MONGO DB

DROPPING A COLLECTION

```
// Drop the characters collection before each test
beforeEach(function(done){
    // Drop the collection
    mongoose.connection.collections.mariochars.drop(function(){
        done();
    });
});
```

ROBO MONGO

- Visual representation of our data in mongo db.
- Install robo mongo .
- When ever data is saved it gets displayed and stored with the object id.
- Each data that is saved has a unique different id.

SO FAR,

- Make a connection
- Set up a simple testing environment with Mocha
- Create model and schema
- Create instance of model and save it to DB
- Drop a collection from DB.

SAVING RECORDS

- Var char= new instance;
- Char.save()
- Save is used on instance
- Find is used on models.

SAVING RECORDS

```
const assert = require('assert');
    const MarioChar = require('../models/mariochar');
 3
    // Describe our tests
 4
 5
    describe('Saving records', function(){
 6
     // Create tests
 8
      it('Saves a record to the database', function(done){
 9
10
        const char = new MarioChar({
          name: 'Mario'
11
12
        });
13
        char.save().then(function(){
14
15
          assert(!char.isNew);
16
          done();
17
        });
18
19
      });
20
21
    });
```

OBJECT ID

- Finding a specific record using name is not feasible.
- We can use object id which is a form of string but actually object.
- So each record in the DB has different ,unique object id.

OBJECT ID

```
it('Finds a record by unique id', function(done){
    MarioChar.findOne({_id: char._id}).then(function(result){
        assert(result._id.toString() === char._id.toString());
        done();
    });
});
```

DELETING RECORDS

- Variable name.remove()
- Model.remove(criteria)
- Model.findOneAndRemove(criteria)

DELETING RECORDS

```
// Create tests
 it('Deletes a record from the database', function(done){
   MarioChar.findOneAndRemove({name: 'Mario'}).then(function(){
     MarioChar.findOne({name: 'Mario'}).then(function(result){
        assert(result === null);
       done();
     });
   });
 });
});
```

UPDATING THE RECORDS

- Variablename.update()
- Model.update(arug1,arug2)
- Modele.findOneAndUpdate(arug1,arug2)

UPDATING THE RECORDS

UPDATE OPERATOR

- Helps us in updating our fields in certain ways.
- Eg: rename, Increment etc

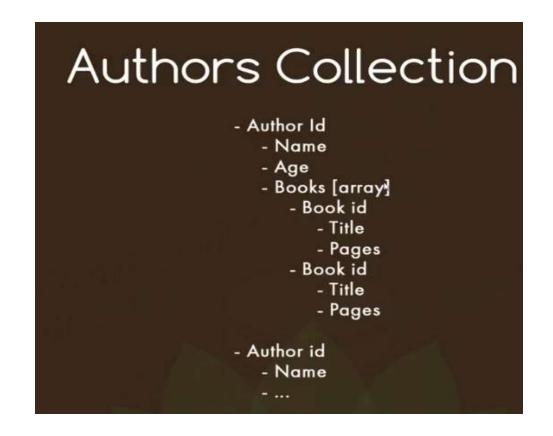
```
it('Adds 1 to the weight of every record', function(done){
    MarioChar.update({}, { $inc: { weight: 1 } }).then(function(){
        MarioChar.findOne({name: 'Mario'}).then(function(record){
            assert(record.weight === 51);
            done();
        });
    });
});
```

RELATIONAL DATA

- When we consider a real time application where we want to store multiple data we will require relation data.
- Eg: when we want to maintain some library database which has n number of authors and n no of books.
- In such cases in RDBMS what we do is we create two tables like author, books.
- Each table will have common attribute to have connection.

RELATIONAL DATA

 But in Mongo what can be done is: both can be created in a schema.



RELATIONAL DATA

```
Object format...
Name: 'Patrick',
Age: 38,
Books: [
   {title: 'Name of the Wind', pages: 400},
   {title: 'Wise Man's Fear', pages: 500}
```

EXAMPLE-CREATING SCHEMA

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
// Create a Schema and a Model
const BookSchema = new Schema({
    title: String,
    pages: Number
});
const AuthorSchema = new Schema({
    name: String,
    books: [BookSchema]
});
const Author = mongoose.model('author', AuthorSchema);
module.exports = Author;
```

NESTING SUB DOCUMENTS

```
// Create tests
it('Creates an author with sub-documents', function(done){
    var pat = new Author({
        name: 'Patrick Rothfuss',
        books: [{title: 'Name of the Wind', pages: 400}]
    });
    pat.save().then(function(){
        Author.findOne({name: 'Patrick Rothfuss'}).then(function(record
            assert(record.books.length === 1);
            done();
        });
    });
});
```

NESTING SUB DOCUMENTS

```
it('Adds a book to an author', function(done){
    var pat = new Author({
        name: 'Patrick Rothfuss',
        books: [{title: 'Name of the Wind', pages: 400}]
    });
    pat.save().then(function(){
        Author.findOne({name: 'Patrick Rothfuss'}).then(function(record
            // add a book to the books collection
            record.books.push({title: "Wise Man's Fear", pages: 500});
            record.save().then(function(){
                Author.findOne({name: 'Patrick Rothfuss'}).then(function)
                    assert(record.books.length === 2);
                    done();
                });
            });
        });
    });
```