

COMP835

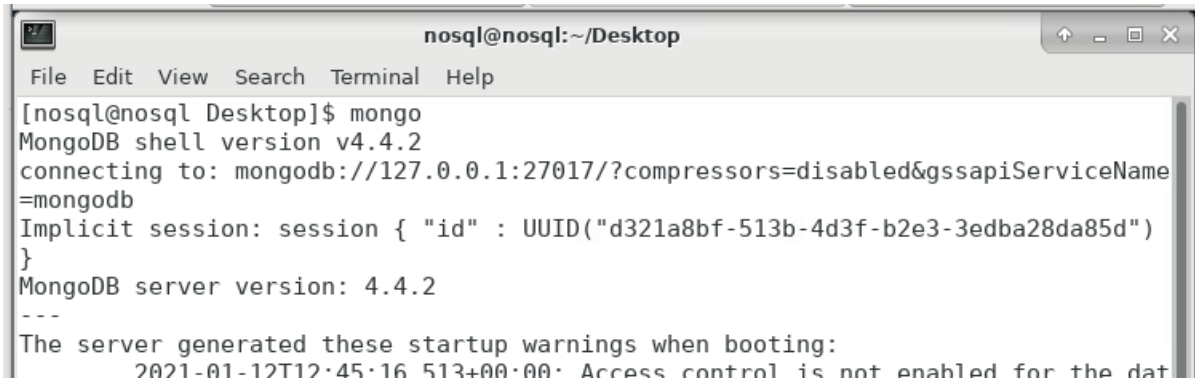
Lab 6. Introduction to MongoDB database

Overview

In this lab you will work with different data types in MongoDB data store.

Lab 6.1 Ensure your MongoDB server is running.

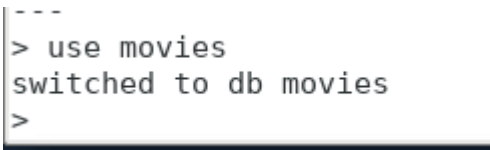
Open in Terminal and start MongoDB, by using command : **mongo**

A screenshot of a terminal window titled 'nosql@nosql: ~/Desktop'. The terminal shows the command 'mongo' being executed, which starts the MongoDB shell (version 4.4.2). It displays the connection details: 'connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb'. It also shows an implicit session ID and the MongoDB server version (4.4.2). At the bottom, it lists startup warnings, including one about access control not being enabled for the data directory.

```
nosql@nosql: ~/Desktop
File Edit View Search Terminal Help
[nosql@nosql Desktop]$ mongo
MongoDB shell version v4.4.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d321a8bf-513b-4d3f-b2e3-3edba28da85d") }
MongoDB server version: 4.4.2
---
The server generated these startup warnings when booting:
  2021-01-12T12:45:16.513+00:00: Access control is not enabled for the data directory
```

Create a database called **movies**

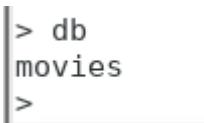
The command for this is: **use movies**, and press Enter.

A screenshot of the MongoDB shell showing the command 'use movies' being entered. The prompt is '>'. The output is 'switched to db movies', and the prompt returns to '>'.

```
---
> use movies
switched to db movies
>
```

You will notice that it will now indicate that you have switched to the db called **movies**.

You can confirm that the database has now been selected by entering the command **db** and then pressing enter.

A screenshot of the MongoDB shell showing the command 'db' being entered. The prompt is '>'. The output is 'movies', and the prompt returns to '>'.

```
> db
movies
>
```

Lab 6.2 Entering data into MongoDB.

6.2.1

We now want to enter some documents into the **movies** database and then add these to a collection call **video_records**.

Our first document will have the following data in it:

- `_id: 1`
- Title: "Les Misérables"
- Director: "Tom Hooper"
- Runtime: 158

There are different ways to enter the document into mongoDB.

First method is to create a variable/document and then use the name of the document in the insert command. Create document called **lesmis** with the following data:

```
lesmis = {_id: 1,
          Title: "Les Miserables",
          Director: "Tom Hooper",
          Runtime: 158
        }
```

```
> lesmis={_id:1,
... Title:"Les Miserables",
... Director:"Tom Hooper",
... Runtime:158
... }
```

Note: Mongo will display your document in response:

```
{
  "_id" : 1,
  "Title" : "Les Miserables",
  "Director" : "Tom Hooper",
  "Runtime" : 158
}
```

After you have created your **lesmis** document, you must insert it into a collection which we have called **video_records**. Use the following command and press enter:

```
> db.video_records.insert(lesmis)
```

Mongo shows the result `WriteResult({ "nInserted" : 1 })`

Verify that your document has been added by using the **find()** command:

```
> db.video_records.find()
```

```
{ "_id" : 1, "Title" : "Les Miserables", "Director" : "Tom Hooper", "Runtime" :
158 }
```

By inserting a first document, the collection **video_records** has been created as well.

You can display collections by using **show collections** command:

```
> show collections
video_records
```

6.2.2

Now create the second document with new document name **skyfall** for the following records details:

```
_id=2
Title= "Skyfall"
Director= "Sam Mendes"
RunTime= 137
```

And then insert the document into **video_records** collection.

Query all documents in the collection video_records. You should get two documents displayed.

```
> db.video_records.find()
{ "_id" : 1, "Title" : "Les Miserables", "Director" : "Tom Hooper", "Runtime" : 158 }
{ "_id" : 2, "Title" : "Skyfall", "Director" : "Sam Mendes", "Runtime" : 137 }
>
```

Note: If are you in the middle of the entering command and you need to exit, use <Ctrl-C>

6.2.3

Now you will try another method of inserting the document using **db.video_records.insert** method with the new document specified inside the method:

```
> db.video_records.insert({
...   _id:3,
...   Title:"The Tall Men",
...   Director: "Raoul Walsh",
...   Runtime:117
... })
```

Verify that your collection **video_records** has three documents now. Which command will you use?

Both methods of inserting data are similar, but the use of the document name has a benefit: you can query the data using the document name:

```
> db.video_records.find(lesmis)
{ "_id" : 1, "Title" : "Les Miserables", "Director" : "Tom Hooper", "Runtime" : 158 }
```

6.2.4

Now enter a new document which has some intentional data mistakes in it. Leave them as is.

You can use either method : creating a document first and then inserting it or inserting on fly.

```
_id:4,
Title: "Sam Mendes",
Director: "Skyfall",
Runtime: 143,
Price: 9.99,
Year: 2010
```

Also note the structure of this new document is different. Extra data (**Price** and **Year**) being included before inserting it the **video_records** collection and the database doesn't complain about different structure.

Insert another document for director Sam Mendes:

```
_id:5,
```

Title: "Sam Mendes",
Director: "Spectre",
Runtime: 160,
Price: 12.99,
Year: 2015

As you probably noticed that documents 4 and 5 have mistakes: "Sam Mendes" value is assigned to a wrong key. This was done deliberately, so leave it as it is for the moment.

Verify that you have all 5 documents in the collection:

```
> db.video_records.find()
{ "_id" : 1, "Title" : "Les Miserables", "Director" : "Tom Hooper", "Runtime" : 158 }
{ "_id" : 2, "Title" : "Skyfall", "Director" : "Sam Mendes", "Runtime" : 137 }
{ "_id" : 3, "Title" : "The Tall Men", "Director" : "Raoul Walsh", "Runtime" : 117 }
{ "_id" : 4, "Title" : "Sam Mendes", "Director" : "Skyfall", "Runtime" : 143, "Price" : 9.99, "Year" : 2010 }
{ "_id" : 5, "Title" : "Sam Mendes", "Director" : "Spectre", "Runtime" : 160, "Price" : 12.99, "Year" : 2015 }
```

Use pretty() method to produce better formatted output:

```
> db.video_records.find().pretty()
{
  "_id" : 1,
  "Title" : "Les Miserables",
  "Director" : "Tom Hooper",
  "Runtime" : 158
}
{
  "_id" : 2,
  "Title" : "Skyfall",
  "Director" : "Sam Mendes",
  "Runtime" : 137
}
{
  "_id" : 3,
  "Title" : "The Tall Men",
  "Director" : "Raoul Walsh",
  "Runtime" : 117
}
{
  "_id" : 4,
  "Title" : "Sam Mendes",
  "Director" : "Skyfall",
  "Runtime" : 143,
  "Price" : 9.99,
  "Year" : 2010
}
{
  "_id" : 5,
  "Title" : "Sam Mendes",
  "Director" : "Spectre",
  "Runtime" : 160,
  "Price" : 12.99,
  "Year" : 2015
}
```

Lab 6.3. Querying the Video Records Data with 'where' conditions.

1. Select all documents from the collection `video_records` that have an `_id` of 1

```
> db.video_records.find({_id:1}).pretty()
{
  "_id" : 1,
  "Title" : "Les Miserables",
  "Director" : "Tom Hooper",
  "Runtime" : 158
}
```

2. Select all documents from the collection **video_records** that have a director Raoul Walsh.

```
> db.video_records.find({Director:"Raoul Walsh"}).pretty()
{
  "_id" : 3,
  "Title" : "The Tall Men",
  "Director" : "Raoul Walsh",
  "Runtime" : 117
}
```

3. Select all documents from the video recording collection which have a runtime longer than 140 minutes.

Use the lecture notes to get the correct syntax.

You should get this output:

```
{
  "_id" : 1,
  "Title" : "Les Miserables",
  "Director" : "Tom Hooper",
  "Runtime" : 158
}
{
  "_id" : 4,
  "Title" : "Sam Mendes",
  "Director" : "Skyfall",
  "Runtime" : 143,
  "Price" : 9.99,
  "Year" : 2010
}
{
  "_id" : 5,
  "Title" : "Sam Mendes",
  "Director" : "Spectre",
  "Runtime" : 160,
  "Price" : 12.99,
  "Year" : 2015
}
```

4. Select all documents from the video recording collection which have a runtime between 110 and 140 minutes.

You should get this output:

```
{
  "_id" : 2,
  "Title" : "Skyfall",
  "Director" : "Sam Mendes",
  "Runtime" : 137
}
{
  "_id" : 3,
  "Title" : "The Tall Men",
  "Director" : "Raoul Walsh",
  "Runtime" : 117
}
```

5. Select all documents from the video recording collection which have a runtime of less than 140 and not equal to 117. You should get this output:

```
{
  "_id" : 2,
  "Title" : "Skyfall",
  "Director" : "Sam Mendes",
  "Runtime" : 137
}
```

6. Select all documents from the collection **video_records** that DO NOT have a director Raoul Walsh.

You should get this output:

```
{
  "_id" : 1,
  "Title" : "Les Miserables",
  "Director" : "Tom Hooper",
  "Runtime" : 158
}
{
  "_id" : 2,
  "Title" : "Skyfall",
  "Director" : "Sam Mendes",
  "Runtime" : 137
}
{
  "_id" : 4,
  "Title" : "Sam Mendes",
  "Director" : "Skyfall",
  "Runtime" : 143,
  "Price" : 9.99,
  "Year" : 2010
}
{
  "_id" : 5,
  "Title" : "Sam Mendes",
  "Director" : "Spectre",
  "Runtime" : 160,
  "Price" : 12.99,
  "Year" : 2015
}
```

Lab 6.4 Updating and Removing data.

We left deliberately some mistakes in the documents 4 and 5. So, now we would like to fix them by updating the document 4 and removing document 5 from the collection **video_records**.

Hint: use lecture notes for guidelines.

6.4.1

Update document 4 using UPDATE command.

We need to change the Title of the movie to "Skyfall" and then Director to "Sam Mendes".

```
> db.video_records.update({  
... _id:4},  
... {$set:{Title:"Skyfall"}})
```

You should get a confirmation:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Now update the Director to Sam Mendes

Check your data:

```
> db.video_records.find({_id:4}).pretty()  
{  
  "_id" : 4,  
  "Title" : "Skyfall",  
  "Director" : "Sam Mendes",  
  "Runtime" : 143,  
  "Price" : 9.99,  
  "Year" : 2010  
}
```

6.4.1

Then, delete document 5 using REMOVE command.

You should get a confirmation:

```
WriteResult({ "nRemoved" : 1 })
```

Now, check to see if the collection has the correct data using the command:

```
db.video_records.find().pretty()
```

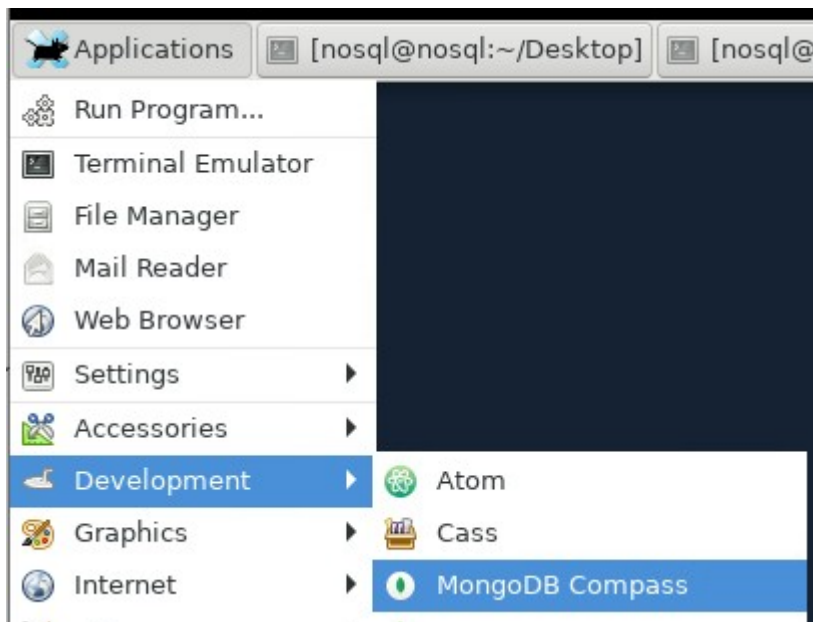
You should have only 4 documents in your collection.

Lab 6.5 Using MongoDB Compass.

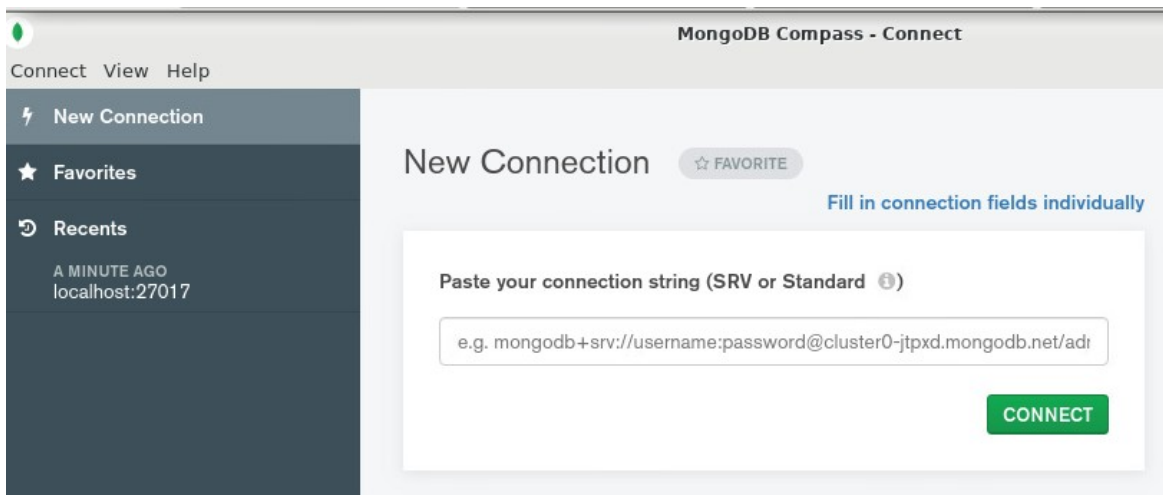
6.5.1

In this step you will use GUI that MongoDB provides with every MongoDB database, called MongoDB Compass.

Navigate to Applications->Development-> MongoDB Compass.



On the New Connection screen, just click on Connect without putting anything into connection field:



On the next screen, put password= **nosql** and press Unlock







It will open the home page for all your databases:


Databases

Performance

CREATE DATABASE

Database Name ^	Storage Size	Collections	Indexes	
admin	32.0KB	0	1	
config	36.0KB	0	2	
local	44.0KB	1	1	
movies	36.0KB	1	1	

Click on movies database, it will open all collections you have in that database:

Collections						
CREATE COLLECTION						
Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
video_records	4	89.5 B	358.0 B	1	36.0 KB	

Click on video_records to see all documents.

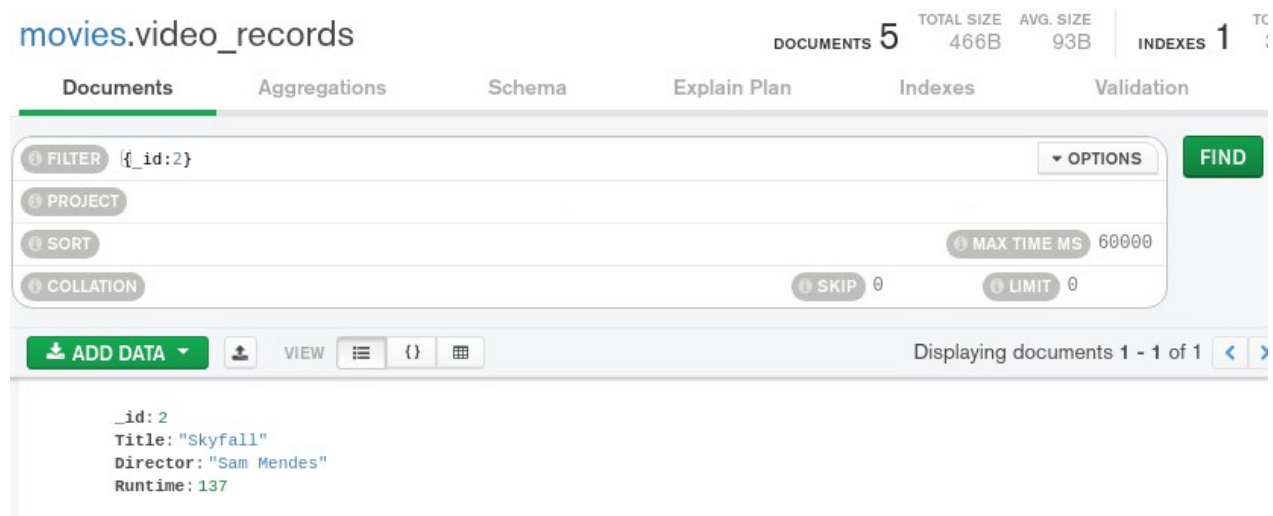
movies.video_records		DOCUMENTS 4	
Documents	Aggregations	Schema	Explain Plan
FILTER			
ADD DATA			
VIEW			
<pre> _id: 1 Title: "Les Miserables" Director: "Tom Hooper" Runtime: 158 </pre>			
<pre> _id: 2 Title: "Skyfall" Director: "Sam Mendes" Runtime: 137 </pre>			
<pre> _id: 3 Title: "The Tall Men" Director: "Raoul Walsh" Runtime: 117 </pre>			
<pre> _id: 4 Title: "Skyfall" Director: "Sam Mendes" Runtime: 143 Price: 9.99 Year: 2010 </pre>			

6.5.2

Now you need to perform some queries with the 'where' conditions, as you did in the command line. MongoDB Compass uses **Filter** for that.

1. We want to see only document #2:

Click on the button **Options** and put the condition in the **FILTER** and the press **FIND**:



Using Lab 6.3 steps as you guide, perform the same queries in the MongoDB Compass:

2. Select all documents from the collection **video_records** that have a director Raoul Walsh.

You should get:

```
_id: 3
Title: "The Tall Men"
Director: "Raoul Walsh"
Runtime: 117
```

3. Select all the documents from the video recording collection which have a runtime longer than 140 minutes.

You should get this:

```
_id: 1
Title: "Les Miserables"
Director: "Tom Hooper"
Runtime: 158
```

```
_id: 4
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 143
Price: 9.99
Year: 2010
```

4. Select all documents from the video recording collection which have a runtime between 110 and 140 minutes. The result should be:

```
_id: 2
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 137
```

```
_id: 3
Title: "The Tall Men"
Director: "Raoul Walsh"
Runtime: 117
```

5. Select all documents from the video recording collection which have a runtime of less than 140 and not equal to 117. You should get this output:

```
_id: 2
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 137
```

6. Select all documents from the collection video_records that DO NOT have a director Raoul Walsh **OR** their Runtime is longer than 140 minutes. You should get this output:

```
_id: 1
Title: "Les Miserables"
Director: "Tom Hooper"
Runtime: 158
```

```
_id: 2
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 137
```

```
_id: 4
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 143
Price: 9.99
Year: 2010
```

6.5.3.

You can insert a document with more complex data type, for example, an array.

Insert a document with the following data:

```
_id: 5,
Title: "True Grit",
Director: ["Joel Coen", "Ethan Coen"],
Runtime: 110,
Year: 2010
```

Then perform some query either in the command line or in MongoDB Compass.

Optional Lab 6.6 Using projection, sort, and skip.

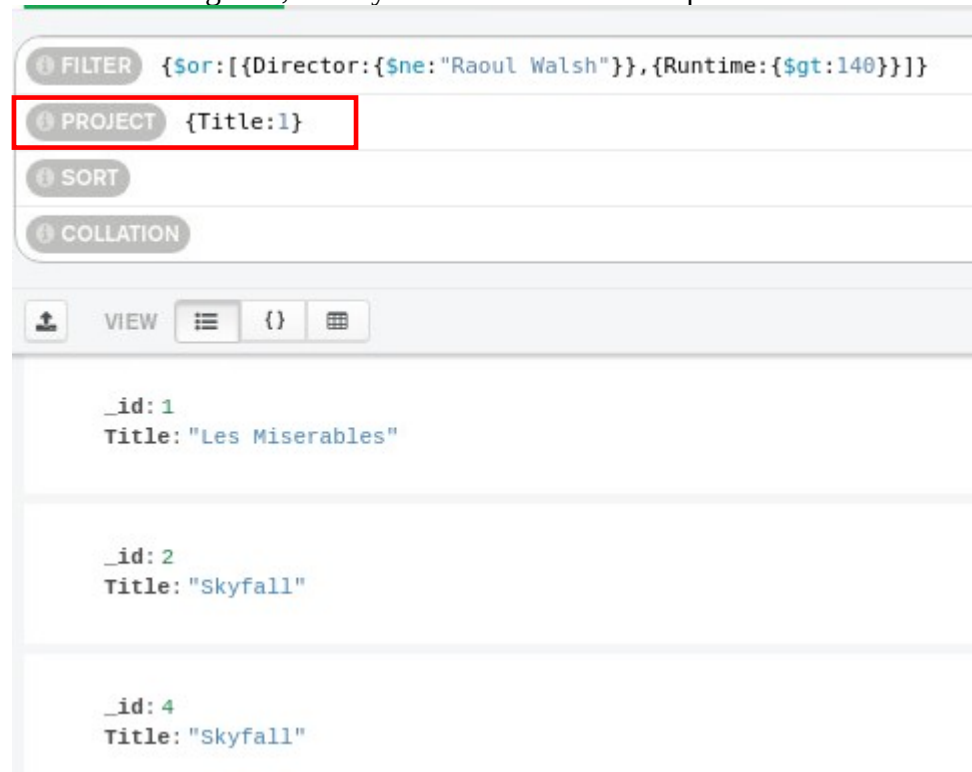
6.6.1

We will be covering projection, sorting data, skipping data, etc in the next week lecture.

But if you want to try, you can use MongoDB Compass and put conditions into PROJECT field to specify which fields should be included into the results. In order to sort your output you will include condition into SORT.

All condition should follow the same rule, key-value pairs.

For example, in our last query, if I include condition **{Title:1}** (1 means true) in the PROJECT area, it will tell MongoDB, to only include Title in the output:



Notice, that _id is still included, as it is mandatory field. If you do not want to display it, you would need to add explicit condition `{_id:0}` (0 means false)

FILTER `{ $or: [{ Director: { $ne: "Raoul Walsh" } }, { Runtime: { $gt: 140 } }] }`

PROJECT `{ Title: 1, _id: 0 }`

SORT

COLLATION

VIEW `{ }`

Title: "Les Miserables"

Title: "Skyfall"

Title: "Skyfall"

6.6.1

Similarly, you can include a key-value pair to notify MongoDB how you would like your output be sorted. For example, if you would like the output to be sorted by the Runtime, you need to add `{ Runtime: 1 }` in the SORT area:

FILTER `{ $or: [{ Director: { $ne: "Raoul Walsh" } }, { Runtime: { $gt: 140 } }] }`

PROJECT

SORT `{ Runtime: 1 }`

COLLATION

ADD DATA VIEW `{ }`

`_id: 2`
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 137

`_id: 4`
Title: "Skyfall"
Director: "Sam Mendes"
Runtime: 143
Price: 9.99
Year: 2010

`_id: 1`
Title: "Les Miserables"
Director: "Tom Hooper"
Runtime: 158

6.6.1

If you specify LIMIT as 1, it will mean to return only the first document that matches your condition:

