Les10a-Mongo part 1 Query
MongoDB Query

Also available on PowerPoints by Nasim

Topics

1 ad-hoc queries using find and findOne

2 Query for

- Range
- Set inclusion
- Inequalities

FIND

Already done a few

Move to mydb

Create a collection blog and insert one piece of data

```
db.blog.insertOne (
{
   "_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
   "title" : "My Blog Post",
   "content" : "Here's my blog post.",
   "date" : ISODate("2012-08-24T21:12:09.982Z")
}
)
```

Add another

```
db.blog.insertOne (
{
   "title" : "My Blog Post 2",
   "content" : "Here's my blog post 2.",
}
)
```

FIND (cont'd)

The *find* function is called to query and see documents in a collection. It returns a subset of documents.

To see documents in *blog* collection:

```
        db.blog.find()
Or
        db.blog.find({})
```

The following output is the result of calling the find function collection:

```
{ "_id" : ObjectId("5037ee4a1084eb3ffeef7228"), "title" : "My Blog Post", "content" : "Here's my blog post.",
"date" : ISODate("2012-08-24T21:12:09.982Z") }
{ "_id" : ObjectId("5fb6d40060924ef8e582c389"), "title" : "My Blog Post 2", "content" : "Here's my blog post
2." }
```

```
> db.blog.find({}).pretty();
{
        "_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
        "title" : "My Blog Post",
        "content" : "Here's my blog post.",
        "date" : ISODate("2012-08-24T21:12:09.982Z")
}
{
        "_id" : ObjectId("5fb6d40060924ef8e582c389"),
        "title" : "My Blog Post 2",
        "content" : "Here's my blog post 2."
}
```

findOne

Shows only one document.

If you run it again it will show the same document

```
db.blog.findOne()
{
    "_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
    "title" : "My Blog Post",
    "content" : "Here's my blog post.",
    "date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

Query with Key/Values Pairs

- We can add key/value pairs to restrict the search.
  - Members match members
  - Booleans match Booleans
  - Strings match strings

- Example:
  1 Find all documents where the value of age is 20:

```
db.users.find({"age" : 20})
```

We have not loaded any

2 Search for documents where the username is “joe”

```
db.users.find({"username" : "joe"})
```

Find with multiple conditions

Simply add more conditions

Example:
We want to find users where their age is 20 and the username is “joe”:

```
> db.users.find({"username" : "joe", "age" : 27})
```

To find items which are out of stock:

```
> db.stock.find({"in_stock" : 0})
```

find some

Limit the Keys/Values to see

First -- Adding another collection ➔ Employee
Done by an array, but not needed yet

```
var myEmployee=
        [       {
                        "Employeeid" : 1,
                        "EmployeeName" : "Smith"
                },
                {
                        "Employeeid"   : 2,
                        "EmployeeName" : "Mohan"
                },
                {
                        "Employeeid"   : 3,
                        "EmployeeName" : "Joe"
                },
        ];

db.Employee.insert(myEmployee);
```

Display json

What Is JSON?

JSON is short for **JavaScript Object Notation** and is *a way to store information in an organized, easy-to-access manner.* In a nutshell, it gives us a human-readable collection of data that we can access in a logical manner.

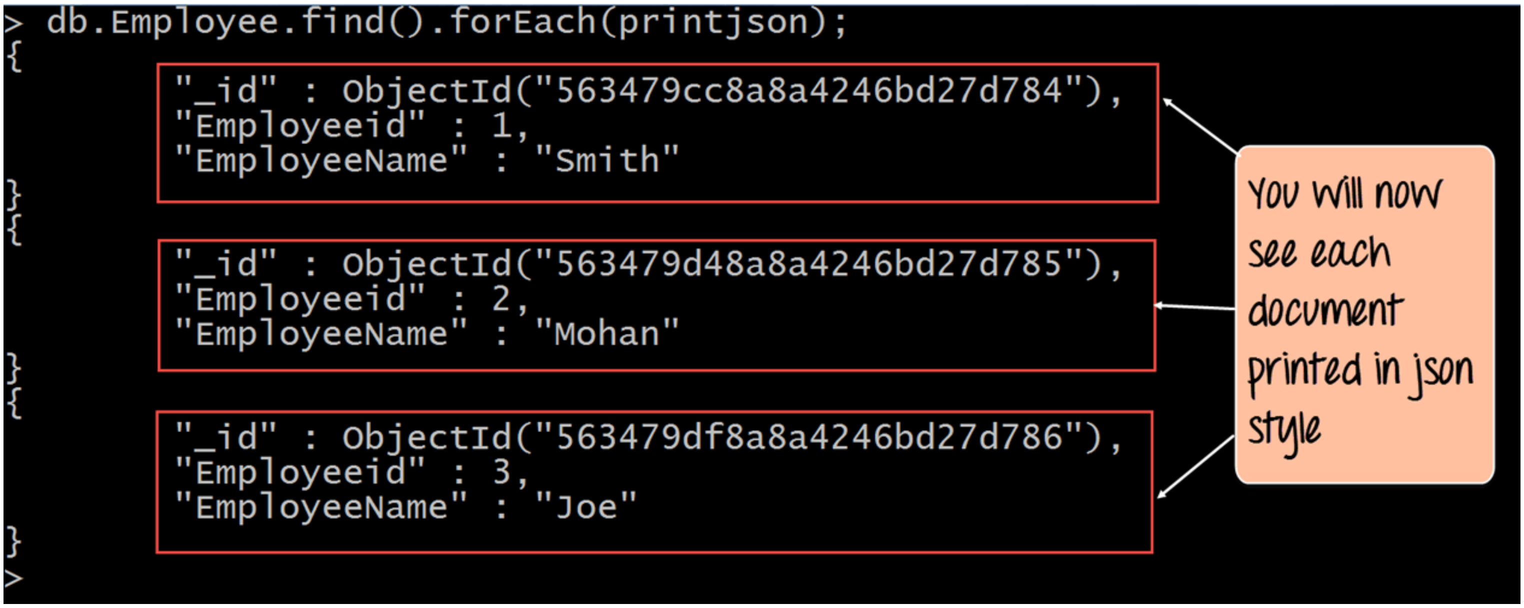
db.Employee.find().forEach(printjson)

Code Explanation:

1. The first change is to append the function called for Each() to the find() function. What this does is that it makes sure to explicitly go through each document in the collection. In this way, you have more control of what you can do with each of the documents in the collection.
2. The second change is to put the printjson command to the forEach statement. This will cause each document in the collection to be displayed in JSON format.

If the command is executed successfully, the following Output will be shown

Output:



```
> db.Employee.find().forEach(printjson);
{
        "_id" : ObjectId("563479cc8a8a4246bd27d784"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
{
        "_id" : ObjectId("563479d48a8a4246bd27d785"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
{
        "_id" : ObjectId("563479df8a8a4246bd27d786"),
        "Employeeid" : 3,
        "EmployeeName" : "Joe"
}
>
```

You will now see each document printed in json style

PRIMARY KEY

What is Primary Key in MongoDB?

In MongoDB, _id field as the primary key for the collection so that each document can be uniquely identified in the collection. The _id field contains a unique ObjectID value.

By default, when inserting documents in the collection, if you do not add a field name with the _id in the field name, then MongoDB will automatically add an Object id field as shown below

How to not have the default key

```
db.Employee.insert({_id:10, "EmployeeName" : "Smith"})
```

Now do a find

```
> db.Employee.find().forEach(printjson)
{
        "_id" : ObjectId("5fb6d84a60924ef8e582c38a"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
{
        "_id" : ObjectId("5fb6d84a60924ef8e582c38b"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
{
        "_id" : ObjectId("5fb6d84a60924ef8e582c38c"),
        "Employeeid" : 3,
        "EmployeeName" : "Joe"
}
{ "_id" : 10,
"EmployeeName" : "Smith" }
```

Queries with Criteria

Range

OR clauses

And Negation

Range

Done the same as any language just the operators are a bit different

Comparison operators:

Operators	Description
“$lt”	<
“$lte”	<=
“$gt”	>
“$gte”	>=
“$ne”	<>

The "$ne" operator can be used with any type.

Conditional Examples

The comparison operators can be combined to create range queries.

The following query look for employees with an Employeeid more than 2

```
db.Employee.find({Employeeid : {$gt:2}})   --- not a big problem at first
                                                but all the bracketing
```

OUTPUT:

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38c"), "Employeeid" : 3,
"EmployeeName" : "Joe" }
```

More than one operator

Range query

db.Employee.find({Employeeid : {$gt:1 , $le:3}}); -- will show an error

Correction:

db.Employee.find({Employeeid : {$gt:1 , $lte:3}});

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38b"), "Employeeid" : 2,
"EmployeeName" : "Mohan" }
```

NOT EQUAL

Look for employee not equal Mohan

db.Employee.find({"EmployeeName" : {"$ne" : "Mohan"} })
(aside: the bracketing needs to be looked at carefully)

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : 10, "EmployeeName" : "Smith" }
```

LIMIT clause

How to limit the number of documents returned
(not on test or exam)

```
db.Employee.find().limit(2)

{ "_id" : ObjectId("5fb6d84a60924ef8e582c38a"), "Employeeid" : 1,
"EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38b"), "Employeeid" : 2,
"EmployeeName" : "Mohan" }
>
```

OR Query

2 ways to do the OR

1	Just like in SQL	➔	IN
2	Use the	➔	OR

"$in"

Used to search for several values of a key

To match more than 1 value use an array of values … [….]

Start with showing what is in the collection

```
        db.Employee.find().pretty();
```

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : 10, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
```

```
db.Employee.find( {"Employeeid": { "$in" : [ 1, 3, 78 ] } });
```

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
```

In with different values

First insert a different row with a string as a key

```
db.Employee.insertOne (
{ "Employeeid" : "Tarr", "EmployeeName" : "RonTarr" })
```

Check for values stored

```
db.Employee.find();
```

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : 10, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : ObjectId("5fbbc55215e69a4c0f0ca16d"), "Employeeid" : "Tarr", "EmployeeName" : "RonTarr" }
```

This IN can contain numeric and character values

```
db.Employee.find( {"Employeeid": { "$in" : [ 3, 78, "Tarr" ] } });
```

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16c"), "Employeeid" : 3, "EmployeeName" : "Joe" }
{ "_id" : ObjectId("5fbbc55215e69a4c0f0ca16d"), "Employeeid" : "Tarr", "EmployeeName" : "RonTarr" }
```

The following statements are equivalent:

```
{ticket_no : {$in : [725]}}

{ticket_no : 725}
```

"$nin" Operator

NOT IN

The operator returns documents that do not match any of the criteria in the array.

Use the previous example changing IN to a NIN

```
db.Employee.find( {"Employeeid": { "$nin" : [ 3, 78, "Tarr" ] } });
```

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : 10, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16a"), "Employeeid" : 1, "EmployeeName" : "Smith" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
```

"$or"   "$and"   Operators

As in other languages
OR -- checks for possibilities and as long as one of them is true the document is returned

AND -- checks that both values are true in order to return the document

OR

```
db.Employee.find({"$or": [ { "Employeeid" : "Tarr" },{"EmployeeName" : "Mohan"} ]});
```

```
{ "_id" : ObjectId("5fb6d84a60924ef8e582c38b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : ObjectId("5fbbb4df15e69a4c0f0ca16b"), "Employeeid" : 2, "EmployeeName" : "Mohan" }
{ "_id" : ObjectId("5fbbc55215e69a4c0f0ca16d"), "Employeeid" : "Tarr", "EmployeeName" : "RonTarr" }
```

AND

```
db.Employee.find({"$and": [ { "Employeeid" : "Tarr" },{"EmployeeName" : "Mohan"} ]});
```

there is not anything matching the AND condition

NOT

"$not" Operator

SYNTAX: { field: { $not: { <expression> } } }

NULL

Null means the value of a key is unknown.

Assume the following documents:

```
db.c.find()
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }
{ "_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }
```

- To find documents with the NULL value for the “y” key:

```
db.c.find({"y" : null})

{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
```

To find all documents that a specific key does not exist among their keys.

```
 db.c.find({"z" : null})
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }
{ "_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }
```

$exists

We want or find documents that have the “z” key, but its value is null.

We want to exclude any documents that does not contain the “z” key.

```
 db.c.find ({"z" : {"$in" : [null], "$exists" : true}})
```

$all

Assume the following documents in the *food* collection:

```
db.food.insert({"_id" : 1, "fruit" : ["apple", "banana", "peach"]})
db.food.insert({"_id" : 2, "fruit" : ["apple", "kumquat", "orange"]})
db.food.insert({"_id" : 3, "fruit" : ["cherry", "banana", "apple"]})
```

We want or find all documents with both apple and banana elements.

```
 db.food.find({fruit : {$all : ["apple", "banana"]}})
      {"_id" : 1, "fruit" : ["apple", "banana", "peach"]}
      {"_id" : 3, "fruit" : ["cherry", "banana", "apple"]}
```

“$size” Operator

To query arrays for a given size, the “$size” operator is used.

```
> db.food.find({"fruit" : {"$size" : 3}})
```

You cannot combine the "$size" operator with other $ conditional operators.

There are some more topics ...see Nasim's PPT