figure 16):

```
db.getCollection('books').find({}).sort({_id: 1})
```

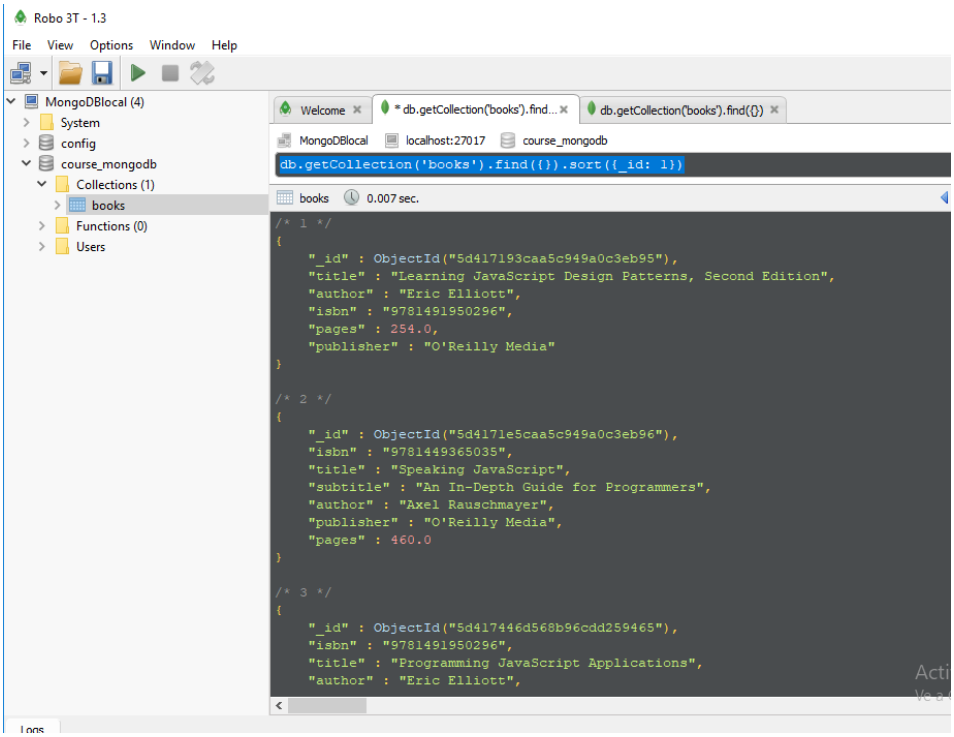

Figure 16

Descending (see Figure 17)

```
db.getCollection('books').find({}).sort({_id: -1})
```

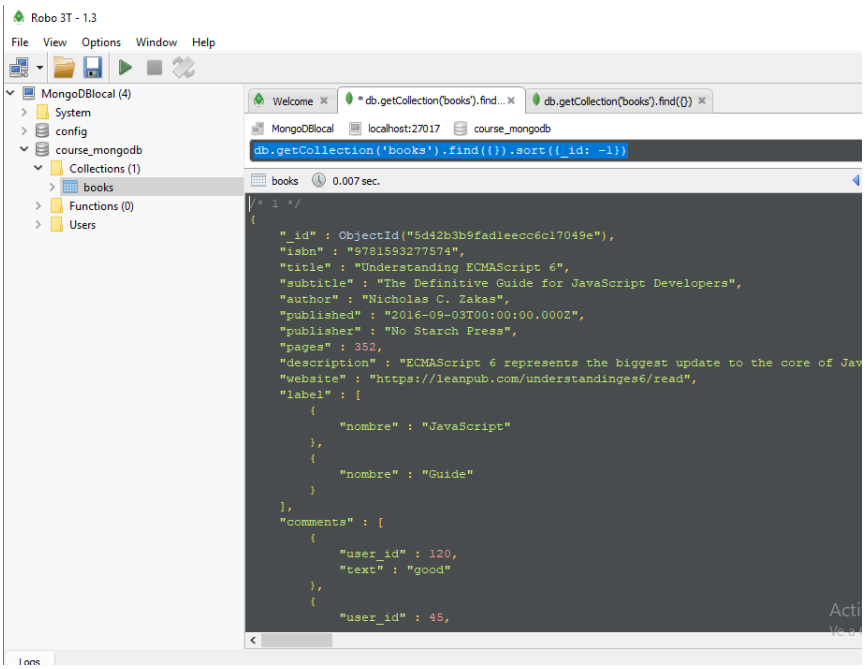

Figure 17

3. Recover different document fields

Retrieve the authors of all the documents in the “books” collection (see Figure 18)

```
db.getCollection('books').find(null, {author:1})
```

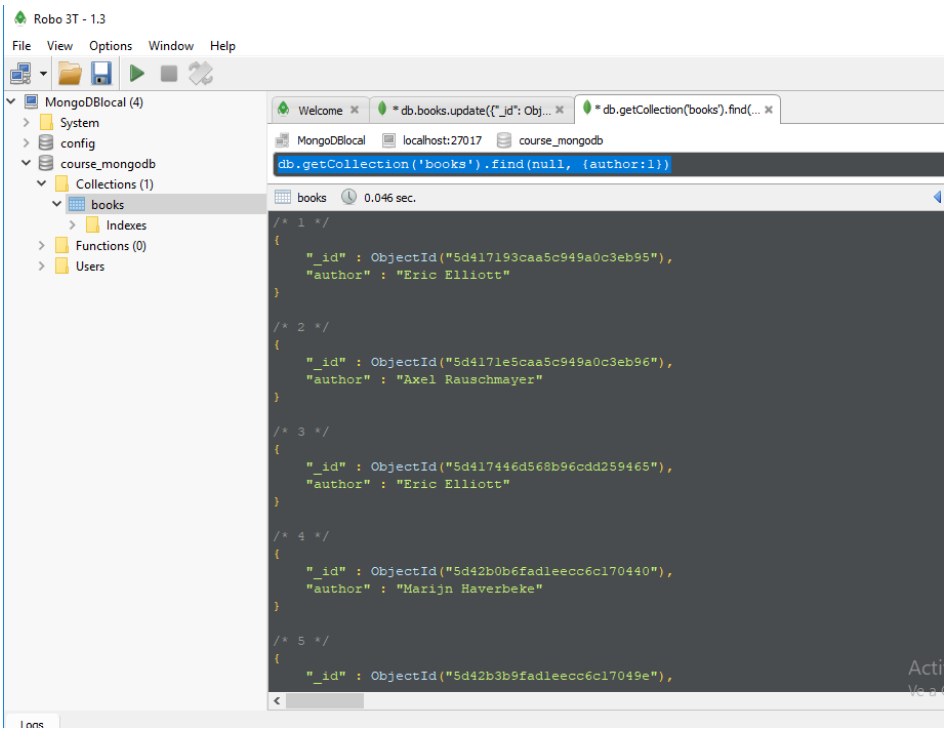

Figure 18

The same query, but without the id field (see Figure 19):

```
db.getCollection('books').find(null, {author:1, _id:0})
```

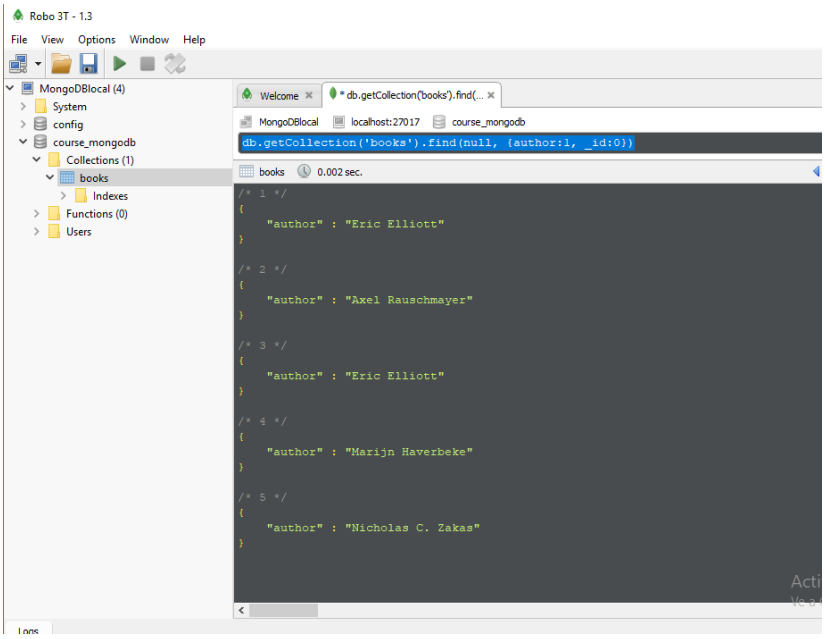

Figure 19

Retrieve the authors and publisher of all the documents in the “books” collection (see Figure 20)

```
db.getCollection('books').find(null, {author:1, _id:0, publisher: 1})
```

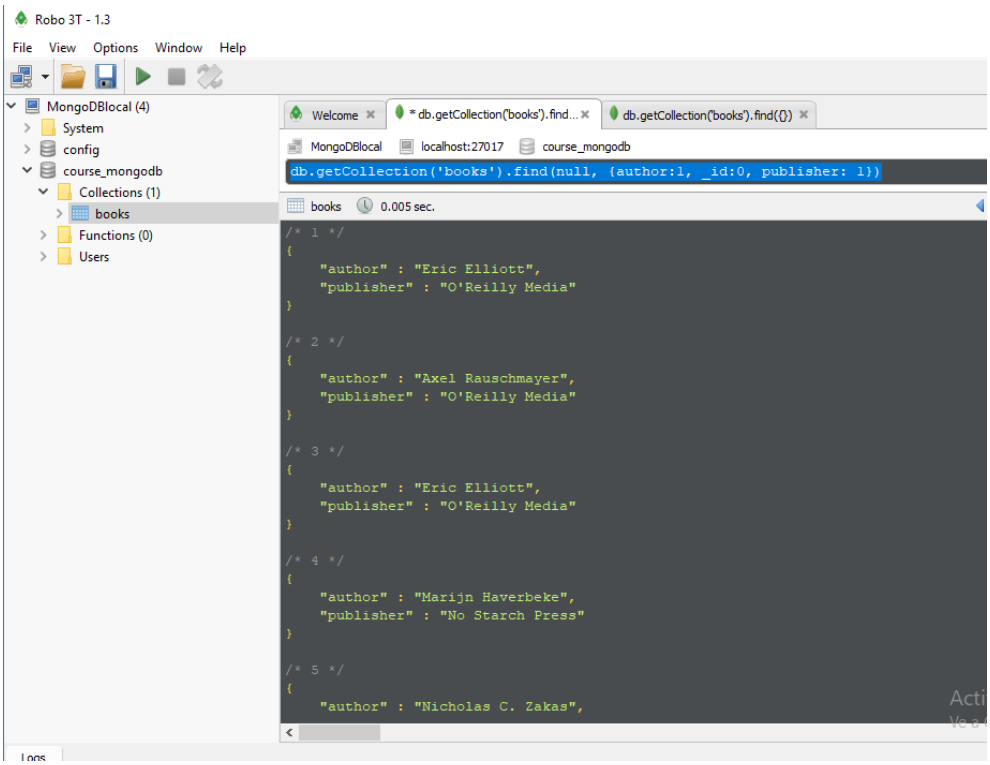

Figure 20

Retrieve documents when el name of the labels is “JavaScript” (Figure 21)

```
db.getCollection('books').find({"label.nombre" : "JavaScript"})
```

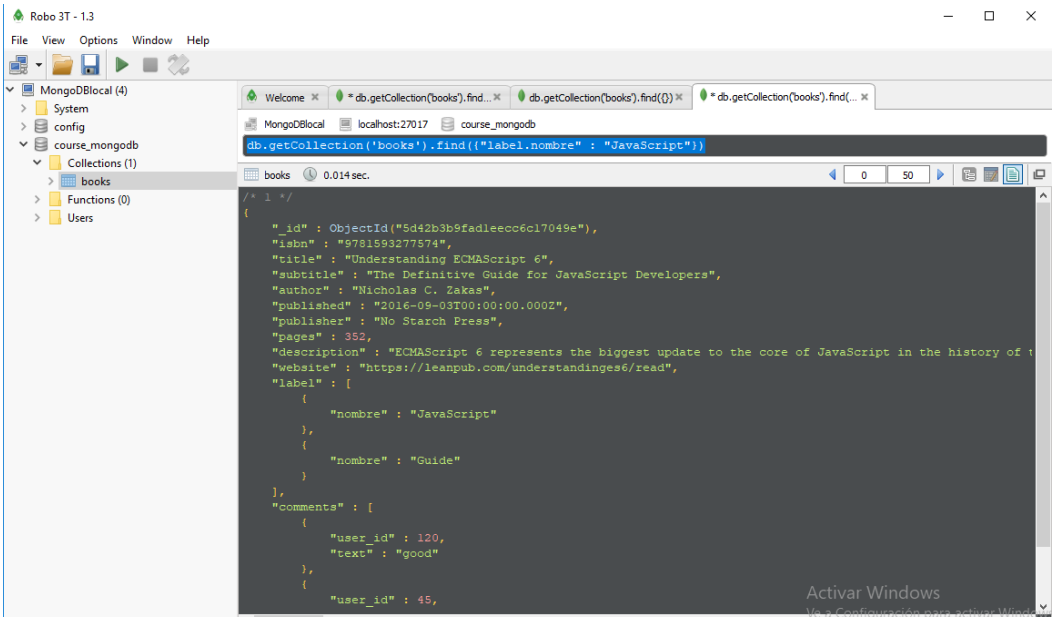

Figure 21

4. Operators in a query

$gt (“greater than”)

Recover documents when the date is greater than 2015.

```
db.getCollection('books').find({date: {$gt: 2015}})
```

$gte (“greater than equals”)

Recover documents when the date is greater than equals 2015.

```
db.getCollection('books').find({ date: {$gte: 2015}})
```

5. Update of many documents

Update the date to 2019 in the documents where the publisher is “O'Reilly Media” (see Figure 22)

```
db.getCollection('books').update(
    {"publisher" : "O'Reilly Media"},
    { $set: {"date": 2019} },
    { multi: true }
    )
```

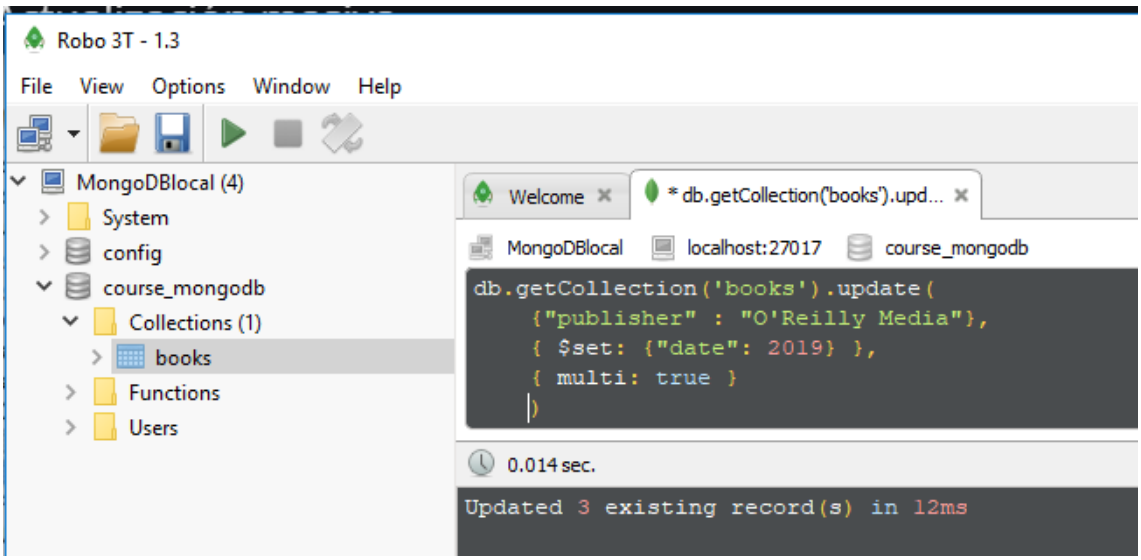

Figure 22

6. Count documents

(see Figure 23)

```
db.getCollection('books').find({}).count();
```

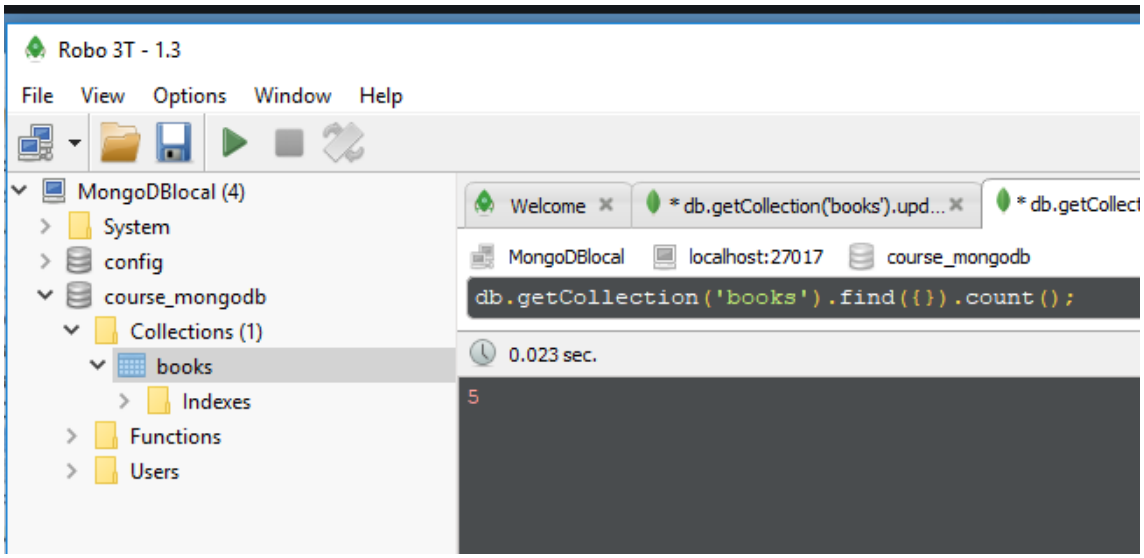

Figure 23

Count the documents dated 2019 (see Figure 24)

```
db.getCollection('books').find({ "date" : 2019.0}).count()
```

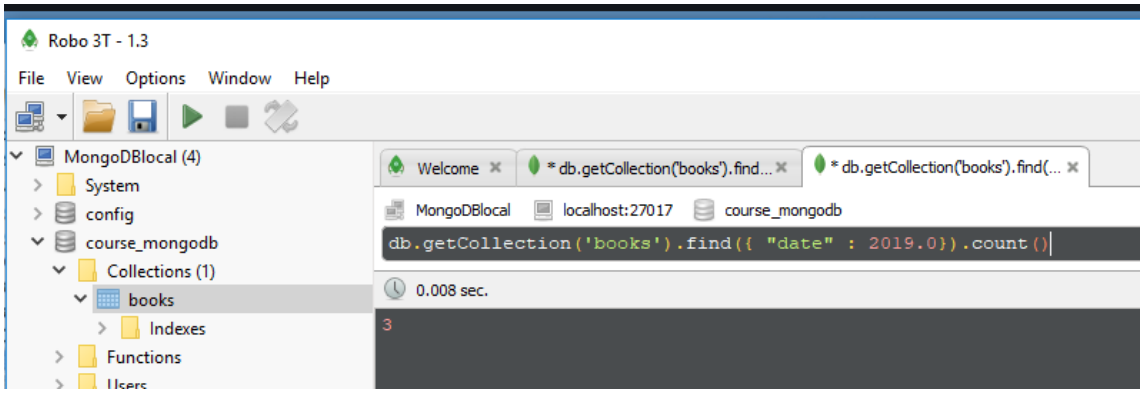

Figure 24