**3. Saving Data – Commands**

Contents

[1. Collections 1](#_Toc130131158)

[2. ObjectId 4](#_Toc130131159)

[3. Insert 5](#_Toc130131160)

[4. Insert with Id 6](#_Toc130131161)

[5. Complex Document 7](#_Toc130131162)

[6. Save Danger 8](#_Toc130131163)

[7. Update Command 9](#_Toc130131164)

[8. Set Operator 11](#_Toc130131165)

[9. Unset Operator 12](#_Toc130131166)

[10. Rename Operator 13](#_Toc130131167)

[11. Push Operator 14](#_Toc130131168)

[12. Pull Operator 17](#_Toc130131169)

[13. Pop Operator 18](#_Toc130131170)

[14. Array Type 20](#_Toc130131171)

[15. Multi Update 20](#_Toc130131172)

[16. Find And Modify 23](#_Toc130131173)

[17. Query With Sort 23](#_Toc130131174)

[18. Documentation 28](#_Toc130131175)

# Collections

Show current database and collections

test> db

test

test> show collections

Insert a document

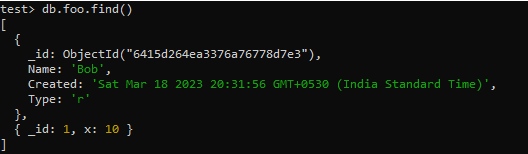
==========================

db.foo.insertOne({\_id:1, x:10})

{ acknowledged: true, insertedId: 1}

Find documents

db.foo.find()



test> db.bar.insert({\_id:1, x:10})





**MongoDB 3.0 deprecates direct access to the system.indexes collection.**

List all indexes on all collections in a database

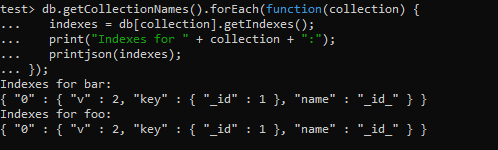
db.getCollectionNames().forEach(**function**(collection) {

indexes = db[collection].getIndexes();

print("Indexes for " + collection + ":");

printjson(indexes);

});



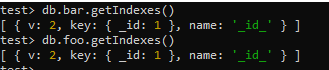
List of all indexes on a collection

================================

[db.collection.getIndexes()](https://mongoing.com/docs/reference/method/db.collection.getIndexes.html#db.collection.getIndexes)

db.bar.getIndexes()

db.foo.getIndexes()



DataTypes supported by \_id field

=========================================

db.foo.insertOne({\_id:1})

db.foo. insertOne ({\_id:3.14})

db.foo. insertOne ({\_id:"Hello"})

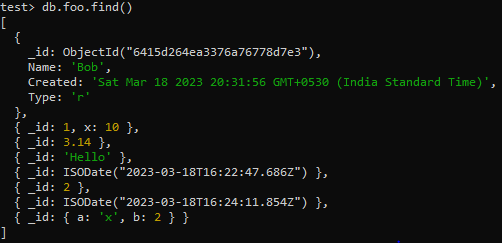
db.foo. insertOne ({\_id:ISODate()})



db.foo. insertOne ({\_id:{a:'x', b2}})



db.foo.find()



**Array datatype cannot be used for the \_id field**

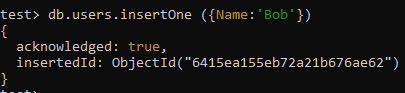
=================================================

db.foo.insertOne({\_id:[1,2,3]})



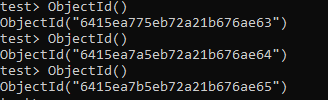
# ObjectId

db.users.insertOne ({Name:'Bob'})



Get new ObjectID from the shell

test> ObjectId()



Get Timestamp

test> ObjectId().getTimestamp()



# Insert

Remove All Documents from a Collection

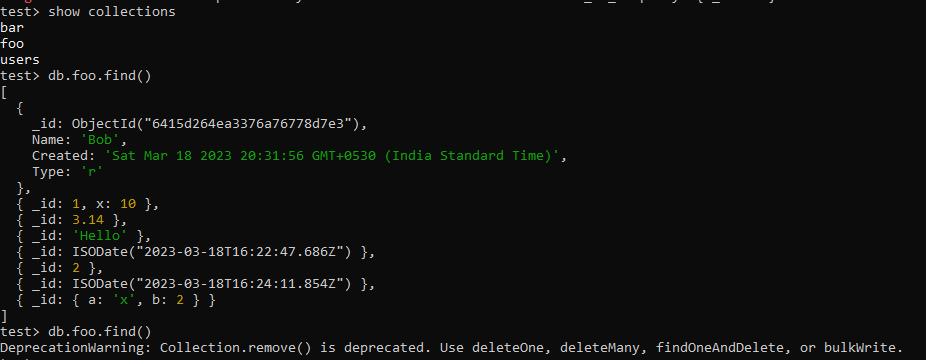
===============================================

**db.foo.remove( { } )**

test> db.foo.find()

test> db.foo.remove( { } )

test> db.foo.find()

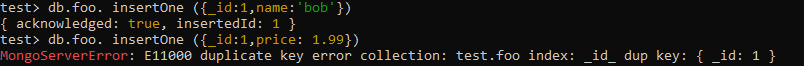


Save 2 different documents with the same Id

===========================================

db.foo. insertOne ({\_id:1,name:'bob'})

db.foo. insertOne ({\_id:1,price: 1.99})



db.foo.find()



# Insert with Id

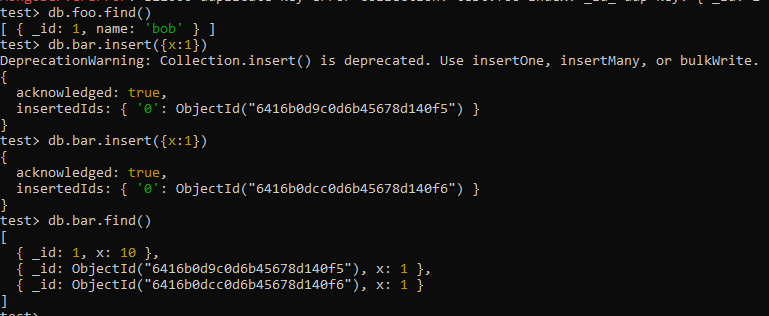
Insert 2 exact same documents without Id

==========================================

db.bar.insert({x:1})

db.bar.insert({x:1})

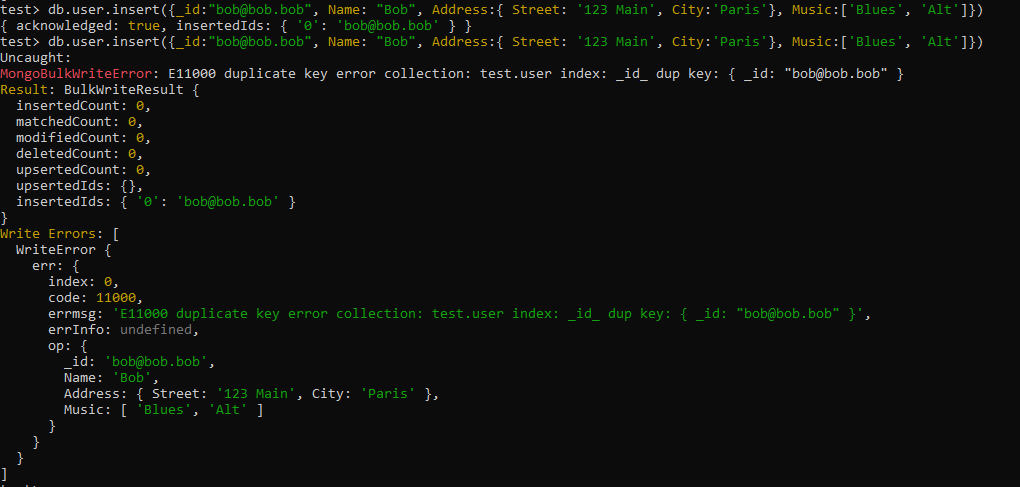
db.bar.find()



Tracking Users - use the email address as an ID

=======================================

db.user.insert({\_id:"bob@bob.bob", Name: "Bob", Address:{ Street: '123 Main', City:'Paris'}, Music:['Blues', 'Alt']})

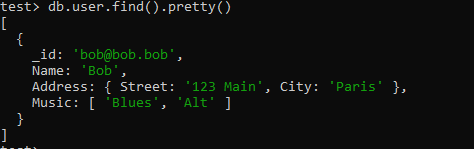


Use the insert command so that that if somebody double-clicked a button on the front-end or something, the second insert will fail.

# Complex Document

This is a document that contains an address, which is a subdocument, it contains fields of its own, and it contains an array of music that the user might like, maybe they like the blues and the alternative genres.

db.user.find().pretty()



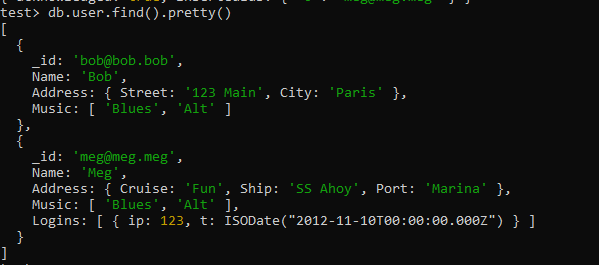
Insert another user with different format

===========================================

db.user.insert({ \_id:"meg@meg.meg", Name: "Meg", Address:{ Cruise: 'Fun', Ship:'SS Ahoy', Port: 'Marina'}, Music:['Blues', 'Alt'],Logins:[{ip:123, t: ISODate("2012-11-10")}]})



db.user.find().pretty()



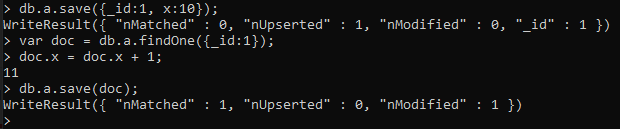
**\*\* Documents in MongoDB are free to define their own schema.**

# Save Danger

For versions priortoMongoDB 6.0.

Scenario 1

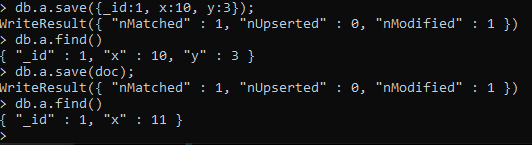
**=================**



If somebody else on another thread in another location tried to read the same document as I was reading it in increment x by themselves, I now have a stale value of x. So when I'm going to say save, I'm going to slam their value and assign x an incorrect number. My intent was to increment whatever was in the database, but my version of what was in the database when I read it is already gone and there's a new version there, I just don't know it. So that's a concurrency management issue and there is no versioning built in by default and I can't very well hold that record under lock for a long while, so this is not a great scenario.

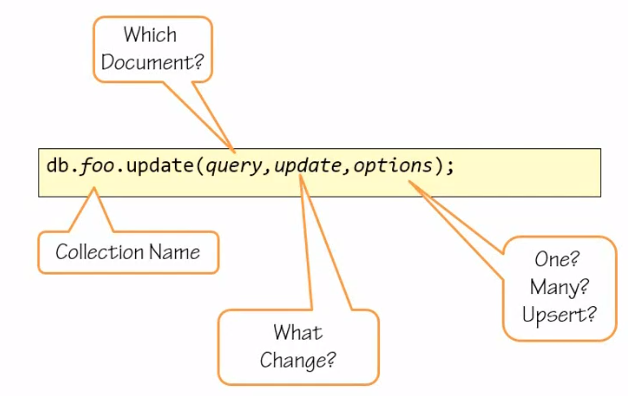
Scenario 2

**=================**



Another stale condition may occur if another client gets a version of the document and attempts to save it, this time with an extra field, at this point there is the version y because that's what I just saved, but if the first client had the version of the document with x incremented, then the version saved by them will only contain the field x with a value 11 and not have the field y. So we just deleted some data that somebody else wanted to save.

# Update Command

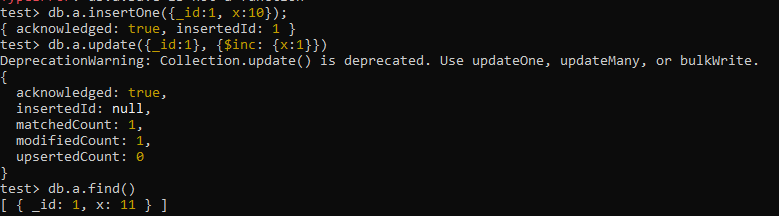


* First, you need to specify which collection you're going to update.
* Second, you will need to specify which document or documents you are targeting so that you don't update all the documents in that collection. You do that by issuing a Mongo query.
* Third, you will need to specify what change you want to see enacted, that's the update parameter.
* Lastly, you may specify other options such as do you want to change only one, the first document found matching the query, or multiple of them, any document matching the query. Do you want to upsert, meaning do you want to save a new record in case the query does not match any document, do you want to generate a document on the fly and inserted as the change specified. The options parameter in itself is optional. If you omit this parameter, Mongo will update only one document and Mongo will not create a new document if one does not exist that matches the query.

db.a.insertOne({\_id:1, x:10});

db.a.update({\_id:1}, {$inc: {x:1}})

db.a.find()



If two concurrent clients would have issued an increment, both of them would have incremented eventually one or the other before the other, but the field would have been incremented from whatever it was inside the server. It's not a stale version, it's the server's version, it finds the record, updates it in place.

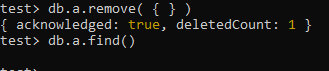
# Set Operator

Remove All Documents from a Collection

===============================================

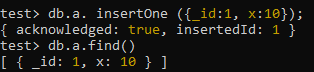
db.a.remove( { } )

db.a.find()



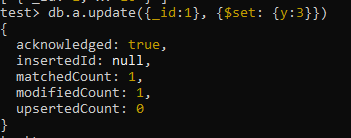
db.a. insertOne ({\_id:1, x:10});

db.a.find()



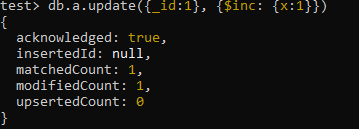
The second scenario, one client wants to add a field to the document while another client is trying to increment. So, we're going to have ourselves a record in there, db. a. find with only one field, x equal 10.

**db.a.update({\_id:1}, {$set: {y:3}})**



Now comes along the client that wants to add a field. Instead of saving a new record, they issue an update command. Within the update command the change will be set a field name y that did not exist before to the value of 3.

db.a.update({\_id:1}, {$inc: {x:1}})



db.a.find()



Now if that same client as before wanted to increment the value of x, only specifying that x should be incremented, they don't need a full version of the document, they just send over a change request saying find me that document and increment that field, don't touch anything else, and that change has been carried out without touching the field y.

# Unset Operator

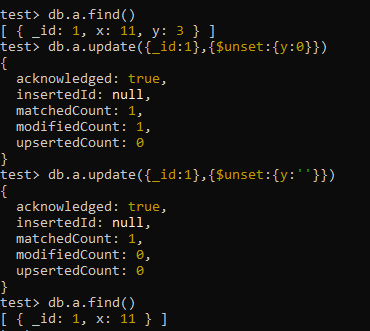
db.a.find()

db.a.update({\_id:1},{$unset:{y:0}})

**or**

db.a.update({\_id:1},{$unset:{y:''}})

db.a.find()



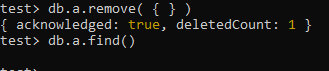
# Rename Operator

Remove All Documents from a Collection

===============================================

db.a.remove( { } )

db.a.find()



Insert a document with incorrectly spelled field name

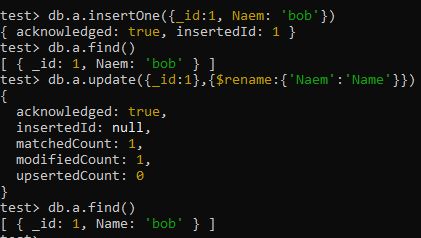
===============================================

db.a.insertOne({\_id:1, Naem: 'bob'})

db.a.find()

db.a.update({\_id:1},{$rename:{'Naem':'Name'}})

db.a.find()



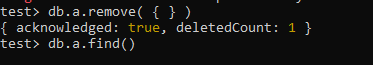
# Push Operator

Remove All Documents from a Collection

===============================================

db.a.remove( { } )

db.a.find()

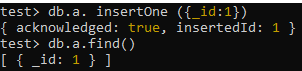


**Start with a document that has only an id**

**==========================================**

db.a. insertOne ({\_id:1})

db.a.find()

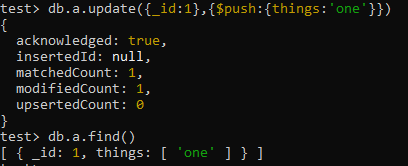


Create an array containing some values in this document. Issue the update with a push command to add an item to an array. It will create an array if one did not exist.

=====================================================================================

**db.a.update({\_id:1},{$push:{things:'one'}})**

db.a.find()



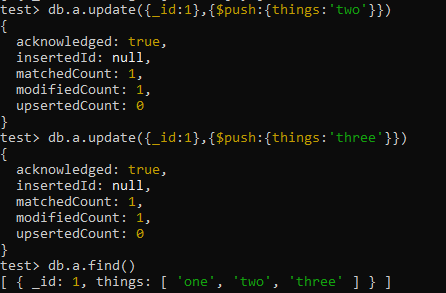
Push some more items to the array

=====================================

**db.a.update({\_id:1},{$push:{things:'two'}})**

**db.a.update({\_id:1},{$push:{things:'three'}})**

db.a.find()

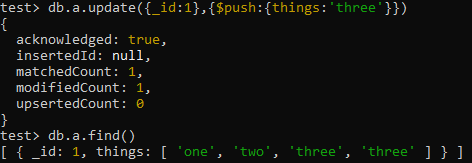


Push the item three again

===============================

db.a.update({\_id:1},{$push:{things:'three'}})

db.a.find()



addToSet operator

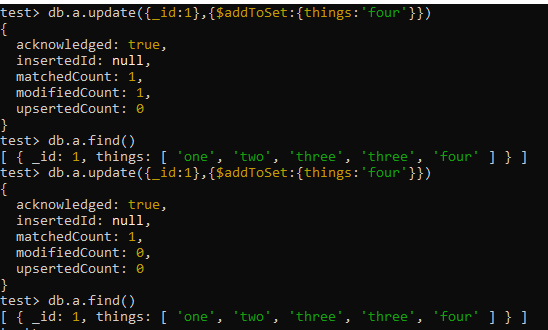
==========================

**db.a.update({\_id:1},{$addToSet:{things:'four'}})**

**db.a.find()**

db.a.update({\_id:1},{$addToSet:{things:'four'}})

db.a.find()



# Pull Operator

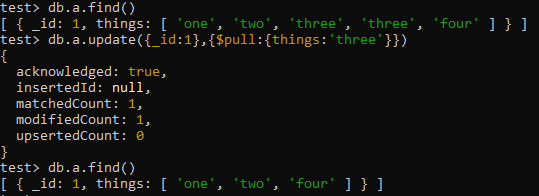
Remove duplicate elements

==========================

db.a.find()

db.a.update({\_id:1},{$pull:{things:'three'}})

db.a.find()



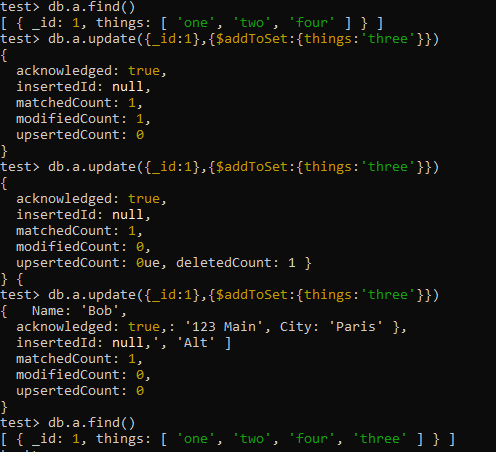
Then use addToSet operator

db.a.update({\_id:1},{$addToSet:{things:'three'}})

db.a.update({\_id:1},{$addToSet:{things:'three'}})

db.a.update({\_id:1},{$addToSet:{things:'three'}})

db.a.find()



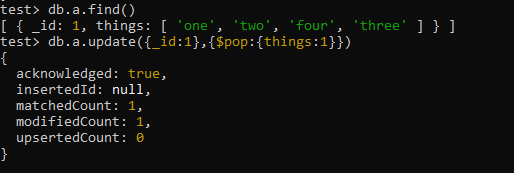
# Pop Operator

Remove the last element in an array

**=================================**

db.a.find()

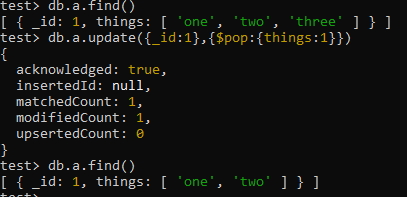
db.a.update({\_id:1},{$pop:{things:1}})



db.a.find()

**db.a.update({\_id:1},{$pop:{things:-1}})**

db.a.find()



db.a.update({\_id:1},{$pop:{things:-1}})

db.a.update({\_id:1},{$pop:{things:-1}})

db.a.update({\_id:1},{$pop:{things:-1}})

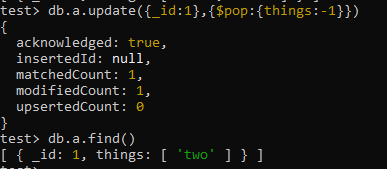
db.a.find()

Remove the first element in an array

**=================================**

**db.a.update({\_id:1},{$pop:{things:-1}})**

db.a.find()



Pop can be issued multiple times (even on an empty array)

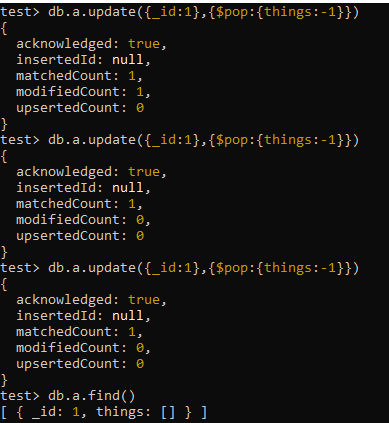
=======================================================

db.a.update({\_id:1},{$pop:{things:-1}})

db.a.update({\_id:1},{$pop:{things:-1}})

db.a.update({\_id:1},{$pop:{things:-1}})

db.a.find()



# Array Type

Preceding operations work only on Array types

=============================================

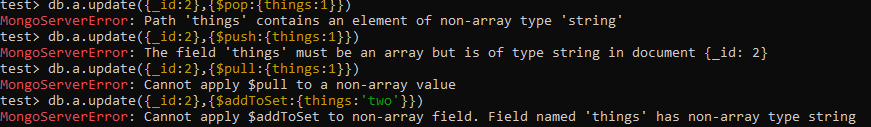
db.a.insertOne({\_id:2, things:"Hello"})

db.a.update({\_id:2},{$pop:{things:1}})

db.a.update({\_id:2},{$push:{things:1}})

db.a.update({\_id:2},{$pull:{things:1}})

db.a.update({\_id:2},{$addToSet:{things:'two'}})



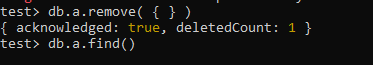
# Multi Update

Remove All Documents from a Collection

===============================================

db.a.remove( { } )

db.a.find()



Apply updates for multiple records

==================================

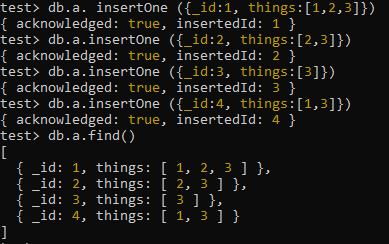
db.a.insertOne ({\_id:1, things:[1,2,3]})

db.a.insertOne ({\_id:2, things:[2,3]})

db.a.insertOne ({\_id:3, things:[3]})

db.a.insertOne ({\_id:4, things:[1,3]})

db.a.find()



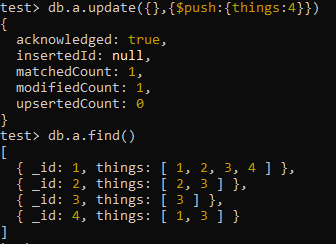
Use the update command with an empty query to say anything matches

===================================================================

Only one record was effected. This is because the default options for an update is to effect only one record.

db.a.update({},{$push:{things:4}})

db.a.find()

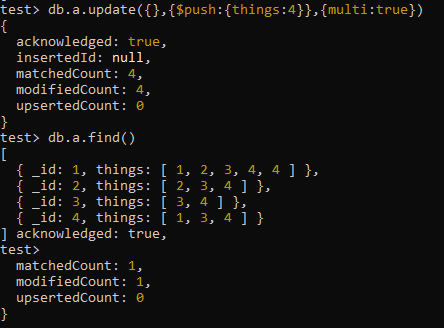


Update multiple fields with options multi field set to true

==================================================

db.a.update({},{$push:{things:4}},{multi:true})

db.a.find()



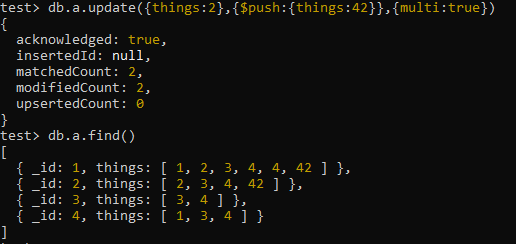
The first one has the number 4 twice, because the push command was used rather than addToSet.

Update the documents that have the element 2 in the array things

=========================================================

db.a.update({things:2},{$push:{things:42}},{multi:true})

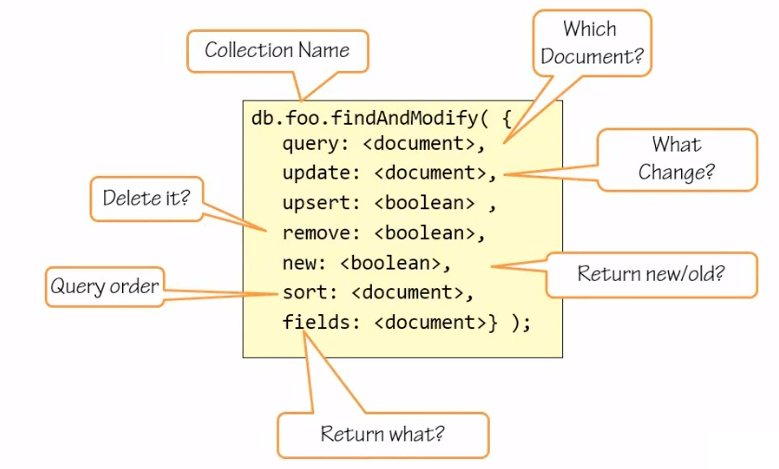
db.a.find()



# Find And Modify

**Signature of findAndModify**

**============================**

****

# Query With Sort

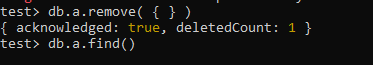
Now in the case the query you give may net more than one document, you want to also specify a sort order so the exact record that you want to retrieve will come back. If there are multiple records that match the exact criteria, only one of them would be modified, so you want to make sure that either your query is such that it will return only one document or that you give a sort order, meaning only the first document or the last document matching that criteria may be modified. If it's one of the interior ones in the batch, you can't really target it with any sorting order, therefore, you will need a more concise query, which will net exactly that document. Next we'd want to say what is the change we want to make to the document. We may want to upsert, set the upsert to true, meaning create a new record if one does not exist already that matches the criteria, or we might want to say remove that document, meaning the delete the document that matches this criteria, but delete exactly one record. It doesn't make sense to both upsert and remove obviously. The new parameter is interesting. FindAndModify also returns the record that you were going to update from the database. By default, find and modify returns you the version of the document before the change was made to it. If you set new to true, it will return the document after the change was made to it. Because findAndModify returns a document to you, you might want to say that only a part of the document is of interest, thereby reducing the traffic on the wire. Remember a document can be up to 16 MB in size, and you probably don't want to return the whole thing if you only updated a counter on it or something like that.

Remove All Documents from a Collection

===============================================

db.a.remove( { } )

db.a.find()



Insert multiple records

==================================

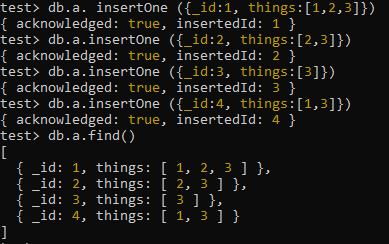
db.a.insertOne ({\_id:1, things:[1,2,3]})

db.a.insertOne ({\_id:2, things:[2,3]})

db.a.insertOne ({\_id:3, things:[3]})

db.a.insertOne ({\_id:4, things:[1,3]})

db.a.find()



Create an object which will represent the modification

=============================================

**var mod = {**

**query :**

**{**

**things :1,**

**}**

**,**

**update:**

**{**

**$set:{**

**touched:true**

**}**

**},**

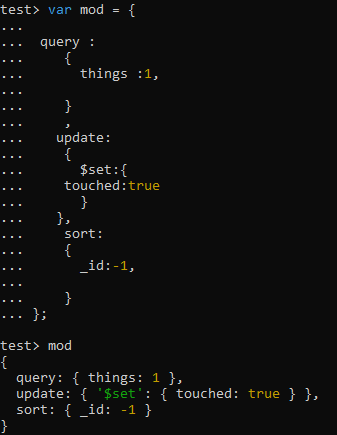
**sort:**

**{**

**\_id:-1,**

**}**

**};**



The object mod has a query, find me something that has a things element 1. It has an update -set the field touched to the value true, sort by the \_id in descending order, the -1 means in descending order, 1 would mean in ascending order.

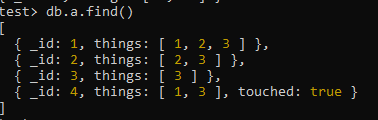
**Issue findAndModify command**

**================================**

**db.a.findAndModify(mod)**



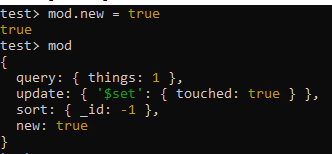
db.a.find()



**Set touched field to false. Return record after it was modified.**

mod.new = true

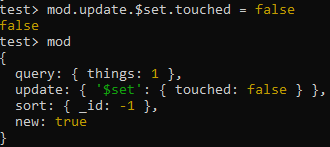
mod



**Update the touched field to be false**

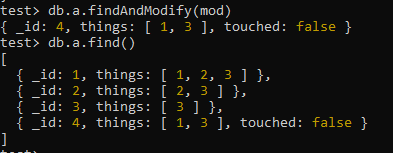
mod.update.$set.touched = false

mod



db.a.findAndModify(mod)

db.a.find()

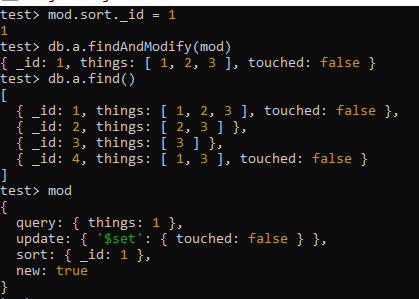


**Change the sort to find the first record rather than the last record matching the query**

mod.sort.\_id = 1

db.a.findAndModify(mod)

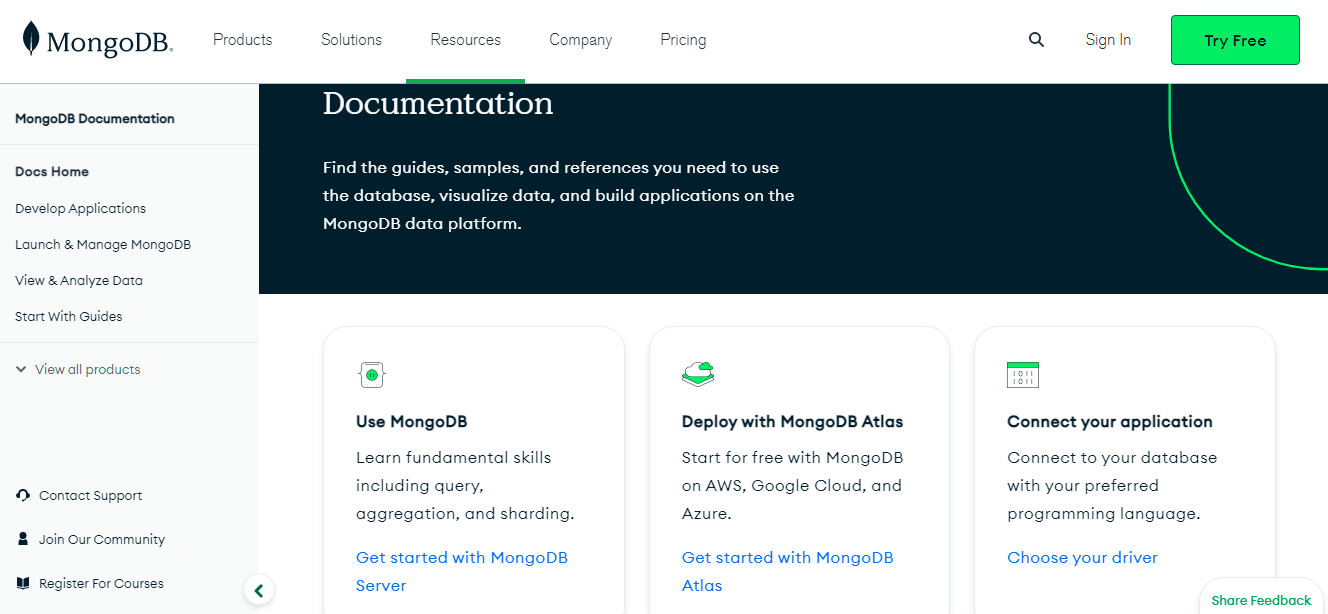
db.a.find()



The document with \_id 1 has been modified.

# Documentation

[**https://www.mongodb.com/docs/**](https://www.mongodb.com/docs/)



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***