MongoDB Lab part 1

Week 11, Advanced Databases.

Dr. Pierpaolo Dondio

Connect to the MongoDB Instance (see instruction on Webcourses).

Switch to your database. There is a database for each student, identified by your student number (all small letters) For example:

```
use c1234567
```

Manipulating and Querying a collection

We are going to create a collection named nettuts in your database.

The script pop.js on Webcourses contains a series of insert statements to create the collection.

Login into Mongo instance using your credential:

```
mongo -u <studentid> -p <studentid> localhost:27017/admin
```

Load the script pop.js:

```
load('pop.js')
```

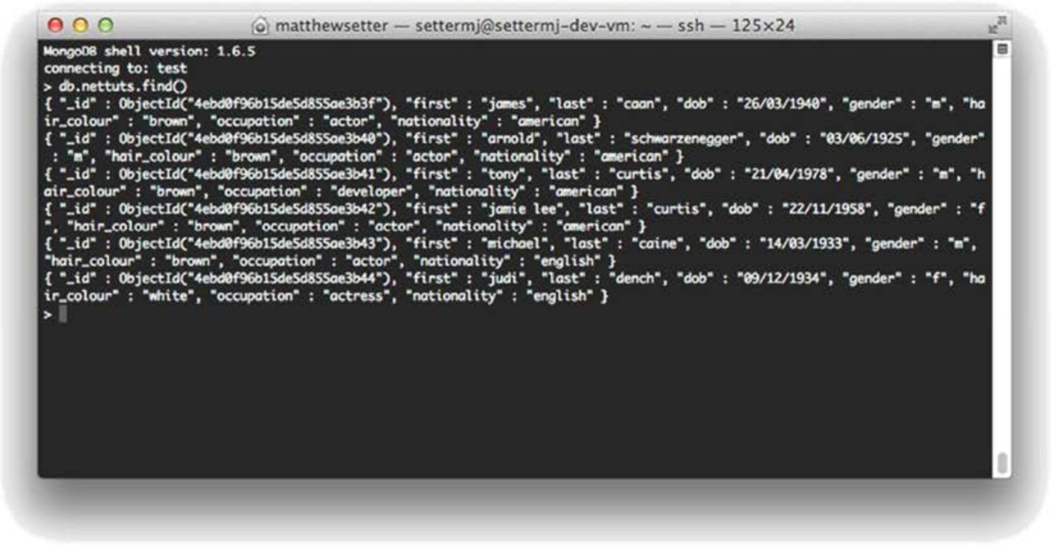
Enter the shell. Verify that the collection has been created with:

```
show collections
```

In order to show the content of a collections, execute the following query (equivalent to a select * in SQL):

```
db.nettuts.find()
```

then you will see the an output similar to this one:



```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find()
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b40"), "first" : "arnold", "last" : "schwarzenegger", "dob" : "03/06/1925", "gender"
 : "m", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b41"), "first" : "tony", "last" : "curtis", "dob" : "21/04/1978", "gender" : "m", "h
air_colour" : "brown", "occupation" : "developer", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b42"), "first" : "jamie lee", "last" : "curtis", "dob" : "22/11/1958", "gender" : "f
", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b44"), "first" : "judi", "last" : "dench", "dob" : "09/12/1934", "gender" : "f", "ha
ir_colour" : "white", "occupation" : "actress", "nationality" : "english" }
> 
```

The output could be formatted better by using `db.nettuts.find().pretty()` (not available in the online shell)

This shows that all of the records were created in the database. One thing to note before we go any further is the `id` field. This is auto generated by Mongo for you, if you don't specify
an id. The reason is that every record must have a unique `id` field.

You can see that we have one record for each of the ones that we insert – now we're ready to start querying them.

Searching For Records

You remember the previous command? It retrieved and displayed every record in the database. Helpful, yes, but how do you be more specific? How do you find all female actors, filtering out the males? That's a good question and the answer is selectors.

Selectors

Selectors are to Mongo what `where` clauses are to SQL. As with where clauses, Mongo selectors allow us to do the following:

- specify criteria that `MUST` match. i.e., an `AND` clause
- specify criteria that `CAN` optionally match. i.e., an `OR` clause
- specify criteria that `MUST` exist
- and much more...

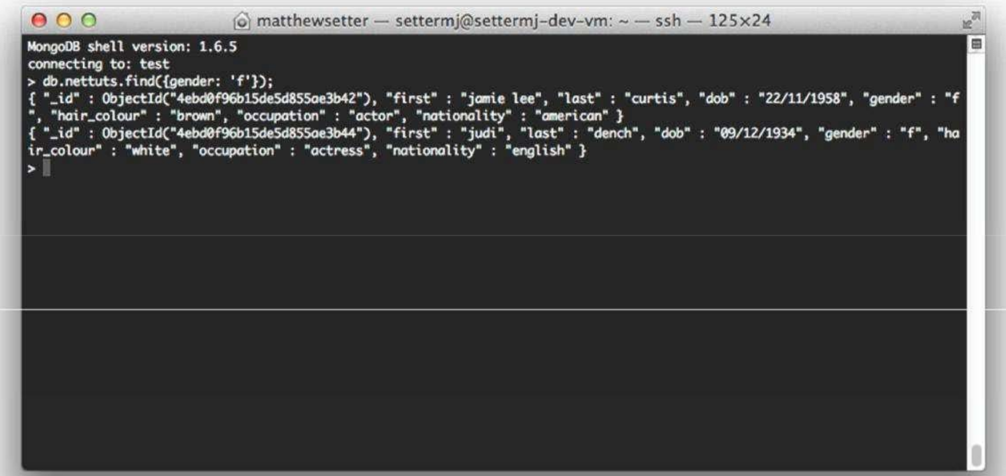
Records That MUST Match

Let's start with a basic selector. Say that we want to find `all actors that are female`. To accomplish that, you'll need to run the following command:

```
db.nettuts.find({gender: 'f'});
```

Here we have specified that gender must be equal 'f'.

Running that command will return the following output:

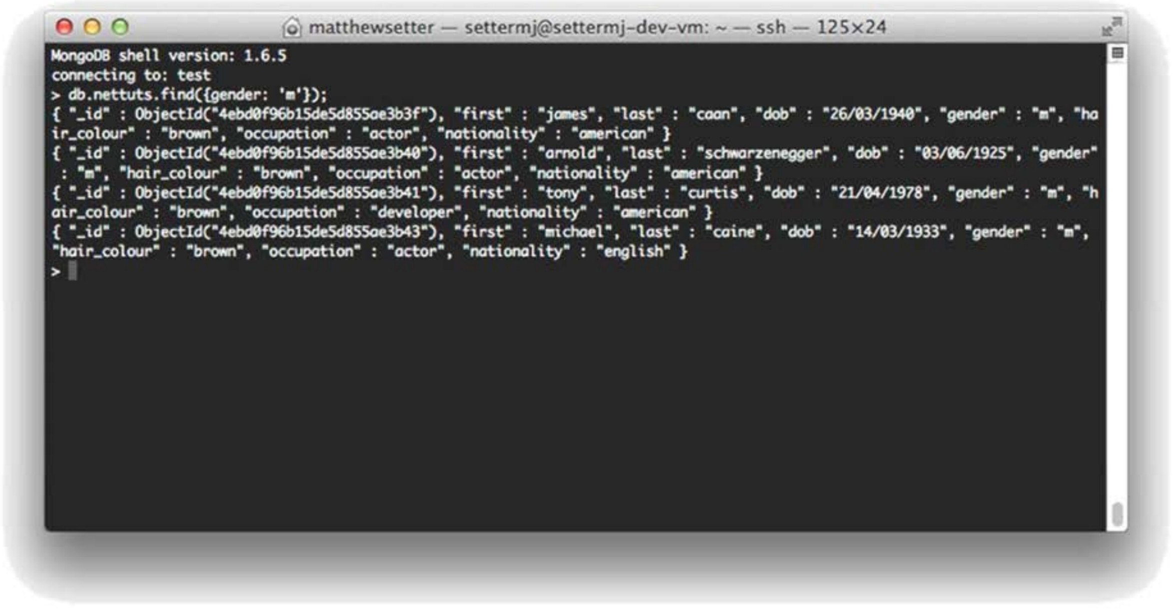


```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'f'});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b42"), "first" : "jamie lee", "last" : "curtis", "dob" : "22/11/1958", "gender" : "f
", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b44"), "first" : "judi", "last" : "dench", "dob" : "09/12/1934", "gender" : "f", "ha
ir_colour" : "white", "occupation" : "actress", "nationality" : "english" }
> 
```

What if we wanted to search for male actors? Run the following command:

```
db.nettuts.find({gender: 'm'});
```

We'll get the following results:



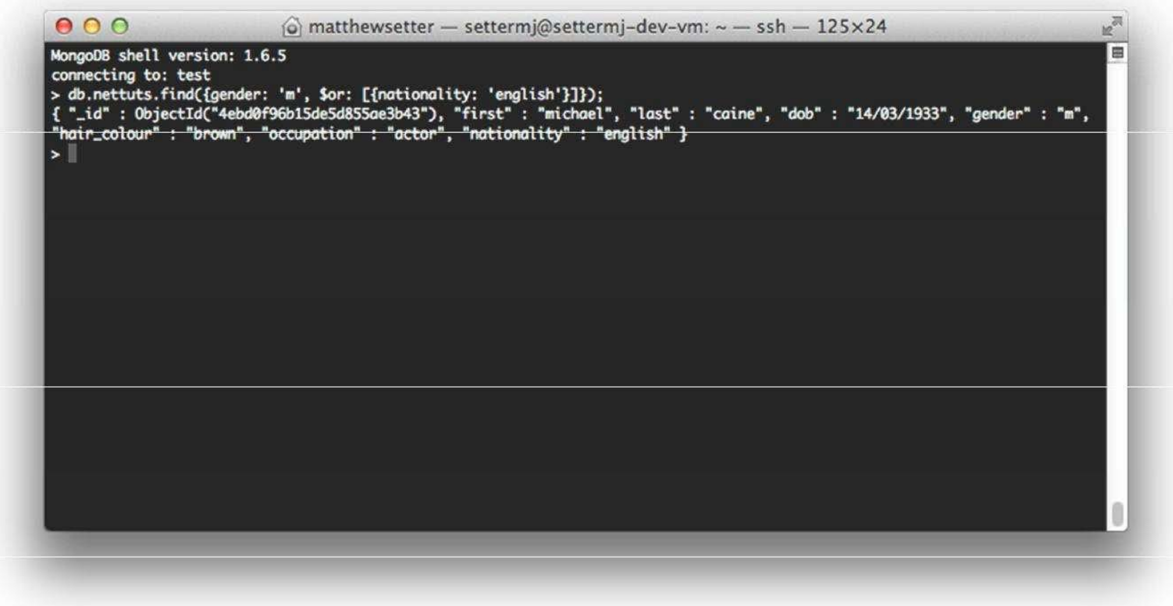
```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'm'});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b40"), "first" : "arnold", "last" : "schwarzenegger", "dob" : "03/06/1925", "gender"
 : "m", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b41"), "first" : "tony", "last" : "curtis", "dob" : "21/04/1978", "gender" : "m", "h
air_colour" : "brown", "occupation" : "developer", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
>
```

Searching with Multiple Criteria

Let's step it up a notch. We'll look for male actors that are English.

```
db.nettuts.find({gender: 'm', $or: [{nationality: 'english'}]});
```

Running that will return the following results:

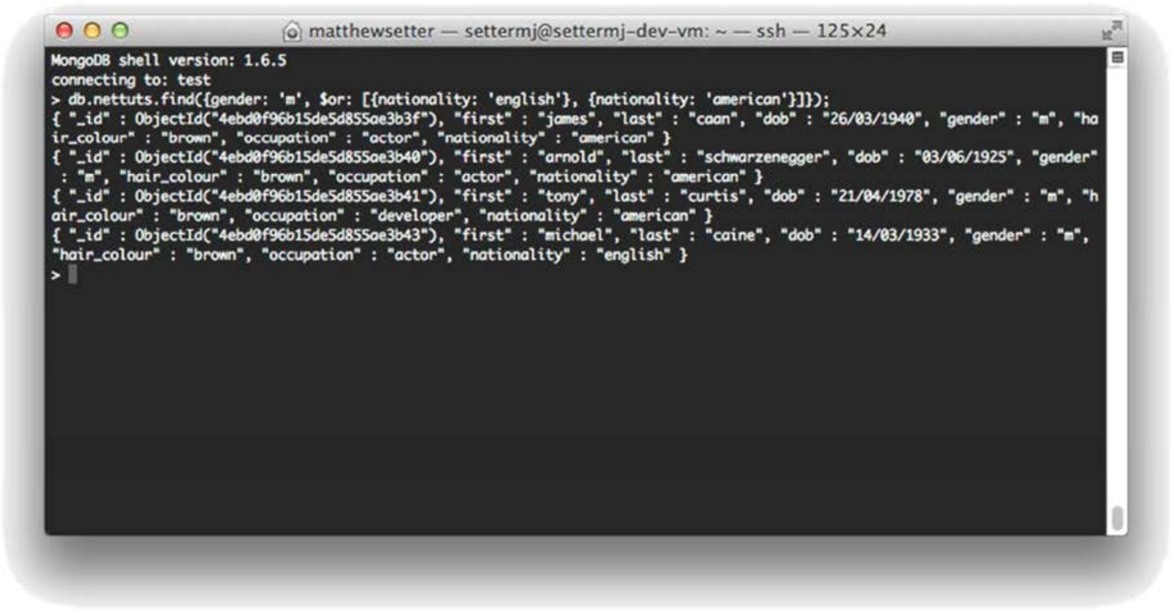


```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'm', $or: [{nationality: 'english'}]});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
>
```

What about male actors who are English or American. Easy! Let's adjust our earlier command to include the Americans:

```
db.nettuts.find({gender: 'm', $or: [{nationality: 'english'},
{nationality: 'american'}]});
```

For that query, we'll see the following results:



```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'm', $or: [{nationality: 'english'}, {nationality: 'american'}]});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b40"), "first" : "arnold", "last" : "schwarzenegger", "dob" : "03/06/1925", "gender"
 : "m", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b41"), "first" : "tony", "last" : "curtis", "dob" : "21/04/1978", "gender" : "m", "h
air_colour" : "brown", "occupation" : "developer", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
> 
```

Sorting Records

What if we want to sort records, say by first name or nationality? Similar to SQL, Mongo provides the `sort` command. The command, like the `find` command takes a list of options to determine the sort order.

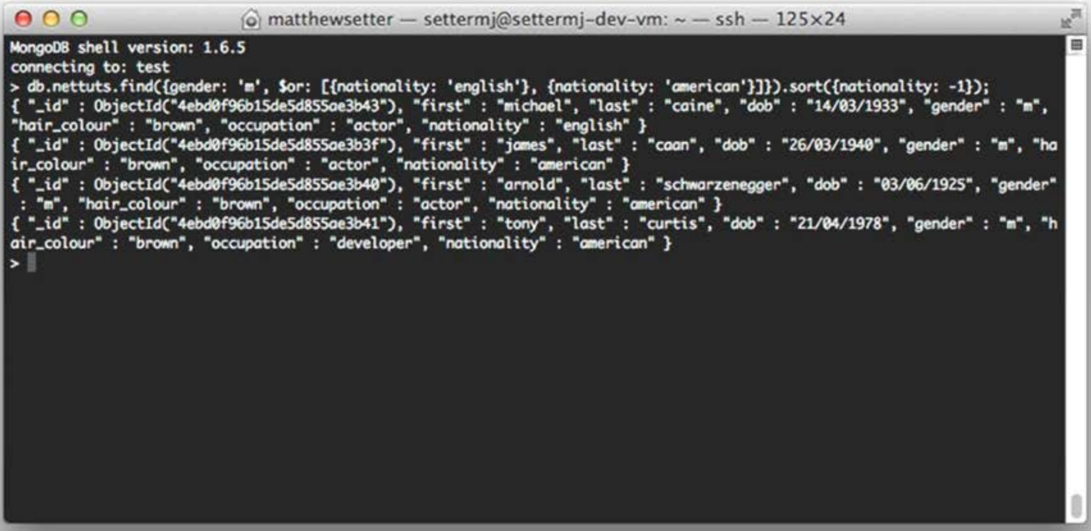
Unlike SQL, however we specify ascending and descending differently. We do that as follows:

- Ascending: -1
- Descending: 1

Let's have a look at an example:

```
db.nettuts.find({gender: 'm', $or: [{nationality: 'english'},
{nationality: 'american'}]}).sort({nationality: -1});
```

This example retrieves all male, English or American, actors and sorts them in descending order of nationality.

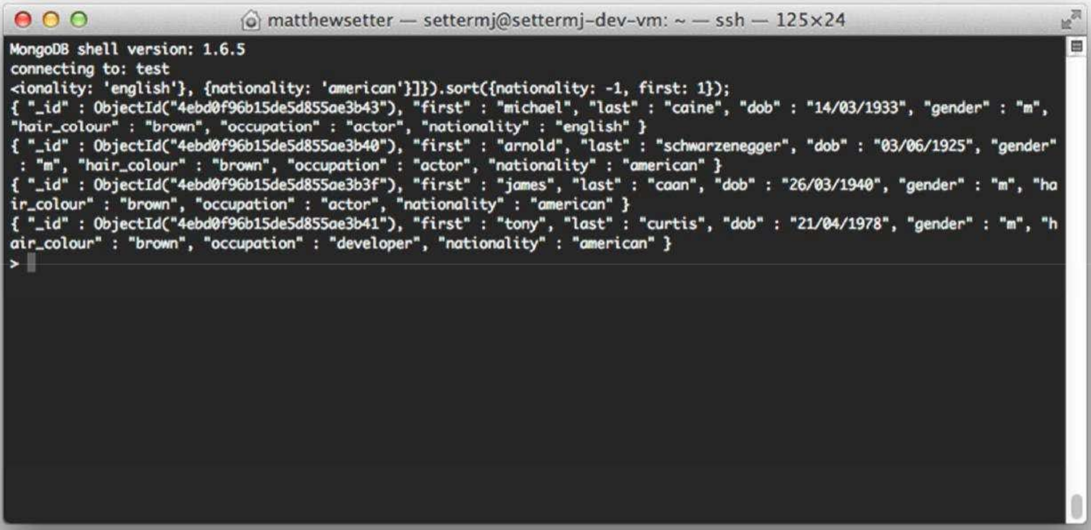


```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'm', $or: [{nationality: 'english'}, {nationality: 'american'}]}).sort({nationality: -1});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b40"), "first" : "arnold", "last" : "schwarzenegger", "dob" : "03/06/1925", "gender"
 : "m", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b41"), "first" : "tony", "last" : "curtis", "dob" : "21/04/1978", "gender" : "m", "h
air_colour" : "brown", "occupation" : "developer", "nationality" : "american" }
>
```

What about sorting by nationality in descending order and name in ascending order? No problem at all! Take a look at the query below, which is a modified version of the last one we ran.

```
db.nettuts.find({gender: 'm', $or: [{nationality: 'english'},
{nationality: 'american'}]}).sort({nationality: -1, first: 1});
```

This time we retrieve the following results et:



```
MongoDB shell version: 1.6.5
connecting to: test
<ionality: 'english'}, {nationality: 'american'}]}).sort({nationality: -1, first: 1});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b40"), "first" : "arnold", "last" : "schwarzenegger", "dob" : "03/06/1925", "gender"
 : "m", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b41"), "first" : "tony", "last" : "curtis", "dob" : "21/04/1978", "gender" : "m", "h
air_colour" : "brown", "occupation" : "developer", "nationality" : "american" }
>
```

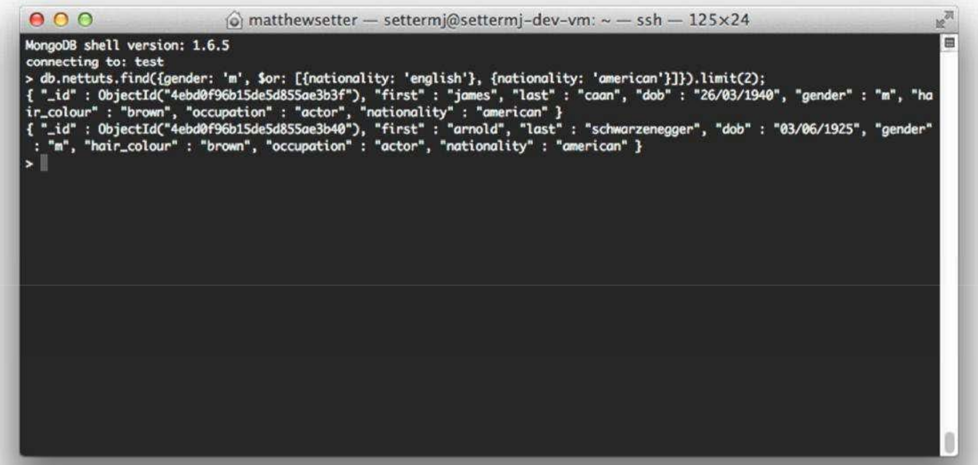
You can see that this time Arnold Schwarzenegger is placed before Tony Curtis.

Limiting Records

What if we had a pretty big data set (lucky us, we don't) and we wanted to limit the results to just 2? Mongo provides the limit command, similar to MySQL and allows us to do just that. Let's update our previous query and return just 2 records. Have a look at the following command:

```
db.nettuts.find({gender: 'm', $or: [{nationality: 'english'},
{nationality: 'american'}]}).limit(2);
```

From that command, we'll get the following results:

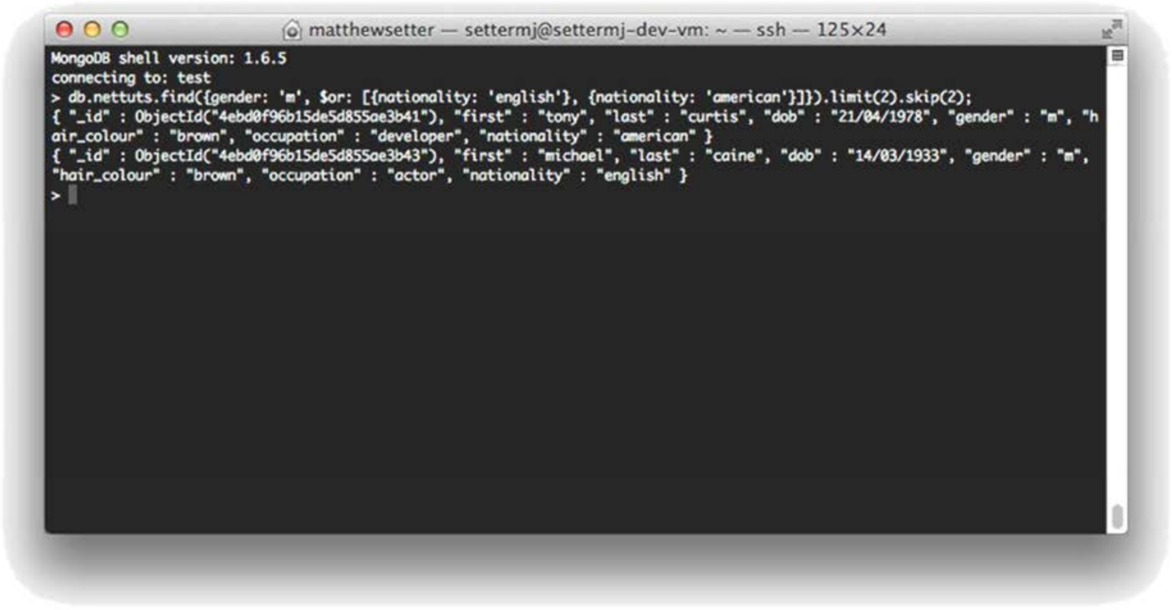


```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'm', $or: [{nationality: 'english'}, {nationality: 'american'}]}).limit(2);
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b40"), "first" : "arnold", "last" : "schwarzenegger", "dob" : "03/06/1925", "gender"
 : "m", "hair_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
> 
```

If we wanted the third and fourth records, i.e., skip over the first two? Once again, Mongo has a function for that. Have a look at the further customisation of the previous command:

```
db.nettuts.find({gender: 'm', $or: [{nationality: 'english'},
{nationality: 'american'}]}).limit(2).skip(2);
```

Running that will return the following results:



```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({gender: 'm', $or: [{nationality: 'english'}, {nationality: 'american'}]}).limit(2).skip(2);
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b41"), "first" : "tony", "last" : "curtis", "dob" : "21/04/1978", "gender" : "m", "h
air_colour" : "brown", "occupation" : "developer", "nationality" : "american" }
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b43"), "first" : "michael", "last" : "caine", "dob" : "14/03/1933", "gender" : "m",
"hair_colour" : "brown", "occupation" : "actor", "nationality" : "english" }
>
```

You can see from the original result set that the first two were skipped.

Updating Records

As expected, Mongo provides an option to update records as well. As with the `find` method and SQL queries, you need to specify the criteria for the record that you want to modify, then the data in that record that's going to be modified.

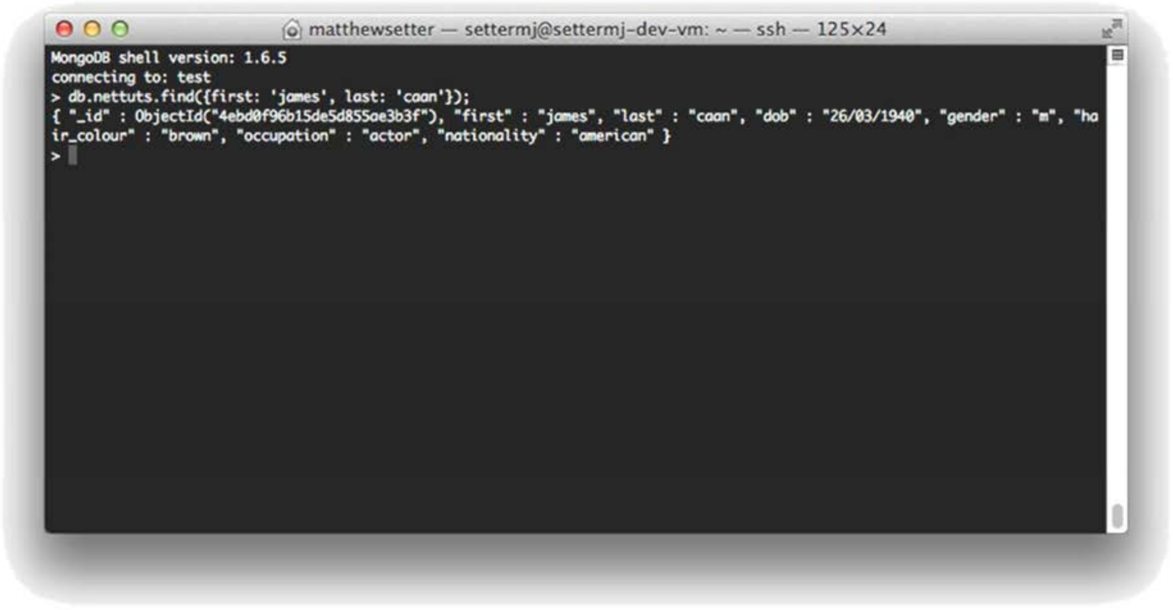
Let's say that we need to update the record for James Caan specifying that his hair is grey, not brown. Well for that we run the update function. Have a look at the example below:

```
db.nettuts.update({first: 'james', last: 'caan'}, {$set:
{hair_colour: 'brown'}});
```

Now when you run that, if all went well, there won't be anything to indicate whether it was a success or failure. To find out if the record was update properly, we need to search for it. So let's do that.

```
    db.nettuts.find({first: 'james', last: 'caan'});
```

After this you will see the following result:



```
MongoDB shell version: 1.6.5
connecting to: test
> db.nettuts.find({first: 'james', last: 'caan'});
{ "_id" : ObjectId("4ebd0f96b15de5d855ae3b3f"), "first" : "james", "last" : "caan", "dob" : "26/03/1940", "gender" : "m", "ha
ir_colour" : "brown", "occupation" : "actor", "nationality" : "american" }
> 
```

This shows that the update worked. One word of caution though, if you don't pass in the `$set` modifier, then you will *replace the record*, if it's available, instead of*updating it*. Be careful!

Important NOTE: if you want to update multiple documents in one single instruction, add {multi:true} in your update statement!

Advanced Queries

Previously we looked at basic queries and were introduced to selectors. Now we're going to get into more advanced queries, by building on the previous work in two key ways:

- Conditional Operators

Each of these successively provide us with more fine-grained control over the queries we can write and, consequently, the information that we can extract from our mongoDB databases.

Conditional Operators

Conditional operators are, as the name implies, operators to collection queries that refine the conditions that the query must match when extracting data from the database. There are a number of them, but today I'm going to focus on 9 key ones. These are:

- **$lt** – value must be less than the conditional
- **$gt** – value must be greater than the conditional
- **$lte** – value must be less than or equal to the conditional
- **$gte** – value must be greater than or equal to the conditional

- **$in** – value must be in a set of conditionals
- **$nin** – value must NOT be in a set of conditionals
  - **$not** – value must be equal to a conditional

Let's look at each one in turn. Open up your terminal and get ready to use the original database from the first part in this series (pre-modifications). To make this tutorial easier, we're going to make a slight alteration to the database. We're going to give each document in our collection an age attribute. To do that, run the following modification query:

```
db.nettuts.update({"first" : 'matthew'}, {"$set" : {"age" : 18 }});
db.nettuts.update({"first" : 'james'}, {"$set" : {"age" : 45 }});
db.nettuts.update({"first" : 'arnold'}, {"$set" : {"age" : 65 }});
db.nettuts.update({"first" : 'tony'}, {"$set" : {"age" : 43 }});
db.nettuts.update({"first" : 'jamie lee'}, {"$set" : {"age" : 22}});
db.nettuts.update({"first" : 'michael'}, {"$set" : {"age" : 45 }});
db.nettuts.update({"first" : 'judi'}, {"$set" : {"age" : 33 }});
```

All being well, you can run a 'find all' and you'll have the following output:

```
db.nettuts.find();

{ "_id" : ObjectId("4ef224be0fec2806da6e9b27"), "age" : 18, "dob"
: "21/04/1978", "first" : "matthew", "gender" : "m",
"hair_colour" : "brown", "last" : "setter", "nationality" :
"australian", "occupation" : "developer" }

{ "_id" : ObjectId("4ef224bf0fec2806da6e9b28"), "age" : 45, "dob"
: "26/03/1940", "first" : "james", "gender" : "m", "hair_colour" :
"brown", "last" : "caan", "nationality" : "american", "occupation"
: "actor" }

{ "_id" : ObjectId("4ef224bf0fec2806da6e9b29"), "age" : 65, "dob"
: "03/06/1925", "first" : "arnold", "gender" : "m", "hair_colour"
: "brown", "last" : "schwarzenegger", "nationality" : "american",
"occupation" : "actor" }

{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2a"), "age" : 43, "dob" :
"21/04/1978", "first" : "tony", "gender" : "m", "hair_colour" :
"brown", "last" : "curtis", "nationality" : "american", "occupation"
: "developer" }

{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2b"), "age" : 22, "dob" :
"22/11/1958", "first" : "jamie lee", "gender" : "f", "hair_colour" :
"brown", "last" : "curtis", "nationality" : "american", "occupation"
: "actor" }

{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2c"), "age" : 45, "dob"
: "14/03/1933", "first" : "michael", "gender" : "m", "hair_colour"
: "brown", "last" : "caine", "nationality" : "english",
"occupation" : "actor" }

{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2d"), "age" : 33, "dob"
: "09/12/1934", "first" : "judi", "gender" : "f", "hair_colour" :
"white", "last" : "dench", "nationality" : "english", "occupation"
: "actress" }
```

Try an EXERCISE:

Update the age of all the American people by adding 2 years

$lt/$lte

Now let's find all the actors who are less than 40. To do that, run the following query:

```
db.nettuts.find( { "age" : { "$lt" : 40 } } );
```

After running that query, you'll see the following output:

```
{ "_id" : ObjectId("4ef224be0fec2806da6e9b27"), "age" : 18, "dob"
: "21/04/1978", "first" : "matthew", "gender" : "m",
"hair_colour" : "brown", "last" : "setter", "nationality" :
"australian", "occupation" : "developer" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2b"), "age" : 22, "dob" :
"22/11/1958", "first" : "jamie lee", "gender" : "f", "hair_colour" :
"brown", "last" : "curtis", "nationality" : "american", "occupation" :
"actor" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2d"), "age" : 33, "dob"
: "09/12/1934", "first" : "judi", "gender" : "f", "hair_colour" :
"white", "last" : "dench", "nationality" : "english", "occupation"
: "actress" }
```

What about the ones who are less than 40 inclusive? Run the following query to return that result:

```
db.nettuts.find( { "age" : { "$lte" : 40 } } );
```

This returns the following list:

```
{ "_id" : ObjectId("4ef224be0fec2806da6e9b27"), "age" : 18, "dob"
: "21/04/1978", "first" : "matthew", "gender" : "m",
"hair_colour" : "brown", "last" : "setter", "nationality" :
"australian", "occupation" : "developer" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2b"), "age" : 22, "dob" :
"22/11/1958", "first" : "jamie lee", "gender" : "f", "hair_colour" :
"brown", "last" : "curtis", "nationality" : "american", "occupation"
: "actor" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2d"), "age" : 33, "dob"
: "09/12/1934", "first" : "judi", "gender" : "f", "hair_colour" :
"white", "last" : "dench", "nationality" : "english", "occupation"
: "actress" }
```

$gt/$gte

Now let's find all the actors who are older than 47. Run the following query to find that list:

```
db.nettuts.find( { 'age' : { '$gt' : 47 } } );
```

You'll then get the following output:

```
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b29"), "age" : 65, "dob"
: "03/06/1925", "first" : "arnold", "gender" : "m", "hair_colour"
: "brown", "last" : "schwarzenegger", "nationality" : "american",
"occupation" : "actor" }
```

What about inclusive of 40?

```
db.nettuts.find( { 'age' : { '$gte' : 47 } } );
```

As there's only one person over 47, the data returned doesn't change.

$in/$nin

What about finding information based on a list of criteria? These first ones have been ok, but arguably, quite trivial. Let's now look to see which of the people we have are either actors or developers. With the following query, we'll find that out (to make it a bit easier to read, we've limited the keys that are returned to just first and last names):

```
db.nettuts.find( { 'occupation' : { '$in' : [ "actor",
"developer" ] } }, { "first" : 1, "last" : 1 } );
```

This query, yields the following output:

```
{ "_id" : ObjectId("4ef224be0fec2806da6e9b27"), "first" : "matthew",
"last" : "setter" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b28"), "first" : "james",
"last" : "caan" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b29"), "first" : "arnold",
"last" : "schwarzenegger" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2a"), "first" : "tony",
"last" : "curtis" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2b"), "first" : "jamie
lee", "last" : "curtis" }
{ "_id" : ObjectId("4ef224bf0fec2806da6e9b2c"), "first" : "michael",
"last" : "caine" }
```

You can see that we can get the inverse of this by using `$nin` just as simply.

Let's make this a bit more fun and combine some of the operators. Let's say that we want to look for all the people, who are either male or developers, they're less than 40 years of age.

Now that's a bit of a mouthful, but with the operators that we've used so far – quite readily achievable. Let's work through it and you'll see. Have a look at the query below:

```
db.nettuts.find( { $or : [ { "gender" : "m", "occupation" :
"developer" } ], "age" : { "$gt" : 40 } }, { "first" : 1, "last" :
1, "occupation" : 1, "dob" : 1 } );
```

You can see that we've stipulated that the either the gender can be male or the occupation can be a developer in the `$or` condition and then added an and condition of the age

being greater than 4.

For that, we get the following results:

```
{ "_id" : ObjectId("4ef22e522893ba6797bf8cb6"), "first" :
"matthew", "last" : "setter", "dob" : "21/04/1978", "occupation"
: "developer" }
{ "_id" : ObjectId("4ef22e522893ba6797bf8cb9"), "first" : "tony",
"last" : "curtis", "dob" : "21/04/1978", "occupation" : "developer"}
```

Pipeline

Using pipeline, you can aggregate data using built-in functionality of mongoDB.

A pipeline is a sequence of functions called inside the aggregate mongoDB methods. In a pipeline, the output of a function is the input for the next one.

It can be used to compute complex queries requiring aggregation of data.

A pipeline is implemented with the mongoDB command:

```
db.collection_name.aggregate([
  function1,


  function2,
…. ])
```

A typical pipeline is a match function followed by a group function . You might have other functions, and if the match function is omitted, all the collection is used.

Look at the example here: http://docs.mongodb.org/manual/core/aggregation-pipeline/

Inside the group function you can use a lot of aggregation operators such as $sum, $max, $min …

Here is an example using max:

http://docs.mongodb.org/manual/reference/operator/aggregation/max/

$sort , $limit, $skip can be used in a pipeline.

For istance, to get only the first 5 elements:

```
db.nettuts.aggregate([  { $limit : 5 } ]);
```

To sort the elements by first name ascending and take only first one (=find the minimum value!)

```
db.nettuts.aggregate([ { $sort : {first : 1} }, { $limit : 5 } ]);
```

//The following will take all the persons above 37, group by nationalities and count them

```
db.nettuts.aggregate( [ { $match: { 'age' : { '$gte' : 37 }}},
{ $group: { _id: '$nationality', total : { $sum : 1} }}] );
```

The command $out can be used to redirect the output of the aggregation into a collection

```
{$out : “my_collection”}
```

Mongo Documentation on Pipeline:

Aggregation pipeline: http://docs.mongodb.org/manual/core/aggregation-pipeline/

Accumulator operators:
http://docs.mongodb.org/manual/reference/operator/aggregation- group/

$group: http://docs.mongodb.org/manual/reference/operator/aggregation/group/

Exercises with Pipeline:

Create a collections bygender that count the person by gender using a pipeline

Create a collections bygender that display the average age of the persons by gender using a pipeline. Use the function $avg

Select the older male and the older male using a pipeline

DESIGN EXERCISE (you are required to submit this as part of your labwork)

You are given the following 3 tables of a relational database
Table Students

Student_ID	Name	Surname	Nationality	Age
1	Mary	Murray	Irish	45
2	Bill	Bellichick	American	32
3	Tom	Brady	Canadian	22
4	John	Bale	English	24

Table Courses

Course_ID	Course_Name	Credits
DB	Databases	10
MA	Maths	5
PR	Programming	15

Table Marks

Student_ID	Course_ID	Mark	ExamDate
1	DB	56	10/10/2011
1	MA	76	09/11/2012
1	PR	45	02/07/2014
2	DB	55	10/10/2011
2	MA	87	09/11/2012
2	PR	45	10/10/2011
3	DB	34	09/11/2012
3	MA	56	02/07/2014
4	DB	71	10/10/2011
4	MA	88	10/10/2011
4	PR	95	09/11/2012

Design a MongoDB data schema for the above 3 tables by using one collection containing embedded JSON objects.

Create the collection and insert the data in MongoDB using your database.

Note the redundancy of the data if you store the data in a single JSON.

Name the collection your_studentid_schema

Execute the following query and save them into a collection

1. Find all the students that failed
2. Find the number of people that passed each exam
3. Find the student with the highest average mark