**4. Finding Documents - Commands**

Contents

[1. find() 2](#_Toc130214665)

[2. Equality 2](#_Toc130214666)

[3. Projection 3](#_Toc130214667)

[4. Comparison 4](#_Toc130214668)

[5. $not 5](#_Toc130214669)

[6. $in 5](#_Toc130214670)

[7. Arrays 6](#_Toc130214671)

[8. $all 6](#_Toc130214672)

[9. Dot Notation 7](#_Toc130214673)

[10. Sub-Document 7](#_Toc130214674)

[11. null and $exists 9](#_Toc130214675)

[12. And 11](#_Toc130214676)

[13. More Projection 11](#_Toc130214677)

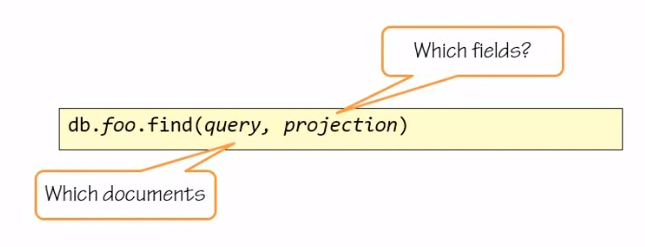
[14. Cursor 12](#_Toc130214678)

[15. limit() 16](#_Toc130214679)

[16. skip() 17](#_Toc130214680)

[17. findOne() 17](#_Toc130214681)

# find()



The find command signature is very simple. It has a query parameter and a projection parameter. The query parameter is a filter. It defines the matching criteria to run against the documents. The projection parameter defines of those documents which parts should be returned. The projection is optional. If you don't specify a projection parameter, the whole document or all documents and all their parts are going to be returned to you from the find command.

# Equality

Find a document whose Id is 1 in the collection animals

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

var animals = {

\_id: 1,

name: "cat",

tags : ["land" ,"cute"],

info : {

type: "mammal",

color: "red"

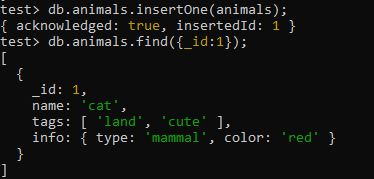
},

};



db.animals.insertOne(animals);

db.animals.find({\_id:1});



# Projection

Include only the ID in your result

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

db.animals.find({\_id:1},{\_id:1});



# Comparison

Find all animals who have an ID greater than 5

db.animals.insertOne({ \_id: 1, name: 'cat', tags:['land','cute'] ,info:{type:'mammal', color:'red' }})

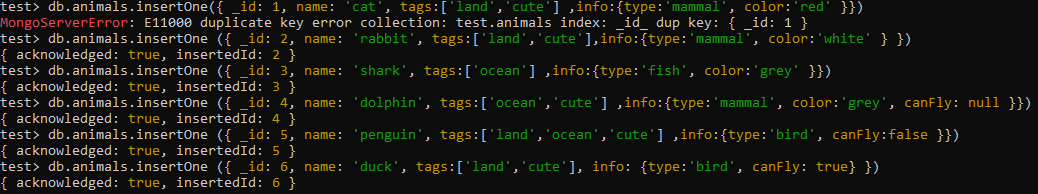
db.animals.insertOne ({ \_id: 2, name: 'rabbit', tags:['land','cute'],info:{type:'mammal', color:'white' } })

db.animals.insertOne ({ \_id: 3, name: 'shark', tags:['ocean'] ,info:{type:'fish', color:'grey' }})

db.animals.insertOne ({ \_id: 4, name: 'dolphin', tags:['ocean','cute'] ,info:{type:'mammal', color:'grey', canFly: null }})

db.animals.insertOne ({ \_id: 5, name: 'penguin', tags:['land','ocean','cute'] ,info:{type:'bird', canFly:false }})

db.animals.insertOne ({ \_id: 6, name: 'duck', tags:['land','cute'], info: {type:'bird', canFly: true} })



Find by Comparison Operators

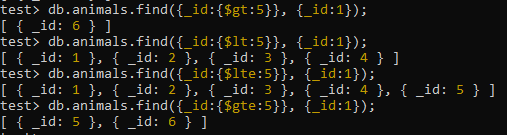
=============================

db.animals.find({\_id:{$gt:5}}, {\_id:1});

db.animals.find({\_id:{$lt:5}}, {\_id:1});

db.animals.find({\_id:{$lte:5}}, {\_id:1});

db.animals.find({\_id:{$gte:5}}, {\_id:1});

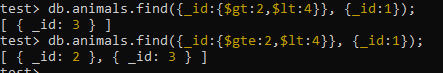


Specify a Range

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

db.animals.find({\_id:{$gt:2,$lt:4}}, {\_id:1});

db.animals.find({\_id:{$gte:2,$lt:4}}, {\_id:1});



# $not

Negate a selection

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

db.animals.find({\_id:{$not:{$gt:2}}}, {\_id:1});



# $in

Match against discrete values

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

db.animals.find({\_id:{$in:[1,3]}}, {\_id:1});



Find all those whose values are not 1 or 3

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

db.animals.find({\_id:{$nin:[1,3]}}, {\_id:1});



**\*\* The greater to and equal and less than also work on strings.**

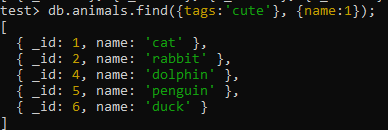
# Arrays

Find an animal who has the tag cute

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

The animal documents contains a tags field, and the tags field contains an array of tags.

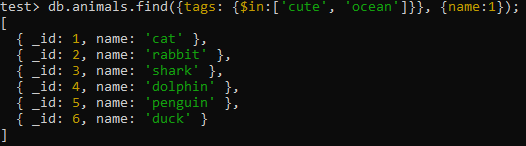
db.animals.find({tags:'cute'}, {name:1});



Find animals who are cute or in the ocean

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

db.animals.find({tags: {$in:['cute', 'ocean']}}, {name:1});



# $all

Find animals who are cute and in the ocean

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

db.animals.find({tags: {$all:['cute', 'ocean']}}, {name:1});



Find animals who are not cute (not in operator)

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

db.animals.find({tags: {$nin:['cute']}}, {name:1});



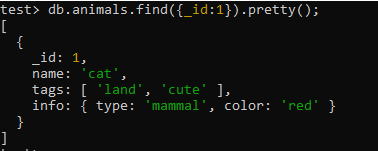
# Dot Notation

Find an animal whose info.canFly field is set to true

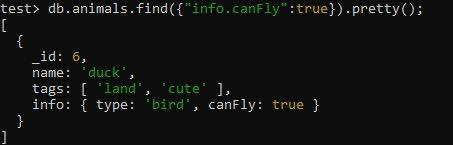
================================================

\*\* The info field contains a subdocument.

db.animals.find({\_id:1}).pretty();



db.animals.find({"info.canFly":true}).pretty();



# Sub-Document

Find an animal whose info is a type bird and a canFly of true

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

db.animals.find({info:{type:'bird',canFly:true}}, {name:1});



Find an animal whose info is a canFly true and a type of bird

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

db.animals.find({info:{canFly:true, type:'bird'}}, {name:1});



No document in response. For the field info, Mongo tries to match the sub-document type: bird, canFly: true exactly as it is

Mongo stores its data as BSON and memory mapped files. So the sub-document is just one BLOB for it. . It doesn't break it apart and try to find each field matching each of those fields in the database. It just tries to get this value exactly. If it finds it, then it's a match. And if not, it's not. It's not only about the order of the fields here, it's also about the existence of all of them.

Refer <https://bsonspec.org/>.

[http://mongodb.github.io/node-mongodb-native/schema/chapter3/#:~:text=Memory%20Mapped%20Files,of%20a%20file%20or%20file).](http://mongodb.github.io/node-mongodb-native/schema/chapter3/" \l ":~:text=Memory%20Mapped%20Files,of%20a%20file%20or%20file).)

Find an animal whose info is a type of bird

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

db.animals.find({info:{type:'bird'}}, {name:1});

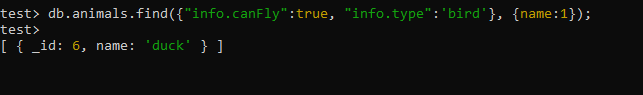


Mongo will not match that. It doesn't understand that I'm saying I want the field info to contain a subdocument with a field type bird.

Use Dot Notation

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

db.animals.find({"info.canFly":true, "info.type":'bird'}, {name:1});



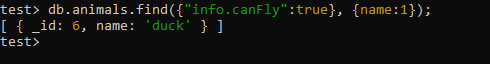
\*\* the canFly of the info field and the type of the info field must match the criteria given above.

# null and $exists

Find an animal that can fly

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

db.animals.find({"info.canFly":true}, {name:1});



Find an animal that cannot fly

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

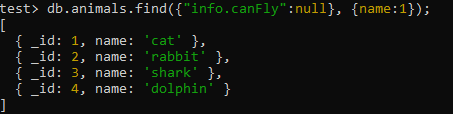
db.animals.find({"info.canFly":false}, {name:1});



Find an animal(s) that has canFly set to null

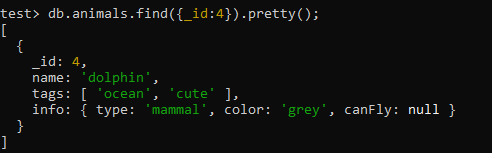
=====================================

db.animals.find({"info.canFly":null}, {name:1});

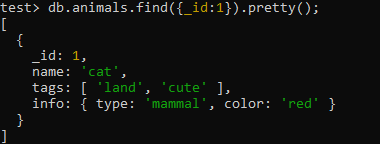


Check specific records for canFly value

db.animals.find({\_id:4}).pretty();



db.animals.find({\_id:1}).pretty();



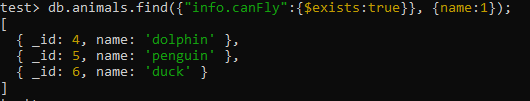
So when we execute **db.animals.find({"info.canFly":null}, {name:1});**

it returned us both records that had that field set to null and records that did not have the field.

Use the exists operator to be stricter

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

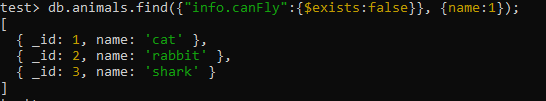
db.animals.find({"info.canFly":{$exists:true}}, {name:1});



Find documents that do not have the specified field

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

db.animals.find({"info.canFly":{$exists:false}}, {name:1});



# And

Find documents that match multiple criteria

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

db.animals.find({"info.type":'bird',tags:'cute'}, {name:1});



Find documents that match info type of bird and a tag of ocean

db.animals.find({"info.type":'bird',tags:'ocean'}, {name:1});



\*\* The comma between the criteria in this case performs like an **and**, it's the ***and*** of all the criteria specified in the query.

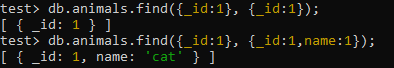
# More Projection

Fields to be included have a value of 1

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

db.animals.find({\_id:1}, {\_id:1});

db.animals.find({\_id:1}, {\_id:1,name:1});



Fields to be excluded have a value of 0

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

db.animals.find({\_id:1}, {\_id:0,name:0, info:0});



\*\* Excluded all fields except for the tags, so only got the tags back.

Omitting Id field has no effect

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

db.animals.find({\_id:1}, {name:1, info:1});



The ID is a special field. When specifying which fields to include, specifically exclude the ID field.

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

db.animals.find({\_id:1}, {\_id:0,name:1,info:1});



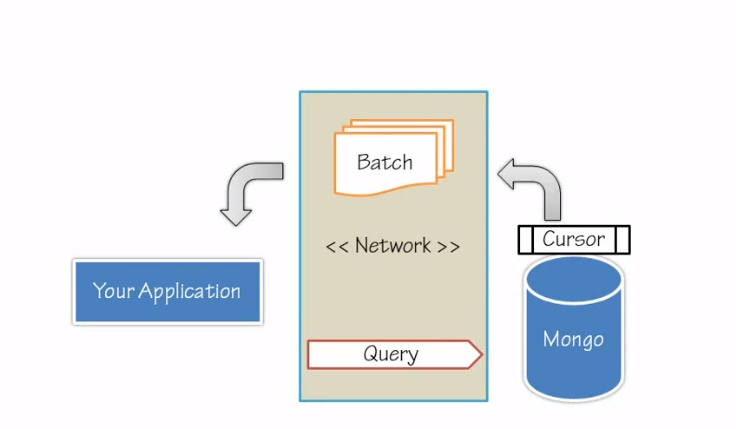
Cannot mix and match. Choose whether you want to include or exclude.

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

db.animals.find({\_id:1}, {\_id:0,name:1,info:0});



# Cursor

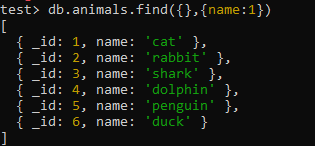


Now what exactly happens when you're shooting a query at the Mongo server? Your application will want to communicate over the network. It's going to shoot a query over the network to Mongo, and that query might cover many, many documents, more than you even need, more than fit in your memory. To support efficient retrieval of documents, Mongo uses a cursor. Mongo will populate a cursor and give you a batch of documents each time. On the client side, the Mongo shell and every other driver will retrieve a batch of documents from the cursor and finally close the cursor. A client can then signal the server to close the cursor before all the documents have been exhausted, thereby saving many, many bytes over the network and a lot of time, also freeing the server to service other calls.

Find documents that match an empty criteria

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

db.animals.find({},{name:1})



\*\* The Mongo server returned a cursor, a pointer to the client where the client can go back to the server and request a batch at the time.

Capture this cursor into a variable

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

var c = db.animals.find({},{name:1})



Look at cursor size

c.size



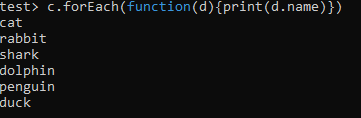
To support iterating through the cursor, query the cursor to see if it has items,

c.hasNext()



For Each method

c.forEach(function(d){print(d.name)})



Check if cursor is exhausted

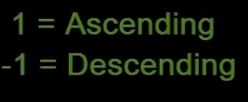
========================

c.hasNext()

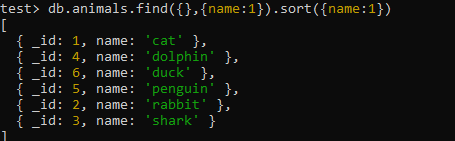


Sort animals by name in ascending order

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

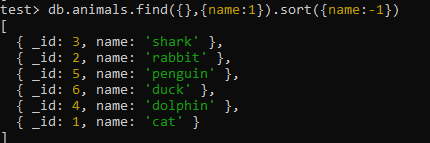


db.animals.find({},{name:1}).sort({name:1})



Sort animals by name in descending order

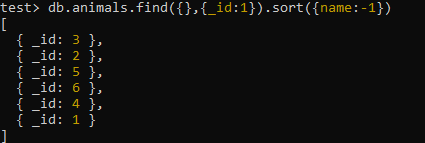
===================================



Sort animals by name but return only the \_id field

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

db.animals.find({},{\_id:1}).sort({name:-1})

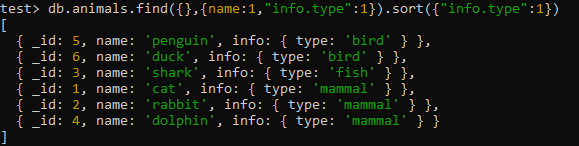


\*\* This proves that the sorting is done on the server side because the field name is not even returned, so how would Mongo sort it correctly.

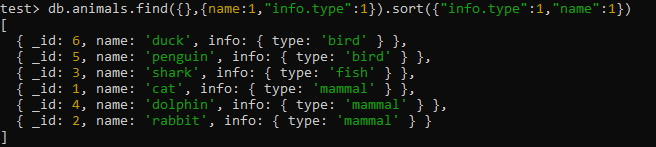
Sort on Sub-Document fields

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

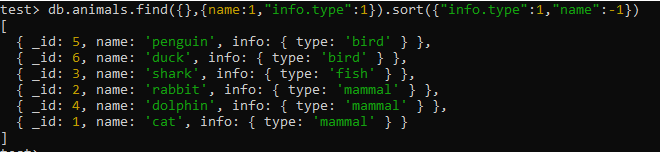
db.animals.find({},{name:1,"info.type":1}).sort({"info.type":1})



db.animals.find({},{name:1,"info.type":1}).sort({"info.type":1,"name":1})



db.animals.find({},{name:1,"info.type":1}).sort({"info.type":1,"name":-1})



\*\* If we reverse the name sorting criteria, we will see the penguin and duck switch locations on the top, that's because first we sorted on the info. type bird, both begin with b, and penguin with a p sorted higher when we sorted it ascending and lower when we sorted it descending.

# limit()

Return specified number of documents

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

db.animals.find({},{\_id:1}).sort({\_id:-1}).limit(3)



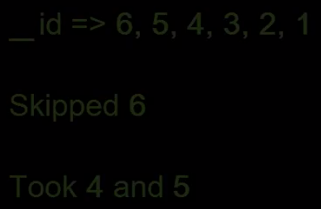
# skip()

Server side Paging

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

db.animals.find({},{\_id:1}).sort({\_id:-1}).skip(1).limit(2)





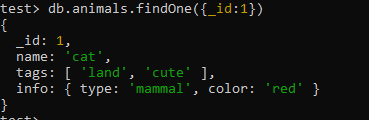
\*\* Using the skip and limit is a much more efficient way to page through documents than retrieving all of them into the client and paging on the client side.

# findOne()

For retrieving exactly one document, you don't even need a cursor.

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

db.animals.findOne({\_id:1})

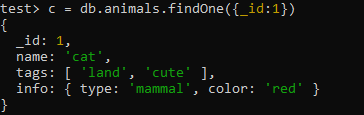


\*\* findOne command will return to you exactly one record, not a cursor, but a single record.

findOne returns an object

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

c = db.animals.findOne({\_id:1})



c.hasNext()



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