

➔ **Download Studio 3T from**

<https://studio3t.com/download/>

Request a student license from the company.

Expect email with a license key from the company.

Start up Studio 3T.

In Studio 3T go to Help > License Manager > Import New License

Copy and paste the license key from the email into the window and click OK.

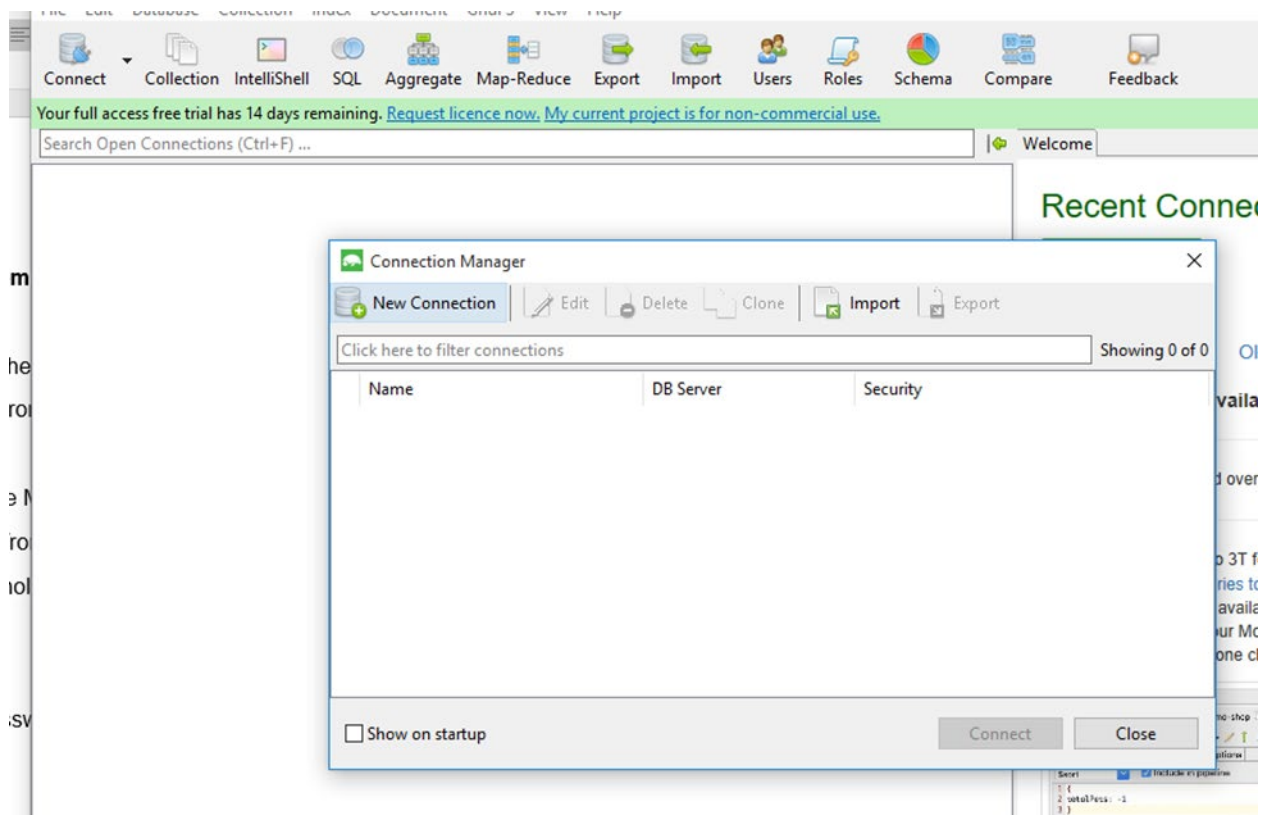
NOTE: You have to copy "the whole thing" not just the encrypted part. From --- to ---

➔ **Register the Server**

Wait for a **user name and for password information from your professor**. This is YET ANOTHER password.

Click Connect

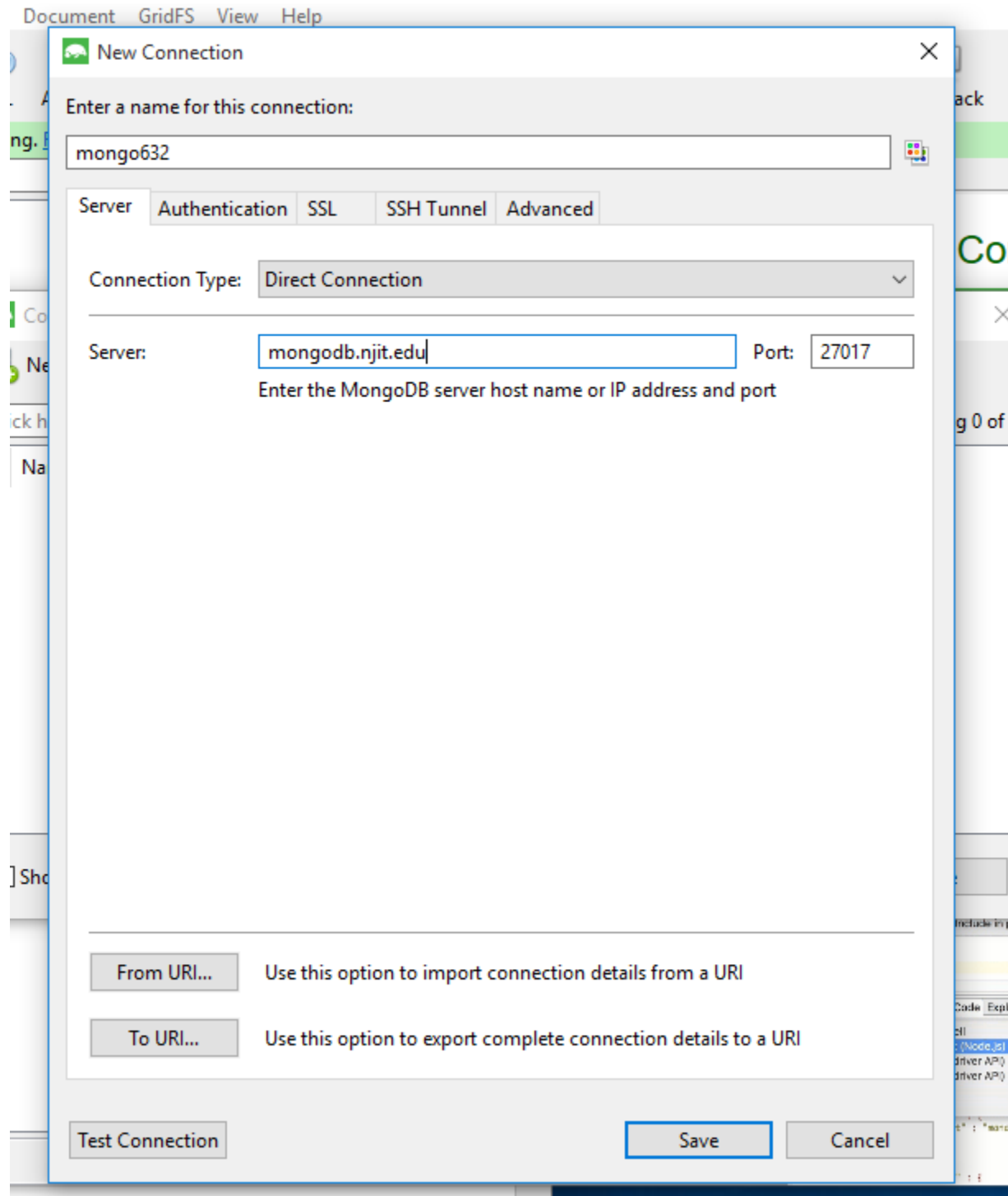
Click New Connection



Invent a name for your connection. How about mongo632?

Your server is mongodb.njit.edu (not localhost).

Then click on the Authentication tab.



Choose Authentication Mode:

Legacy

Type in the user name and password above in **red**.

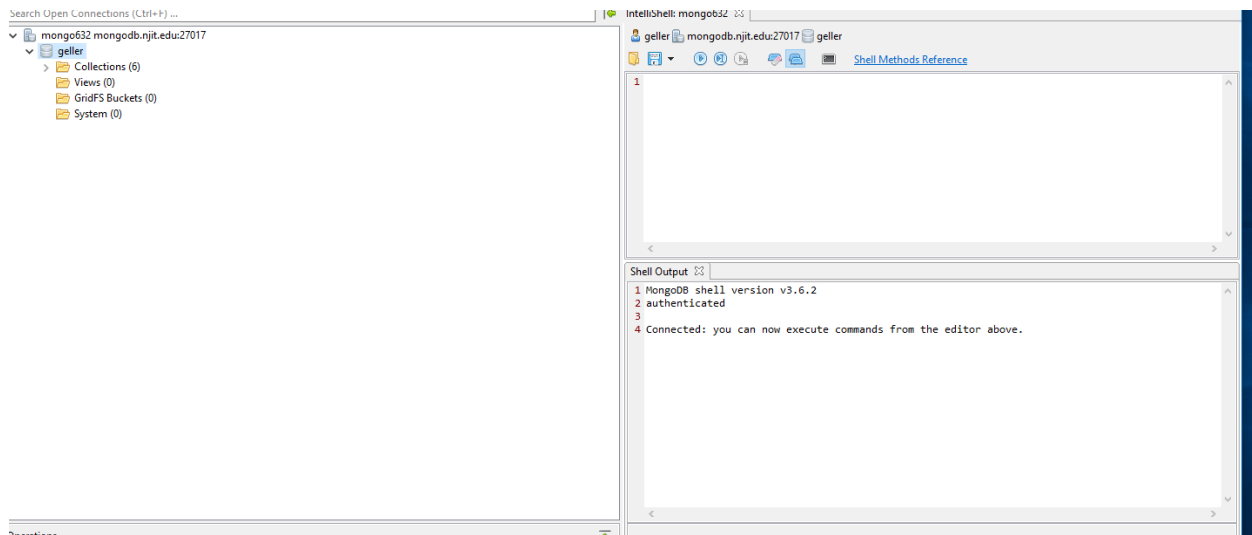
The authentication database is the same as your username.

Click on Save (if there is a Save) and then click on Connect.

Now right click on your database. (Next to the picture of a disk stack.)

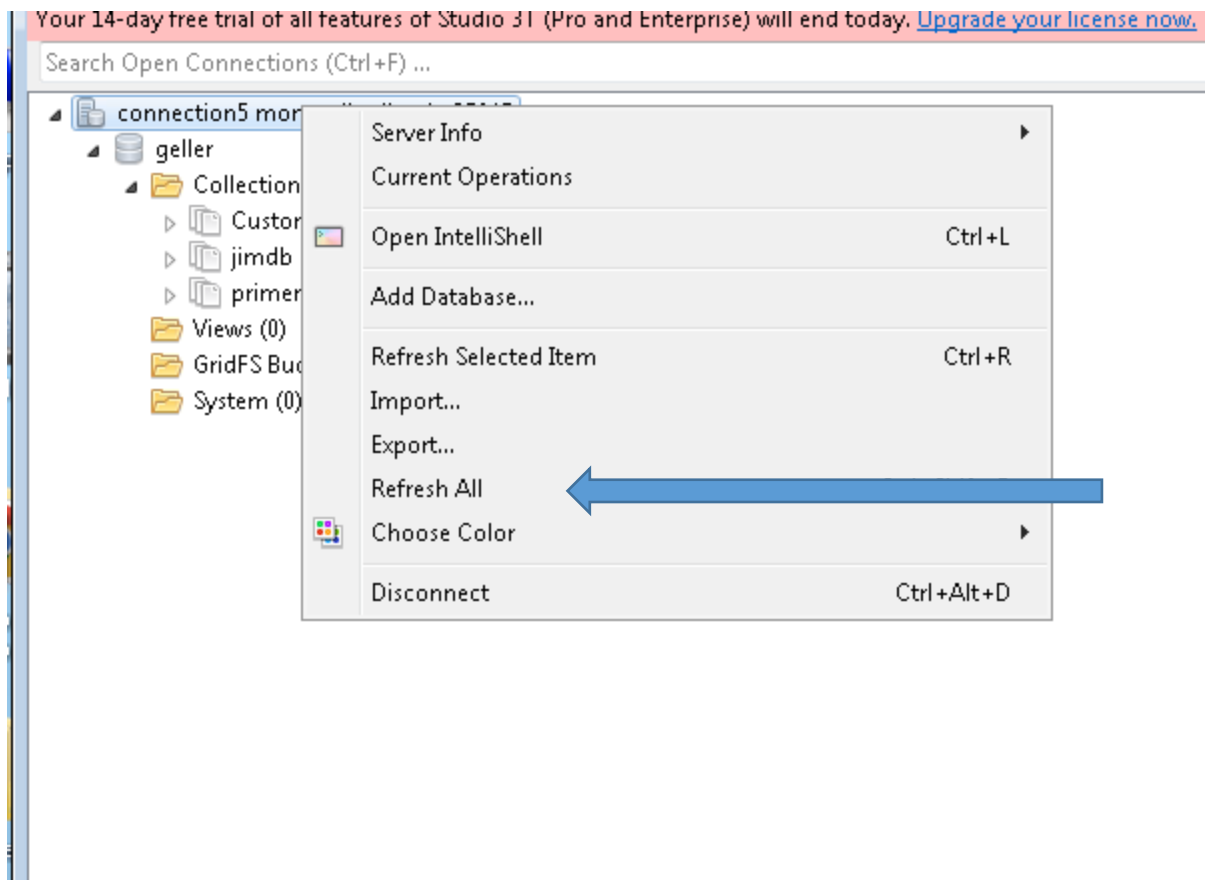
Click on Open Intellishell.

It should look something like this:



Now you are connected and can insert data or query data.

- ➔ **Reconnect:** If you already have a server registered, right click on the server connection and click Refresh All. See below.



➔ Next we need to create a collection.

A collection (of documents) corresponds to a table in MongoDB.

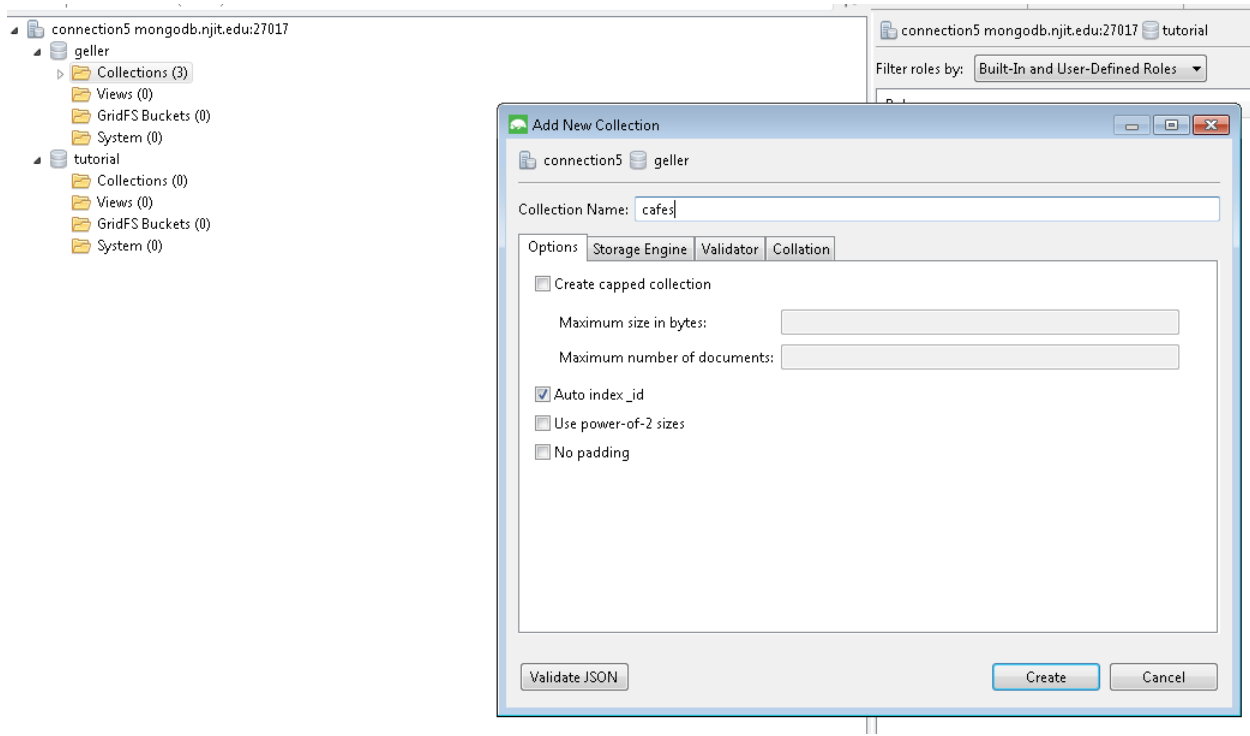
There should be a database with your user name. For me it is **geller**.

Under it there should be Collections.

Right click on Collections.

Click on Add Collection

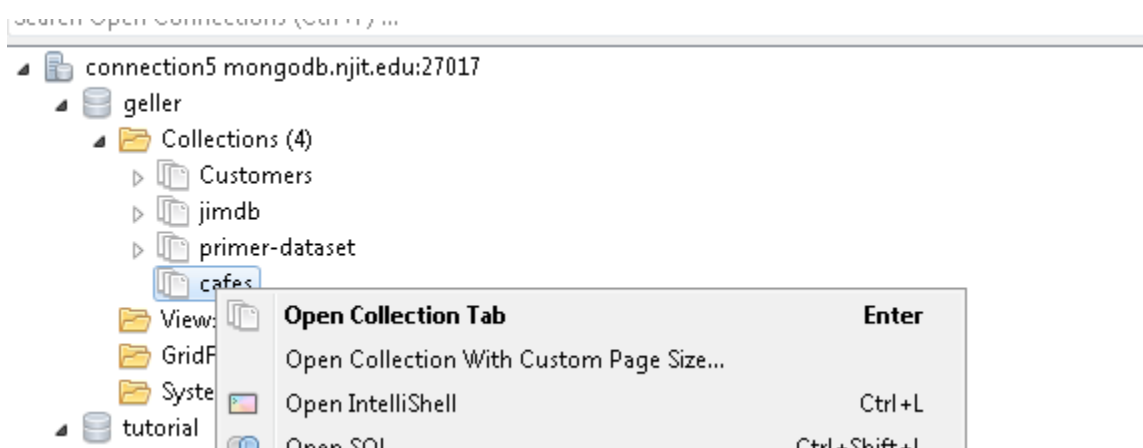
Type in a name. I used **cafes**.



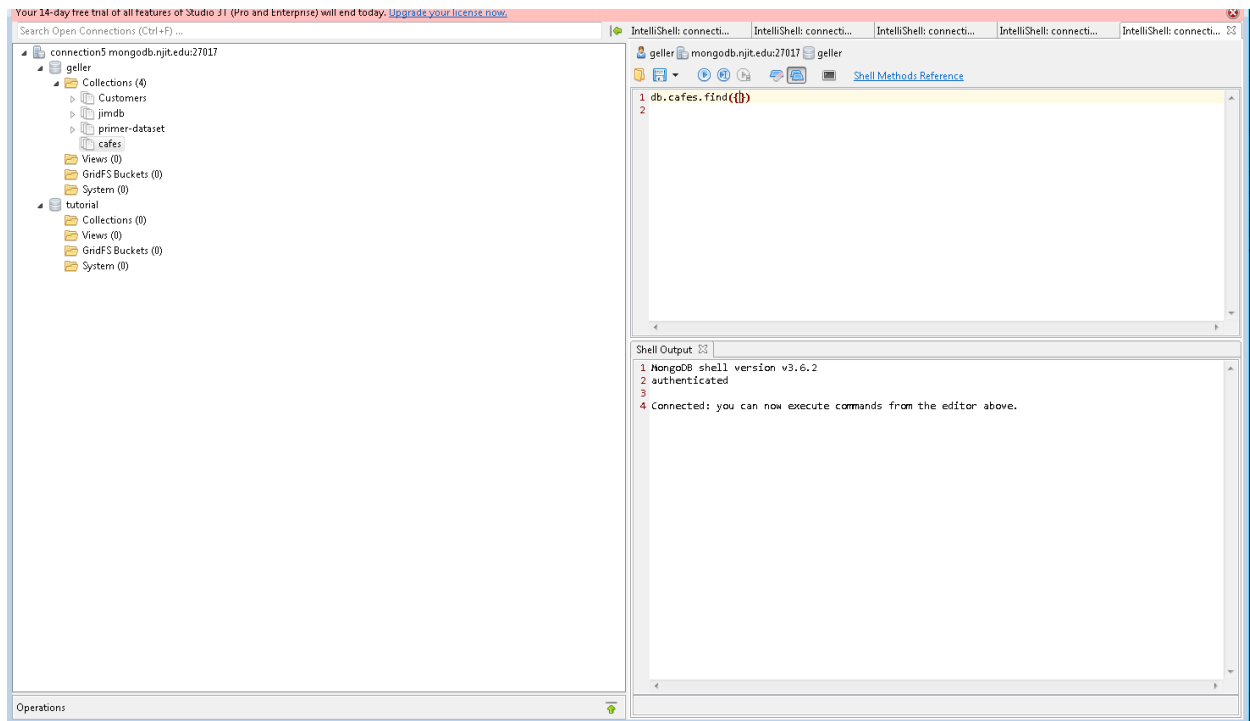
Click on Create.

Right click on cafes.

Click on Open in Intellishell



Now it should look like this. It will look a little simpler for you, because I have some older tabs already.



The first query is already in the query editor. It corresponds to a select * from cafes

However, your database is still empty.

Click on the second blue arrow, and you will see no data is returned.

➔ Now we need to insert some data into this collection.

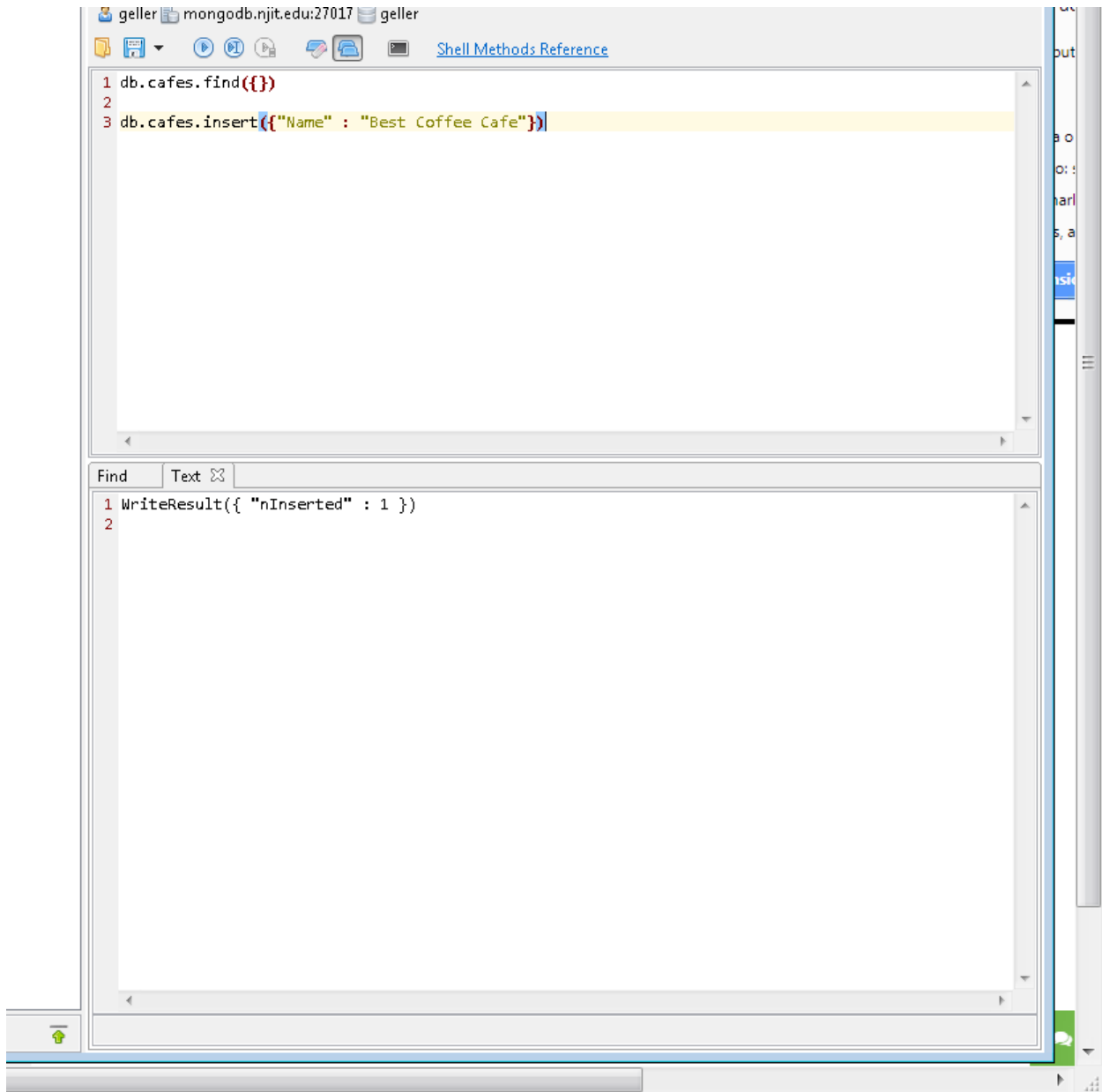
We start simple.

Type this

```
db.cafes.insert({"Name" : "Best Coffee Cafe"})
```

and hit the second blue arrow.

Note that keys are case sensitive, even when they are not enclosed in "" quotes.



Now move the mouse over the find statement and click the second right arrow.

The screenshot shows a MongoDB Shell window titled "Shell Methods Reference". The command prompt contains the following code:

```
1 db.cafes.find({})
2
3 db.cafes.insert({"Name" : "Best Coffee Cafe"})
```

Below the command prompt is a toolbar with navigation icons, a search field containing "50", and a "JSON View" dropdown menu. The output area displays the result of the insert operation as a JSON document:

```
1 {
2   "_id" : ObjectId("5ab293ec280bea0f5ca96904"),
3   "Name" : "Best Coffee Cafe"
4 }
5
```

Note that the system added an object ID `_id`

Adding one at a time is tiring. Of course you can import from a file. But we get to that later. For now we will do a multi-insert.

```
db.cafes.insertMany(
[{"Name" : "Dream Cafe"},
{"Name" : "My Favorite Cafe"},
{"Name" : "Cafe Europe"},
{"Name" : "French Flavor"}])
```

Click the second blue button.

Now move the mouse over the find statement and click the second right arrow.

You should see five simple documents.

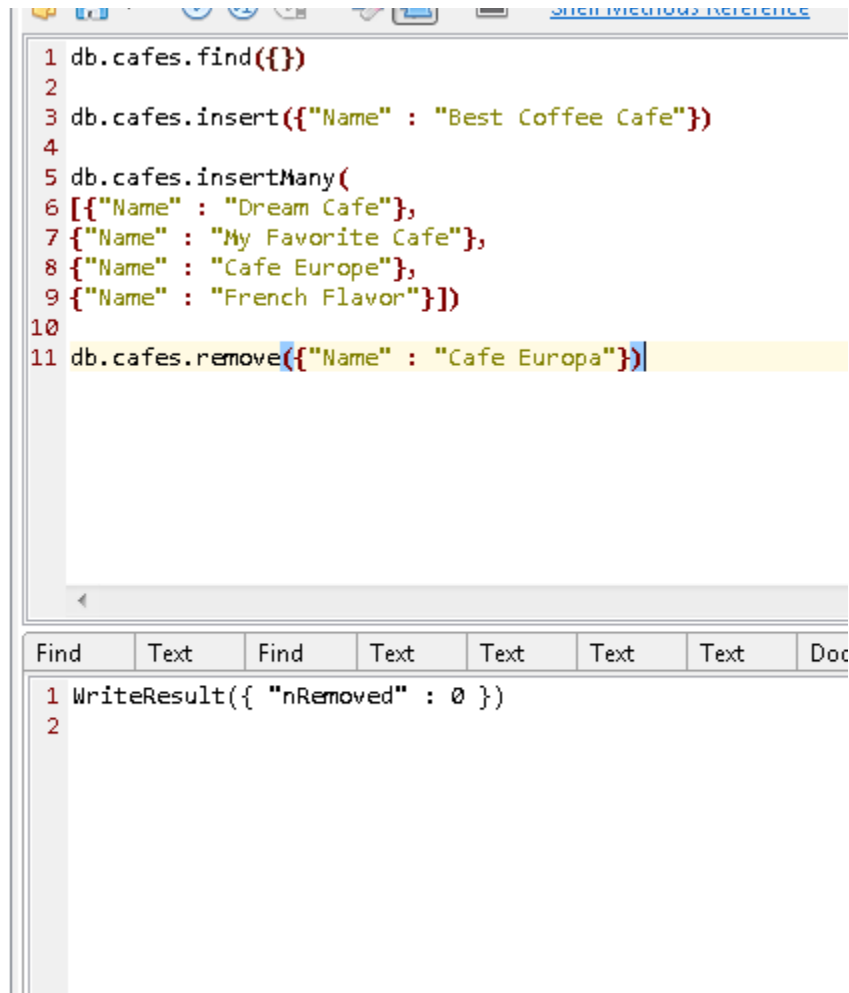
Completing the CRUD operations: We will do a simple delete and two simple updates.

➔ Simple delete

Unfortunately the Cafe Europa closed and we have to delete it.

```
db.cafes.remove({"Name" : "Cafe Europa"})
```

Second blue arrow.



```
1 db.cafes.find({})
2
3 db.cafes.insert({"Name" : "Best Coffee Cafe"})
4
5 db.cafes.insertMany(
6 [{"Name" : "Dream Cafe"},
7 {"Name" : "My Favorite Cafe"},
8 {"Name" : "Cafe Europa"},
9 {"Name" : "French Flavor"}])
10
11 db.cafes.remove({"Name" : "Cafe Europa"})
```

Find	Text	Find	Text	Text	Text	Text	Doc
1	WriteResult({ "nRemoved" : 0 })						
2							

It says it removed "0". That is because I mistyped Europe.

Let's try this one more time. This time correctly.

```
db.cafes.remove({"Name" : "Cafe Europe"})
```

Sec. Blu. Arr.

```
Shell Methods Referen
1 db.cafes.find({})
2
3 db.cafes.insert({"Name" : "Best Coffee Cafe"})
4
5 db.cafes.insertMany(
6 [{"Name" : "Dream Cafe"},
7 {"Name" : "My Favorite Cafe"},
8 {"Name" : "Cafe Europe"},
9 {"Name" : "French Flavor"}])
10
11 db.cafes.remove({"Name" : "Cafe Europe"})
```

Find	Text	Find	Text	Text	Text	Text
1	WriteResult({ "nRemoved" : 1 })					
2						

Now 1 was removed.

Move cursor to the first row.

Second blue arrow.

Cafe Europe is gone.

Now click on Count Documents on the bottom of the Studio 3T.

The screenshot shows the MongoDB Compass interface. The main area displays a JSON array of four documents, each representing a cafe. The documents are:

```
1 {
2   "_id" : ObjectId("5ab293ec280bea0f5ca96904"),
3   "Name" : "Best Coffee Cafe"
4 }
5 {
6   "_id" : ObjectId("5ab295c5280bea0f5ca96905"),
7   "Name" : "Dream Cafe"
8 }
9 {
10  "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
11  "Name" : "My Favorite Cafe"
12 }
13 {
14  "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
15  "Name" : "French Flavor"
16 }
17 }
```

At the bottom of the interface, there is a status bar with the text "0 documents selected" on the left and "Count Documents" on the right. A blue arrow points to the "Count Documents" button, which is currently showing a count of 4 and a time of 0.002s.

```
1 {
2   "_id" : ObjectId("5ab293ec280bea0f5ca96904"),
3   "Name" : "Best Coffee Cafe"
4 }
5 {
6   "_id" : ObjectId("5ab295c5280bea0f5ca96905"),
7   "Name" : "Dream Cafe"
8 }
9 {
10  "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
11  "Name" : "My Favorite Cafe"
12 }
13 {
14  "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
15  "Name" : "French Flavor"
16 }
17
```

4 documents | 0

It tells you that it found four documents.

Of course you can find this also with a command. We will do that later.

➔ Now we will do two simple updates.

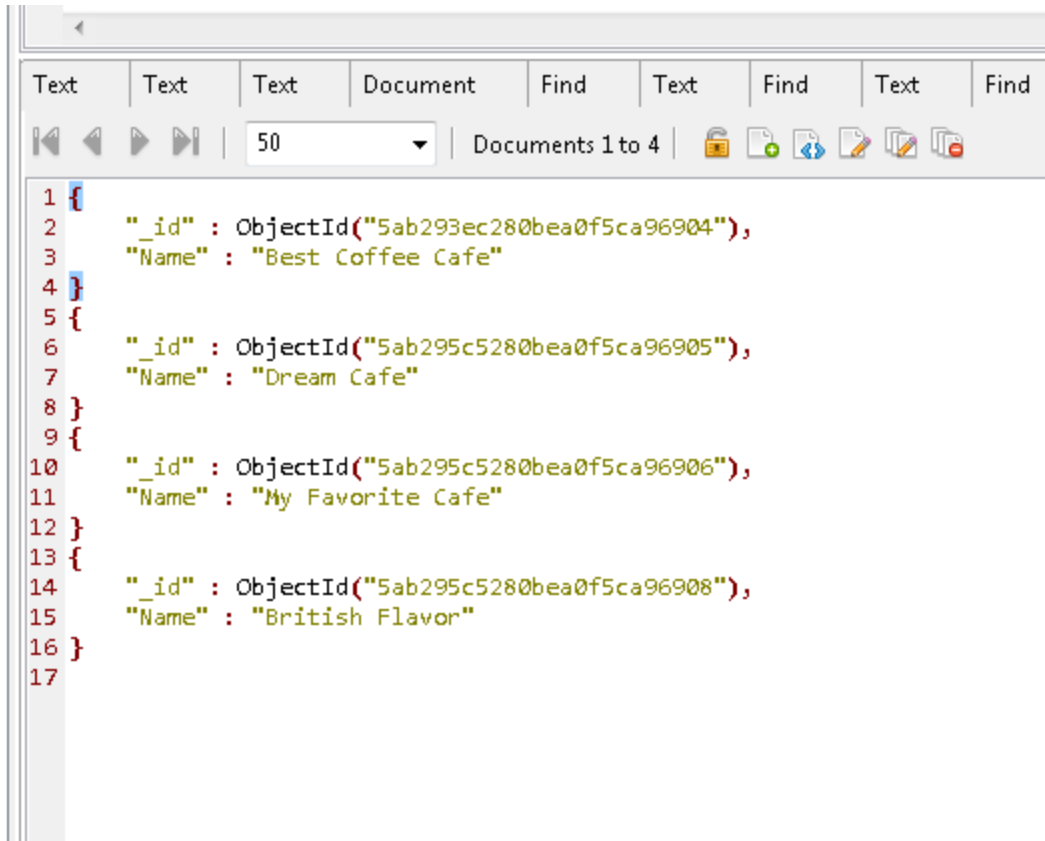
First, the French Flavor cafe was taken over by the British.

```
db.cafes.update({"Name" : "French Flavor"}, {$set: {"Name" : "British Flavor"}})
```

```
12
13 db.cafes.update({"Name" : "French Flavor"}, {$set: {"Name" : "British Flavor"}})
```

Text	Text	Text	Text	Document	Find	Text	Find	Text	Find	Fin
1	WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })									
2										

As usual we check the whole database to make sure this really happened.



The image shows a code editor window with a toolbar at the top. The toolbar includes buttons for 'Text', 'Document', 'Find', and 'Text', along with a search input field containing '50' and a dropdown menu. Below the toolbar, the editor displays a JSON array of four cafe documents. The documents are as follows:

```
1 {
2   "_id" : ObjectId("5ab293ec280bea0f5ca96904"),
3   "Name" : "Best Coffee Cafe"
4 }
5 {
6   "_id" : ObjectId("5ab295c5280bea0f5ca96905"),
7   "Name" : "Dream Cafe"
8 }
9 {
10  "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
11  "Name" : "My Favorite Cafe"
12 }
13 {
14  "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
15  "Name" : "British Flavor"
16 }
17
```


Now we will do a more sophisticated update. Remember there is no schema. So we can make the structure more complicated by an update.

We are going to add a style key:value pair for the Dream Cafe.

```
db.cafes.update({"Name" : "Dream Cafe"}, {$set: {"Style" : "Bistro"}})
```

The result:

```
1 {
2   "_id" : ObjectId("5ab293ec280bea0f5ca96904"),
3   "Name" : "Best Coffee Cafe"
4 }
5 {
6   "_id" : ObjectId("5ab295c5280bea0f5ca96905"),
7   "Name" : "Dream Cafe",
8   "Style" : "Bistro"
9 }
10 {
11   "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
12   "Name" : "My Favorite Cafe"
13 }
14 {
15   "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
16   "Name" : "British Flavor"
17 }
18
```



We can even do this with a nested document.

So let's give the Dream Cafe a basic address first.

```
db.cafes.update({"Name" : "Dream Cafe"},
{$set: {"Address" : {"City" : "New York", "State" : "New York"}}})
```

```
4 }
5 {
6   "_id" : ObjectId("5ab295c5280bea0f5ca96905"),
7   "Name" : "Dream Cafe",
8   "Style" : "Bistro",
9   "Address" : {
10     "City" : "New York",
11     "State" : "New York"
12   }
13 }
14 {
15   "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
16   "Name" : "My Favorite Cafe"
17 }
18 {
19   "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
20   "Name" : "British Flavor"
21 }
```

But we would like to have a street! So we need to do one more update.

```
db.cafes.update({"Name" : "Dream Cafe"}, {$set: {"Address" : {"Number" : "255", "Street": "Fifth Ave.", "City" : "New York", "State" : "New York"}}})
```

```
1 {
  "_id" : ObjectId("5ab293ec280bea0f5ca96904"),
  "Name" : "Best Coffee Cafe"
}
{
  "_id" : ObjectId("5ab295c5280bea0f5ca96905"),
  "Name" : "Dream Cafe",
  "Style" : "Bistro",
  "Address" : {
    "Number" : "255",
    "Street" : "Fifth Ave.",
    "City" : "New York",
    "State" : "New York"
  }
}
{
  "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
  "Name" : "My Favorite Cafe"
}
{
  "_id" : ObjectId("5ab295c5280bea0f5ca96908")
}
```

With this we have covered the basic CRUD operations.

However... JSON/MongoDB also work with arrays. We need to do the CRUD operations now for arrays.

➔ Array Insert.

A cafe offers different kinds of coffee.

For simplicity we create a new cafe.

```
db.cafes.insert({"Name" : "Morning Glory",
"Drinks" : ["Cappuccino", "Espresso", "Mocha"]})
```



```
20 {
21   "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
22   "Name" : "British Flavor"
23 }
24 {
25   "_id" : ObjectId("5ab29f91280bea0f5ca96909"),
26   "Name" : "Morning Glory",
27   "Drinks" : [
28     "Cappuccino",
29     "Espresso",
30     "Mocha"
31   ]
32 }
33
```

1 document selected

Now we want to

- ➔ Find all the drinks offered at Cafe Morning Glory

For this we have to refine the find() command by providing a "projection" like in SQL.

```
db.cafes.find({"Name" : "Morning Glory"}, {"Drinks" : 1})
```

The "1" means TRUE. Return this.

```
26
27 db.cafes.find({})
28
29 db.cafes.find({"Name" : "Morning Glory"}, {"Drinks" : 1})
30
```

Text Find Text Find Text Find Text Find Text

50 Documents 1 to 1

```
1 {
2   "_id" : ObjectId("5ab29f91280bea0f5ca96909"),
3   "Drinks" : [
4     "Cappuccino",
5     "Espresso",
6     "Mocha"
7   ]
8 }
9
```

Note that the `_id` is returned even though I did not ask for it.

I can suppress the `_id`:

```
db.cafes.find({"Name" : "Morning Glory"}, {"Drinks" : 1, "_id" : 0})
```

```
--
27 db.cafes.find({})
28
29 db.cafes.find({"Name" : "Morning Glory"}, {"Drinks" : 1, "_id" : 0})
30
```

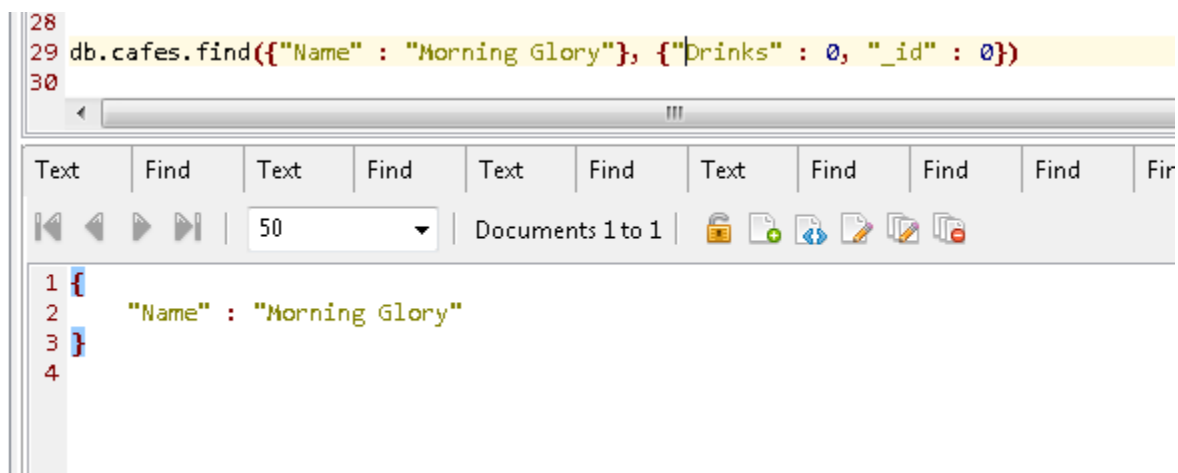


```
1 {
2   "Drinks" : [
3     "Cappuccino",
4     "Espresso",
5     "Mocha"
6   ]
7 }
8
```

Not surprisingly, 0 means FALSE.

Something surprising happens if I set Drinks to 0.

```
28
29 db.cafes.find({"Name" : "Morning Glory"}, {"Drinks" : 0, "_id" : 0})
30
```

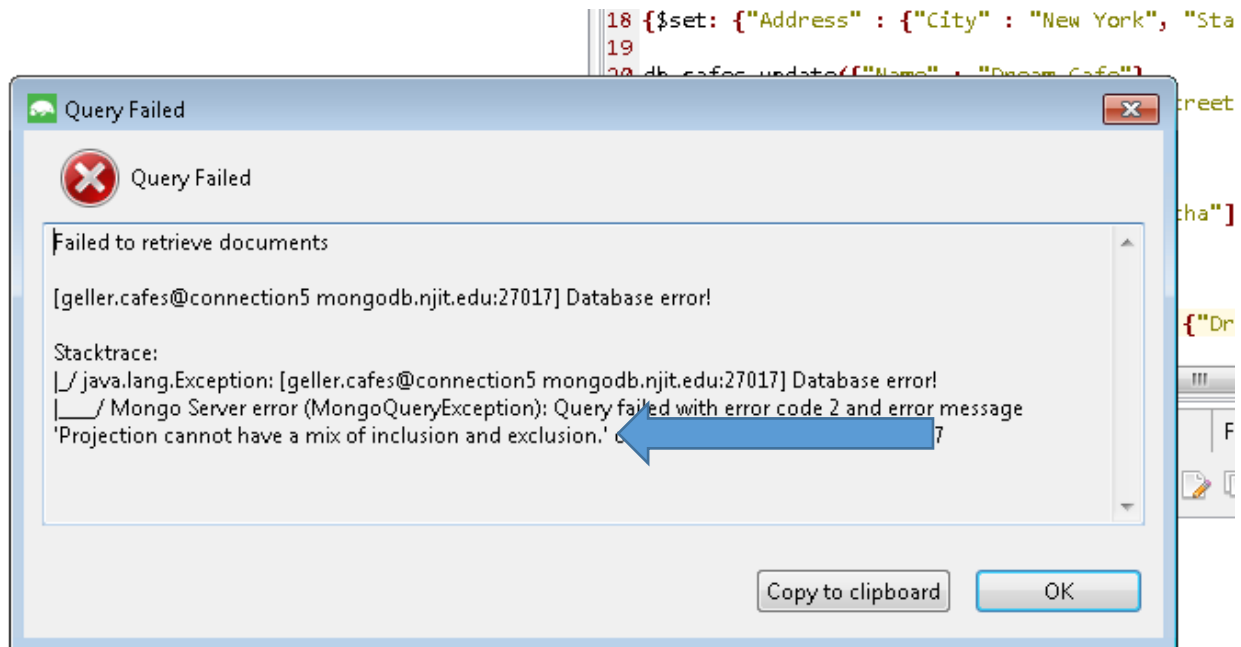


```
1 {
2   "Name" : "Morning Glory"
3 }
4
```

I DID NOT ASK FOR Name !!!

Even more surprising is this error message:

```
db.cafes.find({"Name" : "Morning Glory"}, {"Drinks" : 0, "_id" : 1})
```



It appears that you cannot mix "0" and "1" in one single projection operation!!!

➔ Remove all the Drinks (and recreate the drink list for further use)

Next we want to remove the drink list completely.

Let's start with a warning:

`db.cafes.remove({})` REMOVES ALL DATA FROM cafes

`db.cafes.drop()` REMOVES EVERYTHING

We don't want to do those things!!!

We want to remove one key/value pair only.

It does not have to be an array. It just happens to be an array.

```
db.cafes.update({"Name" : "Morning Glory"},  
{$unset: {"Drinks" : 1}})
```

```
1  }
2  "Name" : "British Flavor"
3  }
4  {
5  "_id" : ObjectId("5ab29f91280bea0f5ca96909"),
6  "Name" : "Morning Glory"
7  }
8
```

We need to recreate the previous structure now to continue with experimenting.

```
db.cafes.update({"Name" : "Morning Glory"},
{$set: {"Drinks" : ["Cappuccino", "Espresso", "Mocha"]}})
```

➔ Remove only one Drink (and recreate the drink list)

This is ambiguous. Do we want remove a specific drink? Or do we want to remove the first drink in the list?

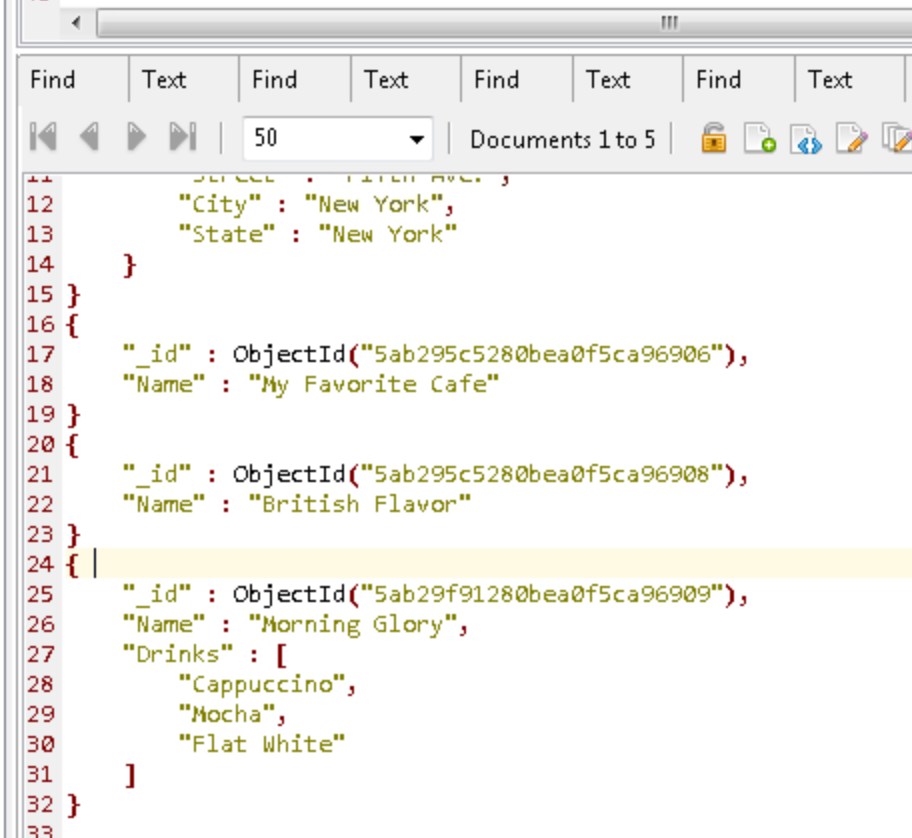
Remove a specific drink from a list:

```
db.cafes.update({"Name" : "Morning Glory"},
{$pull: {"Drinks" : {$in: ["Espresso"]}}})
```

```

39
40 db.cafes.update({"Name" : "Morning Glory"},
41 {$pull: {"Drinks" : {$in: ["Espresso"]}, }})
42
43

```



```

11
12   "City" : "New York",
13   "State" : "New York"
14 }
15 }
16 {
17   "_id" : ObjectId("5ab295c5280bea0f5ca96906"),
18   "Name" : "My Favorite Cafe"
19 }
20 {
21   "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
22   "Name" : "British Flavor"
23 }
24 {
25   "_id" : ObjectId("5ab29f91280bea0f5ca96909"),
26   "Name" : "Morning Glory",
27   "Drinks" : [
28     "Cappuccino",
29     "Mocha",
30     "Flat White"
31   ]
32 }
33

```

While doing this another surprise: Some error messages appear in the result field.

Some pop out in their own window.

And some errors cause no error message at all.

And there was a wrong example in the documentation. There were no [] in the example.

➔ Remove the first drink from the list:

Now we remove the first drink from the list.

```

db.cafes.update({"Name" : "Morning Glory"},
{$pop : {"Drinks" : -1 }})

```

```
19 }
20 {
21   "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
22   "Name" : "British Flavor"
23 }
24 {
25   "_id" : ObjectId("5ab29f91280bea0f5ca96909"),
26   "Name" : "Morning Glory",
27   "Drinks" : [
28     "Mocha",
29     "Flat White"
30   ]
31 }
32
```

1 document selected

Note that Cappuccino is gone!

To \$pop from the end of a list you need to do

: 1

instead of

: -1

➔ Add one more Drink to the end of the list.

I recreated the original list of Drinks. This is not shown here. I put "Flat White" at the end.

```
db.cafe.update({"Name" : "Morning Glory"},
{$push: {"Drinks" : "Flat White"}})
```

I typed cafe instead of cafes. NO ERROR MESSAGE. Now correct:

```
db.cafe.update({"Name" : "Morning Glory"},
{$push: {"Drinks" : "Flat White"}})
```

```
18   "Name" : "My Favorite Cafe"
19 }
20 {
21   "_id" : ObjectId("5ab295c5280bea0f5ca96908"),
22   "Name" : "British Flavor"
23 }
24 {
25   "_id" : ObjectId("5ab29f91280bea0f5ca96909"),
26   "Name" : "Morning Glory",
27   "Drinks" : [
28     "Cappuccino",
29     "Espresso",
30     "Mocha",
31     "Flat White"
32   ]
33 }
34
```

1 document selected

→ Simplest aggregate.

How many documents are in the database?

```
db.cafes.find({}).count()
```

```
45
46 db.cafes.find({}).count()
47
48
49
```

Text	Find	Text	Text	Text	Text	Text
1	5					
2						

Previously we found this in the user interface.

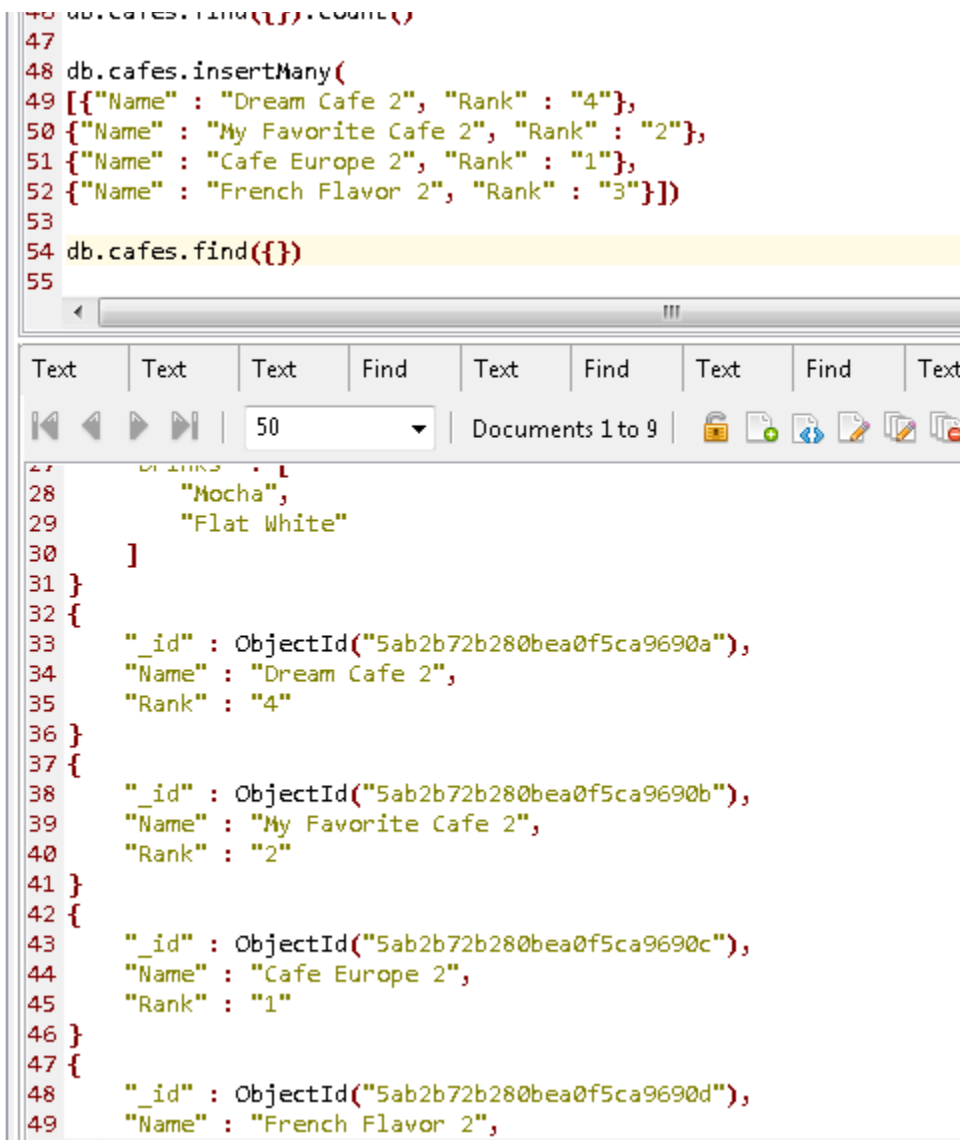
→ Sorting data.

For this we need some new data.

```
db.cafes.insertMany(
[{"Name" : "Dream Cafe 2", "Rank" : "4"},
```

```
{"Name" : "My Favorite Cafe 2", "Rank" : "2"},  
{"Name" : "Cafe Europe 2", "Rank" : "1"},  
{"Name" : "French Flavor 2", "Rank" : "3"}])
```

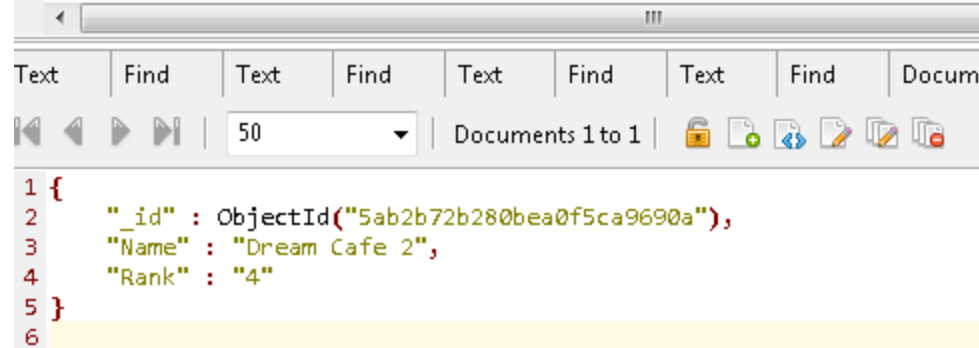
```
46 db.cafes.find({}).count()  
47  
48 db.cafes.insertMany(  
49 [{"Name" : "Dream Cafe 2", "Rank" : "4"},  
50 {"Name" : "My Favorite Cafe 2", "Rank" : "2"},  
51 {"Name" : "Cafe Europe 2", "Rank" : "1"},  
52 {"Name" : "French Flavor 2", "Rank" : "3"}])  
53  
54 db.cafes.find({})  
55
```



```
28     "Mocha",  
29     "Flat White"  
30 ]  
31 }  
32 {  
33   "_id" : ObjectId("5ab2b72b280bea0f5ca9690a"),  
34   "Name" : "Dream Cafe 2",  
35   "Rank" : "4"  
36 }  
37 {  
38   "_id" : ObjectId("5ab2b72b280bea0f5ca9690b"),  
39   "Name" : "My Favorite Cafe 2",  
40   "Rank" : "2"  
41 }  
42 {  
43   "_id" : ObjectId("5ab2b72b280bea0f5ca9690c"),  
44   "Name" : "Cafe Europe 2",  
45   "Rank" : "1"  
46 }  
47 {  
48   "_id" : ObjectId("5ab2b72b280bea0f5ca9690d"),  
49   "Name" : "French Flavor 2",
```

Just making sure that this worked:


```
56 db.cafes.find({"Rank" : "4"})
57 |
58
59
```



```
1 {
2   "_id" : ObjectId("5ab2b72b280bea0f5ca9690a"),
3   "Name" : "Dream Cafe 2",
4   "Rank" : "4"
5 }
6
```

Well, that showed us only one element.

Let's look for all of them:

```
db.cafes.find({"Rank" : {$in: ["1", "2", "3", "4"]}})
```

We are finally getting to the actual sorting.

```
db.cafes.find().sort( {Rank : -1})
```

This command sorts in DESCENDING ORDER

```
db.cafes.find().sort( {Rank : 1})
```

```
51 }
52 {
53   "_id" : ObjectId("5ab2b72b280bea0f5ca9690c"),
54   "Name" : "Cafe Europe 2",
55   "Rank" : "1"
56 }
57 {
58   "_id" : ObjectId("5ab2b72b280bea0f5ca9690b"),
59   "Name" : "My Favorite Cafe 2",
60   "Rank" : "2"
61 }
62 {
63   "_id" : ObjectId("5ab2b72b280bea0f5ca9690d"),
64   "Name" : "French Flavor 2",
65   "Rank" : "3"
66 }
67 {
68   "_id" : ObjectId("5ab2b72b280bea0f5ca9690a"),
69   "Name" : "Dream Cafe 2",
70   "Rank" : "4"
71 }
72
```

1 document selected | Count D

This command sorts in ASCENDING ORDER

I AM CHEATING on the above output.

There's one big problem: All the above sort even elements that have no Rank at all.

I just did not show the complete output!

So we need to say we want only things with Rank.

But for that we have to have pairs

Rank: SOMETHING

And that requires that we use regular expressions.

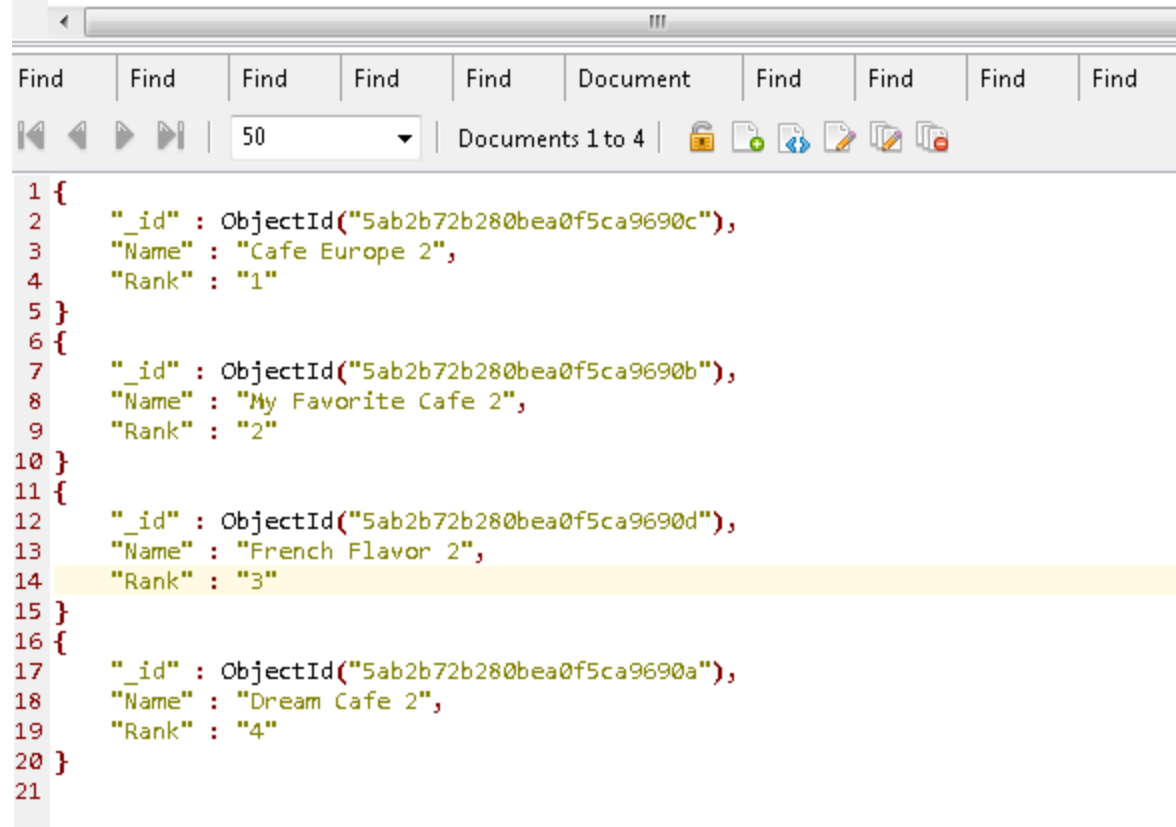
Remember regular expressions?

So this works!

```
db.cafes.find({"Rank" : {$regex : /[0-9]/}}).sort( {Rank : 1})
```

This means: Find me anything that has a Rank, and I don't know what the Rank is, as long as it is a number between 0 and 9. And after you find these, sort them in ascending order.

```
>/
58 db.cafes.find({"Rank" : {$regex : /[0-9]/}}).sort( {Rank : 1})
59
```



```
1 {
2   "_id" : ObjectId("5ab2b72b280bea0f5ca9690c"),
3   "Name" : "Cafe Europe 2",
4   "Rank" : "1"
5 }
6 {
7   "_id" : ObjectId("5ab2b72b280bea0f5ca9690b"),
8   "Name" : "My Favorite Cafe 2",
9   "Rank" : "2"
10 }
11 {
12   "_id" : ObjectId("5ab2b72b280bea0f5ca9690d"),
13   "Name" : "French Flavor 2",
14   "Rank" : "3"
15 }
16 {
17   "_id" : ObjectId("5ab2b72b280bea0f5ca9690a"),
18   "Name" : "Dream Cafe 2",
19   "Rank" : "4"
20 }
21
```

And this time I was not cheating.

ONLY these four elements were returned.

```
55
56 db.cafes.find({"Rank" : {$in: ["1", "2", "3", "4"]}})
57
58
59
```

```
1 {
2   "_id" : ObjectId("5ab2b72b280bea0f5ca9690a"),
3   "Name" : "Dream Cafe 2",
4   "Rank" : "4"
5 }
6 {
7   "_id" : ObjectId("5ab2b72b280bea0f5ca9690b"),
8   "Name" : "My Favorite Cafe 2",
9   "Rank" : "2"
10 }
11 {
12   "_id" : ObjectId("5ab2b72b280bea0f5ca9690c"),
13   "Name" : "Cafe Europe 2",
14   "Rank" : "1"
15 }
16 {
17   "_id" : ObjectId("5ab2b72b280bea0f5ca9690d"),
18   "Name" : "French Flavor 2",
19   "Rank" : "3"
20 }
21
```

Note that this query did NOT work with [1, 2, 3, 4] because the numbers are given as strings!

So "1" is not 1.

Let's experiment with number data.

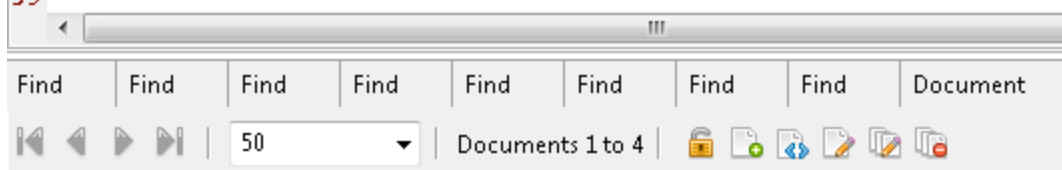
```
db.cafes.insertMany(
[{"Name" : "Dream Cafe 3", "Rank" : 4},
{"Name" : "My Favorite Cafe 3", "Rank" : 2},
{"Name" : "Cafe Europe 3", "Rank" : 1},
{"Name" : "French Flavor 3", "Rank" : 3}])
```

```
db.cafes.find({"Rank" : {$in: [1, 2, 3, 4]}})
```

```

56 db.cafes.find({"Rank" : {$in: [1, 2, 3, 4]}})
57
58
59

```



```

1 {
2   "_id" : ObjectId("5ab2ba07280bea0f5ca9690e"),
3   "Name" : "Dream Cafe 3",
4   "Rank" : 4.0
5 }
6 {
7   "_id" : ObjectId("5ab2ba07280bea0f5ca9690f"),
8   "Name" : "My Favorite Cafe 3",
9   "Rank" : 2.0
10 }
11 {
12   "_id" : ObjectId("5ab2ba07280bea0f5ca96910"),
13   "Name" : "Cafe Europe 3",
14   "Rank" : 1.0
15 }
16 {
17   "_id" : ObjectId("5ab2ba07280bea0f5ca96911"),
18   "Name" : "French Flavor 3",
19   "Rank" : 3.0
20 }
21

```

➔ What if I want ALL documents that have a "Rank" and I don't care at all what the value is?

```
db.cafes.find({"Rank" : {$exists : 1}})
```

or

```
db.cafes.find({"Rank" : {$exists : true}})
```

This will return the ranks as numbers and the ranks as strings.

The window is too small to copy all of them so I am doing a copy and paste of the content here:

```

{
  "_id" : ObjectId("5ab2b72b280bea0f5ca9690a"),
  "Name" : "Dream Cafe 2",
  "Rank" : "4"
}
{
  "_id" : ObjectId("5ab2b72b280bea0f5ca9690b"),
  "Name" : "My Favorite Cafe 2",
  "Rank" : "2"
}
{

```

```

    "_id" : ObjectId("5ab2b72b280bea0f5ca9690c"),
    "Name" : "Cafe Europe 2",
    "Rank" : "1"
  }
  {
    "_id" : ObjectId("5ab2b72b280bea0f5ca9690d"),
    "Name" : "French Flavor 2",
    "Rank" : "3"
  }
  {
    "_id" : ObjectId("5ab2ba07280bea0f5ca9690e"),
    "Name" : "Dream Cafe 3",
    "Rank" : 4.0
  }
  {
    "_id" : ObjectId("5ab2ba07280bea0f5ca9690f"),
    "Name" : "My Favorite Cafe 3",
    "Rank" : 2.0
  }
  {
    "_id" : ObjectId("5ab2ba07280bea0f5ca96910"),
    "Name" : "Cafe Europe 3",
    "Rank" : 1.0
  }
  {
    "_id" : ObjectId("5ab2ba07280bea0f5ca96911"),
    "Name" : "French Flavor 3",
    "Rank" : 3.0
  }
}

```

➔ EXPERIMENT WITH MAPREDUCE

Mapreduce Figure for explanation:

The basis of the figure is from here, but I added missing details!

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
}
orders
```

query

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
}
orders
```

Note: A123 is a VALUE here.

map

```
{ "A123": [ 500, 250 ] }
{ "B212": 200 }
```

Really there are two steps here.
The MAP function that **you** write gives you
{A123, 500}
{A123, 250}
{B212, 200}
Then the system, transparently to you, creates
{A123, [500, 250]}
{B212, [200]}

Note: A123 is a KEY here.

reduce

```
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
}
order_totals
```

Note: A123 is again a VALUE here.

Map Reduce Example from

docs.mongodb.com/manual/tutorial/map-reduce-examples/

```
db.skudata.insertMany(
  [{cust_id: "abc123",
    status: 'A',
    price: 25,
    items: [ {"sku" : "mmm", "qty" : 5 , "price" : 2.5 },
              {"sku" : "nnn", "qty" : 5, "price" : 2.5} ]}],
  {cust_id: "def456",
    status: 'A',
    price: 30,
    items: [ {"sku" : "mmm", "qty" : 6 , "price" : 3.0 },
              {"sku" : "nnn", "qty" : 4, "price" : 3.0} ]}],
  {cust_id: "abc123",
    status: 'A',
    price: 40,
    items: [ {"sku" : "mmm", "qty" : 10 , "price" : 1 },
              {"sku" : "nnn", "qty" : 10, "price" : 3} ]}]})
```

```
var mapFunction1 = function() {
    emit(this.cust_id, this.price);
};
```

```
var reduceFunction1 = function(keyCustId, valuesPrices) {
    return Array.sum(valuesPrices);
};
```

There are strict rules how the above two functions have to look. The use of “this” the parameter lists and the use of “emit” all have to be exactly like this. Array is a data type with an operation sum() that adds all the elements in the array.

The following call uses the above two functions and determines where the result goes to. This invokes the hidden machinery of MapReduce.

```
db.skudata.mapReduce( mapFunction1, reduceFunction1, {out: "mapreduceexample" } )
```

Now do a Refresh All on your connection.

You will see a new collection: mapreduceexample.

The screenshot shows a MongoDB shell window with the following content:

```
1 db.mapreduceexample.find({})
2
```

Below the shell window is a 'Find' toolbar with navigation icons, a limit of 50, and 'Documents 1 to 2'. The results are displayed as follows:

```
1 {
2   "_id" : "abc123",
3   "value" : 65.0
4 }
5 {
6   "_id" : "def456",
7   "value" : 30.0
8 }
9
```

Customer abc123 had two orders with prices: 25 and 40, which correctly adds up to 65.
Customer def456 had one order with price: 30.

Note that the prices that I used are not exactly the prices from the Mongo web site.

➔ Mapreduce Example 2

Note that the count is always "1". We will be counting how many times we went through the loop.

```
var mapFunction2 = function () {
  for (var idx = 0; idx < this.items.length; idx++) {
    var key = this.items[idx].sku;
    var value = {count: 1, qty: this.items[idx].qty };
    emit(key, value);
  }
}
```

```

    }
};

var reduceFunction2 = function(keySKU, countObjVals) {
  reduceVal = {count: 0, qty: 0};
  for (var idx = 0; idx < countObjVals.length; idx++){
    reduceVal.count += countObjVals[idx].count;
    reduceVal.qty += countObjVals[idx].qty;      # This adds 1 for each order
  }
  return reduceVal;
};

```

We now add one more optional MapReduce function called “finalize.”
Our reduceFunction2 computed the total number (quantity) and counted the items.

Below we divide the total number (quantity – qty) by the count to get the average size of each order.

```

var finalizeFunction2 = function (key, reducedVal) {
  reducedVal.avg = reducedVal.qty/reducedVal.count;
  return reducedVal;
};

```

And below this starts the MapReduce machinery again.
We use “merge” as we include the results into the existing output document. Otherwise we would “overwrite” and wipe out data.

```

db.skudata.mapReduce( mapFunction2,
  reduceFunction2, { out: { merge: "mapreduceexample2" },
                    finalize: finalizeFunction2 }
)

```

➔ Mapreduce Example 3 (This is the simplest example).

Implement the MAX example. Who has the highest salary?

```

db.numdata.insertMany(
  [{personid : "person1", salary : 20000},
  {personid : "person2", salary : 50000},
  {personid : "person3", salary : 40000}])

```

The above should be mapped to:

```
{data : {personid : "person1", salary : 20000}}
{data : {personid : "person2", salary : 50000}}
{data : {personid : "person3", salary : 40000}}
```

That is achieved by the mapping function below.

The function does not take any arguments and generates pairs from the current JSON document. The predefined function emit() sends the pairs out. The object **this** refers to the current JSON document and all of them are processed.

```
var mapFunction3 = function() {
  emit("data", this);
};
```

arrayOfPersons = [] // I should not need that. But I think it helped.

Codes is written JavaScript

```
var reduceFunction3 = function(data, arrayOfPersons) {
  maxPair = {person: "none", max : 0};
  for (var idx = 0; idx < arrayOfPersons.length; idx++) {
    if (arrayOfPersons[idx].salary > maxPair.max)
      {maxPair.max = arrayOfPersons[idx].salary;
       maxPair.person = arrayOfPersons[idx].personid;}
  }
  return maxPair;
};
```

Now we are starting the whole MapReduce process, with our own Map function, our own Reduce function and a document that describes where we want the output to go (and how it should be processed).

We create a new collection called "findmax".

```
db.numdata.mapReduce( mapFunction3,
                      reduceFunction3,
                      {out: "findmax" }
)
```

➔ COMPARISON AND LOGICAL OPERATORS

A simple comparison operation does not look the way you may expect.

Note that the \$lt is between Rank and 2. This is an “infix operator” if you ignore the { }

```
db.cafes.find( { Rank: { $lt: 2 } });
```

This means

WHERE Rank < 2

But OR is a prefix operation:

```
db.cafes.find( { $or: [ {Rank: 2}, {Rank: { $gt: 2}} ] } )
```

And note that keys are case sensitive, even if they are not in quotes.

➔ NOW WE WANT TO WORK WITH A REAL, BIG DATA SET

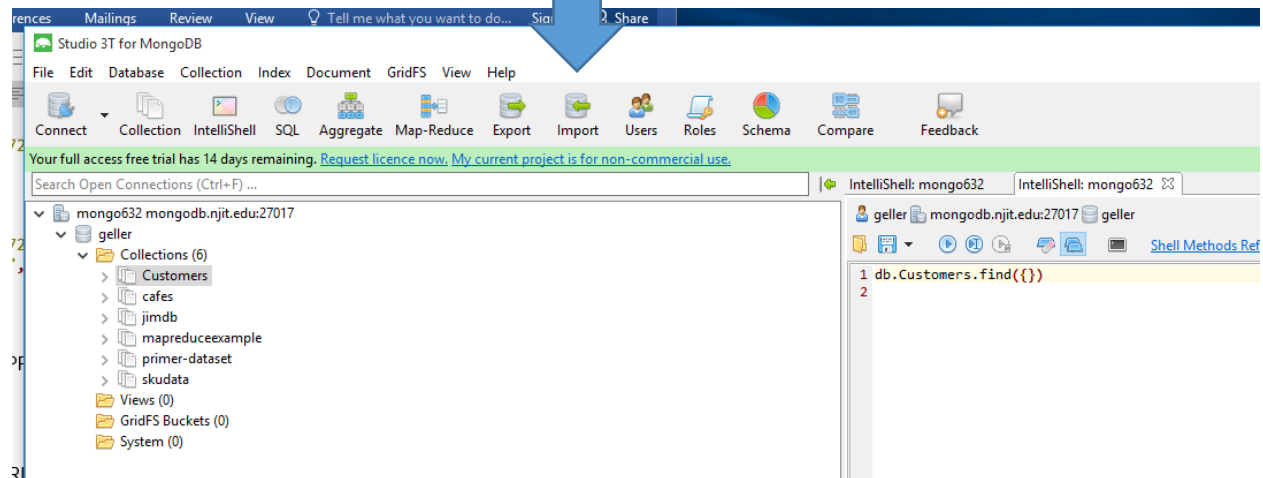
Go to (below) to download the customers data set.

<https://www.dropbox.com/s/f0crtay1kb7zhe1/Customers.json?dl=0>

It will complain that it is too large to open, but eventually I was able to download it.

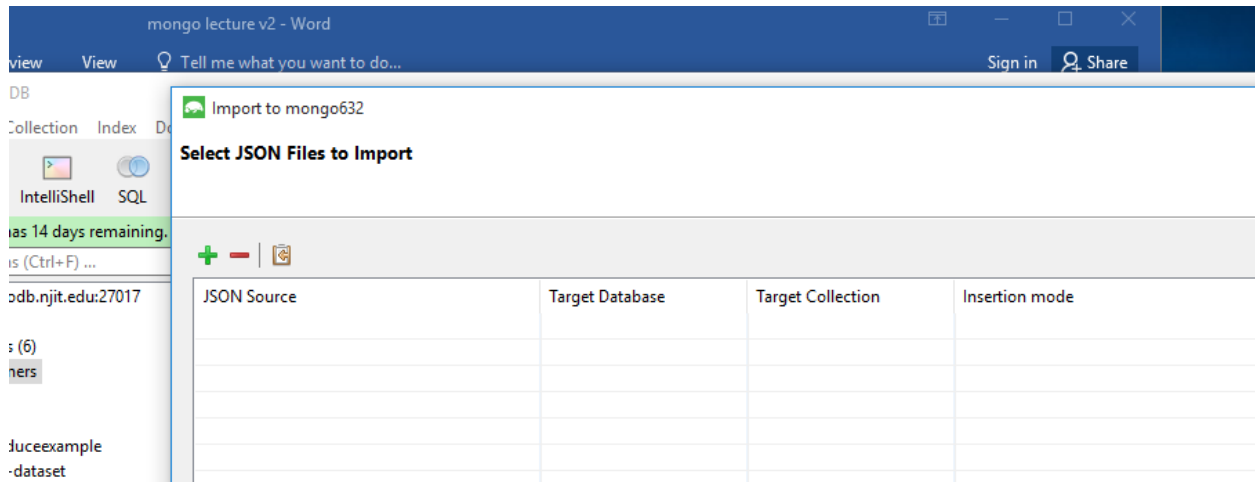
(You might need a free Dropbox account.... No problem. I have one.)

Import the collection.

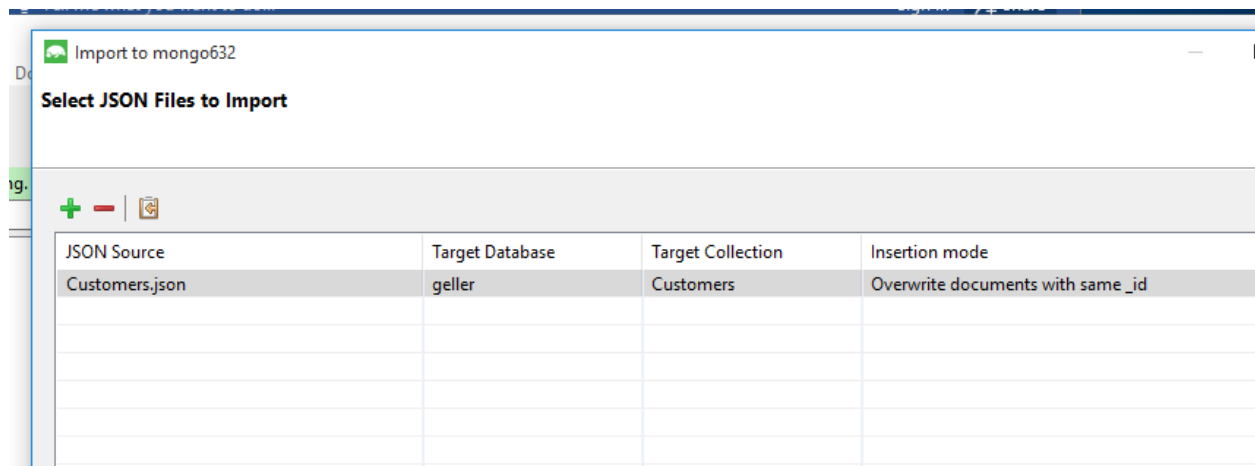


Import in JSON Format.

Click on **+**



Navigate to the place where you saved the download file. (Better to move it into a different directory than Download.)



Click Next (as often as needed)

Click Start Import

Now do this:

```
db.Customers.find({})
```

This shows 3725 rows of data. Presumably the whole database.

```
db.Customers.find({}).limit(1)
```

This shows 1 (the first) element. 74 rows.

```
db.Customers.find({}).limit(2)
```

shows 2 elements as expected.

```
db.Customers.find({}).count()
```

returns 70000

```
db.Customers.find({"Name.Last Name" : "Johnston"},
```

```
 {"Name.First Name" : NumberInt(1),
```

```
 "Name.Last Name" : NumberInt(1)}
```

Returns first name and last name for all Johnstons.

The above shows the `_id` ALSO.

To get rid of this, write:

```
db.Customers.find({"Name.Last Name" : "Johnston"},
```

```
 {"Name.First Name" : NumberInt(1),
```

```
 "Name.Last Name" : NumberInt(1)
```

```
 "_id" : NumberInt(0)})
```

Now we will sort all Johnston's by first name.

```
db.Customers.find({"Name.Last Name" : "Johnston"},
```

```
 {"Name.First Name" : NumberInt(1),
```

```
 "Name.Last Name" : NumberInt(1)
```

```
 "_id" : NumberInt(0))).sort({"Name.First Name" : NumberInt(1)})
```