Exporting and importing MongoDB data

A common need with databases is to be able to move the data from one location to another, perhaps when setting up a new application you need to move your database into production for the first time, move the data from one provider to another or run backups.

Exporting with MongoDump

When you install Mongo it will bring with it a utility called *mongodump*. This doesn't delete anything; what it does do is create a binary export of a Mongo database. As we'll see soon it creates `.bson` files.

So, if we want to export the data from our database, we start with the `mongodump` command - in the command line not the Mongo shell - and pass it the name of the database to work with.

```
mongodump --db meantest
```

Note: if you don't supply the name of a database it will run on all of the databases in the current instance.

By default, mongodump saves output files in a directory named dump in the current working directory.

We can check it out - on the Mac - by going here in the command line:

```
cd ~/dump
ls
```

Checkout the `meantest` folder and the contents.

New in this version of MongoDB - 3.2 - is the ability to gzip the dumps by adding a flag to the command.

First of all delete the `meantest` folder:

```
cd ~/
rm -rf dump/meantest
```

Be *very* careful with the `rm -rf` command!

Then run the dump command again passing the `--gzip` option.

```
mongodump --db meantest --gzip
```

There are loads of options you can specify here, but we won't go through them all. You can export specific collections from a database, create point-in-time snapshots, specify credentials to connect to remote servers and much much more. But for now, we'll stick with this export and move onto importing the data.

Restoring dumped data with MongoRestore

The utility for importing data is called `mongorestore`. We'll build up a query to restore our exported data to a new database:

1. Start by calling `mongorestore`
2. Use the `--db` flag to specify the name of the database to restore to (Mongo will create this if it doesn't already exist
3. Tell `mongorestore` that we're looking for gzipped files by passing the `--gzip` flag
4. The final argument of the `mongorestore` command is the path to the directory containing the exported data.

```
mongorestore --db mean2 --gzip  meantest/
```

Now if we jump back to the Mongo shell we can run `show dbs` and see this new database has been created.

Inserts only, no updates

Note that `mongorestore` only does *inserts* and not *updates*. That is to say, if you restore to an existing database, and a document in the database and a document in the restore data both have the same *_id* value, the document in the database will **not** be updated, regardless of whether the restore data contains others different values.

Exporting JSON with mongoexport

As well as BSON, MongoDB can also export JSON using a utility called `mongoexport`.

This will export a JSON representation of the database. Note that as a rule - particularly for backing up or moving data - MongoDB recommends the "dump and restore" approach as BSON can contain more rich data than JSON.

Nevertheless, `mongoexport` still has its uses, sometimes a JSON representation of the data is very useful - it's what we've been using so far in the application development.

Basic output to console

Need to specify the name of the database `--db` and the collection `--collection` to export.

```
mongoexport --db meantest --collection tech
```

This will send the data to the console log.

Send to a file

Add the `--out` parameter.

```
mongoexport --db meantest --collection tech --out MEAN/api/data/tech.json
```

Check the file in Sublime text.

Create as array

This document is technically invalid, as it's a collection of objects. You can add a `--jsonArray` flag to wrap them up in an array.

```
mongoexport --db meantest --collection tech --out MEAN/api/data/tech.json --jsonArray
```

Make output pretty

Finally, if you want to have the file easily human readable you can make it pretty.

```
mongoexport --db meantest --collection tech --out MEAN/api/data/tech.json --jsonArray --pre
```

End up with something like this:

```
[{
    "_id": {
        "$oid": "56c66049fa3ac25d0bbc0920"
    },
    "name": "MongoDB",
    "role": "Database"
},
{
    "_id": {
        "$oid": "56c66049fa3ac25d0bbc0921"
    },
    "name": "Express",
    "role": "Web application server"
},
{
    "_id": {
        "$oid": "56c66049fa3ac25d0bbc0922"
    },
    "name": "Angular",
    "role": "Front-end framework"
},
{
    "_id": {
        "$oid": "56c66049fa3ac25d0bbc0923"
    },
    "name": "Node.js",
    "role": "Platform"
}]
```

Note the JSON safe IDs

Note in the output that the `_id` fields don't contain `ObjectId` like we've seen when running queries in the Mongo shell. This is because `ObjectId` is invalid JSON, so `mongoexport` converts the object IDs into JSON objects.

Importing JSON with mongoimport

Now we'll see how we can create a database from a JSON file - a file of data.

We'll use the *tech.json* file that we've just exported. We'll use this file to create a new `tech` collections in a new `mean2` database.

In terminal we'll navigate to the application folder: `cd ~/MEAN`.

And then from here we'll build up a `mongoimport` command. We need to specify:

1. The name of the database to use or create
2. The name of the collection to use or create
3. That the file contains an array, so that it will create a number of documents
4. The source file for the data

```
mongoimport --db mean2 --collection tech --jsonArray api/data/tech.json
```

Again, there are loads of different options you can work with here, including specifying a different file types - such as CSV - but we're not going through them all now.

We can validate that this worked by checking in the Mongo shell:

```
show dbs
use mean2
show collections
db.tech.find().pretty()
```

Here we can see that the `_id` values match those from the original database, so MongoDB has transformed the JSON-friendly `"$oid"` properties back to `ObjectId`.
