Exploring MongoDB

Start by opening a command prompt or terminal and connect to your MongoDB instance.

Work through the following commands, taking screen captures where noted for submission.

Commands:

- Show databases available

```
show dbs
```

- connect to a particular database

```
use movie
```

- MongoDB doesn't use tables of records but rather 'collections' of 'documents'. Consider these as roughly equivalent. Type the following to see the collections available in the movie database and take a screen capture of the command prompt or terminal window to show the results.

```
show collections
```

- MongoDB documents are modelled as json data. So instead of fields, you have key-value pairs. You can see this in the result of querying the database. The equivalent of selecting data in MongoDB is .find() using the following full syntax.

```
db.<collection_name>.find() -- same as select * from <table>
```

try the following:

```
db.movies.find()
```

the output from this is hard to read. To get better looking results in proper json style, try the following:

```
db.movies.find().pretty()
```

this syntax is roughly equivalent to select * from <tablename>. We can provide parameters to the find() method to narrow results as we would with SQL. This is done by adding criteria to the find method, enclosed in {}. For example, we can select all of the movies from 2013 by adding {year:2013} to the find method:

```
db.movies.find({year:2013}).pretty()
```

also note that semicolons aren't necessary.

We can also do conditional criteria using special builtin operators such as $lt or $gt. Try adding the following to find() and take a screen capture of the result.

```
{year:{$lt:2000}}
```

If you get a lot of results, you can limit the results using limit(). Run the last query but add `limit(5)` on the end using dot notation. This shows something called method chaining which is used extensively in MongoDB's query language.

You can also choose to display only certain keys of a document. The first {} of the find command is assumed to be the criteria. You can add another {} containing keys you wish to see displayed or explicitly not shown. Simply list the field with a colon and then either 0 or 1 depending if you the key hidden or shown. Try the following and take a screen capture:

```
db.movies.find({year:{$lt:2000}},{title:1,_id:0}).pretty()
```

This syntax means 'only show the title field'. The _id field is special as it is autogenerated by mongoDB as an internal primary key field and populated automatically. To hide it, you have to do so explicitly.

Try the following to search with more than one criteria:

```
db.movieDetails.find({actors:"Leonard Nimoy",year:1982}).pretty()
```

We can also perform DDL-type operations as well. However MongoDB is 'schema-less', meaning it has no predetermined structure(there is still a defacto structure). It also implies that documents don't all have to be the same or all have the same keys. Run the `show collections` command and then execute the following:

```
db.ratings.insert({"title":"Star Trek II: The Wrath of Khan","year":1982,"rated":"PG"})
```

Now run `show collections` again and take a screen capture of the result. Notice that a new collection has been created, not using a creation command but executing an insert. MongoDB does the creation on the fly according to the keys present in the document you inserted. Now try the following:

```
db.ratings.insert({"title":"Star Trek III","year":1990,"rated":"PG","rating":8})
```

Do `db.ratings.find()` to see what these two documents look like. Take a screen capture. Now execute the following, run the find() again and take a screen capture.

```
db.ratings.update({year:1982},{$set:{"rating":9}})
```

See how the document is updated with the new field but see how previously, it was perfectly fine for two documents to have a different structure. MongoDB essentially does DDL on demand. This means that you can modify your 'schema' dynamically. This has made MongoDB a very popular database option for web applications.

You might notice that brackets can get confusing as they are uses quite a lot so formatting is important.

```
db.ratings.insert({"title":"Star Trek III","year":1990,"rated":"PG","rating":8})
```

can be better formatted as

```
db.ratings.insert(
	{
		"title":"Star Trek III",
		"year":1990,
		"rated":"PG",
		"rating":8
	})
```

This isn't bad… more complex queries can be done that can be very error prone because of the number of different types of brackets that can be required.