

# 11. Understanding Sequelize

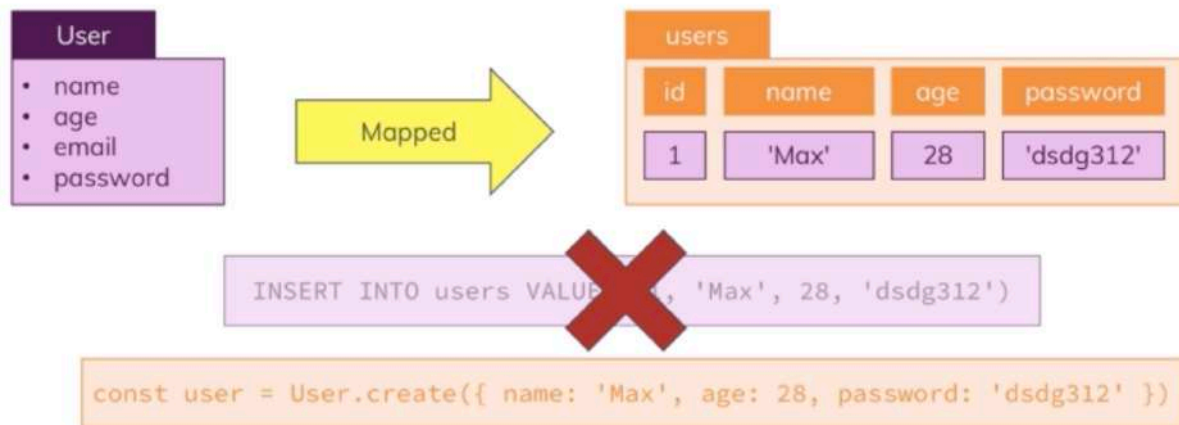
## \* Chapter 146: What Is Sequelize





### What is Sequelize?

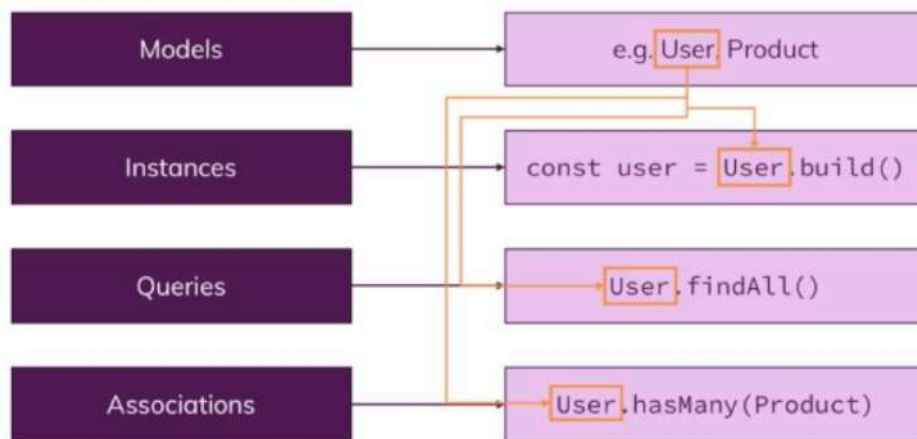
An Object-Relational Mapping Library



- this is long name which means it does all the heavy lifting, all the SQL code behind the scenes for us and maps it into javascript objects with convenience methods which we can call to execute that behind the scenes SQL code. so that we never have to write SQL code on our own.



## Core Concepts



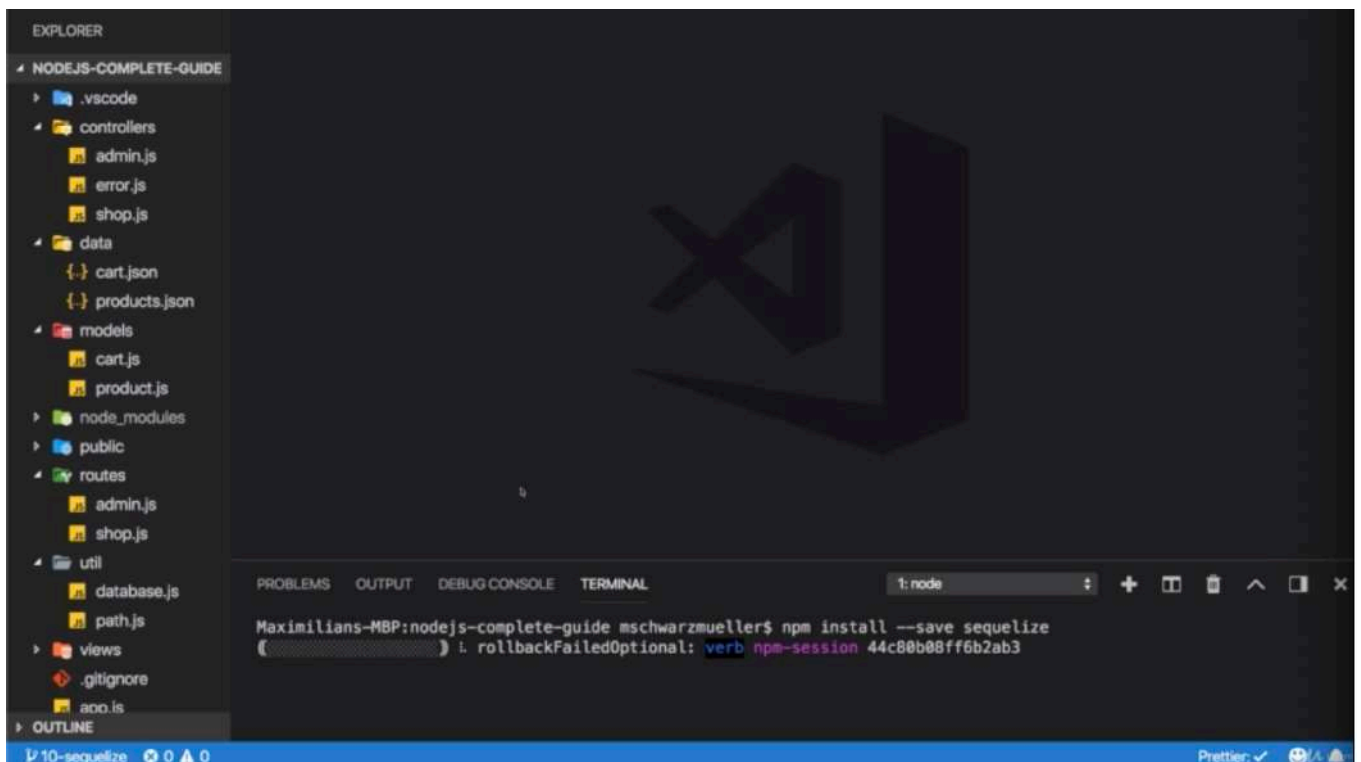
© 2019

- Sequelize offers us the models to work with our database. as i showed you on the last slide and it allows us to define such models. so basically define which data makes up a model and therefore which data will be saved in the database.
- and then we can instantiate these models. so these classes, we can execute the constructor functions or use utility methods to create let's say a new user object based on that model. so we have a connection
- and we can then run queries on that. that could be that we save a new user, but it could also be that we find all users as an example. this always relate back to our model which we define with sequelize.
- and we can also associate our models. for example, we could associate our user model to a product model.

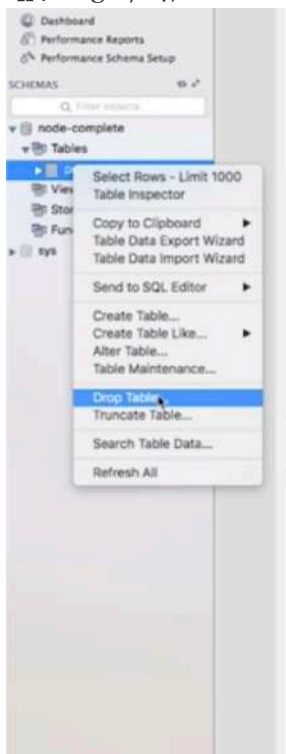
## \* Chapter 147: Connecting To The Database

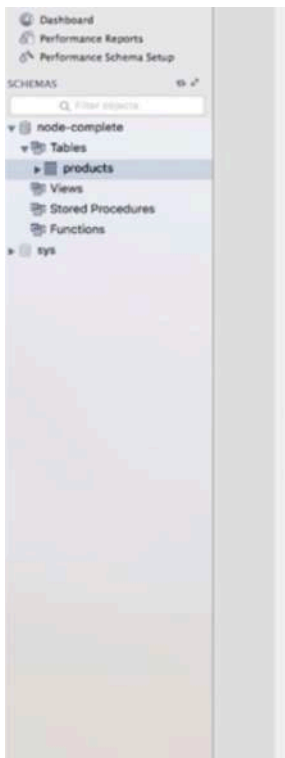
1. update
- ./util/database.js





- we will do this by running 'npm install --save' because this is also a production dependency, it's core dependency of our project and then the name is sequelize.
  - now important, sequelize needs that MySQL 2 package which we already installed. so this MySQL 2 package we installed in the last module needs to be installed.
  - the first step always is that we create a model with sequelize. and also that we connect to the database. now therefore the first step is that when i connect to MySQL database with the workbench which we also used in the last module already
- 
- 





- and in there, i will go into my node complete database and delete the products table by right clicking on it, 'drop table' and then simply click 'drop now'. i do this because i now wanna use sequelize to manage my tables.

```

1  const Sequelize = require('sequelize')
2
3  /** i will connect it to my database.
4   * you see we have to configure it
5   * with the database name -> 'node-complelte'
6   * with a username to connect to it, -> 'root'
7   * with a password. -> 'rldnjs12'
8   * so my schema name which is 'node-complete'
9   *
10  * we can also pass a fourth argument, an option object
11  * and in there you can see, for example, the 'dialect'
12  * we can set this to MySQL to make it clear
13  * that we connect to a MySQL database
14  * because different SQL engine or databases use slightly different SQL syntax
15  *
16  * the one thing i wanna set for now is the 'host'
17  * by default, it would use 'localhost'
18  * so we don't need to set it.
19  * but i will explicitly set this to 'localhost'
20  *
21  * so we are creating a new sequelize object
22  * and it will automatically connect to the database
23  * then it will set up a connection pool just as we did it manually in the last lecture
24  module.
25  */
26  const sequelize = new Sequelize('node-complete', 'root', 'rldnjs12', {
27    dialect: 'mysql',
28    host: 'localhost'
29  })
30
31  /**i can export my sequelize object
32  * which is essentially that database connection pool however managed by sequelize giving us

```

```
a lot of useful features
32 * with that we got the connection setup
33 */
34 module.exports = sequelize
```

## \* Chapter 148: Defining A Model

1. update  
- ./models/product.js

```
1 //./models/product.js
2
3 /** i will require sequelize
4  * and that will give me back a class or constructor function
5  * hence i name this with a capital S
6  */
7 const Sequelize = require('sequelize')
8
9 /** the next thing i will import is my database connection pool
10  * managed by sequelize
11  * i will also name this sequelize but with lowercase s
12  * so i will import what i export in my ./util/database.js file.
13  *
14  * my database connection pool
15  * which is more than a connection pool
16  * it's a fully configured sequelize environment
17  * which does also have the connection pool
18  * but also all the features of the sequelize package.
19  */
20 const sequelize = require('../util/database')
21
22 /**we can define a model that will be managed by sequelize
23  *
24  * we can define a new model by calling 'define()'
25  * the first name is the model name
26  * and the model name is typically a lowercase
27  *
28  * the second argument defines the structure of our model
29  * and therefore also of the automatically created database table
30  * this will be a javascript object
31  * and in there, we simply define the attributes or fields our product should have,
32  * for example i wanna have an ID
33  * now an ID is then in turn defined with an object
34  * where i configured this attribute
35  */
36 const Product = sequelize.define('product', {
37   id: {
38     /**'type' is one of the types defined by the sequelize package
39     * and i would choose 'INT(integer)'
40     * because my ID will be number starting at 1
41     * and then incrementing
42     *
43     * i also configured this attribute to auto-incrementing
44     * by auto-increment to 'true'
45     *
46     * and i don't wanna allow this value to be empty
```

```

47      * so i will 'allowNull' to false
48      * because i don't wanna allowNull value in there.
49      *
50      * and i will set 'primaryKey' to true
51      * to basically define this as the primaryKey of the table
52      * which is an important concept in SQL database
53      * for retrieving the data and then also for later defining relations.
54      */
55      type: Sequelize.INTEGER,
56      autoIncrement: true,
57      allowNull: false,
58      primaryKey: true
59  },
60  /** we can define a javascript object to configure it in detail
61   * if you wanna set type,
62   * you can use sequelize and then the type like below
63   *
64   *   Sequelize.STRING
65   */
66  title: Sequelize.STRING,
67  price: {
68    type: Sequelize.DOUBLE,
69    allowNull: false
70  },
71  imageUrl: {
72    type: Sequelize.STRING,
73    allowNull: false
74  },
75  description: {
76    type: Sequelize.STRING,
77    allowNull: false
78  }
79 })
80 /**with that, we made a huge step forward
81  * and we can now starting using this product.
82  */
83 module.exports = Product

```

## \* Chapter 149: Syncing JS Definition To The Database

1. update  
- app.js





```
17 app.use(bodyParser.urlencoded({ extended: false }));
18 app.use(express.static(path.join(__dirname, 'public')));
19
20 app.use('/admin', adminRoutes);
21 app.use(shopRoutes);
22
23 app.use(errorController.get404);
24
25 sequelize
26   .sync()
27   .then(result => {
28     console.log(result);
29     app.listen(3000);
30   })
31   .catch(err => {
32     console.log(err);
33   });
34
```

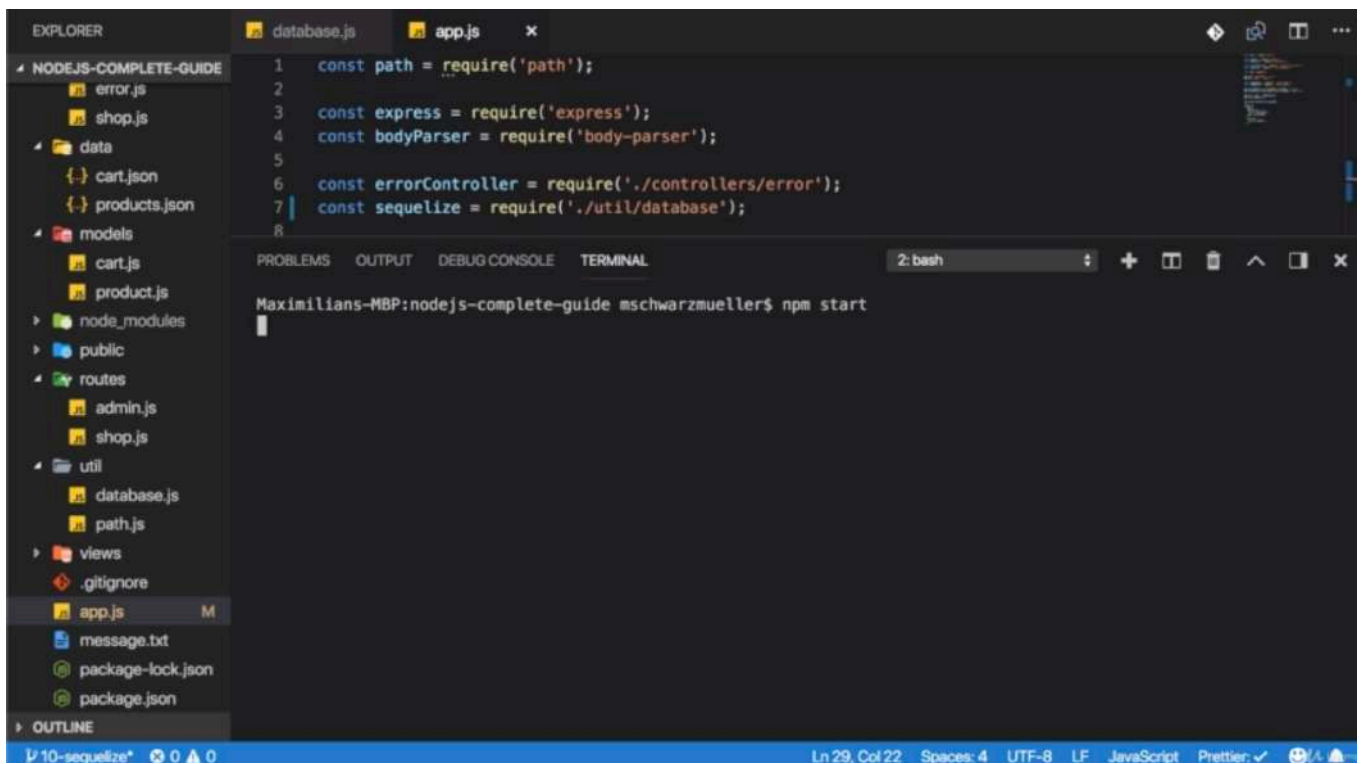
Maximilians-MBP:nodejs-complete-guide mschwarzmueller\$ npm start

> nodejs-complete-guide@1.0.0 start /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide

> nodemon app.js

```
[nodemon] 1.18.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting 'node app.js'
Executing (default): CREATE TABLE IF NOT EXISTS `products` (`id` INTEGER NOT NULL auto_increment , `title` VARCHAR(255), `price` DOUBLE PRECISION NOT NULL, `imageUrl` VARCHAR(255) NOT NULL, `description` VARCHAR(255) NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `products`
Sequelize {
  options:
    { dialect: 'mysql',
      dialectModulePath: null,
      host: 'localhost',
      protocol: 'tcp',
      define: {},
      query: {},
      sync: {},
```

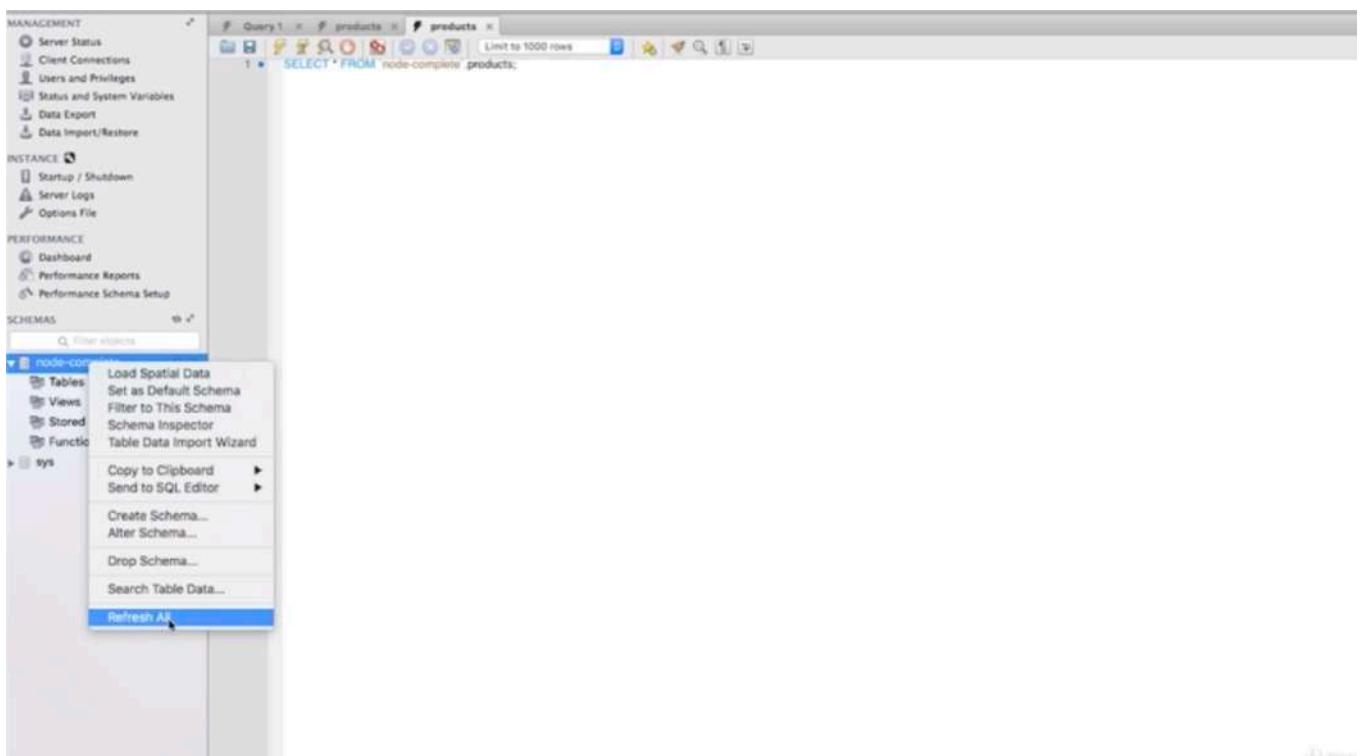
- if i now run 'npm start', it starts up and we can see there is some log output.
  - if you scroll up quite a bit because we got back a complex object, you see that this is a default log thrown by sequelize, it executed this SQL query for us without us writing this query. it created a table, if not exists 'products' yet which it named product, products and that is that automatically inferred name because we named our model product, it automatically pluralizes that and then it assigned a couple of fields which it configured according to our model definition.
  - and then this is the return value we get back, basically our sequelize object you can tell.
- 
- 



The screenshot shows the Visual Studio Code editor with a project named 'NODEJS-COMPLETE-GUIDE'. The file explorer on the left shows a directory structure with files like error.js, shop.js, data, models, routes, util, views, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor window shows the app.js file with the following code:

```
1 const path = require('path');
2
3 const express = require('express');
4 const bodyParser = require('body-parser');
5
6 const errorController = require('./controllers/error');
7 const sequelize = require('./util/database');
8
```

Below the code editor is a terminal window with the command 'npm start' entered. The terminal output shows 'Maximilians-MBP:nodejs-complete-guide mschwarzmueller\$ npm start'.

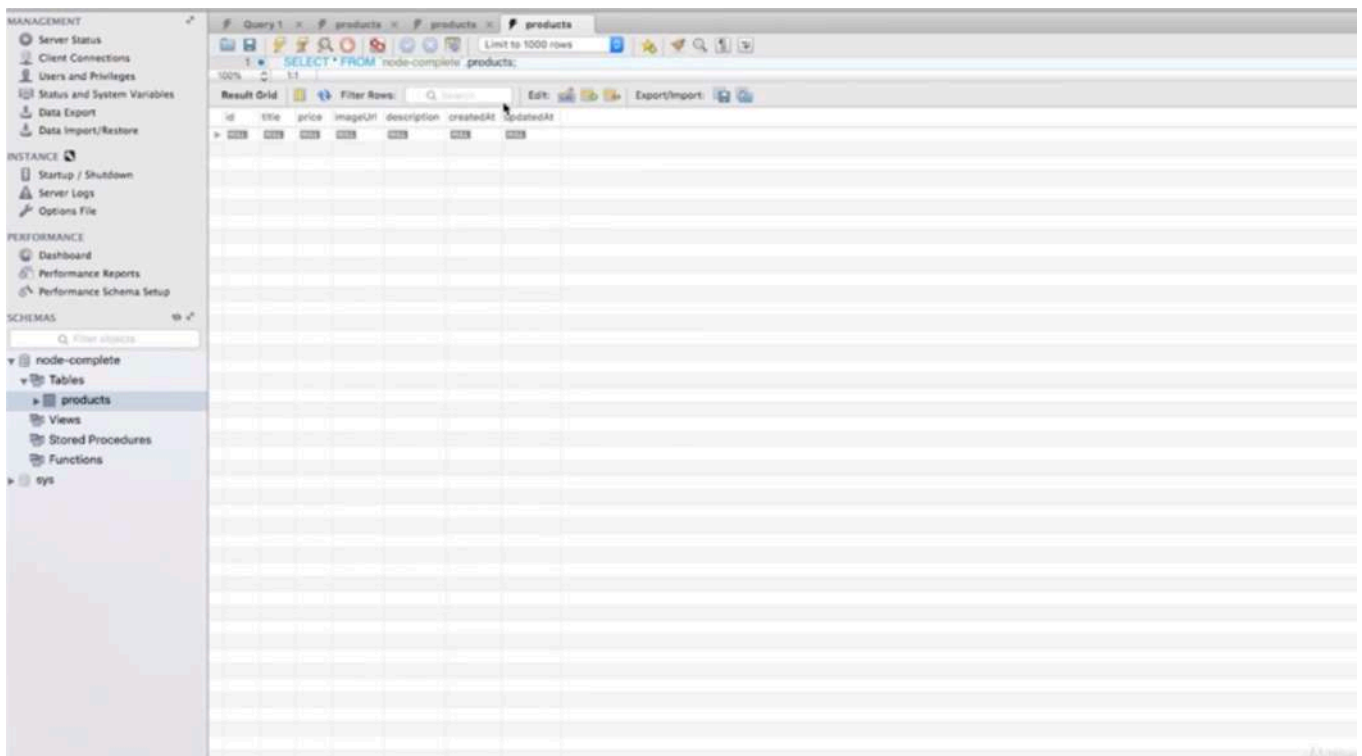
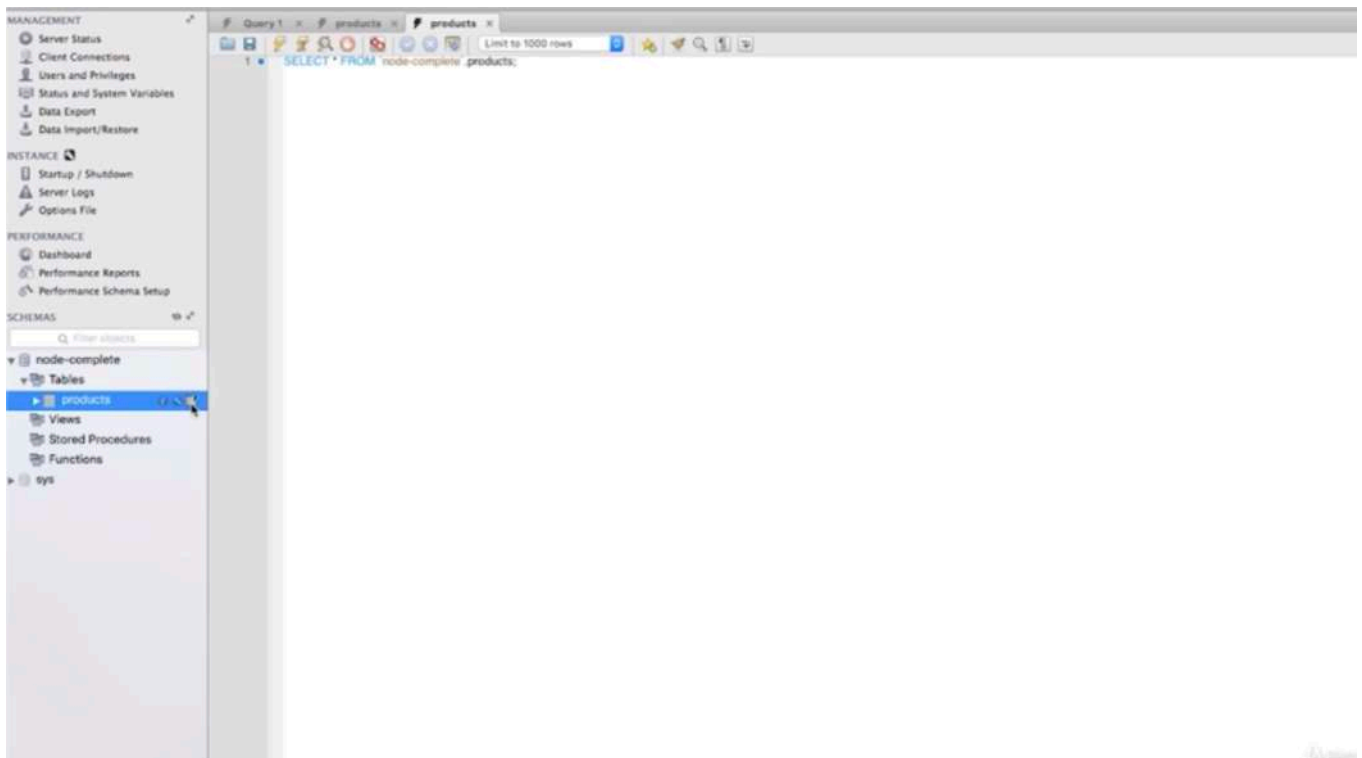


- and if you quit the server and restart, you will see it runs this again. but it does not overwrite the existing table because we have that if not exists check in there automatically. so we can run this again without issues and our server starts up even if this table already exists.









- go to our database, and click on 'refresh all', we can see that under tables, we get a products table and if we inspect that with this icon, we see all the fields we defined and that is added by sequelize to new fields, created at and updated at. so it automatically manages some timestamps for us.
- this is how we sync our tables to the database and what sequelize does for us.

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const sequelize = require('./util/database');

```

```

10
11 const app = express();
12
13 app.set('view engine', 'ejs');
14 app.set('views', 'views');
15
16 const adminRoutes = require('./routes/admin');
17
18 const shopRoutes = require('./routes/shop');
19
20 app.use(bodyParser.urlencoded({ extended: false }));
21 app.use(express.static(path.join(__dirname, 'public')));
22
23 app.use('/admin', adminRoutes);
24 app.use(shopRoutes);
25
26 app.use(errorController.get404);
27
28 /**there's a special method.
29  * the 'sync' method
30  *
31  * The 'sync' method has a look at all the models you defined.
32  * and keep in mind that you defined your models in your ./models/product.js files
33  * by calling 'sequelize' defined on that same sequelize object.
34  *
35  * so it is aware of all your models
36  * and it then basically creates tables for them.
37  * it syncs your models to the database
38  * by creating the appropriate tables.
39  * and if you have them, relations.
40  */
41 sequelize.sync().then(result => {
42     //console.log(result)
43     app.listen(3000);
44 })
45 .catch(err => {
46     console.log(err)
47 })

```

## \* Chapter 150: Inserting Data & Creating A Product

1. update  
- ./controllers/admin.js





Shop Products Cart Orders **Add Product** Admin Products

Title

First Book

Image URL

https://images.pexels.com/photos/2900/book-reading-learning-letters-2900.jpg?cs=srgb&dl=book-learning-letters-2900.jpg&fm=jpg

Price

59.99

Description

This is the first book I add through Sequelize!

Add Product

The screenshot shows the VS Code editor with the `admin.js` file open. The file contains the following code:

```
5   pageTitle: 'Add Product',
6   path: '/admin/add-product',
7   editing: false
8 });
9 };
10
11 exports.postAddProduct = (req, res, next) => {
12   const title = req.body.title;
13   const imageUrl = req.body.imageUrl;
14   const price = req.body.price;
```

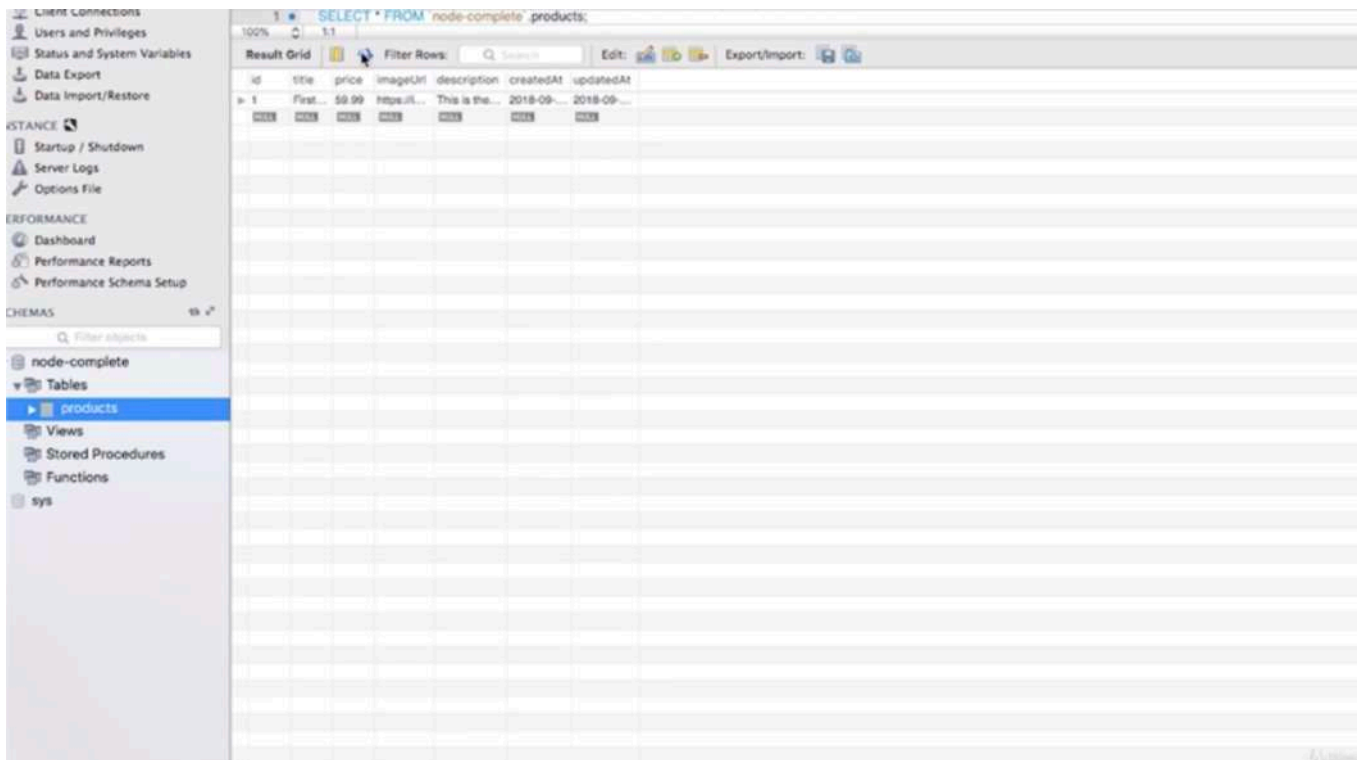
The terminal output shows the following logs:

```
at SendStream.emit (events.js:182:13)
at SendStream.error (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/send/index.js:270:17)
at SendStream.onStatError (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/send/index.js:421:12)
at next (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/send/index.js:764:28)
at /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/send/index.js:772:23
at FSReqWrap.oncomplete (fs.js:152:21)
Executing (default): INSERT INTO `products` (`id`,`title`,`price`,`imageUrl`,`description`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?, ?, ?, ?, ?, ?);
product {
  dataValues:
    { id: 1,
      title: 'First Book',
      price: '59.99',
      imageUrl:
        'https://images.pexels.com/photos/2900/book-reading-learning-letters-2900.jpg?cs=srgb&dl=book-learning-letters-2900.jpg&fm=jpg',
```

- if i click 'Add Product', we go back to node.js, this seems to have succeeded because we see the SQL statement it executed.







- and we can prove that by going to our products table and refresh this and we should see our book being added there, so this indeed succeeded.

```

1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
15   const imageUrl = req.body.imageUrl;
16   const price = req.body.price;
17   const description = req.body.description;
18   /**i will now create a new Product
19    * by calling one of the methods provided by sequelize
20    * and we got 'create' for example.
21    * 'create' creates a new element based on that model
22    * and immediately save it to the database.
23    * there's also 'build' which creates a new object
24    * based on the model
25    * but only in javascript
26    * and then we need to save it manually
27    *
28    * 'create()' take some arguments
29    * that we need to pass per our model definition
30    * so i can pass in a javascript object
31    */
32   Product.create({

```

```

33     /**i don't need to assign an ID
34     * that will be managed automatically
35     *
36     * this will be immediately saved into database.
37     */
38     title: title,
39     price: price,
40     imageUrl: imageUrl,
41     description: description
42 })
43 .then(result => {
44     //console.log(result)
45     console.log('Created Product')
46 })
47 .catch(err => {
48     console.log(err)
49 })
50 };
51
52 exports.getEditProduct = (req, res, next) => {
53     const editMode = req.query.edit;
54     if (!editMode) {
55         return res.redirect('/');
56     }
57     const prodId = req.params.productId;
58     Product.findById(prodId, product => {
59         if (!product) {
60             return res.redirect('/');
61         }
62         res.render('admin/edit-product', {
63             pageTitle: 'Edit Product',
64             path: '/admin/edit-product',
65             editing: editMode,
66             product: product
67         });
68     });
69 };
70
71 exports.postEditProduct = (req, res, next) => {
72     const prodId = req.body.productId;
73     const updatedTitle = req.body.title;
74     const updatedPrice = req.body.price;
75     const updatedImageUrl = req.body.imageUrl;
76     const updatedDesc = req.body.description;
77     const updatedProduct = new Product(
78         prodId,
79         updatedTitle,
80         updatedImageUrl,
81         updatedDesc,
82         updatedPrice
83     );
84     updatedProduct.save();
85     res.redirect('/admin/products');
86 };
87
88 exports.getProducts = (req, res, next) => {

```

```

89 Product.fetchAll(products => {
90   res.render('admin/products', {
91     prods: products,
92     pageTitle: 'Admin Products',
93     path: '/admin/products'
94   });
95 });
96 };
97
98 exports.postDeleteProduct = (req, res, next) => {
99   const prodId = req.body.productId;
100  Product.deleteById(prodId);
101  res.redirect('/admin/products');
102 };
103

```

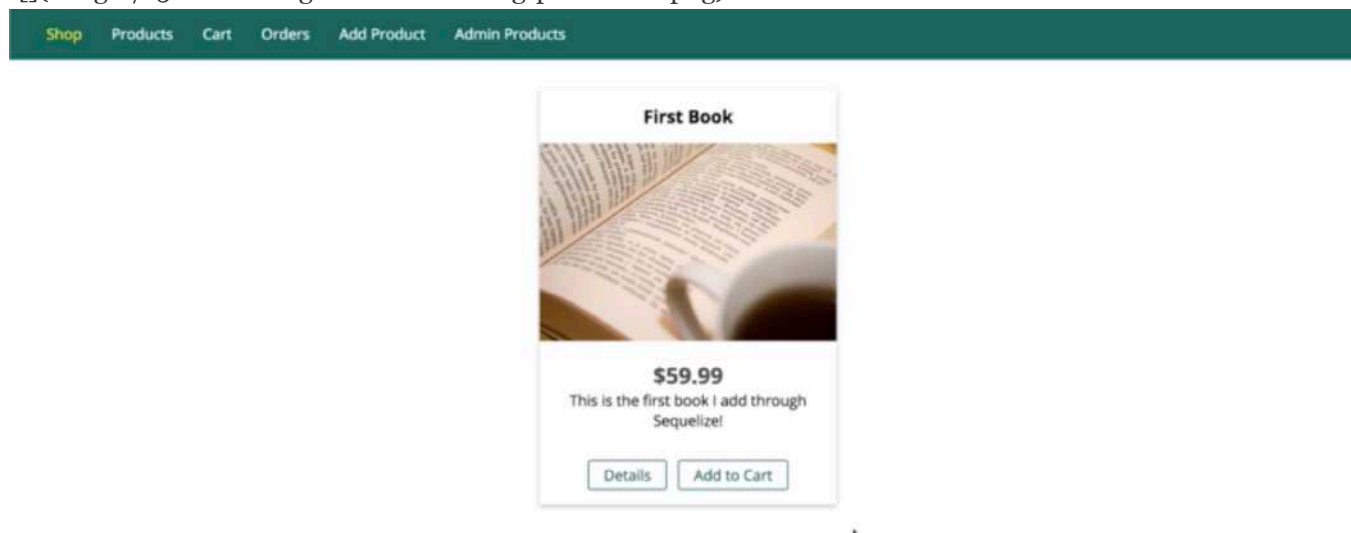
## \* Chapter 151: MUST READ: findById() in Sequelize 5

- With Sequelize v5, 'findById()' (which we will use in this course) was replaced by 'findPk()'
- You use it in the same way, so you can simply replace all occurrences of 'findById()' with 'findPk()'

## \* Chapter 152: Retrieving Data & Finding Products

1. update
  - ./controllers/shop.js



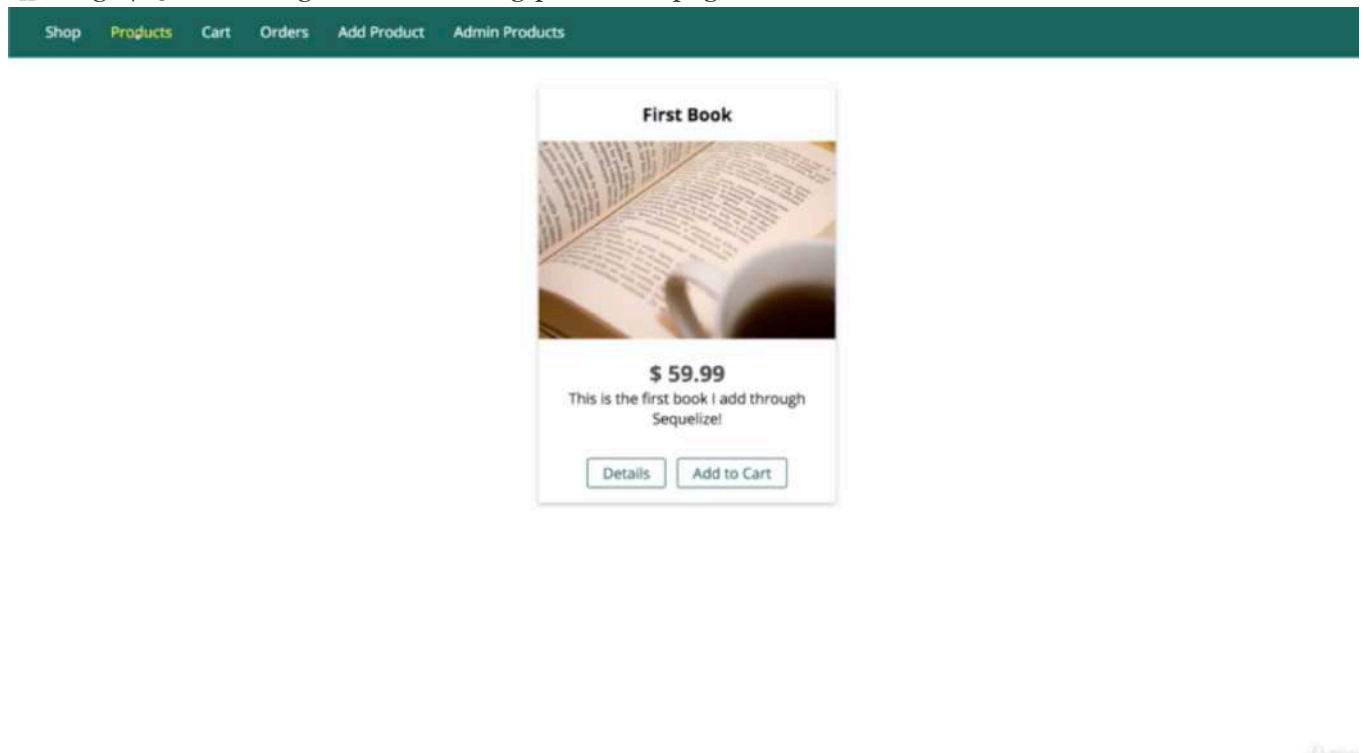


- it retrieves the data. and the data still has the same field names as before and therefore rendering this



automatically works.





15m

```
1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Cart = require('../models/cart');
5
6 exports.getProducts = (req, res, next) => {
7
8   Product.findAll()
9     .then(products => {
10       res.render('shop/product-list', {
11         prods: products,
12         pageTitle: 'All Products',
13         path: '/products'
14       });
15     })
16     .catch(err => {
17       console.log(err)
18     })
19 }
20
21 exports.getProduct = (req, res, next) => {
22   const prodId = req.params.productId;
23   Product.findById(prodId)
24     .then([product]) => {
25       res.render('shop/product-detail', {
26         product: product[0],
27         pageTitle: product.title,
28         path: '/products'
29       });
30     })
31     .catch(err => console.log(err));
32 };
```



```

33
34 exports.getIndex = (req, res, next) => {
35   /**sequelize models have plenty of methods for getting data
36    * and instead of 'fetchAll()',
37    * they, for example, have 'findAll()' to get all the records for this model.
38    * 'findAll()' also gives us back a promise where we can use the result.
39    *
40    * we can pass our options
41    * and we could define a 'where' condition
42    * to also restrict the kind of data we retrieve
43    *
44    * in the 'then()' block,
45    * we should have our products
46    *
47    */
48   Product.findAll()
49     .then(products => {
50     res.render('shop/index', {
51       prods: products,
52       pageTitle: 'Shop',
53       path: '/'
54     });
55   })
56   .catch(err => {
57     console.log(err)
58   })
59 };
60
61 exports.getCart = (req, res, next) => {
62   Cart.getCart(cart => {
63     Product.fetchAll(products => {
64       const cartProducts = [];
65       for (product of products) {
66         const cartProductData = cart.products.find(
67           prod => prod.id === product.id
68         );
69         if (cartProductData) {
70           cartProducts.push({ productData: product, qty: cartProductData.qty });
71         }
72       }
73       res.render('shop/cart', {
74         path: '/cart',
75         pageTitle: 'Your Cart',
76         products: cartProducts
77       });
78     });
79   });
80 };
81
82 exports.postCart = (req, res, next) => {
83   const prodId = req.body.productId;
84   Product.findById(prodId, product => {
85     Cart.addProduct(prodId, product.price);
86   });
87   res.redirect('/cart');
88 };

```

```

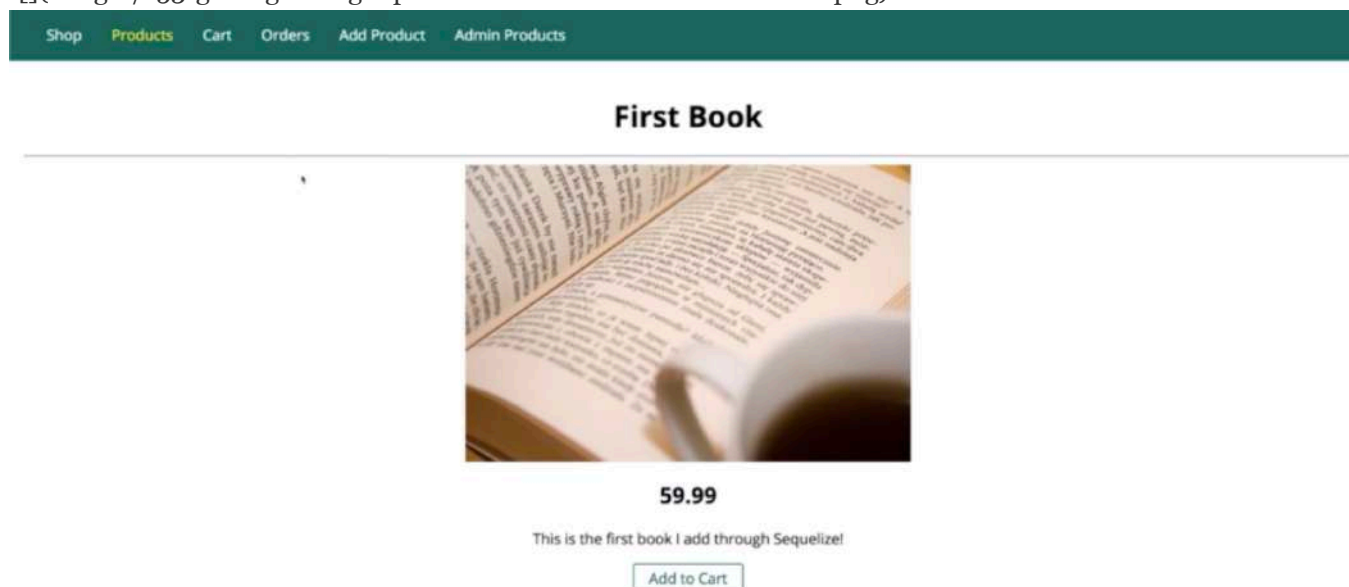
89
90 exports.postCartDeleteProduct = (req, res, next) => {
91   const prodId = req.body.productId;
92   Product.findById(prodId, product => {
93     Cart.deleteProduct(prodId, product.price);
94     res.redirect('/cart');
95   });
96 };
97
98 exports.getOrders = (req, res, next) => {
99   res.render('shop/orders', {
100     path: '/orders',
101     pageTitle: 'Your Orders'
102   });
103 };
104
105 exports.getCheckout = (req, res, next) => {
106   res.render('shop/checkout', {
107     path: '/checkout',
108     pageTitle: 'Checkout'
109   });
110 };

```

## \* Chapter 153: Getting A Single Product With The ‘Where’ Condition

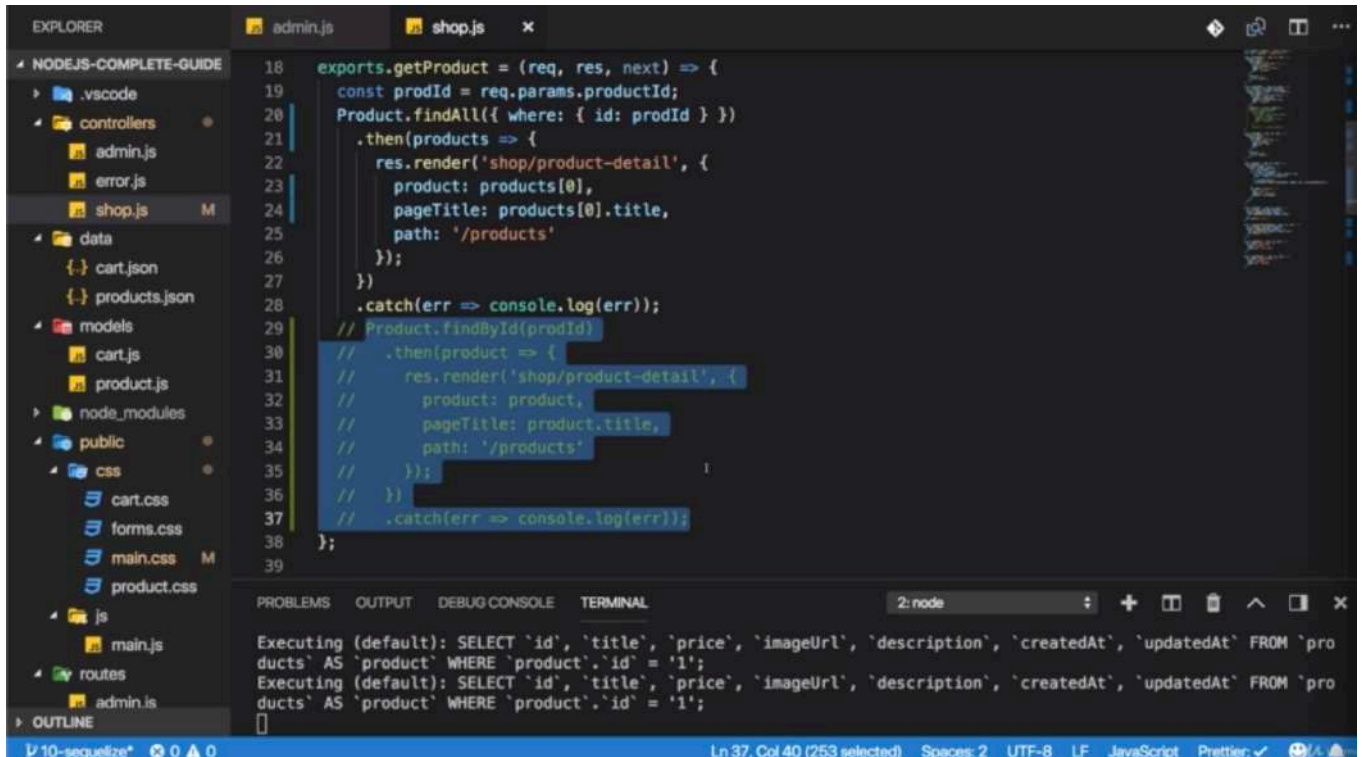
1. update
  - ./controllers/shop.js
  - ./views/shop/product-detail.ejs
  - ./public/css/main.css





- if we save that and let it restart therefore, now we can reload this page and it works as before. but now we are using 'findAll()' and 'where' query. i simply wanted to show you alternative approach.





The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like controllers, data, models, node\_modules, public, and routes. The code editor shows the shop.js file with the following code:

```
18 exports.getProduct = (req, res, next) => {
19   const prodId = req.params.productId;
20   Product.findAll({ where: { id: prodId } })
21     .then(products => {
22       res.render('shop/product-detail', {
23         product: products[0],
24         pageTitle: products[0].title,
25         path: '/products'
26       });
27     })
28     .catch(err => console.log(err));
29   // Product.findById(prodId)
30   // .then(product => {
31   //   res.render('shop/product-detail', {
32   //     product: product,
33   //     pageTitle: product.title,
34   //     path: '/products'
35   //   });
36   // })
37   // .catch(err => console.log(err));
38 };
```

The terminal at the bottom shows the following output:

```
Executing (default): SELECT `id`, `title`, `price`, `imageUrl`, `description`, `createdAt`, `updatedAt` FROM `products` AS `product` WHERE `product`.`id` = '1';
Executing (default): SELECT `id`, `title`, `price`, `imageUrl`, `description`, `createdAt`, `updatedAt` FROM `products` AS `product` WHERE `product`.`id` = '1';
```

- it's perfectly fine and even preferable in this case to use 'findById()'.  
- so i will actually switch back to that other approach. it's good to know how you can query though.

```
1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Cart = require('../models/cart');
5
6 exports.getProducts = (req, res, next) => {
7   Product.findAll()
8     .then(products => {
9       res.render('shop/product-list', {
10         prods: products,
11         pageTitle: 'All Products',
12         path: '/products'
13       });
14     })
15     .catch(err => {
16       console.log(err)
17     })
18 }
19
20 exports.getProduct = (req, res, next) => {
21   const prodId = req.params.productId;
22   /**we only have one product with that ID
23    * but i wanna show you that 'where' syntax
24    * so any object we can pass to 'findAll()'
25    *
26    * and there you got a rich query language
27    * or rich amount of options you can use to configure this
28    *
29    * the one important thing is by default 'findAll({where: {id: prodId}})' gives us an
    array
30    * because even though we know that only one product will have this ID,
```

```

31  * 'findAll()' per definition always gives you multiple items
32  * even if it's an array with only one element.
33  */
34  // Product.findAll({where: {id: prodId}})
35  //   .then(products => {
36  //     res.render('shop/product-detail', {
37  //       product: products[0],
38  //       pageTitle: products[0].title,
39  //       path: '/products'
40  //     });
41  //   })
42  //   .catch(err => console.log(err));
43  Product.findById(prodId)
44    .then(product => {
45      res.render('shop/product-detail', {
46        product: product,
47        pageTitle: product.title,
48        path: '/products'
49      });
50    })
51    .catch(err => console.log(err));
52 };
53
54 exports.getIndex = (req, res, next) => {
55   Product.findAll()
56     .then(products => {
57       res.render('shop/index', {
58         prods: products,
59         pageTitle: 'Shop',
60         path: '/'
61       });
62     })
63     .catch(err => {
64       console.log(err)
65     })
66 };
67
68 exports.getCart = (req, res, next) => {
69   Cart.getCart(cart => {
70     Product.fetchAll(products => {
71       const cartProducts = [];
72       for (product of products) {
73         const cartProductData = cart.products.find(
74           prod => prod.id === product.id
75         );
76         if (cartProductData) {
77           cartProducts.push({ productData: product, qty: cartProductData.qty });
78         }
79       }
80       res.render('shop/cart', {
81         path: '/cart',
82         pageTitle: 'Your Cart',
83         products: cartProducts
84       });
85     });
86   });

```

```

87 };
88
89 exports.postCart = (req, res, next) => {
90   const prodId = req.body.productId;
91   Product.findById(prodId, product => {
92     Cart.addProduct(prodId, product.price);
93   });
94   res.redirect('/cart');
95 };
96
97 exports.postCartDeleteProduct = (req, res, next) => {
98   const prodId = req.body.productId;
99   Product.findById(prodId, product => {
100     Cart.deleteProduct(prodId, product.price);
101     res.redirect('/cart');
102   });
103 };
104
105 exports.getOrders = (req, res, next) => {
106   res.render('shop/orders', {
107     path: '/orders',
108     pageTitle: 'Your Orders'
109   });
110 };
111
112 exports.getCheckout = (req, res, next) => {
113   res.render('shop/checkout', {
114     path: '/checkout',
115     pageTitle: 'Checkout'
116   });
117 };

```

```

1 <!--./views/shop/product-detail.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   </head>
5
6   <body>
7     <%- include('../includes/navigation.ejs') %>
8     <main class="centered">
9       <h1><%= product.title %></h1>
10      <hr>
11      <div class="image">
12        ">
13      </div>
14      <h2><%= product.price %></h2>
15      <p><%= product.description %></p>
16      <%- include('../includes/add-to-cart.ejs') %>
17      <%- include('../includes/end.ejs') %>

```

```

1 /*./public/css/main.css*/
2
3 @import url('https://fonts.googleapis.com/css?family=Open+Sans:400,700');
4
5 * {
6   box-sizing: border-box;
7 }

```

```
8
9 body {
10   padding: 0;
11   margin: 0;
12   font-family: 'Open Sans', sans-serif;
13 }
14
15 main {
16   padding: 1rem;
17   margin: auto;
18 }
19
20 form {
21   display: inline;
22 }
23
24 .centered {
25   text-align: center;
26 }
27
28 .image {
29   height: 20rem;
30 }
31
32 .image img {
33   height: 100%;
34 }
35
36 .main-header {
37   width: 100%;
38   height: 3.5rem;
39   background-color: #00695c;
40   padding: 0 1.5rem;
41   display: flex;
42   align-items: center;
43 }
44
45 .main-header__nav {
46   height: 100%;
47   display: none;
48   align-items: center;
49 }
50
51 .main-header__item-list {
52   list-style: none;
53   margin: 0;
54   padding: 0;
55   display: flex;
56 }
57
58 .main-header__item {
59   margin: 0 1rem;
60   padding: 0;
61 }
62
63 .main-header__item a {
```

```
64   text-decoration: none;
65   color: white;
66 }
67
68 .main-header__item a:hover,
69 .main-header__item a:active,
70 .main-header__item a.active {
71   color: #ffeb3b;
72 }
73
74 .mobile-nav {
75   width: 30rem;
76   height: 100vh;
77   max-width: 90%;
78   position: fixed;
79   left: 0;
80   top: 0;
81   background: white;
82   z-index: 10;
83   padding: 2rem 1rem 1rem 2rem;
84   transform: translateX(-100%);
85   transition: transform 0.3s ease-out;
86 }
87
88 .mobile-nav.open {
89   transform: translateX(0);
90 }
91
92 .mobile-nav__item-list {
93   list-style: none;
94   display: flex;
95   flex-direction: column;
96   margin: 0;
97   padding: 0;
98 }
99
100 .mobile-nav__item {
101   margin: 1rem;
102   padding: 0;
103 }
104
105 .mobile-nav__item a {
106   text-decoration: none;
107   color: black;
108   font-size: 1.5rem;
109   padding: 0.5rem 2rem;
110 }
111
112 .mobile-nav__item a:active,
113 .mobile-nav__item a:hover,
114 .mobile-nav__item a.active {
115   background: #00695c;
116   color: white;
117   border-radius: 3px;
118 }
119
```

```
120 #side-menu-toggle {
121   border: 1px solid white;
122   font: inherit;
123   padding: 0.5rem;
124   display: block;
125   background: transparent;
126   color: white;
127   cursor: pointer;
128 }
129
130 #side-menu-toggle:focus {
131   outline: none;
132 }
133
134 #side-menu-toggle:active,
135 #side-menu-toggle:hover {
136   color: #ffeb3b;
137   border-color: #ffeb3b;
138 }
139
140 .backdrop {
141   position: fixed;
142   top: 0;
143   left: 0;
144   width: 100%;
145   height: 100vh;
146   background: rgba(0, 0, 0, 0.5);
147   z-index: 5;
148   display: none;
149 }
150
151 .grid {
152   display: flex;
153   flex-wrap: wrap;
154   justify-content: space-around;
155   align-items: stretch;
156 }
157
158 .card {
159   box-shadow: 0 2px 8px rgba(0, 0, 0, 0.26);
160 }
161
162 .card__header,
163 .card__content {
164   padding: 1rem;
165 }
166
167 .card__header h1,
168 .card__content h1,
169 .card__content h2,
170 .card__content p {
171   margin: 0;
172 }
173
174 .card__image {
175   width: 100%;
```



```

176 }
177
178 .card__image img {
179     width: 100%;
180 }
181
182 .card__actions {
183     padding: 1rem;
184     text-align: center;
185 }
186
187 .card__actions button,
188 .card__actions a {
189     margin: 0 0.25rem;
190 }
191
192 .btn {
193     display: inline-block;
194     padding: 0.25rem 1rem;
195     text-decoration: none;
196     font: inherit;
197     border: 1px solid #00695c;
198     color: #00695c;
199     background: white;
200     border-radius: 3px;
201     cursor: pointer;
202 }
203
204 .btn:hover,
205 .btn:active {
206     background-color: #00695c;
207     color: white;
208 }
209
210 @media (min-width: 768px) {
211     .main-header__nav {
212         display: flex;
213     }
214
215     #side-menu-toggle {
216         display: none;
217     }
218 }
219

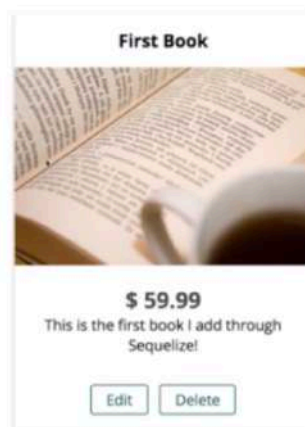
```

## \* Chapter 154: Fetching Admin Products

1. update

- ./controllers/admin.js





```

1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
15   const imageUrl = req.body.imageUrl;
16   const price = req.body.price;
17   const description = req.body.description;
18   Product.create({
19     title: title,
20     price: price,
21     imageUrl: imageUrl,
22     description: description
23   })
24   .then(result => {
25     //console.log(result)
26     console.log('Created Product')
27   })
28   .catch(err => {
29     console.log(err)
30   })
31 };
32
33 exports.getEditProduct = (req, res, next) => {
34   const editMode = req.query.edit;
35   if (!editMode) {

```

```

36     return res.redirect('/');
37 }
38 const prodId = req.params.productId;
39 Product.findById(prodId, product => {
40     if (!product) {
41         return res.redirect('/');
42     }
43     res.render('admin/edit-product', {
44         pageTitle: 'Edit Product',
45         path: '/admin/edit-product',
46         editing: editMode,
47         product: product
48     });
49 });
50 };
51
52 exports.postEditProduct = (req, res, next) => {
53     const prodId = req.body.productId;
54     const updatedTitle = req.body.title;
55     const updatedPrice = req.body.price;
56     const updatedImageUrl = req.body.imageUrl;
57     const updatedDesc = req.body.description;
58     const updatedProduct = new Product(
59         prodId,
60         updatedTitle,
61         updatedImageUrl,
62         updatedDesc,
63         updatedPrice
64     );
65     updatedProduct.save();
66     res.redirect('/admin/products');
67 };
68
69 exports.getProducts = (req, res, next) => {
70     Product.findAll()
71         .then(products => {
72             res.render('admin/products', {
73                 prods: products,
74                 pageTitle: 'Admin Products',
75                 path: '/admin/products'
76             })
77         })
78         .catch(err => console.log(err))
79 };
80
81 exports.postDeleteProduct = (req, res, next) => {
82     const prodId = req.body.productId;
83     Product.deleteById(prodId);
84     res.redirect('/admin/products');
85 };
86

```

## \* Chapter 155: Updating Products

1. update  
- ./controllers/admin.js



[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

First Book

Image URL

https://images.pexels.com/photos/2900/be

Price

59.99

Description

This is the first book I add through Sequelize!

Update Product

- if i reload this page, this is looking good. the fields get populated with our values.
- we just need to make sure that if we do change something and we save it, this does get saved to the database correctly. for that, we have to have a look at 'postEditProduct' which gets called once we submit this page.





[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

First Book!!!

Image URL

https://images.pexels.com/photos/2900/be

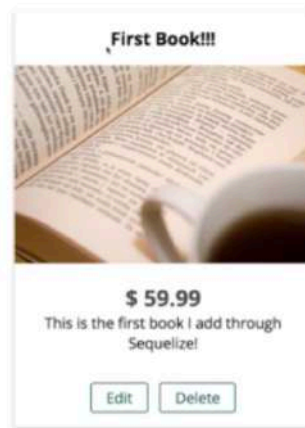
Price

59.99

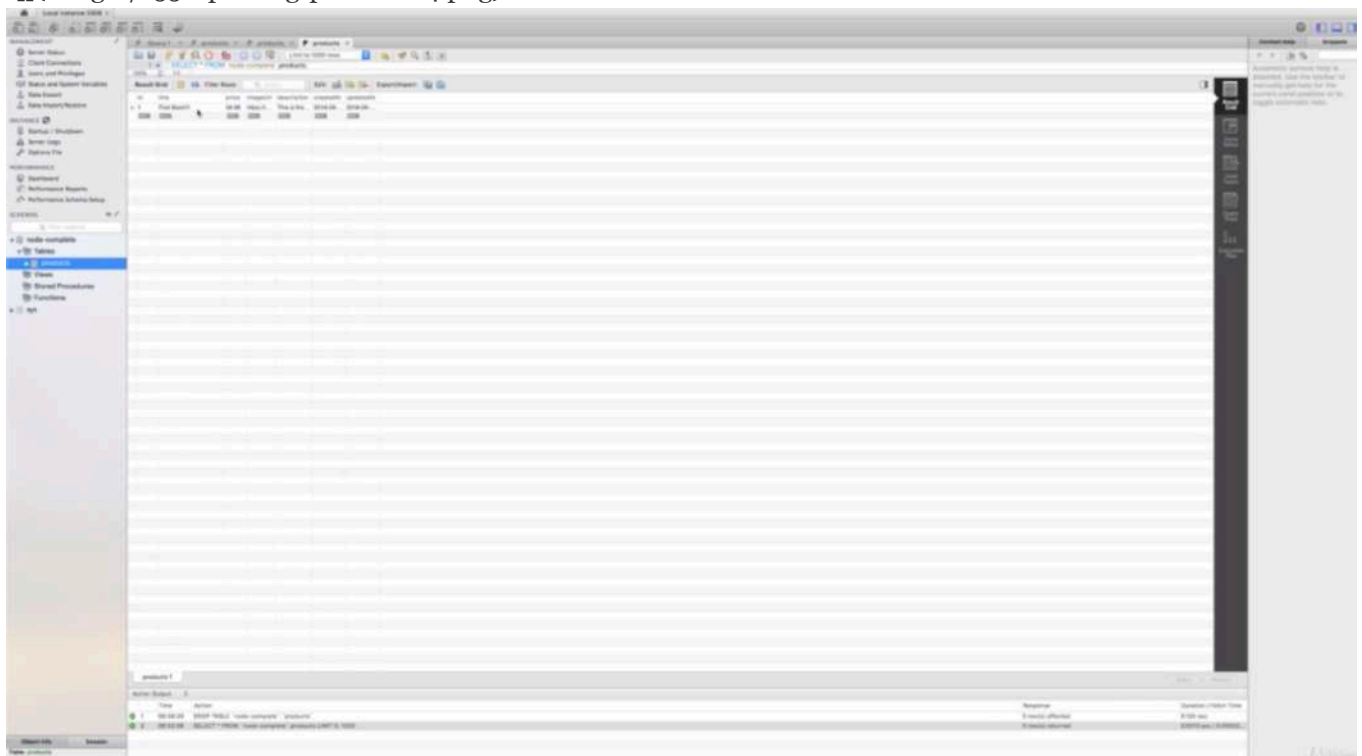
Description

This is the first book I add through Sequelize!

Update Product



- if we click 'updated product', we get redirected to the products page. and we can see the updated thing.  

- and we also can see that change in here.





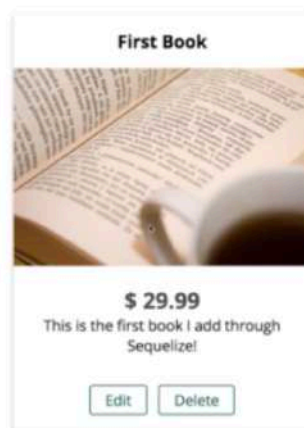
Title  
First Book

Image URL  
<https://images.pexels.com/photos/2900/bk>

Price  
29.99

Description  
This is the first book I add through Sequelize!

Update Product



```

1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;

```

```

15  const imageUrl = req.body.imageUrl;
16  const price = req.body.price;
17  const description = req.body.description;
18  Product.create({
19    title: title,
20    price: price,
21    imageUrl: imageUrl,
22    description: description
23  })
24    .then(result => {
25      // console.log(result);
26      console.log('Created Product');
27    })
28    .catch(err => {
29      console.log(err);
30    });
31  };
32
33  exports.getEditProduct = (req, res, next) => {
34    const editMode = req.query.edit;
35    if (!editMode) {
36      return res.redirect('/');
37    }
38    const prodId = req.params.productId;
39    /**With Sequelize v5, 'findById()' (which we will use in this course) was replaced by
    'findByPk()'
40     * You use it in the same way, so you can simply replace all occurrences of 'findById()'
    with 'findByPk()'
41     */
42    Product.findByPk(prodId)
43      .then(product => {
44        if (!product) {
45          return res.redirect('/');
46        }
47        res.render('admin/edit-product', {
48          pageTitle: 'Edit Product',
49          path: '/admin/edit-product',
50          editing: editMode,
51          product: product
52        });
53      })
54      .catch(err => console.log(err));
55  };
56
57  exports.postEditProduct = (req, res, next) => {
58    const prodId = req.body.productId;
59    const updatedTitle = req.body.title;
60    const updatedPrice = req.body.price;
61    const updatedImageUrl = req.body.imageUrl;
62    const updatedDesc = req.body.description;
63    /**let's handle any error by logging them for now
64     * and in then, let's work with the product we retrieved
65     * that product now needs to be updated
66     *
67     * we can simply do that by saying 'product.title = updatedTitle'
68     * so we can work with all the attributes our product has per our model definition and

```

change them

```
69  * note that this will not directly change the data in the database,
70  * it will only do it locally in our app in our javascript app for the moment.
71  *
72  */
73  Product.findByPk(prodId)
74    .then(product => {
75      product.title = updatedTitle;
76      product.price = updatedPrice;
77      product.description = updatedDesc;
78      product.imageUrl = updatedImageUrl;
79      /**as i said, this will not directly edit it in the database
80       * to do that, we have to call 'product.save()'
81       * this is another method provided by sequelize
82       * those takes the product as we edit it and save it back to the database.
83       *
84       * if the product doesn't exist yet,
85       * it will create a new one,
86       * if it does as this one,
87       * then it will overwrite or update the old one with our new values.
88       *
89       * we can again chain then and catch
90       * but to not start nesting our promises
91       * which would yield the same ugly picture as nesting callbacks,
92       * we can 'return' this,
93       * so we return the promise which is returned by 'save()'
94       * and we can add 'then()' block below.
95       */
96      return product.save();
97    })
98    .then(result => {
99      console.log('UPDATED PRODUCT!');
100      res.redirect('/admin/products');
101    })
102    /**this 'catch()' block would catch errors both for this first '.then()' and second
103     '.then()'
104     * the second 'then()' block will handle any success response from this 'save()' promise
105     *
106     */
107    .catch(err => console.log(err));
108  };
109
110 exports.getProducts = (req, res, next) => {
111   Product.findAll()
112     .then(products => {
113       res.render('admin/products', {
114         prods: products,
115         pageTitle: 'Admin Products',
116         path: '/admin/products'
117       });
118     })
119     .catch(err => console.log(err));
120 }
121
122 exports.postDeleteProduct = (req, res, next) => {
123   const prodId = req.body.productId;
```



```
123 Product.deleteById(prodId);
124 res.redirect('/admin/products');
125 };
126
```

## \* Chapter 156: Deleting Products

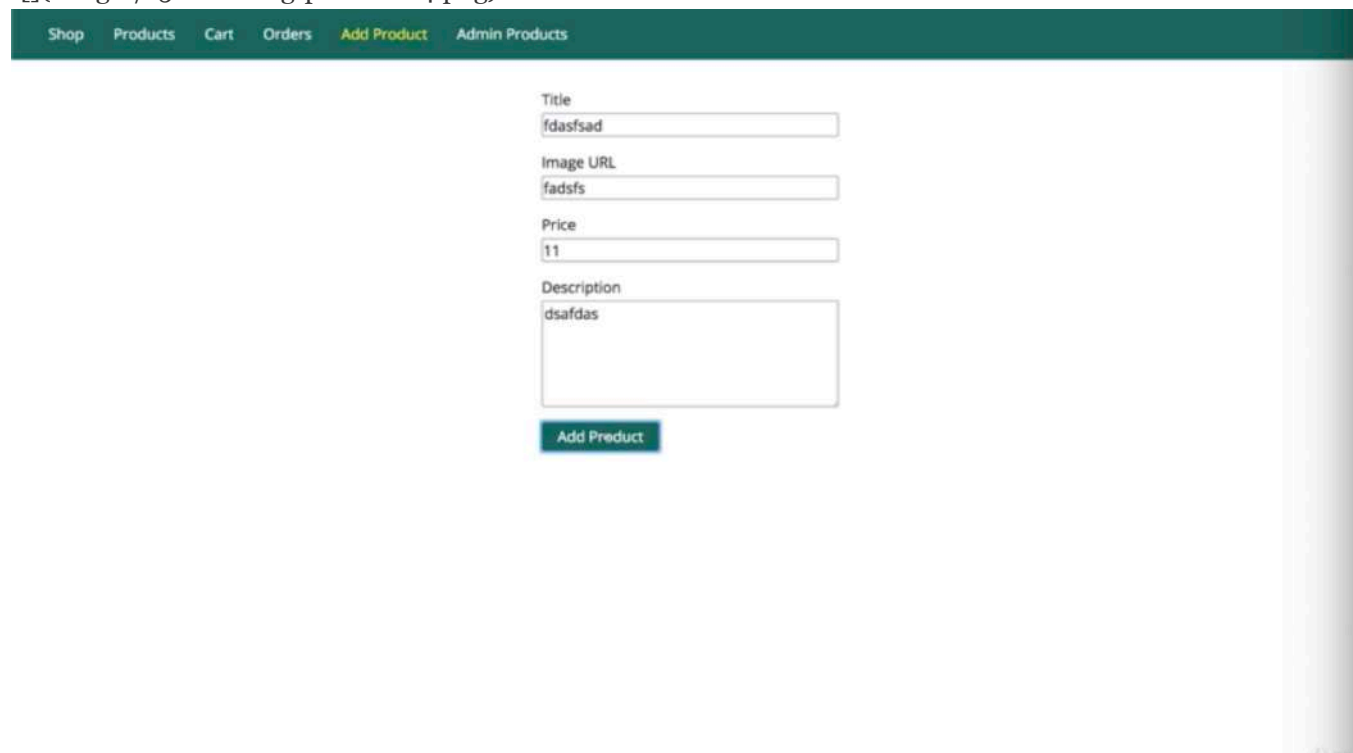
1. update  
- ./controllers/admin.js



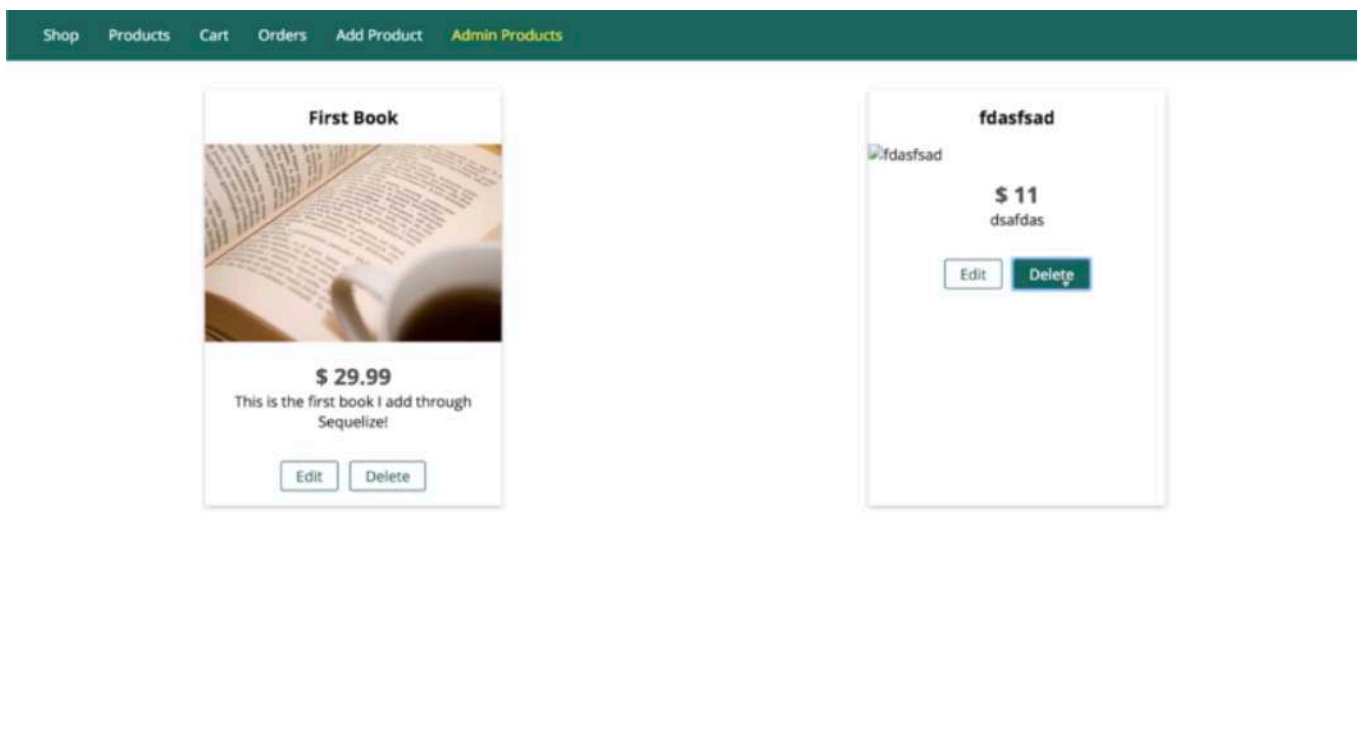
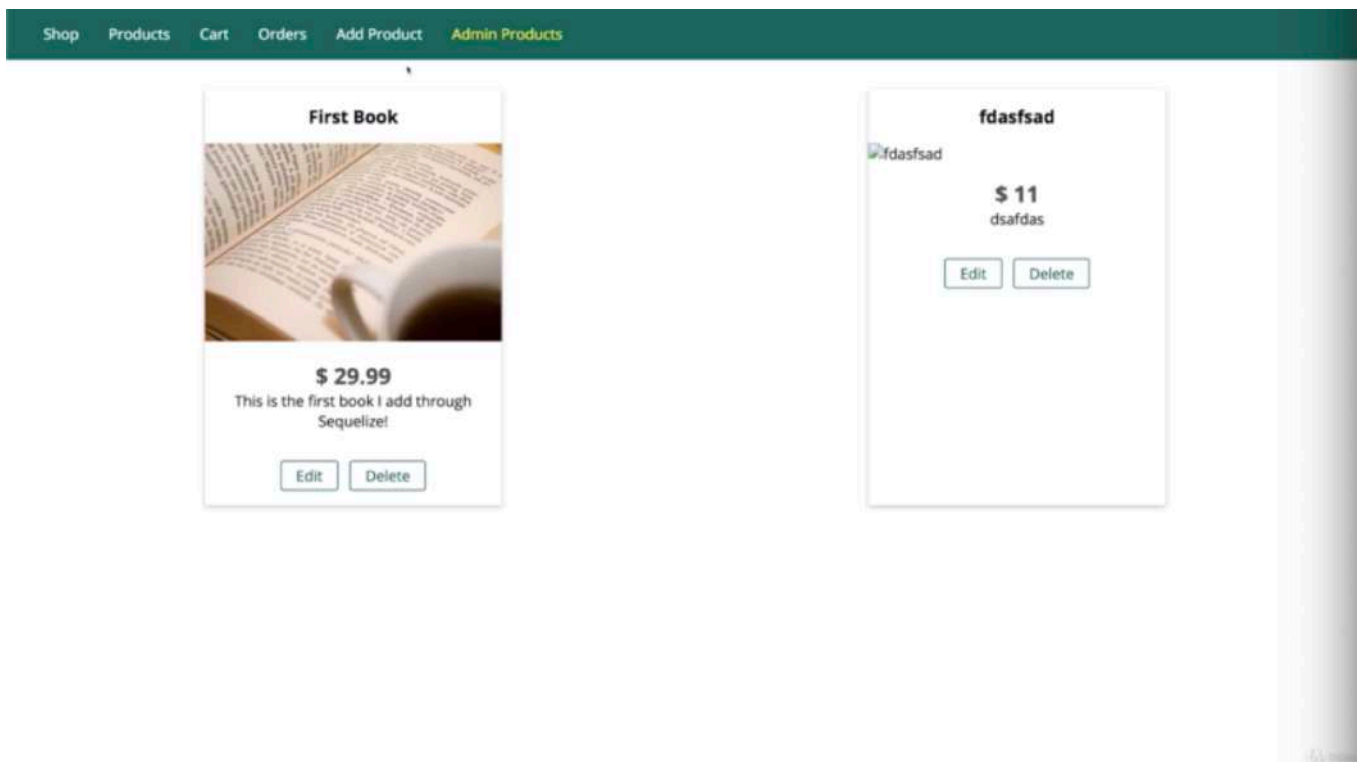








The screenshot shows a web application interface with a dark green header bar containing navigation links: Shop, Products, Cart, Orders, Add Product (highlighted in yellow), and Admin Products. Below the header, the 'Add Product' form is displayed. It includes four input fields: 'Title' with the value 'fdasfsad', 'Image URL' with the value 'fadsfs', 'Price' with the value '11', and 'Description' with the value 'dsafdas'. A green 'Add Product' button is located at the bottom of the form.



- if we go to 'Admin Products' and click 'Delete', this works

```
1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
```

```

14  const title = req.body.title;
15  const imageUrl = req.body.imageUrl;
16  const price = req.body.price;
17  const description = req.body.description;
18  Product.create({
19    title: title,
20    price: price,
21    imageUrl: imageUrl,
22    description: description
23  })
24    .then(result => {
25      // console.log(result);
26      console.log('Created Product');
27      res.redirect('/admin/products')
28    })
29    .catch(err => {
30      console.log(err);
31    });
32 };
33
34 exports.getEditProduct = (req, res, next) => {
35   const editMode = req.query.edit;
36   if (!editMode) {
37     return res.redirect('/');
38   }
39   const prodId = req.params.productId;
40   Product.findById(prodId)
41     .then(product => {
42       if (!product) {
43         return res.redirect('/');
44       }
45       res.render('admin/edit-product', {
46         pageTitle: 'Edit Product',
47         path: '/admin/edit-product',
48         editing: editMode,
49         product: product
50       });
51     })
52     .catch(err => console.log(err));
53 };
54
55 exports.postEditProduct = (req, res, next) => {
56   const prodId = req.body.productId;
57   const updatedTitle = req.body.title;
58   const updatedPrice = req.body.price;
59   const updatedImageUrl = req.body.imageUrl;
60   const updatedDesc = req.body.description;
61   Product.findById(prodId)
62     .then(product => {
63       product.title = updatedTitle;
64       product.price = updatedPrice;
65       product.description = updatedDesc;
66       product.imageUrl = updatedImageUrl;
67       return product.save();
68     })
69     .then(result => {

```

```

70     console.log('UPDATED PRODUCT!');
71     res.redirect('/admin/products');
72   })
73   .catch(err => console.log(err));
74 };
75
76 exports.getProducts = (req, res, next) => {
77   Product.findAll()
78     .then(products => {
79       res.render('admin/products', {
80         prods: products,
81         pageTitle: 'Admin Products',
82         path: '/admin/products'
83       });
84     })
85     .catch(err => console.log(err));
86 };
87
88 exports.postDeleteProduct = (req, res, next) => {
89   const prodId = req.body.productId;
90   /**'deleteById' doesn't exist in a sequelize
91    * instead on the product, we can call 'destroy'
92    * and 'destroy' allows us to destroy any product we find through our options
93    * and these options, for example, allow us to add a 'where' condition to narrow down
    which product to delete
94    *
95    * we can use different approach
96    * instead of calling destroy like this
97    * and adding a condition which product to find which is fine
98    * we can use 'findById' again to find a product by that ID
99   */
100   Product.findById(prodId)
101     .then(product => {
102       /**we can 'return' this
103        * because this will also yield a promise
104        * and therefore add another 'then()' block
105        * which will execute once the destruction succeeded
106        */
107       return product.destroy()
108     })
109     .then(result => {
110       /**we should redirect to make sure we only redirect once the deletion succeeded. */
111       console.log('DESTROYED PRODUCT')
112       res.redirect('/admin/products');
113     })
114     .catch(err => console.log(err))
115 };
116

```

## \* Chapter 157: Creating A User Model

1. update  
- ./models/user.js

```
1 //./models/user.js
```

```
2
```

```

3  /**we have no real authentication process,
4   * so we will only work with one dummy user
5   * who doesn't really have to log in.
6   *
7   * i wanna show you how you could have a user
8   * who did create a product and who therefore is connected to that product
9   * and at the same time, a user should own a cart
10  * and that cart will hold multiple products
11  * and this is how we can then overall connect everything.
12  */
13
14  /**first of all, requiring the sequelize constructor or class
15   * and then also with lowercase s
16   */
17  const Sequelize = require('sequelize')
18
19  const sequelize = require('../util/database')
20
21  const User = sequelize.define('user', {
22    id: {
23      type: Sequelize.INTEGER,
24      autoIncrement: true,
25      allowNull: false,
26      primaryKey: true
27    },
28    name: Sequelize.STRING,
29    email: Sequelize.STRING
30  })
31
32  module.exports = User

```

## \* Chapter 158: Adding A One-To-Many Relationship

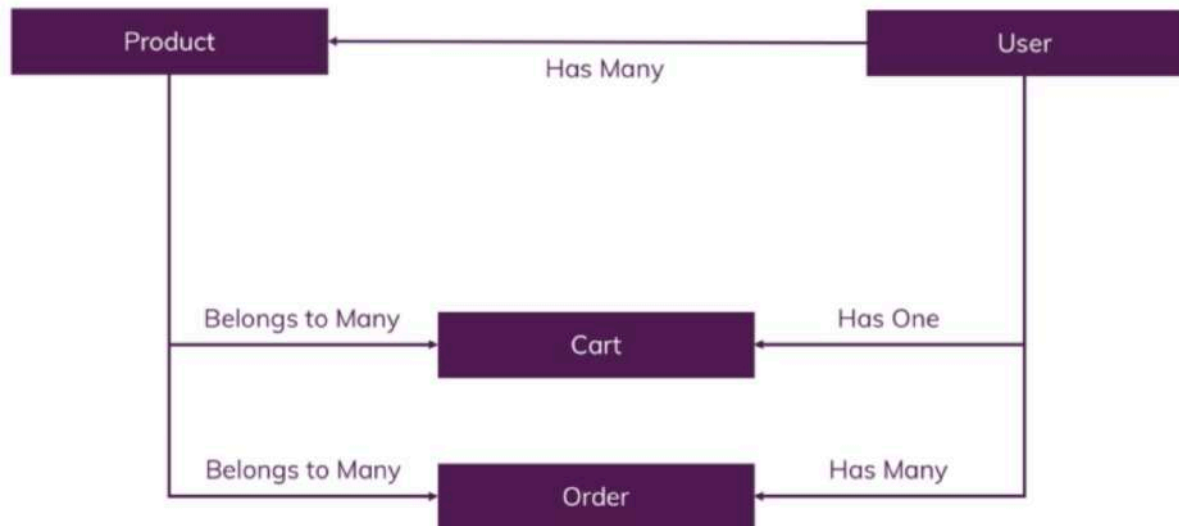
1. update

- app.js





## Associations



The screenshot shows the VS Code editor with the following code in `app.js`:

```

20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/admin', adminRoutes);
23 app.use(shopRoutes);
24
25 app.use(errorController.get404);
26
27 Product.belongsTo(User, { constraints: true, onDelete: 'CASCADE' });
28 User.hasMany(Product);
29
30 sequelize
31   .sync({ force: true })
32   .then(result => {
33     // console.log(result);
34     app.listen(3000);
35   })
  
```

The terminal output shows the following commands and results:

```

R(255), 'price' DOUBLE PRECISION NOT NULL, 'imageUrl' VARCHAR(255) NOT NULL, 'description' VARCHAR(255) NOT NULL,
'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL, PRIMARY KEY ('id')) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `products`
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Executing (default): DROP TABLE IF EXISTS `products`;
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): CREATE TABLE IF NOT EXISTS `users` (`id` INTEGER NOT NULL auto_increment, `name` VARCHAR(25
5), `email` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGIN
E=InnoDB;
Executing (default): SHOW INDEX FROM `users`
  
```

- after restarting, we see a couple of statement were executed.
  - first of all, it dropped any existing tables
- 

```

20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/admin', adminRoutes);
23 app.use(shopRoutes);
24
25 app.use(errorController.get404);
26
27 Product.belongsTo(User, { constraints: true, onDelete: 'CASCADE' });
28 User.hasMany(Product);
29
30 // sequelize
31 .sync({ force: true })
32 .then(result => {
33   // console.log(result);
34   app.listen(3000);
35 })

```

```

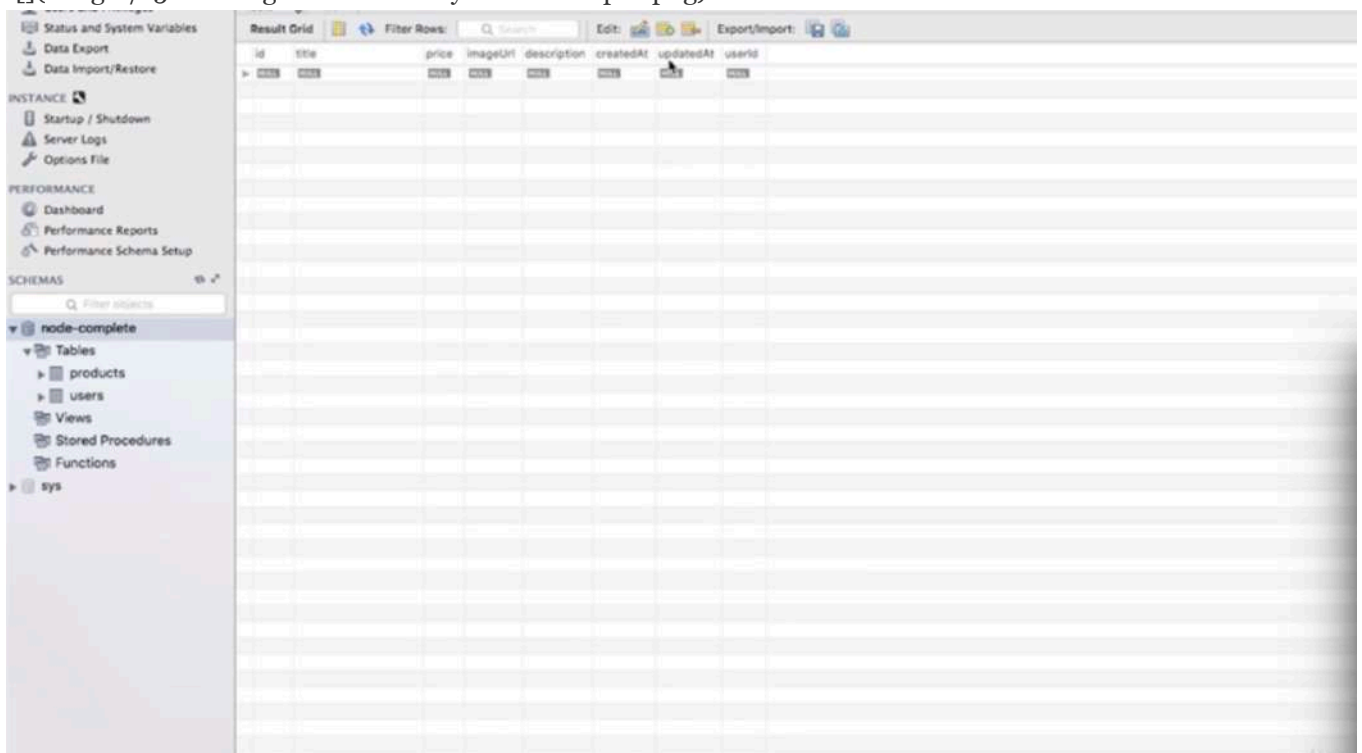
Executing (default): DROP TABLE IF EXISTS `products`;
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): DROP TABLE IF EXISTS `users`;
Executing (default): CREATE TABLE IF NOT EXISTS `users` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `email` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `users`
Executing (default): DROP TABLE IF EXISTS `products`;
Executing (default): CREATE TABLE IF NOT EXISTS `products` (`id` INTEGER NOT NULL auto_increment , `title` VARCHAR(255), `price` DOUBLE PRECISION NOT NULL, `imageUrl` VARCHAR(255) NOT NULL, `description` VARCHAR(255) NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `userId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`userId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB;

```

- and then it created a new table, 'users' with all the setup  

- and then it also creates a new products table  

- and beside adding all the fields there, it also defined that there is a new field, the user id fields which is an INTEGER and which is a FOREIGN KEY that references the ID field in the users table.
- and that 'ON DELETE', it should 'CASCADE' and 'ON UPDATE CASCADE' is the default.
- so this is some meta setup in the database which sequelize now also added to connect our tables  
  
  

- if we go to workbench and we right click on our database and set call 'refresh All', we see there 2 tables now.
- if we inspect products, we see that our product is gone because it recreated the table.

- but now besides createdAt, updatedAt that were added by sequelize
- there is a userId field which was also added by sequelize and this will automatically be populated by sequelize once we create products that are related to a user.
- let's make sure we have a user because that table is empty and that we then can connect users and products in our app.

```
1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const sequelize = require('./util/database');
10 const Product = require('./models/product')
11 const User = require('./models/user')
12
13 const app = express();
14
15 app.set('view engine', 'ejs');
16 app.set('views', 'views');
17
18 const adminRoutes = require('./routes/admin');
19
20 const shopRoutes = require('./routes/shop');
21
22 app.use(bodyParser.urlencoded({ extended: false }));
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 app.use('/admin', adminRoutes);
26 app.use(shopRoutes);
27
28 app.use(errorController.get404);
29
30 /**with the 2 models('Product', 'User'),
31 * we can relate them and i will relate them in the same place where i sequelize
32 * but before i sync it.
33 */
34
35 /**we can also configure this by passing a second argument which is optional
36 * we can define how this relationship would be managed
37 * and very important, we can define so-called constraints
38 *
39 * and for example, 'onDelete'
40 * so if a user is deleted, what should happen to any connected products?
41 * and here we can say 'CASCADE' which means the deletion would then also be executed for
the product
42 * so if we delete a User, any price related to User would be gone
43 */
44
45 Product.belongsTo(User, {constraint: true, onDelete: 'CASCADE'})
46 /**you can also define the inverse
47 * and say that one User has many products
48 * because one user can add more than one product to the shop
49 *
```



```

50  * this is optional, you don't need that.
51  * you can replace 'belongsTo' with a 'hasMany' call
52  * but here i also like to define both directions to really make it clear how this relation
works.
53  */
54  User.hasMany(Product)
55
56  /**sequelize sync will not just create tables for our models
57  * but also define the relation in our database as we define them.
58  *
59  * The one problem is that we already created the products table
60  * and therefore will not override it with the new information
61  * and we can ensure that it will by setting 'force' to true
62  * a setting you wouldn't use in production
63  * because you don't always wanna overwrite your tables all the time.
64  */
65  sequelize
66    .sync({force: true})
67    .then(result => {
68      //console.log(result)
69      app.listen(3000);
70    })
71    .catch(err => {
72      console.log(err)
73    })

```

## \* Chapter 159: Creating & Managing A Dummy User

1. update

- app.js



- here's my user creation result that i get back



- if i go back to my workbench database and i refresh the user's table, we see the user here

- if i do restart my server with 'npm start' and that still works and it doesn't create a new user because we already have one.

```

1  //app.js
2
3  const path = require('path');
4
5  const express = require('express');
6  const bodyParser = require('body-parser');
7
8  const errorController = require('./controllers/error');
9  const sequelize = require('./util/database');
10 const Product = require('./models/product')
11 const User = require('./models/user')
12
13 const app = express();
14

```

```

15 app.set('view engine', 'ejs');
16 app.set('views', 'views');
17
18 const adminRoutes = require('./routes/admin');
19
20 const shopRoutes = require('./routes/shop');
21
22 app.use(bodyParser.urlencoded({ extended: false }));
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 /**i will register a new middleware
26  * because i wanna store that user in my request
27  * so that i can use it from anywhere in my app conveniently
28  */
29 app.use((req, res, next) => {
30     /**i wanna reach out to my database
31     * and retrieve my user with 'User.findById(1)'
32     *
33     * 'app.use()' only register a middleware
34     * so for incoming request, we will then execute this function
35     *
36     * 'npm start' runs this code for the first time
37     * and 'npm start' is what runs sequelize, not incoming request.
38     * incoming requests are only funneled through our middleware
39     * so 'npm start' starts runs below which sets up our database
40     * but never this anonymous function which just register it as middleware for incoming
    request.
41     *
42     *      sequelize
43     *      .sync()
44     *      .then(result => {
45     *          return User.findById(1)
46     *      })
47     *      .then(user => {
48     *          if(!user){
49     *              return User.create({ name: 'Max', email: 'test@test.com'})
50     *          }
51     *          return user
52     *      })
53     *      .then(user => {
54     *          app.listen(3000)
55     *      })
56     *      .catch(err => {
57     *          console.log(err)
58     *      })
59     *
60     */
61
62     /**so 'User.findById(1)' will only run for incoming requests
63     * which, on the other hand, can only reach this
64     * if we did successfully start our server with 'app.listen(3000)'
65     * and that in turn is only true
66     * if we are done with our initialization code here.
67     *
68     * so we are guaranteed to find a user
69     */

```

```

70     User.findById(1)
71     .then(user => {
72         /**i wanna store it in a request.
73         * we can add a new field to our request object
74         * we should make sure we don't overwrite an existing one like body
75         *
76         * but 'user' is undefined by default
77         * now i'm storing the user i retrieved from the database in there.
78         *
79         * keep in mind that the user we are retrieving from the database is not just a
javascript object with values stored in a database.
80         * it's sequelize object with the value stored in the database
81         * and with all these utility methods sequelize added, like destroy,
82         * so we are storing this sequelize object here in the request
83         * and not just a javascript object with the field values
84         *
85         * so whenever we call 'req.user' in the future in our app,
86         * we can also execute method like 'destroy'
87         */
88         req.user = user
89         /**call 'next()' so that we can continue with the next step
90         * if we get our user and stored it.
91         */
92         next()
93     })
94     .catch(err => console.log(err))
95 })
96
97 app.use('/admin', adminRoutes);
98 app.use(shopRoutes);
99
100 app.use(errorController.get404);
101
102 Product.belongsTo(User, {constraint: true, onDelete: 'CASCADE'})
103 User.hasMany(Product)
104
105 sequelize
106     .sync({force: true})
107     .sync()
108     .then(result => {
109         /**i will use my User model
110         * and first of all, check if i find a user with the ID 1
111         * and this is just some dummy code to see if i do have 1 User
112         * because i only need 1 for now as we have no authentication
113         * if i do have it, i will not create a new one,
114         * if i don't have it, i will.
115         */
116         return User.findById(1)
117         //console.log(result)
118     })
119     .then(user => {
120         if(!user){
121             return User.create({ name: 'Max', email: 'test@test.com' })
122         }
123         /**now we are inconsistent
124         * because now this anonymous function either returns a promise or just an object

```

```

125     * we should always return the same
126     * so that we can chain then here successfully
127     *
128     * and therefore i will call 'Promise.resolve' which is essentially a promise
129     * that will immediately resolve to user
130     *
131     * technically you can omit 'Promise.resolve()'
132     * because if you return a value like 'return user' in a 'then()' block,
133     * it is automatically wrapped into a new promise
134     * just wanted to highlight that you should make sure that the value are equal
135     */
136     return user
137   })
138   .then(user => {
139     //console.log(user)
140     app.listen(3000);
141   })
142   .catch(err => {
143     console.log(err)
144   })
145

```

## \* Chapter 160: Using Magic Association Methods







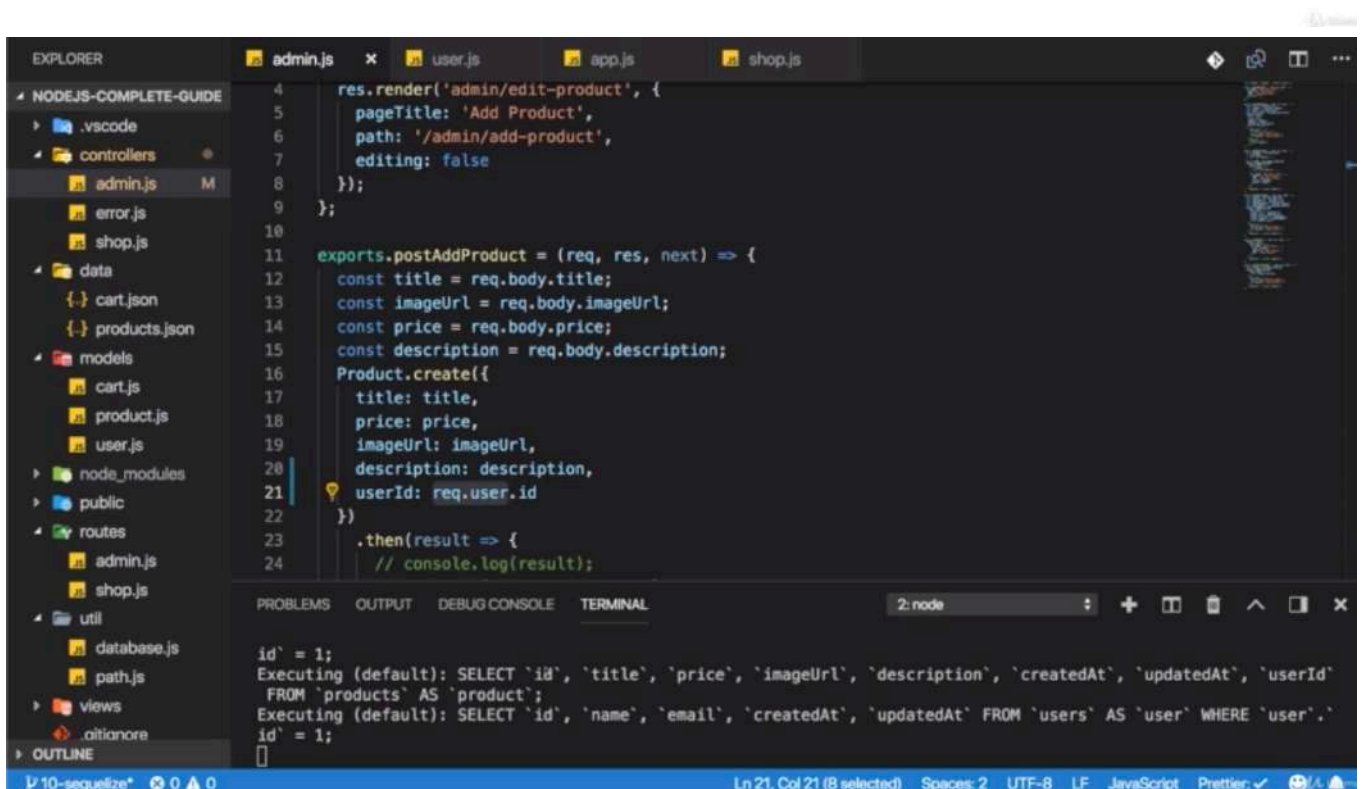
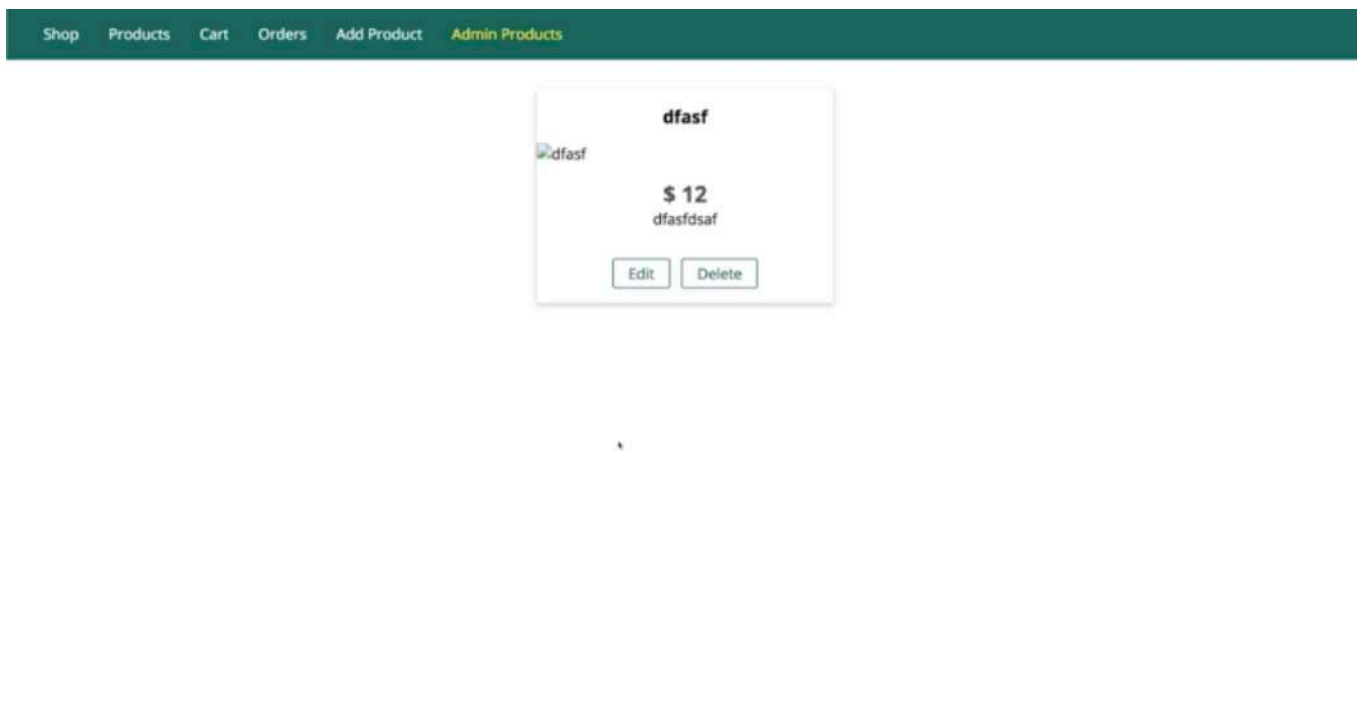
[Shop](#)
[Products](#)
[Cart](#)
[Orders](#)
[Add Product](#)
[Admin Products](#)

Title

Image URL

Price

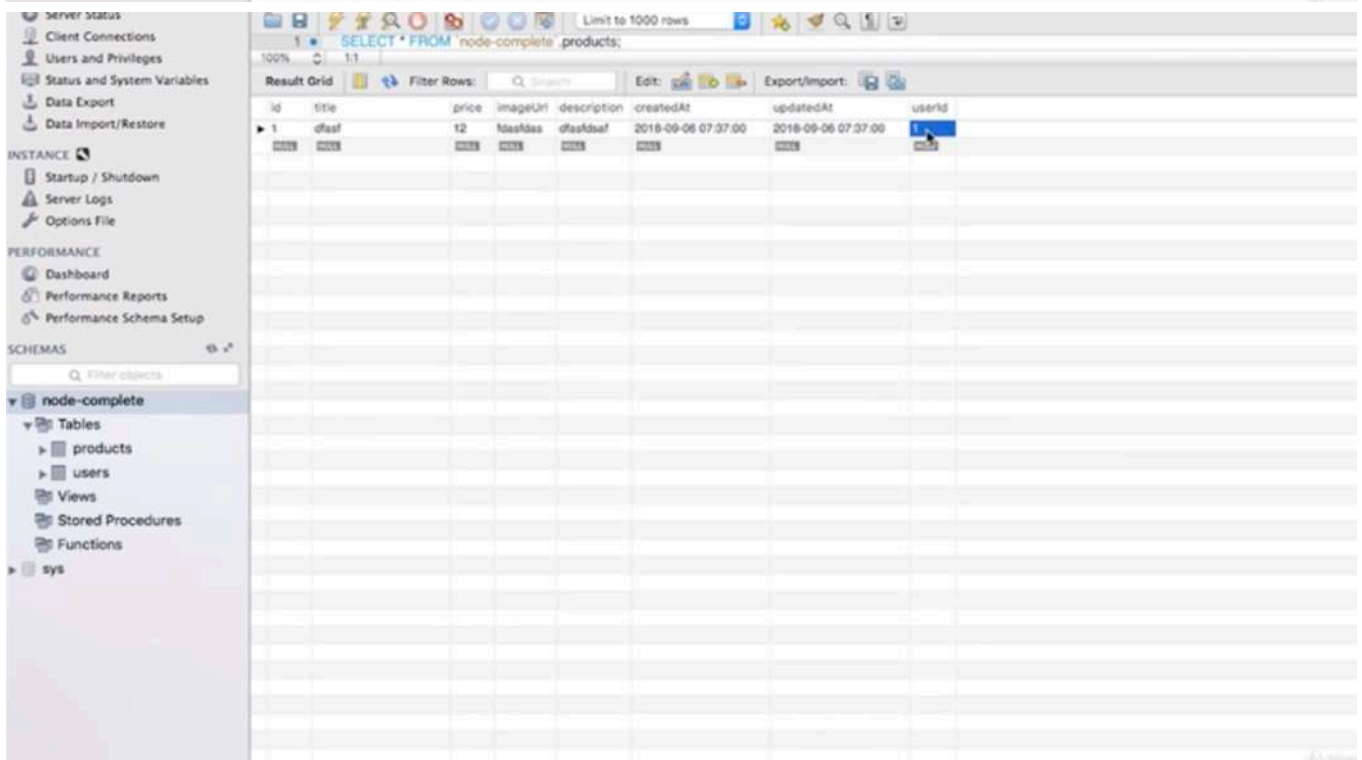
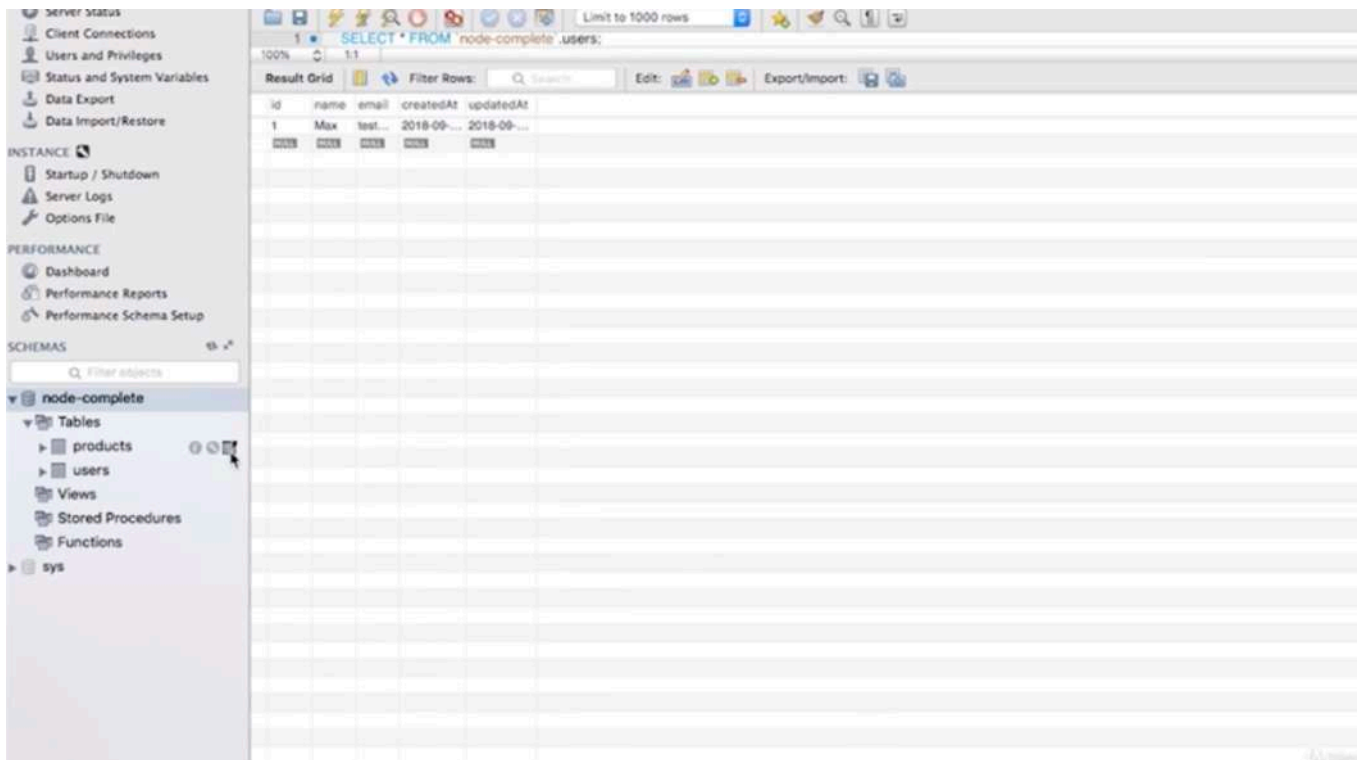
Description



- no error here.







- if we look into our products table by clicking on this icon here, we have the user ID stored here.









Title

fadsfdas

Image URL

fasdfasdf

Price

12

Description

fdsafads

Add Prbduct

dfasf

dfasf

\$ 12

dfasfdsaf

EditDelete

fadsfdas

fadsfdas

\$ 12

fdsafads

EditDelete

```
11 exports.postAddProduct = (req, res, next) => {
12   const title = req.body.title;
13   const imageUrl = req.body.imageUrl;
14   const price = req.body.price;
15   const description = req.body.description;
16   req.user
17     .createProduct({
18     title: title,
19     price: price,
20     imageUrl: imageUrl,
21     description: description
22   })
23   .then(result => {
24     // console.log(result);
25     console.log('Created Product');
26     res.redirect('/admin/products');
27   })
28   .catch(err => {
29     console.log(err);
30   });
31 }
```

FROM `products` AS `product`;  
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`id` = 1;  
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`id` = 1;

id	title	price	imageUrl	description	createdAt	updatedAt	userId
1	dfoaf	12	fdoafdoaf	fdoafdoaf	2018-09-06 07:37:00	2018-09-06 07:37:00	1
2	fdoafdoaf	12	fdoafdoaf	fdoafdoaf	2018-09-06 07:39:23	2018-09-06 07:39:23	1

- we see we also get the 'userId' even though we didn't set it explicitly.
- this is done by sequelize with this magic way of connecting it.

```
1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const sequelize = require('./util/database');
10 const Product = require('./models/product')
11 const User = require('./models/user')
12
```



```

13 const app = express();
14
15 app.set('view engine', 'ejs');
16 app.set('views', 'views');
17
18 const adminRoutes = require('./routes/admin');
19
20 const shopRoutes = require('./routes/shop');
21
22 app.use(bodyParser.urlencoded({ extended: false }));
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 app.use((req, res, next) => {
26   /**from now on, all new products that are created should be associated to the currently
    logged in user
27   * and for now, this below will only be this one dummy user
28   *
29   *     User.findById(1)
30   *       .then(user => {
31   *         req.user = user
32   *         next()
33   *       })
34   *       .catch(err => console.log(err))
35   *   })
36   *
37   * that means if i'm in the ./controllers/admin.js,
38   * when we create a new product in 'postAddProduct',
39   * we will not create the product like this anymore
40   */
41   User.findById(1)
42     .then(user => {
43       req.user = user
44       next()
45     })
46     .catch(err => console.log(err))
47 })
48
49 app.use('/admin', adminRoutes);
50 app.use(shopRoutes);
51
52 app.use(errorController.get404);
53
54 Product.belongsTo(User, {constraint: true, onDelete: 'CASCADE'})
55 User.hasMany(Product)
56
57 sequelize
58   .sync({force: true})
59   .sync()
60   .then(result => {
61     return User.findById(1)
62     //console.log(result)
63   })
64   .then(user => {
65     if(!user){
66       return User.create({ name: 'Max', email: 'test@test.com' })
67     }

```

```

68         return user
69     })
70     .then(user => {
71         //console.log(user)
72         app.listen(3000);
73     })
74     .catch(err => {
75         console.log(err)
76     })
77

```

```

1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6     res.render('admin/edit-product', {
7         pageTitle: 'Add Product',
8         path: '/admin/add-product',
9         editing: false
10    });
11 };
12
13 /** that means if i'm in the ./controllers/admin.js,
14  * when we create a new product in 'postAddProduct',
15  * we will not create the product like this anymore
16  * we need to pass in extra information regarding our user that is associated.
17  *
18  * one way of doing this is that we set this new user id field we got
19  * the user ID was added as a database field
20  * because we now have a relation set up
21  * and we set this to req.user.id
22  *
23  * keep in mind that req.user is the sequelize user object
24  * which holds both the database data for that user as well as helper methods
25  *
26  * this should create new products with that user being associated to it.*/
27 exports.postAddProduct = (req, res, next) => {
28     const title = req.body.title;
29     const imageUrl = req.body.imageUrl;
30     const price = req.body.price;
31     const description = req.body.description;
32     /** we have another cool feature of sequelize,
33      * we can use our user object as it's stored in the request
34      *
35      * keep in mind that this is a sequelize object with all the magic features
36      * and there we will actually have 'req.user.createProduct()' method
37      *
38      * sequelize add special methods depending on the association you added
39      * and for 'Belongs-To' has Many association as we did,
40      * sequelize adds methods that allows us, for example, to create a new associated object
41      * so since a user has many products or a product belongs to a user as the setup in
42      app.js,
43      * since we have that relation defined like below
44      *
45      * Product.belongsTo(User, { constraints: true, onDelete: 'CASCADE' })
46      * User.hasMany(Product)

```

```

46  *
47  * sequelize automatically adds a createProduct() method to the user
48  * 'createProduct()' because our model is named product
49  * and create is then automatically added at the beginning of the method name.
50  * that is by sequelize.
51  */
52
53  /** the rest doesn't change
54   * but this now automatically creates a connected model.
55   */
56  req.user.createProduct({
57    title: title,
58    price: price,
59    imageUrl: imageUrl,
60    description: description
61  })
62  .then(result => {
63    // console.log(result);
64    console.log('Created Product');
65    res.redirect('/admin/products')
66  })
67  .catch(err => {
68    console.log(err);
69  });
70 };
71
72 exports.getEditProduct = (req, res, next) => {
73   const editMode = req.query.edit;
74   if (!editMode) {
75     return res.redirect('/');
76   }
77   const prodId = req.params.productId;
78   Product.findById(prodId)
79     .then(product => {
80       if (!product) {
81         return res.redirect('/');
82       }
83       res.render('admin/edit-product', {
84         pageTitle: 'Edit Product',
85         path: '/admin/edit-product',
86         editing: editMode,
87         product: product
88       });
89     })
90     .catch(err => console.log(err));
91 };
92
93 exports.postEditProduct = (req, res, next) => {
94   const prodId = req.body.productId;
95   const updatedTitle = req.body.title;
96   const updatedPrice = req.body.price;
97   const updatedImageUrl = req.body.imageUrl;
98   const updatedDesc = req.body.description;
99   Product.findById(prodId)
100     .then(product => {
101       product.title = updatedTitle;

```

```

102     product.price = updatedPrice;
103     product.description = updatedDesc;
104     product.imageUrl = updatedImageUrl;
105     return product.save();
106   })
107   .then(result => {
108     console.log('UPDATED PRODUCT!');
109     res.redirect('/admin/products');
110   })
111   .catch(err => console.log(err));
112   });
113
114 exports.getProducts = (req, res, next) => {
115   Product.findAll()
116     .then(products => {
117       res.render('admin/products', {
118         prods: products,
119         pageTitle: 'Admin Products',
120         path: '/admin/products'
121       });
122     })
123     .catch(err => console.log(err));
124   });
125
126 exports.postDeleteProduct = (req, res, next) => {
127   const prodId = req.body.productId;
128   Product.findById(prodId)
129     .then(product => {
130       return product.destroy()
131     })
132     .then(result => {
133       console.log('DESTROYED PRODUCT')
134       res.redirect('/admin/products');
135     })
136     .catch(err => console.log(err))
137   });
138

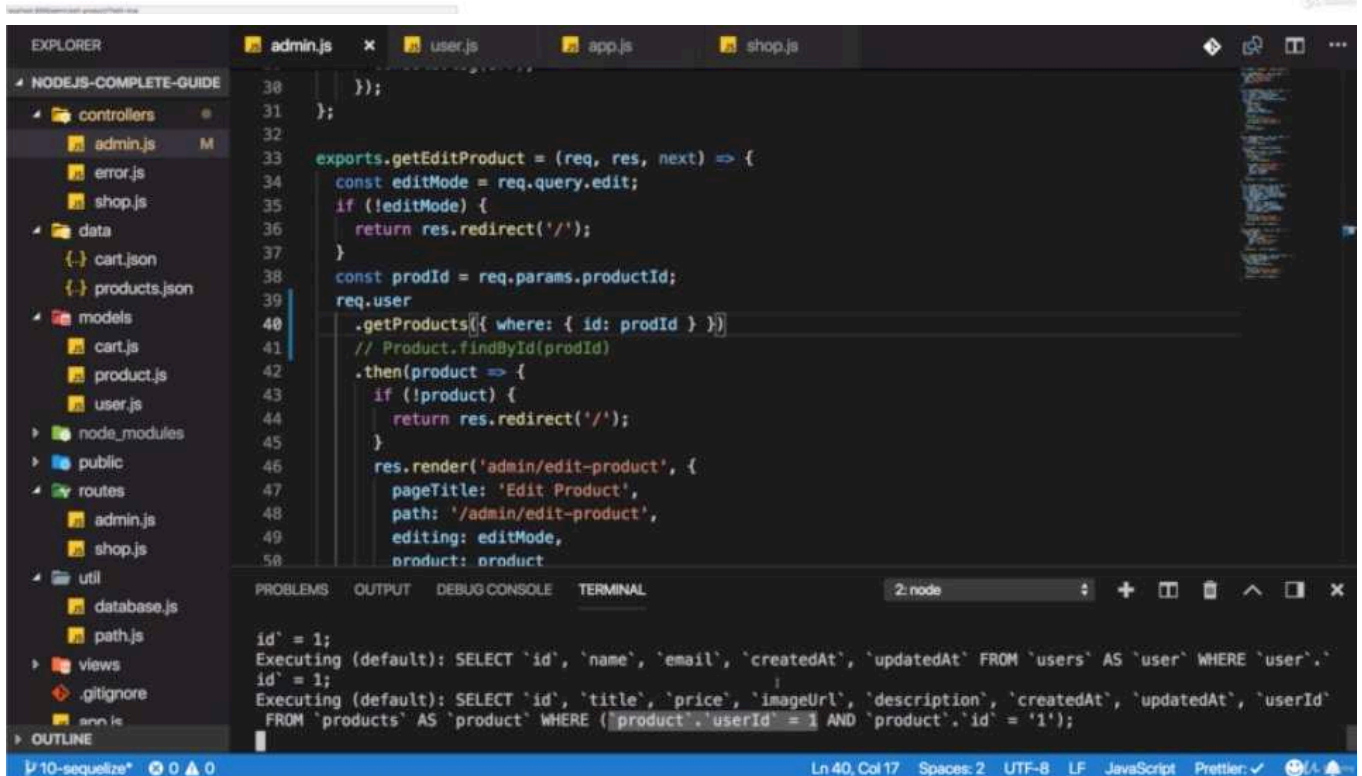
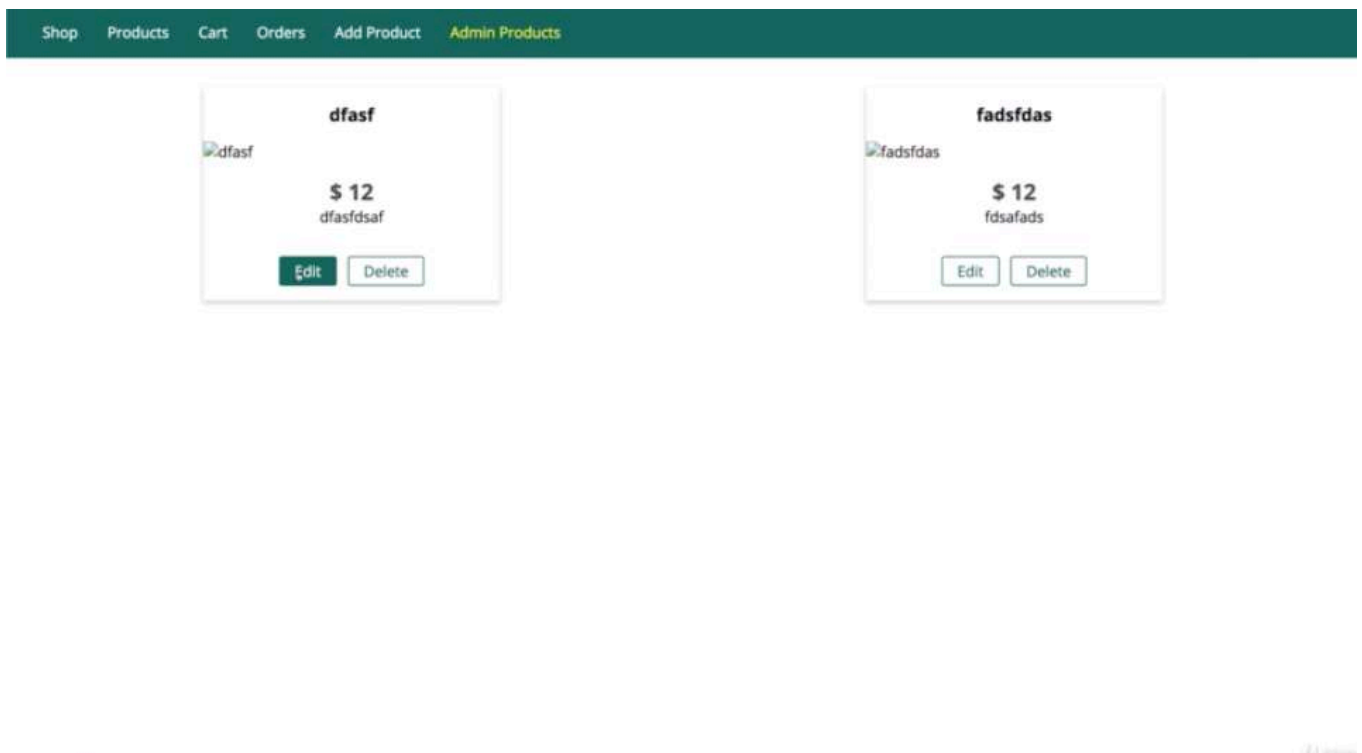
```

## \* Chapter 161: Fetching Related Products

1. update  
- ./controllers/admin.js







- we see the SQL statement here where it looks for a product with the `userId = 1` that is not the condition we wrote, we are responsible for this part `(product, 'id' = '1')` where it then also narrows down the product id.
  - but `user id = 1` was added by sequelize because we use `'getProducts'` on the user
  - keep in mind we get back an array even if it only holds one element. so we got products and therefore we know that one product, the one we are interested in will always be the first element. so we have to store that separately in a new constant.
- 

Title

dfasf

Image URL

fdasfdas

Price

12

Description

dfasfdsaf

Update Product

- so if we reload, this works





dfasf



\$ 12

dfasfdsaf

EditDelete

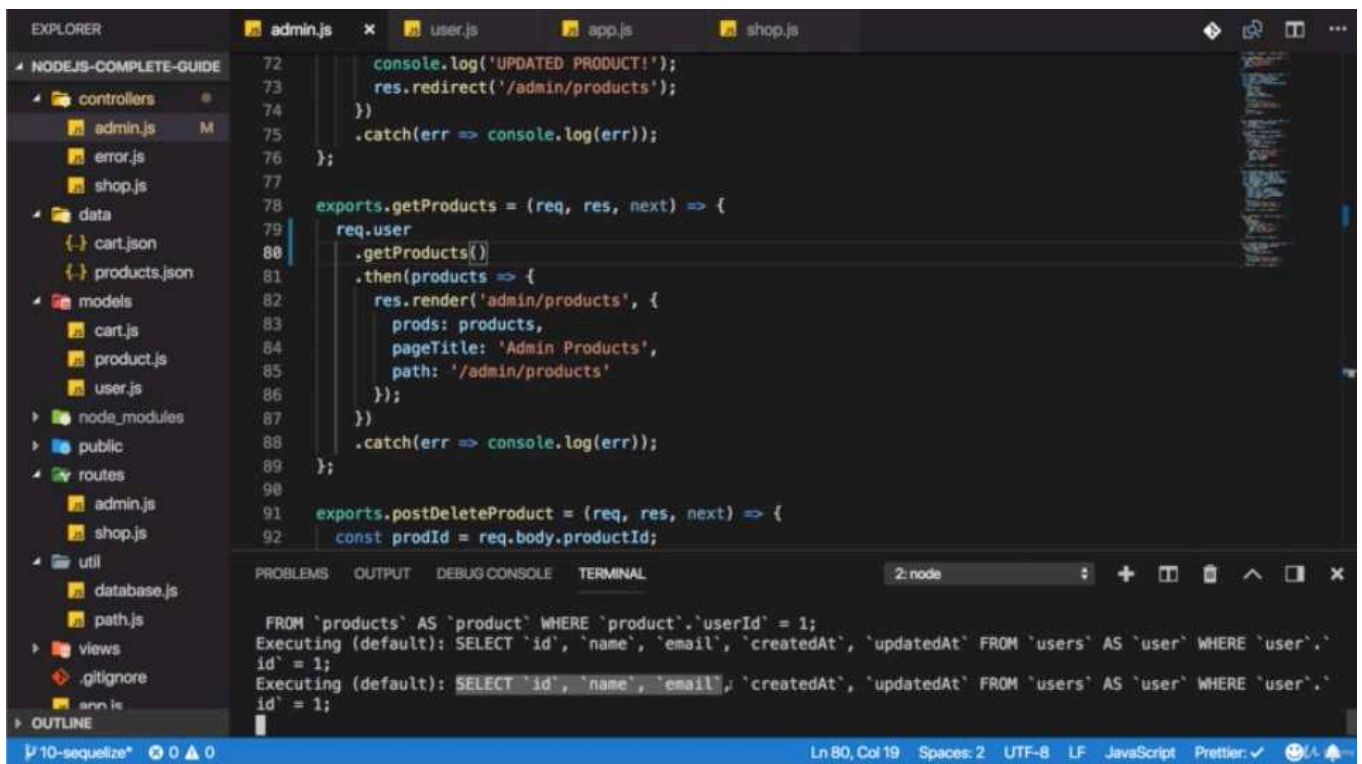
fadsfdas



\$ 12

fdsafads

EditDelete



The screenshot shows a VS Code editor with the following structure:

- EXPLORER: NODEJS-COMPLETE-GUIDE, controllers (admin.js, error.js, shop.js), data (cart.json, products.json), models (cart.js, product.js, user.js), node\_modules, public, routes (admin.js, shop.js), util (database.js, path.js, views), .gitignore, .env file, OUTLINE.
- admin.js (selected):

```
72 console.log('UPDATED PRODUCT!');
73 res.redirect('/admin/products');
74 })
75 .catch(err => console.log(err));
76 };
77
78 exports.getProducts = (req, res, next) => {
79   req.user
80     .getProducts()
81     .then(products => {
82       res.render('admin/products', {
83         prods: products,
84         pageTitle: 'Admin Products',
85         path: '/admin/products'
86       });
87     })
88     .catch(err => console.log(err));
89 };
90
91 exports.postDeleteProduct = (req, res, next) => {
92   const prodId = req.body.productId;
```
- TERMINAL:

```
FROM `products` AS `product` WHERE `product`.`userId` = 1;
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`id` = 1;
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`id` = 1;
```

```
1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
15   const imageUrl = req.body.imageUrl;
16   const price = req.body.price;
17   const description = req.body.description;
18   req.user.createProduct({
19     title: title,
20     price: price,
21     imageUrl: imageUrl,
22     description: description
23   })
24   .then(result => {
25     // console.log(result);
26     console.log('Created Product');
27     res.redirect('/admin/products')
28   })
29   .catch(err => {
30     console.log(err);
31   });
32 };
33
34 exports.getEditProduct = (req, res, next) => {
35   const editMode = req.query.edit;
```

```

36   if (!editMode) {
37     return res.redirect('/');
38   }
39   const prodId = req.params.productId;
40   req.user
41     .getProducts({ where: { id: prodId } })
42     //Product.findByPk(prodId)
43     /**keep in mind we get back an array even if it only holds one element.
44      * so we got 'products'
45      * and therefore we know that one product,
46      * the one we are interested in will always be the first element
47      * so we have to store that separately in a new constant */
48     .then(products => {
49       const product = products[0]
50       if (!product) {
51         return res.redirect('/');
52       }
53       res.render('admin/edit-product', {
54         pageTitle: 'Edit Product',
55         path: '/admin/edit-product',
56         editing: editMode,
57         product: product
58       });
59     })
60     .catch(err => console.log(err));
61   });
62
63   exports.postEditProduct = (req, res, next) => {
64     const prodId = req.body.productId;
65     const updatedTitle = req.body.title;
66     const updatedPrice = req.body.price;
67     const updatedImageUrl = req.body.imageUrl;
68     const updatedDesc = req.body.description;
69     Product.findByPk(prodId)
70       .then(product => {
71         product.title = updatedTitle;
72         product.price = updatedPrice;
73         product.description = updatedDesc;
74         product.imageUrl = updatedImageUrl;
75         return product.save();
76       })
77       .then(result => {
78         console.log('UPDATED PRODUCT!');
79         res.redirect('/admin/products');
80       })
81       .catch(err => console.log(err));
82   });
83
84   exports.getProducts = (req, res, next) => {
85     req.user
86       .getProducts()
87       .then(products => {
88         res.render('admin/products', {
89           prods: products,
90           pageTitle: 'Admin Products',
91           path: '/admin/products'

```



```

92     });
93   })
94   .catch(err => console.log(err));
95 };
96
97 exports.postDeleteProduct = (req, res, next) => {
98   const prodId = req.body.productId;
99   Product.findById(prodId)
100   .then(product => {
101     return product.destroy()
102   })
103   .then(result => {
104     console.log('DESTROYED PRODUCT')
105     res.redirect('/admin/products');
106   })
107   .catch(err => console.log(err))
108 };
109

```

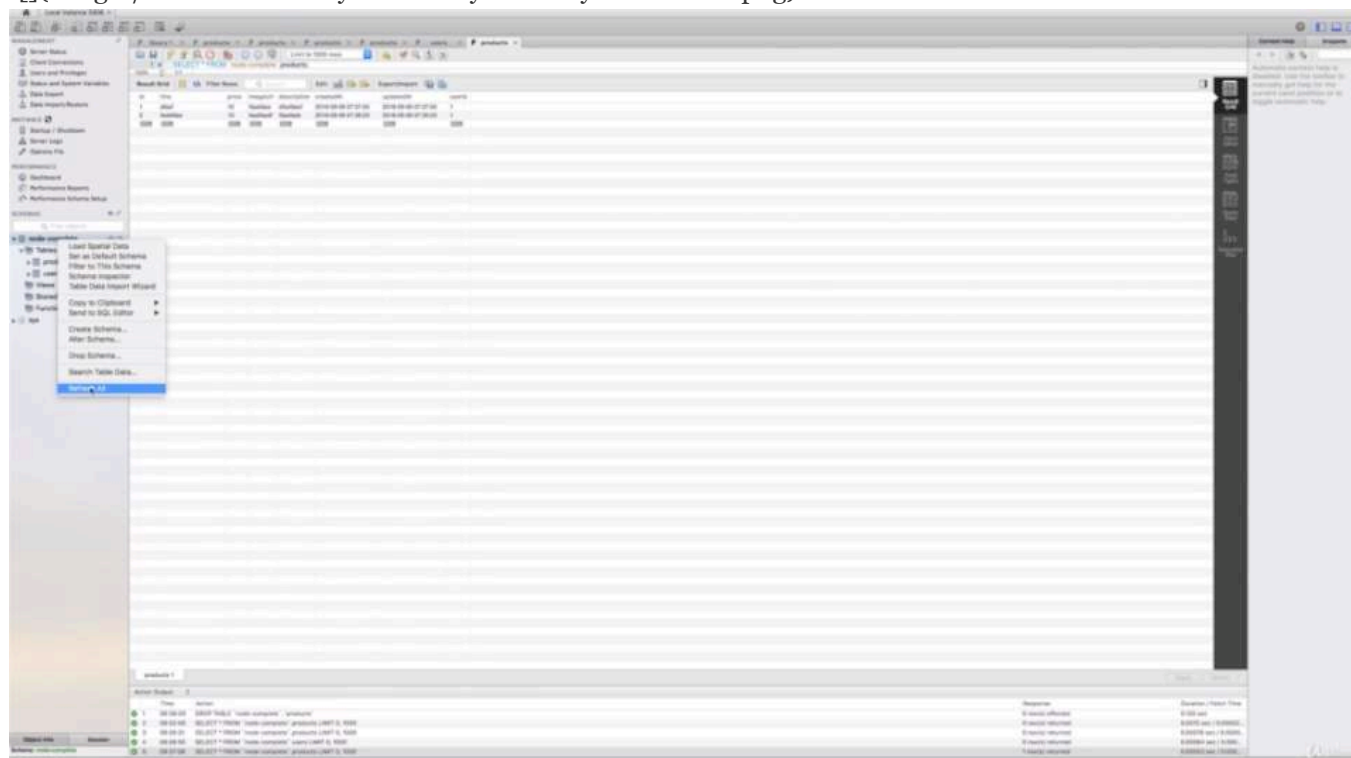
## \* Chapter 162: One-To-Many & Many-To-Many Relations

1. update

- ./models/cart.js
- ./models/cart-item.js
- app.js







id	title	price	imageUrl	description	createdAt	updatedAt	userId
1	dhasf	12	fasdfas	dhasdfasf	2018-09-06 07:37:00	2018-09-06 07:37:00	1
2	fasdfas	12	fasdfasf	fasdfas	2018-09-06 07:39:23	2018-09-06 07:39:23	1

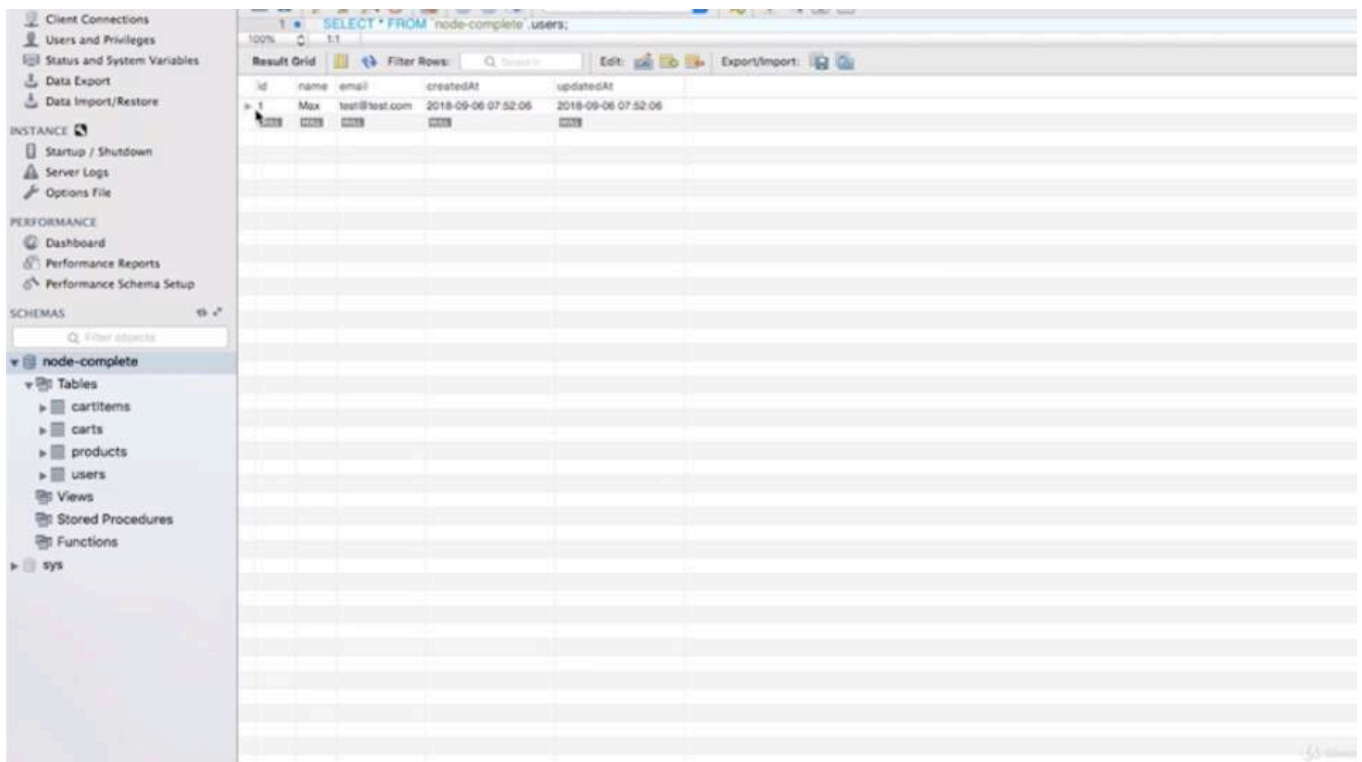
- if we go back and refresh all, now we see carts and cartitems



id	createdAt	updatedAt	userId
1	2018-09-06 07:37:00	2018-09-06 07:37:00	1

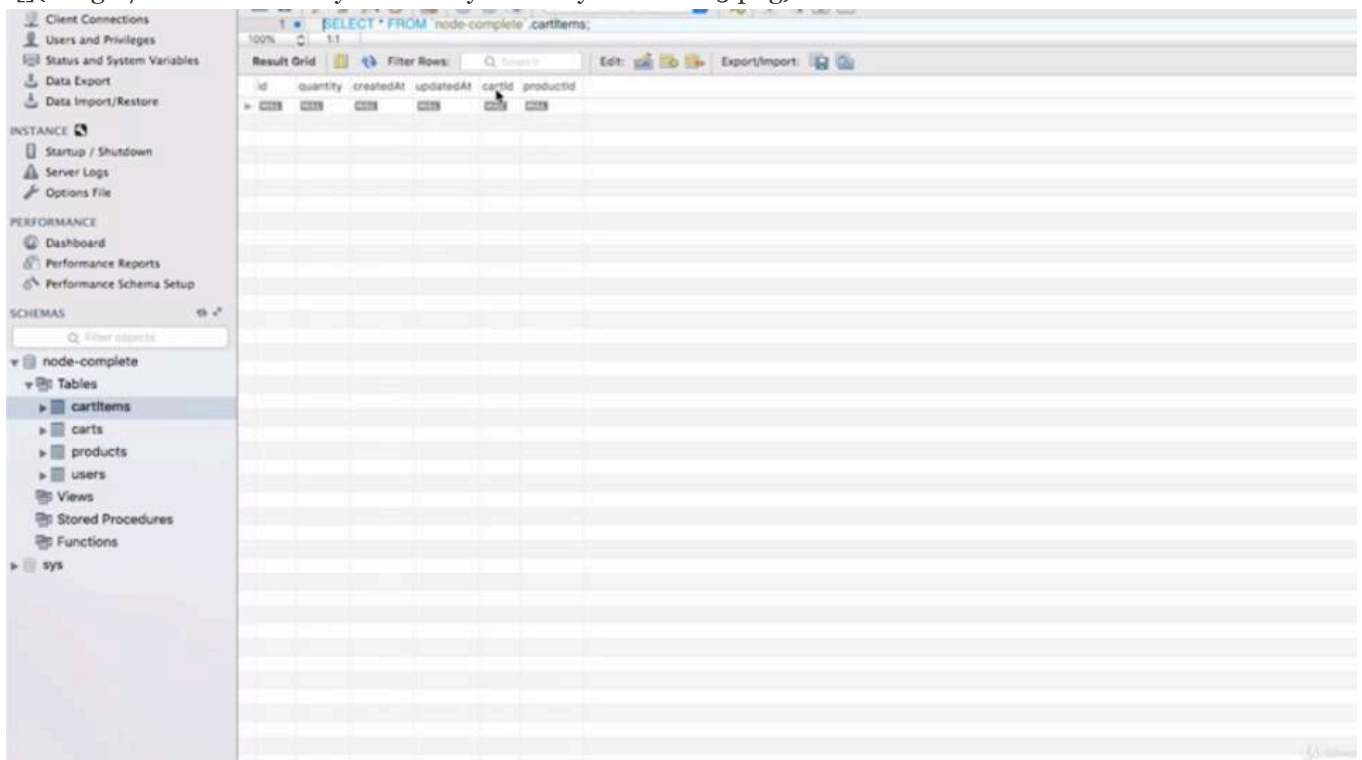
- and we can see that in carts, we only got id, createdAt, updatedAt, userId to which the cart belongs.





- in users, we don't care about that, we only have the user information





- and in cartitems, we have a combination of the cartId to which this cartitem belongs and the productId

- so now we get everything we need to manage our cart items for the carts we need

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const sequelize = require('./util/database');

```

```

10 const Product = require('./models/product')
11 const User = require('./models/user')
12 const Cart = require('./models/cart')
13 const CartItem = require('./models/cart-item')
14
15 const app = express();
16
17 app.set('view engine', 'ejs');
18 app.set('views', 'views');
19
20 const adminRoutes = require('./routes/admin');
21
22 const shopRoutes = require('./routes/shop');
23
24 app.use(bodyParser.urlencoded({ extended: false }));
25 app.use(express.static(path.join(__dirname, 'public')));
26
27 app.use((req, res, next) => {
28     User.findByPk(1)
29         .then(user => {
30             req.user = user
31             next()
32         })
33         .catch(err => console.log(err))
34 })
35
36 app.use('/admin', adminRoutes);
37 app.use(shopRoutes);
38
39 app.use(errorController.get404);
40
41 Product.belongsTo(User, {constraint: true, onDelete: 'CASCADE'})
42 User.hasMany(Product)
43 /**this is basically the inverse of this relation
44  * and it's optional
45  * you don't need to add it.
46  * one direction is enough.
47  */
48 User.hasOne(Cart)
49 Cart.belongsTo(User)
50 /**it's a many-to-many relationship
51  * because one cart can hold multiple products
52  * and a single product can be part of multiple different carts
53  * this only works with an intermediate table that connects them
54  * which basically stores a combination of Product Ids and Cart Ids
55  * for that, i created my CartItem model
56  *
57  * and we add the 'through' key telling sequelize
58  * where these connection should be stored
59  * and that is my CartItem model.
60  * so i will add that to both 'belongsToMany' calls here.
61  */
62 Cart.belongsToMany(Product, { through: CartItem })
63 Product.belongsToMany(Cart, { through: CartItem })
64
65 sequelize

```

```

66     .sync({force: true})
67     //.sync()
68     .then(result => {
69         return User.findByPk(1)
70         //console.log(result)
71     })
72     .then(user => {
73         if(!user){
74             return User.create({ name: 'Max', email: 'test@test.com' })
75         }
76         return user
77     })
78     .then(user => {
79         //console.log(user)
80         app.listen(3000);
81     })
82     .catch(err => {
83         console.log(err)
84     })
85

```

```

1  //./models/cart-item.js
2
3  const Sequelize = require('sequelize');
4
5  const sequelize = require('../util/database');
6
7  const CartItem = sequelize.define('cartItem', {
8      id: {
9          type: Sequelize.INTEGER,
10         autoIncrement: true,
11         allowNull: false,
12         primaryKey: true
13     },
14     /**now the ID of the cart to which this is related doesn't have to be added by us
15     * because we will again create an association
16     * and let sequelize manage this
17     * so it's time for some associations.
18     */
19     quantity: Sequelize.INTEGER
20 });
21
22 module.exports = CartItem;
23

```

```

1  //./models/cart.js
2
3  const Sequelize = require('sequelize');
4
5  const sequelize = require('../util/database');
6
7  const Cart = sequelize.define('cart', {
8      id: {
9          type: Sequelize.INTEGER,
10         autoIncrement: true,
11         allowNull: false,
12         primaryKey: true

```

```

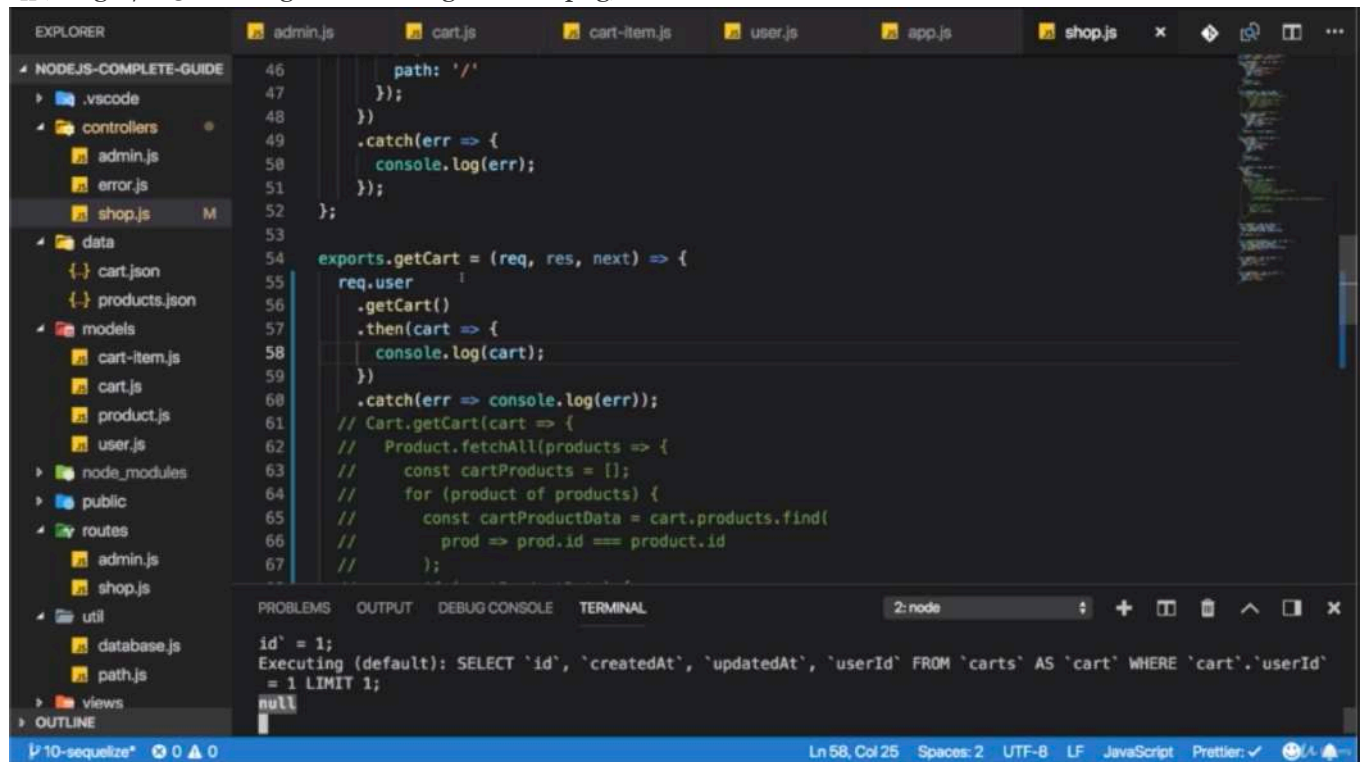
13   }
14   });
15
16   module.exports = Cart;

```

## \* Chapter 163: Creating & Fetching A Cart

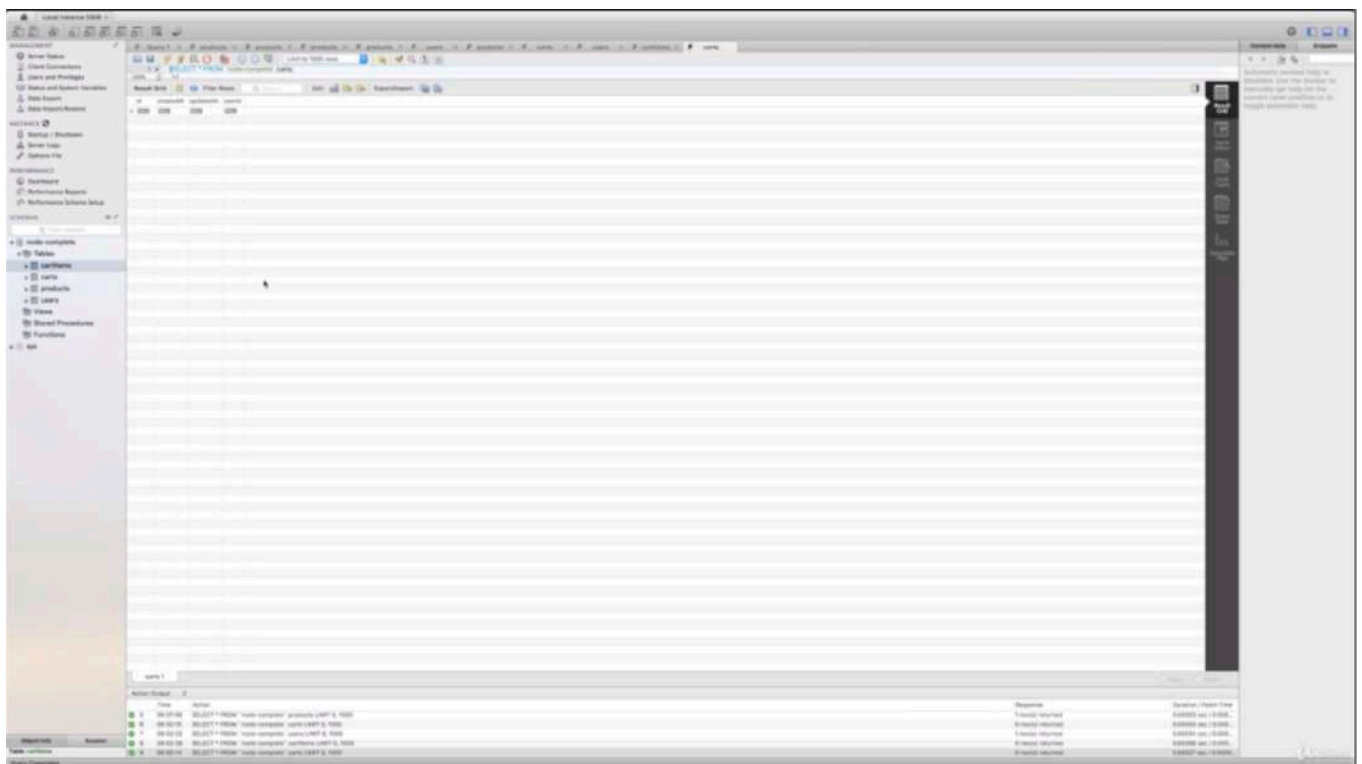
1. update
  - ./controllers/shop.js
  - app.js





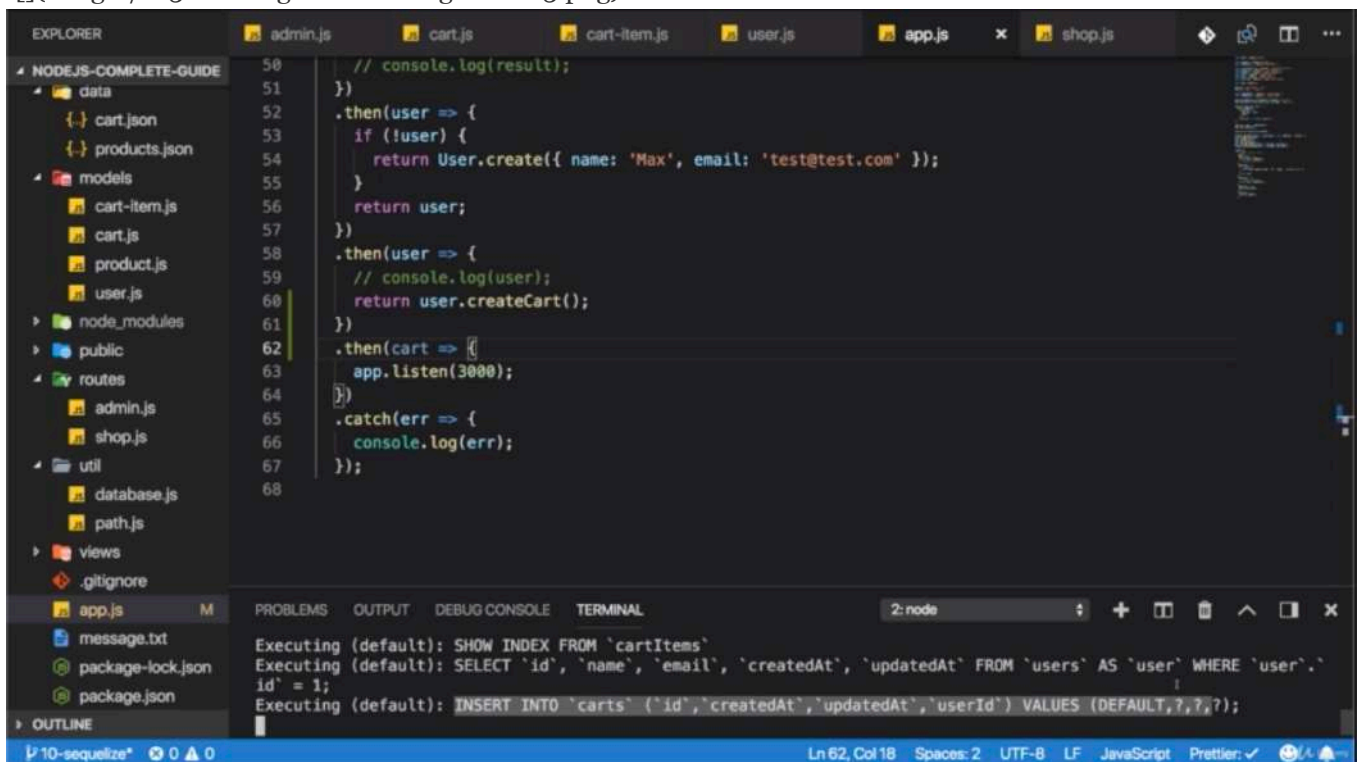
- the reason why we don't find anything is that we got no carts yet.





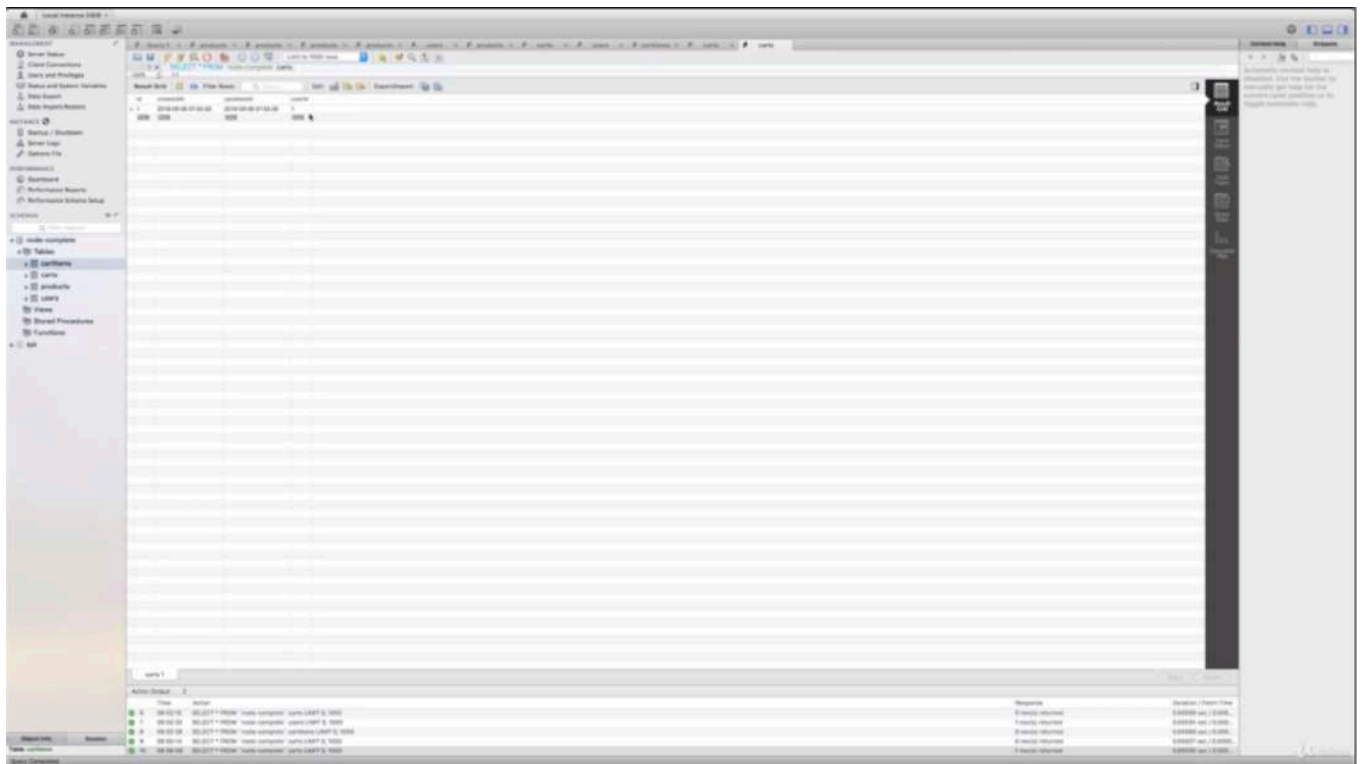
- if we look into our database, carts is totally empty.





- you see an 'INSERT INTO 'carts'' call is done here





- if we have a look into our carts, we see we get a cart associated to our user with the ID 1







This page isn't working

localhost didn't send any data.  
ERR\_EMPTY\_RESPONSE

Reload



```
51  });
52  };
53
54  exports.getCart = (req, res, next) => {
55    req.user
56      .getCart()
57      .then(cart => {
58        console.log(cart);
59      })
60      .catch(err => console.log(err));
61    // Cart.getCart(cart => {
```

```
TEGER, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `cartId` INTEGER, `productId` INTEGER, UNIQUE
E `cartItems_productId_cartId_unique` (`cartId`, `productId`), PRIMARY KEY (`id`), FOREIGN KEY (`cartId`) REFEREN
CES `cards` (`id`) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (`productId`) REFERENCES `products` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `cartItems`
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`
id` = 1;
Executing (default): INSERT INTO `cards` (`id`,`createdAt`,`updatedAt`,`userId`) VALUES (DEFAULT,?,7,?);
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`
id` = 1;
Executing (default): SELECT `id`, `createdAt`, `updatedAt`, `userId` FROM `cards` AS `card` WHERE `card`.`userId`
= 1 LIMIT 1;
card {
  dataValues:
    { id: 1,
      createdAt: 2018-09-06T07:55:58.000Z,
      updatedAt: 2018-09-06T07:55:58.000Z,
      userId: 1 },
```

- and if i try to reload that carts web page again, we get some output and this output is stemming from our 'getCart()' call here.







```
54  exports.getCart = (req, res, next) => {
55    console.log(req.user.cart);
56    req.user
57      .getCart()
58      .then(cart => {
59        console.log(cart);
60      })
61      .catch(err => console.log(err));
62    // Cart.getCart(cart => {
63    //   Product.fetchAll(products => {
64    //     const cartProducts = [];
65    //     for (product of products) {
66    //       const cartProductData = cart.products.find(
67    //         prod => prod.id === product.id
68    //       );
69    //       if (cartProductData) {
70    //         cartProducts.push({ productData: product, qty: cartProductData.qty });
71    //       }
72    //     }
73    //     res.render('shop/cart', {
```

```
_schema: null,
_schemaDelimiter: '',
raw: true,
attributes: [ 'id', 'createdAt', 'updatedAt', 'userId' ],
isNewRecord: false }
```

```

53 //
54 exports.getCart = (req, res, next) => {
55   console.log(req.user.cart);
56   req.user
57     .getCart()
58     .then(cart => {
59       console.log(cart);
60     })
61     .catch(err => console.log(err));
62   // Cart.getCart(cart => {
63     //   Product.fetchAll(products => {

```

```

id = 1;
Executing (default): INSERT INTO `cards` (`id`,`createdAt`,`updatedAt`,`userId`) VALUES (DEFAULT,?,?,?);
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`id` = 1;
undefined
Executing (default): SELECT `id`, `createdAt`, `updatedAt`, `userId` FROM `cards` AS `card` WHERE `card`.`userId` = 1 LIMIT 1;
card {
  dataValues:
    { id: 1,
      createdAt: 2018-09-06T07:55:58.000Z,
      updatedAt: 2018-09-06T07:55:58.000Z,
      userId: 1 },
  _previousDataValues:
    { id: 1,
      createdAt: 2018-09-06T07:55:58.000Z,
      updatedAt: 2018-09-06T07:55:58.000Z,
      userId: 1 },

```

```

54 exports.getCart = (req, res, next) => {
55   req.user
56     .getCart()
57     .then(cart => {
58       return cart
59         .getProducts()
60         .then(products => {
61           res.render('shop/cart', {
62             path: '/cart',
63             pageTitle: 'Your Cart',
64             products: products
65           });
66         })
67     })
68     .catch(err => console.log(err));
69   // Cart.getCart(cart => {
70   //   Product.fetchAll(products => {

```

```

Executing (default): SELECT `id`, `createdAt`, `updatedAt`, `userId` FROM `cards` AS `card` WHERE `card`.`userId` = 1 LIMIT 1;
Executing (default): SELECT `product`.`id`, `product`.`title`, `product`.`price`, `product`.`imageUrl`, `product`.`description`, `product`.`createdAt`, `product`.`updatedAt`, `product`.`userId`, `cardItem`.`id` AS `cardItem.id`, `cardItem`.`quantity` AS `cardItem.quantity`, `cardItem`.`createdAt` AS `cardItem.createdAt`, `cardItem`.`updatedAt` AS `cardItem.updatedAt`, `cardItem`.`cardId` AS `cardItem.cardId`, `cardItem`.`productId` AS `cardItem.productId` FROM `products` AS `product` INNER JOIN `cardItems` AS `cardItem` ON `product`.`id` = `cardItem`.`productId` AND `cardItem`.`cardId` = 1;

```

- if we console log req.user.cart, and if i reload here, this still is the old log, but if we scroll above it, we still see 'undefined'
- so we can't access 'cart' as a property but we can call 'getCart()' to work with the cart
- we can see that this is the statement executed by sequelize. and if you didn't see it before, here we can definitely see the advantage we have by using a package like this. we don't have to write that SQL statement on our own. we use sequelize and let it do that behind the scenes

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Cart = require('../models/cart');
5
6 exports.getProducts = (req, res, next) => {
7   Product.findAll()

```

```

8      .then(products => {
9          res.render('shop/product-list', {
10              prods: products,
11              pageTitle: 'All Products',
12              path: '/products'
13          });
14      })
15      .catch(err => {
16          console.log(err);
17      });
18  };
19
20  exports.getProduct = (req, res, next) => {
21      const prodId = req.params.productId;
22      // Product.findAll({ where: { id: prodId } })
23      // .then(products => {
24      //     res.render('shop/product-detail', {
25      //         product: products[0],
26      //         pageTitle: products[0].title,
27      //         path: '/products'
28      //     });
29      // })
30      // .catch(err => console.log(err));
31      Product.findById(prodId)
32          .then(product => {
33              res.render('shop/product-detail', {
34                  product: product,
35                  pageTitle: product.title,
36                  path: '/products'
37              });
38          })
39          .catch(err => console.log(err));
40  };
41
42  exports.getIndex = (req, res, next) => {
43      Product.findAll()
44          .then(products => {
45              res.render('shop/index', {
46                  prods: products,
47                  pageTitle: 'Shop',
48                  path: '/'
49              });
50          })
51          .catch(err => {
52              console.log(err);
53          });
54  };
55
56  /**i wanna use the Cart associated with my existing user to get all the products in it
57   * and renderthem to the screen
58   */
59  exports.getCart = (req, res, next) => {
60      console.log(req.user.cart)
61      req.user.getCart()
62          .then(cart => {
63              /**we can use it to fetch the products that are inside of it

```

```

64     * by returning 'cart.getProduct()'
65     *
66     * 'cart' is associated to products in our app.js file
67     * through belongsToMany
68     * and sequelize will do the magic of looking into 'CartItem' and so on.
69     */
70     return cart.getProducts()
71       .then(products => {
72         /**we should have the products that are in this cart
73          * and means that we can now render these products
74          */
75         res.render('shop/cart', {
76           path: '/cart',
77           pageTitle: 'Your Cart',
78           products: Products
79         })
80       })
81       .catch(err => console.log(err))
82     })
83     .catch(err => console.log(err))
84   };
85
86   exports.postCart = (req, res, next) => {
87     const prodId = req.body.productId;
88     Product.findById(prodId, product => {
89       Cart.addProduct(prodId, product.price);
90     });
91     res.redirect('/cart');
92   };
93
94   exports.postCartDeleteProduct = (req, res, next) => {
95     const prodId = req.body.productId;
96     Product.findById(prodId, product => {
97       Cart.deleteProduct(prodId, product.price);
98       res.redirect('/cart');
99     });
100   };
101
102   exports.getOrders = (req, res, next) => {
103     res.render('shop/orders', {
104       path: '/orders',
105       pageTitle: 'Your Orders'
106     });
107   };
108
109   exports.getCheckout = (req, res, next) => {
110     res.render('shop/checkout', {
111       path: '/checkout',
112       pageTitle: 'Checkout'
113     });
114   };
115

```

```

1 //app.js
2
3 const path = require('path');
4

```

```

5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const sequelize = require('./util/database');
10 const Product = require('./models/product')
11 const User = require('./models/user')
12 const Cart = require('./models/cart')
13 const CartItem = require('./models/cart-item')
14
15 const app = express();
16
17 app.set('view engine', 'ejs');
18 app.set('views', 'views');
19
20 const adminRoutes = require('./routes/admin');
21
22 const shopRoutes = require('./routes/shop');
23
24 app.use(bodyParser.urlencoded({ extended: false }));
25 app.use(express.static(path.join(__dirname, 'public')));
26
27 app.use((req, res, next) => {
28     User.findByPk(1)
29         .then(user => {
30             req.user = user
31             next()
32         })
33         .catch(err => console.log(err))
34 })
35
36 app.use('/admin', adminRoutes);
37 app.use(shopRoutes);
38
39 app.use(errorController.get404);
40
41 Product.belongsTo(User, {constraint: true, onDelete: 'CASCADE'})
42 User.hasMany(Product)
43 /**this is basically the inverse of this relation
44  * and it's optional
45  * you don't need to add it.
46  * one direction is enough.
47  */
48 User.hasOne(Cart)
49 Cart.belongsTo(User)
50 /**'cart' is associated to products in our app.js file
51  * through belongsToMany
52  * and sequelize will do the magic of looking into 'CartItem' and so on.
53  */
54 Cart.belongsToMany(Product, { through: CartItem })
55 Product.belongsToMany(Cart, { through: CartItem })
56
57 sequelize
58 /**let's disable that force syncing again
59  * so that we don't always override any data we stored
60  */

```

```

61 //sync({force: true})
62 .sync()
63 .then(result => {
64     return User.findByPk(1)
65     //console.log(result)
66 })
67 .then(user => {
68     if(!user){
69         return User.create({ name: 'Max', email: 'test@test.com' })
70     }
71     return user
72 })
73 .then(user => {
74     //console.log(user)
75     /**i don't need to pass any data in here
76     * because cart in the beginning will not hold any special data
77     * it just needs to be there.
78     */
79     return user.createCart()
80 })
81 .then(cart => {
82     app.listen(3000);
83 })
84 .catch(err => {
85     console.log(err)
86 })
87

```

## \* Chapter 164: Adding New Products To The Cart

1. update
  - ./controllers/shop.js
  - ./models/cart-item.js









Title

fdsafdsa

Image URL

fdsafdsa

Price

12

Description

fasdfdas

Add Product



fdsafdsa

fdsafdsa

\$ 12

fasdfdas

Edit

Delete







- if we refresh or load cartitems, we see a new product was added or a new element was added to the cart with quantity 1 pointing at that cart with ID 1 and the product with ID 1.

```

1  ../controllers/shop.js
2
3  const Product = require('../models/product');
4  const Cart = require('../models/cart');
5
6  exports.getProducts = (req, res, next) => {
7    Product.findAll()
8      .then(products => {
9        res.render('shop/product-list', {
10          prods: products,
11          pageTitle: 'All Products',
12          path: '/products'
13        });
14      })
15      .catch(err => {
16        console.log(err);
17      });
18 };
19
20 exports.getProduct = (req, res, next) => {
21   const prodId = req.params.productId;
22   // Product.findAll({ where: { id: prodId } })
23   // .then(products => {
24   //   res.render('shop/product-detail', {
25   //     product: products[0],
26   //     pageTitle: products[0].title,
27   //     path: '/products'
28   //   });
29   // })
30   // .catch(err => console.log(err));
31   Product.findByPk(prodId)
32     .then(product => {

```



```

33     res.render('shop/product-detail', {
34         product: product,
35         pageTitle: product.title,
36         path: '/products'
37     });
38 })
39 .catch(err => console.log(err));
40 };
41
42 exports.getIndex = (req, res, next) => {
43     Product.findAll()
44     .then(products => {
45         res.render('shop/index', {
46             prods: products,
47             pageTitle: 'Shop',
48             path: '/'
49         });
50     })
51     .catch(err => {
52         console.log(err);
53     });
54 };
55
56 exports.getCart = (req, res, next) => {
57     req.user
58     .getCart()
59     .then(cart => {
60         return cart
61             .getProducts()
62             .then(products => {
63                 res.render('shop/cart', {
64                     path: '/cart',
65                     pageTitle: 'Your Cart',
66                     products: products
67                 });
68             })
69             .catch(err => console.log(err));
70     })
71     .catch(err => console.log(err));
72 };
73
74 exports.postCart = (req, res, next) => {
75     const prodId = req.body.productId;
76     let fetchedCart
77     req.user
78     .getCart()
79     .then(cart => {
80         fetchedCart = cart
81         return cart.getProducts({ where: { id: prodId } })
82     })
83     .then(products => {
84         /**first of all i need to check if products.length > 0
85         * i will write this a bit differently
86         * create a product variable here
87         * and assign a value to that variable if we do have products,
88         * otherwise it will stay undefined.

```

```

89     */
90     let product
91     if(products.length > 0){
92         product = products[0]
93     }
94     let newQuantity = 1
95     if(product){
96         //...
97     }
98     return Product.findByPk(prodId)
99         .then(product => {
100         /**'addProduct()' is another magic method added by sequelize for many-to-many
relationships
101         * i can add a single product and i will add it to this in-between table with its
ID
102         *
103         * i just need to also make sure
104         * that i set this extra field i added to my ./models/cart-item.js file
105         * 'cart-item.js' is in-between table.
106         * but for every entry, i also wanna have quantity
107         * if i have more than just 2 matching IDs,
108         * i need to tell sequelize that there is an extra field that needs to be set
109         *
110         * and i do this by passing an object to 'addProduct()' as the second argument
111         * and there i will add 'through' which we use to tell sequelize which model to
use as the in-between model
112         * and therefore as the in-between table.
113         *
114         * now i'm telling sequelize that for that in-between table,
115         * here's some additional information you need to set the values in there.
116         * so that's another object and i'm setting the keys or the fields that should be
set in the in-between table.
117         *
118         */
119         return fetchedCart.addProduct(product, { through: { quantity: newQuantity } })
120     })
121     .catch(err => console.log(err))
122 })
123 .then(() => {
124     res.redirect('/cart')
125 })
126 .catch(err => console.log(err))
127
128 };
129
130 exports.postCartDeleteProduct = (req, res, next) => {
131     const prodId = req.body.productId;
132     Product.findByPk(prodId, product => {
133         Cart.deleteProduct(prodId, product.price);
134         res.redirect('/cart');
135     });
136 };
137
138 exports.getOrders = (req, res, next) => {
139     res.render('shop/orders', {
140         path: '/orders',

```

```

141     pageTitle: 'Your Orders'
142   });
143 };
144
145 exports.getCheckout = (req, res, next) => {
146   res.render('shop/checkout', {
147     path: '/checkout',
148     pageTitle: 'Checkout'
149   });
150 };
151
1
2  //../models/cart-item.js
3  /** this is in-between table */
4
5  const Sequelize = require('sequelize');
6
7  const sequelize = require('../util/database');
8
9  const CartItem = sequelize.define('cartItem', {
10    id: {
11      type: Sequelize.INTEGER,
12      autoIncrement: true,
13      allowNull: false,
14      primaryKey: true
15    },
16    quantity: Sequelize.INTEGER
17  });
18
19 module.exports = CartItem;
20

```

## \* Chapter 165: Adding Existing Products & Retrieving Cart Items

1. update  
 - ./views/shop/cart.ejs  
 - ./controllers/shop.js

fdsafdsa

Quantity: 1

Delete



fdsafdsa

fdsafdsa

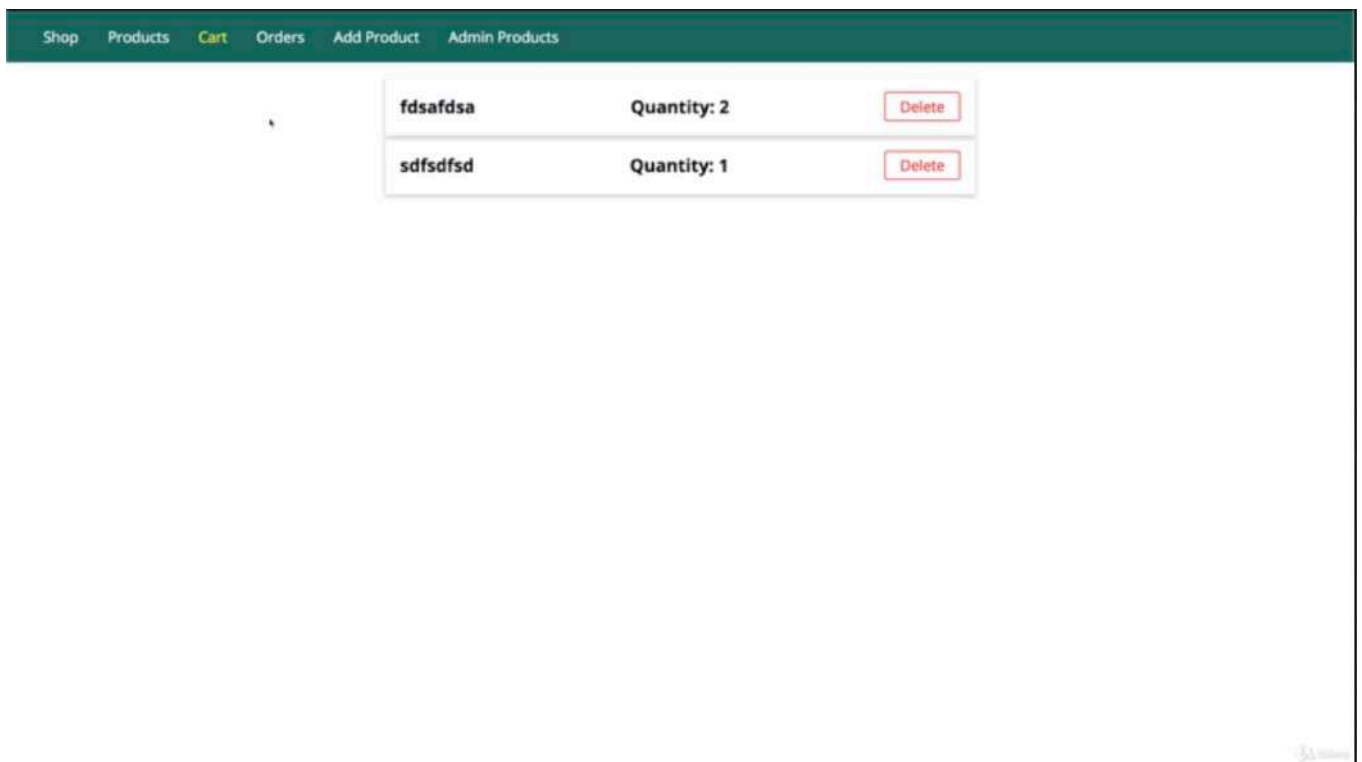
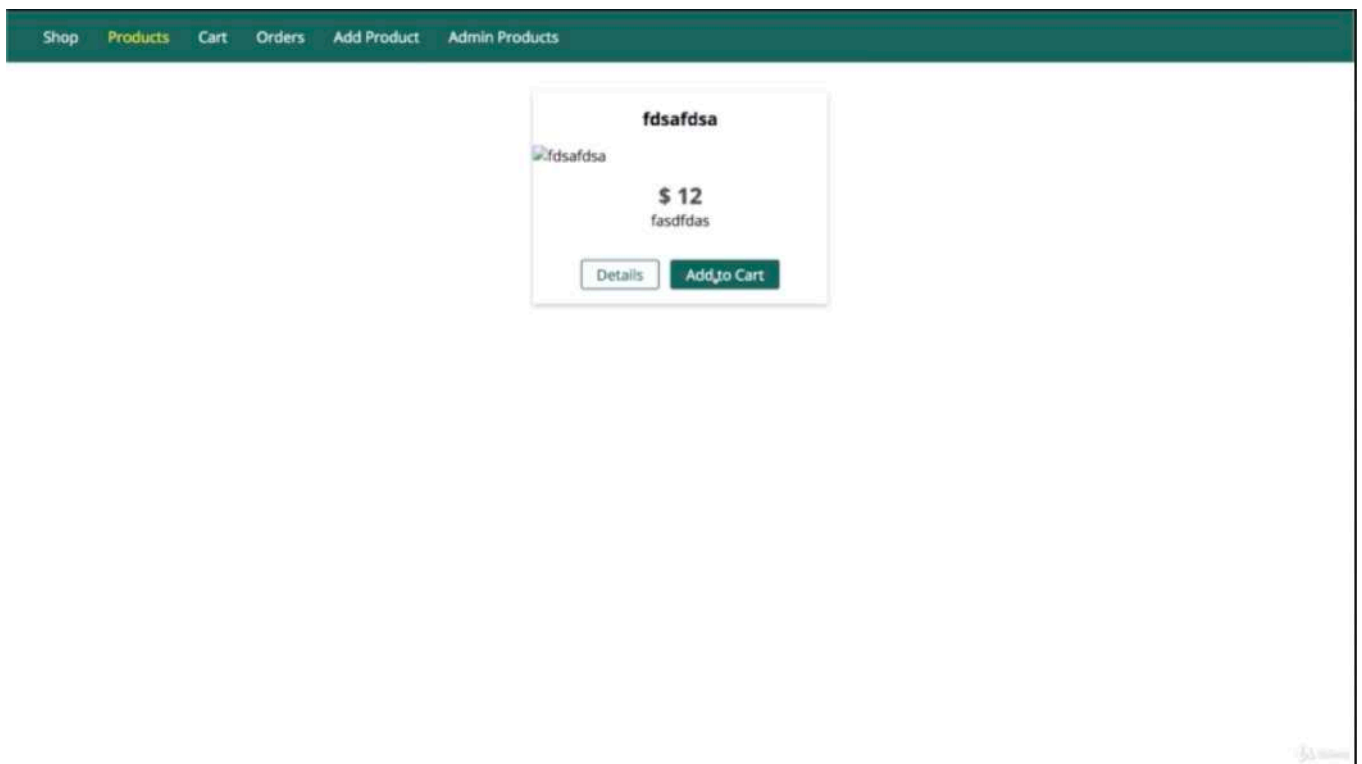
\$ 12

fasdfdas

Details

Add to Cart





```
1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Cart = require('../models/cart');
5
6 exports.getProducts = (req, res, next) => {
7   Product.findAll()
8     .then(products => {
9     res.render('shop/product-list', {
10       prods: products,
11       pageTitle: 'All Products',
12       path: '/products'
13     });
14   })
15 }
```

```

15     .catch(err => {
16         console.log(err);
17     });
18 };
19
20 exports.getProduct = (req, res, next) => {
21     const prodId = req.params.productId;
22     // Product.findAll({ where: { id: prodId } })
23     // .then(products => {
24     //     res.render('shop/product-detail', {
25     //         product: products[0],
26     //         pageTitle: products[0].title,
27     //         path: '/products'
28     //     });
29     // })
30     // .catch(err => console.log(err));
31     Product.findByPk(prodId)
32     .then(product => {
33         res.render('shop/product-detail', {
34             product: product,
35             pageTitle: product.title,
36             path: '/products'
37         });
38     })
39     .catch(err => console.log(err));
40 };
41
42 exports.getIndex = (req, res, next) => {
43     Product.findAll()
44     .then(products => {
45         res.render('shop/index', {
46             prods: products,
47             pageTitle: 'Shop',
48             path: '/'
49         });
50     })
51     .catch(err => {
52         console.log(err);
53     });
54 };
55
56 exports.getCart = (req, res, next) => {
57     req.user
58     .getCart()
59     .then(cart => {
60         return cart
61         .getProducts()
62         .then(products => {
63             res.render('shop/cart', {
64                 path: '/cart',
65                 pageTitle: 'Your Cart',
66                 products: products
67             });
68         })
69         .catch(err => console.log(err));
70     })

```

```

71     .catch(err => console.log(err));
72 };
73
74 exports.postCart = (req, res, next) => {
75     const prodId = req.body.productId;
76     let fetchedCart
77     /**to make sure that we correctly get that data,
78      * we can pull newQuantity out of this anonymous function,
79      * make it a top level variable in this overall function here.
80      *
81      * and therefore newQuantity is available in all 'then()' blocks
82      * and we either leave it at one here.
83      */
84     let newQuantity = 1
85     req.user
86         .getCart()
87         .then(cart => {
88             fetchedCart = cart
89             return cart.getProducts({ where: { id: prodId } })
90         })
91         .then(products => {
92             let product
93             if(products.length > 0){
94                 product = products[0]
95             }
96             if(product){
97                 /**'product.cartItem' is the extra field that gets added by sequelize
98                 * to give us access to this in-between table
99                 * and there to the quantity and sequelize doesn't just give us access to the in-
100 between table
101                 * but to this exact product in the in-between table,
102                 * so therefore we get the quantity for this product as it's stored in the cart
103                 */
104                 const oldQuantity = product.cartItem.quantity
105                 newQuantity = oldQuantity + 1;
106                 /**or if we got a product,
107                 * we also need to return that product here
108                 * because that will then be our product.
109                 * we receive in the 'then()' block.
110                 * it's automatically wrapped by a promise.
111                 */
112                 return product
113             }
114             return Product.findByPk(prodId)
115         })
116         /**now the difference is that
117         * 'data' here should hold both the product that needs to be added and our quantity
118         * because the quantity is calculated differently
119         *
120         * it either stays at one or here
121         * we set it to oldQuantity + 1
122         */
123         .then(product => {
124             return fetchedCart.addProduct(product, {
125                 through: { quantity: newQuantity }

```

```

126   })
127   .then(() => {
128     res.redirect('/cart')
129   })
130   .catch(err => console.log(err))
131 };
132
133 exports.postCartDeleteProduct = (req, res, next) => {
134   const prodId = req.body.productId;
135   Product.findByPk(prodId, product => {
136     Cart.deleteProduct(prodId, product.price);
137     res.redirect('/cart');
138   });
139 };
140
141 exports.getOrders = (req, res, next) => {
142   res.render('shop/orders', {
143     path: '/orders',
144     pageTitle: 'Your Orders'
145   });
146 };
147
148 exports.getCheckout = (req, res, next) => {
149   res.render('shop/checkout', {
150     path: '/checkout',
151     pageTitle: 'Checkout'
152   });
153 };
154

```

```

1 <!--./views/shop/cart.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/cart.css">
5   </head>
6
7   <body>
8     <%- include('../includes/navigation.ejs') %>
9     <main>
10       <% if (products.length > 0) { %>
11         <ul class="cart__item-list">
12           <% products.forEach(p => { %>
13             <li class="cart__item">
14               <h1><%= p.title %></h1>
15               <!--sequelize also gives us a 'cartItem' key for this
16                which stores information about this in-between table
17                and the entry that is related to this product there.
18               -->
19               <h2>Quantity: <%= p.cartItem.quantity %></h2>
20               <form action="/cart-delete-item" method="POST">
21                 <input type="hidden" value="<%= p.id %>" name="productId">
22                 <button class="btn danger" type="submit">Delete</button>
23               </form>
24             </li>
25           <% }) %>
26         </ul>
27       <% } else { %>

```



```

28     <h1>No Products in Cart!</h1>
29   <% } %>
30 </main>
31 <%- include('../includes/end.ejs') %>

```

## \* Chapter 166: Deleting Related Items & Deleting Cart Products

1. update  
- ./controllers/shop.js







```

1  //./controllers/shop.js
2
3  const Product = require('../models/product');
4  const Cart = require('../models/cart');
5
6  exports.getProducts = (req, res, next) => {
7    Product.findAll()
8      .then(products => {
9        res.render('shop/product-list', {
10          prods: products,
11          pageTitle: 'All Products',
12          path: '/products'
13        });
14      })
15      .catch(err => {
16        console.log(err);
17      });
18 };
19
20 exports.getProduct = (req, res, next) => {
21   const prodId = req.params.productId;
22   // Product.findAll({ where: { id: prodId } })
23   // .then(products => {
24   //   res.render('shop/product-detail', {
25   //     product: products[0],
26   //     pageTitle: products[0].title,
27   //     path: '/products'
28   //   });
29   // })
30   // .catch(err => console.log(err));
31   Product.findByPk(prodId)
32     .then(product => {
33       res.render('shop/product-detail', {
34         product: product,
35         pageTitle: product.title,
36         path: '/products'
37       });
38     })
39     .catch(err => console.log(err));
40 };

```

```

41
42 exports.getIndex = (req, res, next) => {
43   Product.findAll()
44     .then(products => {
45       res.render('shop/index', {
46         prods: products,
47         pageTitle: 'Shop',
48         path: '/'
49       });
50     })
51     .catch(err => {
52       console.log(err);
53     });
54 };
55
56 exports.getCart = (req, res, next) => {
57   req.user
58     .getCart()
59     .then(cart => {
60       return cart
61         .getProducts()
62         .then(products => {
63           res.render('shop/cart', {
64             path: '/cart',
65             pageTitle: 'Your Cart',
66             products: products
67           });
68         })
69         .catch(err => console.log(err));
70     })
71     .catch(err => console.log(err));
72 };
73
74 exports.postCart = (req, res, next) => {
75   const prodId = req.body.productId;
76   let fetchedCart
77   let newQuantity = 1
78   req.user
79     .getCart()
80     .then(cart => {
81       fetchedCart = cart
82       return cart.getProducts({ where: { id: prodId } })
83     })
84     .then(products => {
85       let product
86       if(products.length > 0){
87         product = products[0]
88       }
89       if(product){
90         const oldQuantity = product.cartItem.quantity
91         newQuantity = oldQuantity + 1;
92         return product
93       }
94       return Product.findByPk(prodId)
95     })
96     .then(product => {

```

```

97     return fetchedCart.addProduct(product, {
98       through: { quantity: newQuantity }
99     })
100   })
101   .then(() => {
102     res.redirect('/cart')
103   })
104   .catch(err => console.log(err))
105 };
106
107 exports.postCartDeleteProduct = (req, res, next) => {
108   const prodId = req.body.productId;
109   req.user
110     .getCart()
111     .then(cart => {
112       return cart.getProducts({ where: { id: prodId } })
113     })
114     .then(products => {
115       const product = products[0]
116       /**i wanna destroy that product
117        * but not in the products table
118        * but only in that in-between cartItem table that connects my cart with that product
119        *
120        * to do that, i can call 'product.cartItem'
121        * using that magic field sequelize gives me to access the element in the in-between
122        table
123        * and then 'destroy()'
124        * and that will remove it from that in-between table.
125        */
126       return product.cartItem.destroy()
127     })
128     .then(result => {
129       res.redirect('/cart');
130     })
131     .catch(err => console.log(err))
132   };
133
134 exports.getOrders = (req, res, next) => {
135   res.render('shop/orders', {
136     path: '/orders',
137     pageTitle: 'Your Orders'
138   });
139 };
140
141 exports.getCheckout = (req, res, next) => {
142   res.render('shop/checkout', {
143     path: '/checkout',
144     pageTitle: 'Checkout'
145   });
146 };

```

## \* Chapter 167: Adding A Order Model

1. update  
- ./models/order.js

- ./models/order-item.js
- app.js







```

38 app.use(errorController.get404);
39
40 Product.belongsTo(User, { constraints: true, onDelete: 'CASCADE' });
41 User.hasMany(Product);
42 User.hasOne(Cart);
43 Cart.belongsTo(User);
44 Cart.belongsToMany(Product, { through: CartItem });
45 Product.belongsToMany(Cart, { through: CartItem });
46 Order.belongsTo(User);
47 User.hasMany(Order);
48 Order.belongsToMany(Product, { through: OrderItem });
49
50 sequelize
51   .sync({ force: true })
52   .then(result => {
53     return User.findById(1);
54     // console.log(result);
55   })
56   .then(user => {
57     if (!user) {
58       return User.create({ name: 'Max', email: 'test@test.com' });
59     }
60   });

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

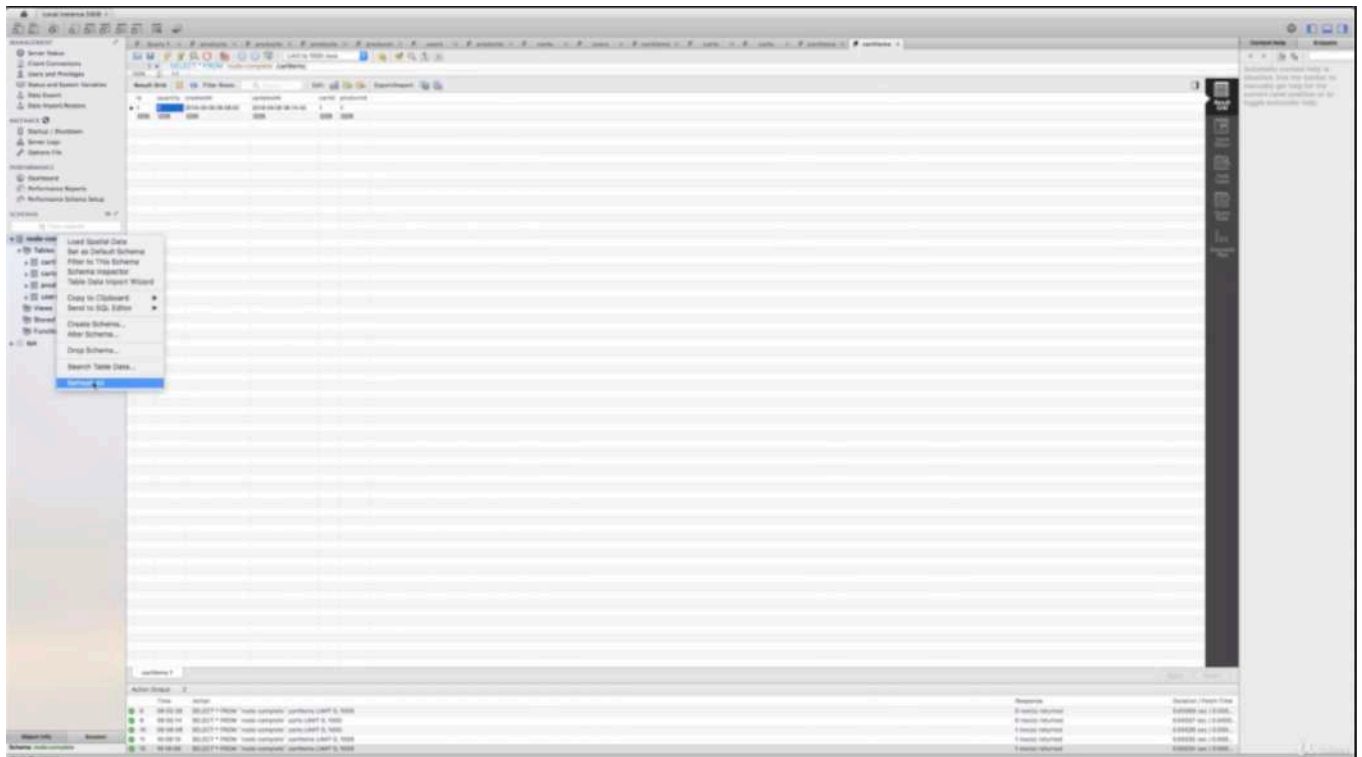
2: node

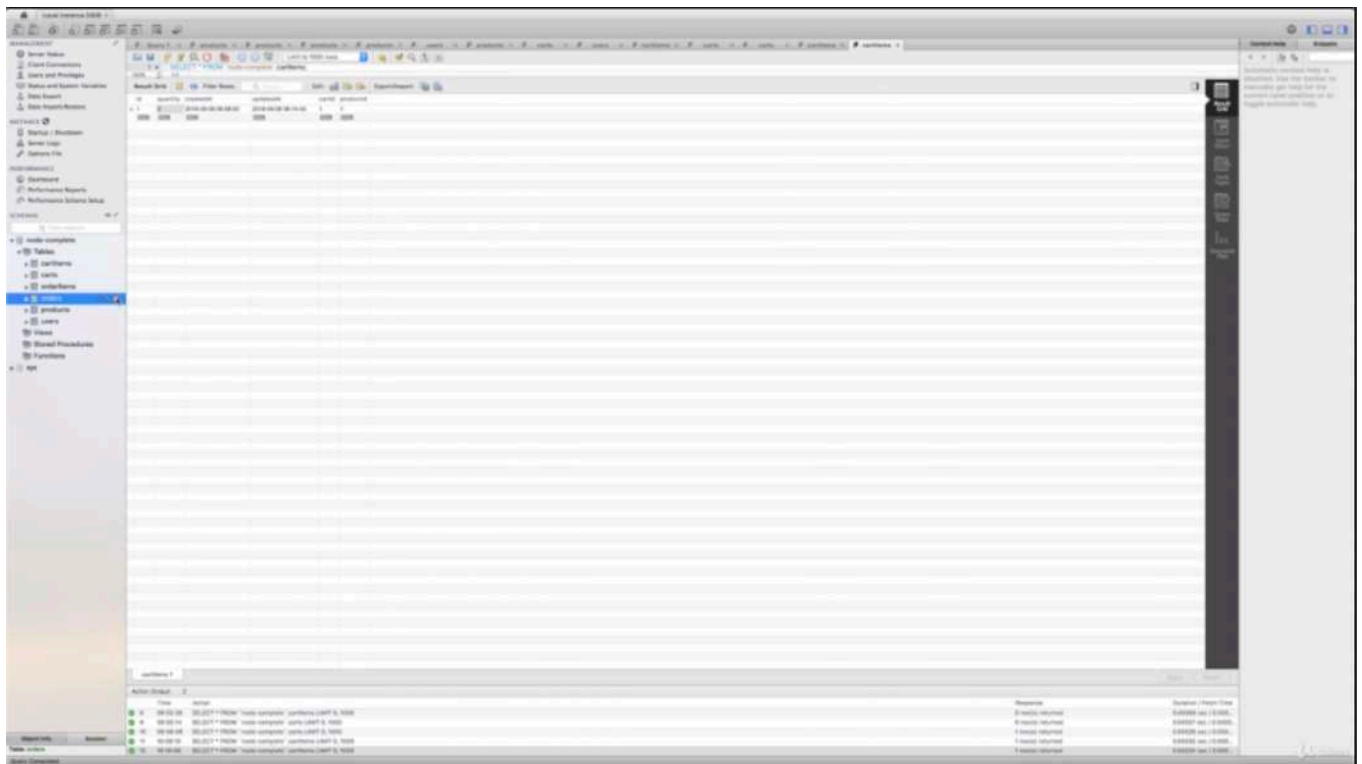
Executing (default): SELECT `id`, `name`, `email`, `createdAt`, `updatedAt` FROM `users` AS `user` WHERE `user`.`id` = 1;

Executing (default): INSERT INTO `carts` (`id`,`createdAt`,`updatedAt`,`userId`) VALUES (DEFAULT,?, ?, ?);

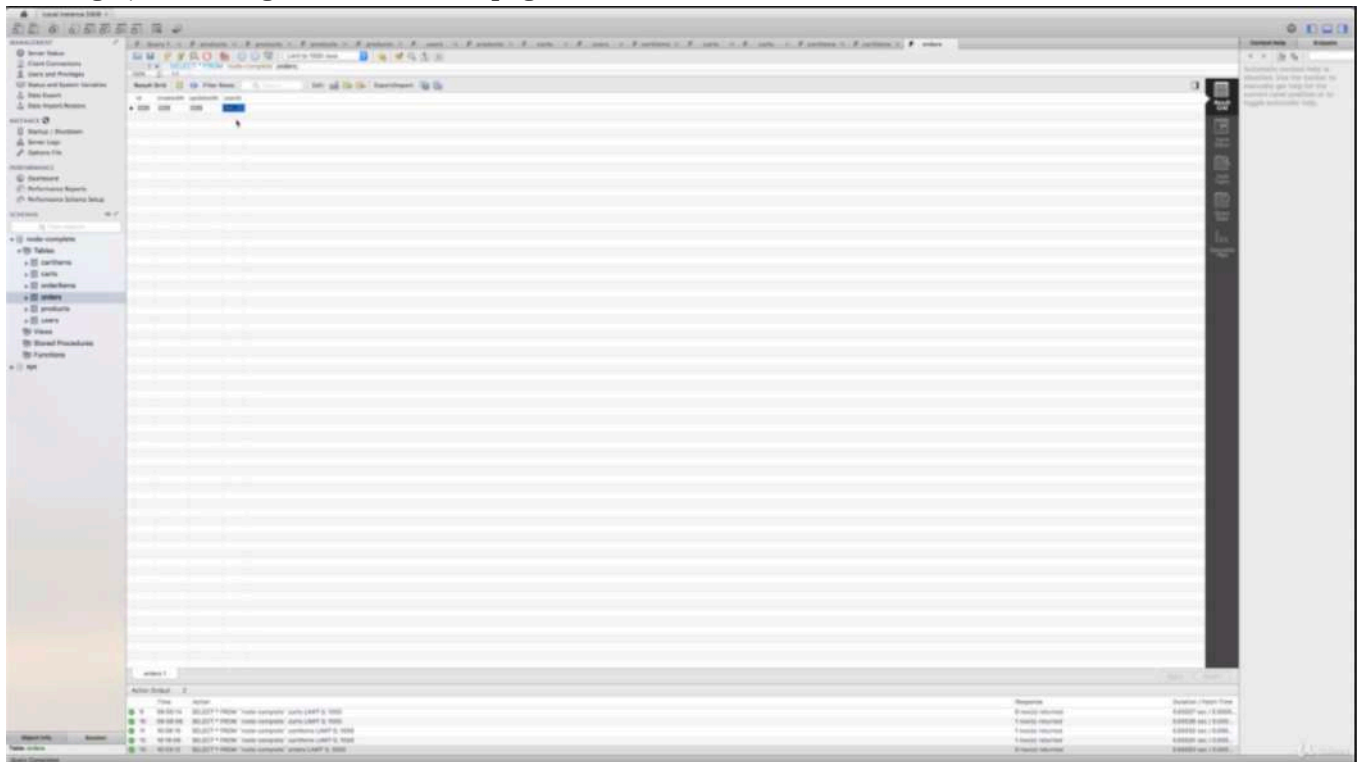
[nodemon] restarting due to changes...

[nodemon] starting 'node app.js'





- let me make sure we can sync again. so let's turn on forcing this.
  - if we now refresh our database, we should have new orders and orderItems
- 



- in orders, we see a connection to a user
- 

The screenshot shows a VS Code editor with a project structure on the left. The main editor displays the `app.js` file. Lines 38-48 define Sequelize models and their associations: `Product` belongs to `User` (with `constraints: true` and `onDelete: 'CASCADE'`); `User` has many `Product`s; `User` has one `Cart`; `Cart` belongs to `User`; `Cart` belongs to many `Product`s (via `CartItem`); `Product` belongs to many `Cart`s (via `CartItem`); `Order` belongs to `User`; `User` has many `Order`s; and `Order` belongs to many `Product`s (via `OrderItem`). Lines 50-59 initialize Sequelize, sync the database (with `force: true`), find a user by ID (1), and create a new user 'Max' with email 'test@test.com'. The bottom terminal shows the execution of these commands, including `SHOW INDEX FROM 'orderItems'` and `INSERT INTO 'carts'`.

- and in orderitems, we see a connection to an order and to our product id.



This screenshot is identical to the previous one, showing the same `app.js` file and terminal output. A red error marker is visible on line 51, indicating a problem with the `.sync({ force: true })` call.

- i will disable forcing this so that we don't overwrite tables all the time now

```
1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const sequelize = require('./util/database');
10 const Product = require('./models/product')
```

```

11 const User = require('./models/user')
12 const Cart = require('./models/cart')
13 const CartItem = require('./models/cart-item')
14 const Order = require('./models/order')
15 const OrderItem = require('./models/order-item')
16
17 const app = express();
18
19 app.set('view engine', 'ejs');
20 app.set('views', 'views');
21
22 const adminRoutes = require('./routes/admin');
23
24 const shopRoutes = require('./routes/shop');
25
26 app.use(bodyParser.urlencoded({ extended: false }));
27 app.use(express.static(path.join(__dirname, 'public')));
28
29 app.use((req, res, next) => {
30     User.findByPk(1)
31         .then(user => {
32             req.user = user
33             next()
34         })
35         .catch(err => console.log(err))
36 })
37
38 app.use('/admin', adminRoutes);
39 app.use(shopRoutes);
40
41 app.use(errorController.get404);
42
43 Product.belongsTo(User, {constraint: true, onDelete: 'CASCADE'})
44 User.hasMany(Product)
45 User.hasOne(Cart)
46 Cart.belongsTo(User)
47 Cart.belongsToMany(Product, { through: CartItem })
48 Product.belongsToMany(Cart, { through: CartItem })
49 /**this is not a 'many-to-many' relationship
50  * it's 'one-to-many' relationship.
51  * One user can have many orders.
52  */
53 Order.belongsTo(User)
54 User.hasMany(Order)
55 /**An order however can belong to many product.
56  * and it does so with an in-between table which we specify with 'through'
57  * which is 'OrderItem'
58  */
59 Order.belongsToMany(Product, { through: OrderItem })
60
61
62 sequelize
63     .sync({force: true})
64     .sync()
65     .then(result => {
66         return User.findByPk(1)

```



```

67     //console.log(result)
68   })
69   .then(user => {
70     if(!user){
71       return User.create({ name: 'Max', email: 'test@test.com' })
72     }
73     return user
74   })
75   .then(user => {
76     //console.log(user)
77     return user.createCart()
78   })
79   .then(cart => {
80     app.listen(3000);
81   })
82   .catch(err => {
83     console.log(err)
84   })

```

```

1  //./models/order-item.js
2
3  const Sequelize = require('sequelize');
4
5  const sequelize = require('../util/database');
6
7  const OrderItem = sequelize.define('orderItem', {
8    id: {
9      type: Sequelize.INTEGER,
10     autoIncrement: true,
11     allowNull: false,
12     primaryKey: true
13   }
14 });
15
16 module.exports = OrderItem;

```

```

1  //./models/order.js
2
3  const Sequelize = require('sequelize');
4
5  const sequelize = require('../util/database');
6
7  /**'order is, in the end, just an in-between table between a user to which the order belongs
8   * and then multiple products that are part of the order
9   * and these products again do have a quantity attached to them
10  */
11  const Order = sequelize.define('order', {
12    id: {
13      type: Sequelize.INTEGER,
14      autoIncrement: true,
15      allowNull: false,
16      primaryKey: true
17    },
18    quantity: Sequelize.INTEGER
19  });
20
21 module.exports = Order;

```



# \* Chapter 168: Storing Caritems As Orderitems

1. update

- ./views/shop/cart.ejs
- ./controllers/shop.js
- ./routes/shop.js
- ./models/order-item.js





















[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

fdasfas

Image URL

fadsfsa

Price

12

Description

fdsafas

Add Product



fdasfas

\$ 12

fdsafas

Edit

Delete

1





fdasfas

\$ 12

fdsafas

Details

Add to Cart



ShopProductsCartOrdersAdd ProductAdmin Products

fdasfas

Quantity: 1

Delete

Order Now!

ShopProductsCartOrdersAdd ProductAdmin Products

Title

Second

Image URL

fdasf

Price

22

Description

fdsfds

Add Product



fdasfas

\$ 12

fdsafas

Details

Add to Cart



Second

\$ 22

fsfsfs

Details

Add to Cart

Source: 2025/05/06/00:00:00





fdasfas

\$ 12

fdsafas

Details

Add to Cart



Second

\$ 22

fsfsfs

Details

Add to Cart



[Shop](#)[Products](#)[Cart](#)[Orders](#)[Add Product](#)[Admin Products](#)

fdasfas


Quantity: 2

Delete

Order Now!

[Shop](#)[Products](#)[Cart](#)[Orders](#)[Add Product](#)[Admin Products](#)

fdasfas



fdasfas


\$ 12

fdsafas

Details

Add to Cart

Second



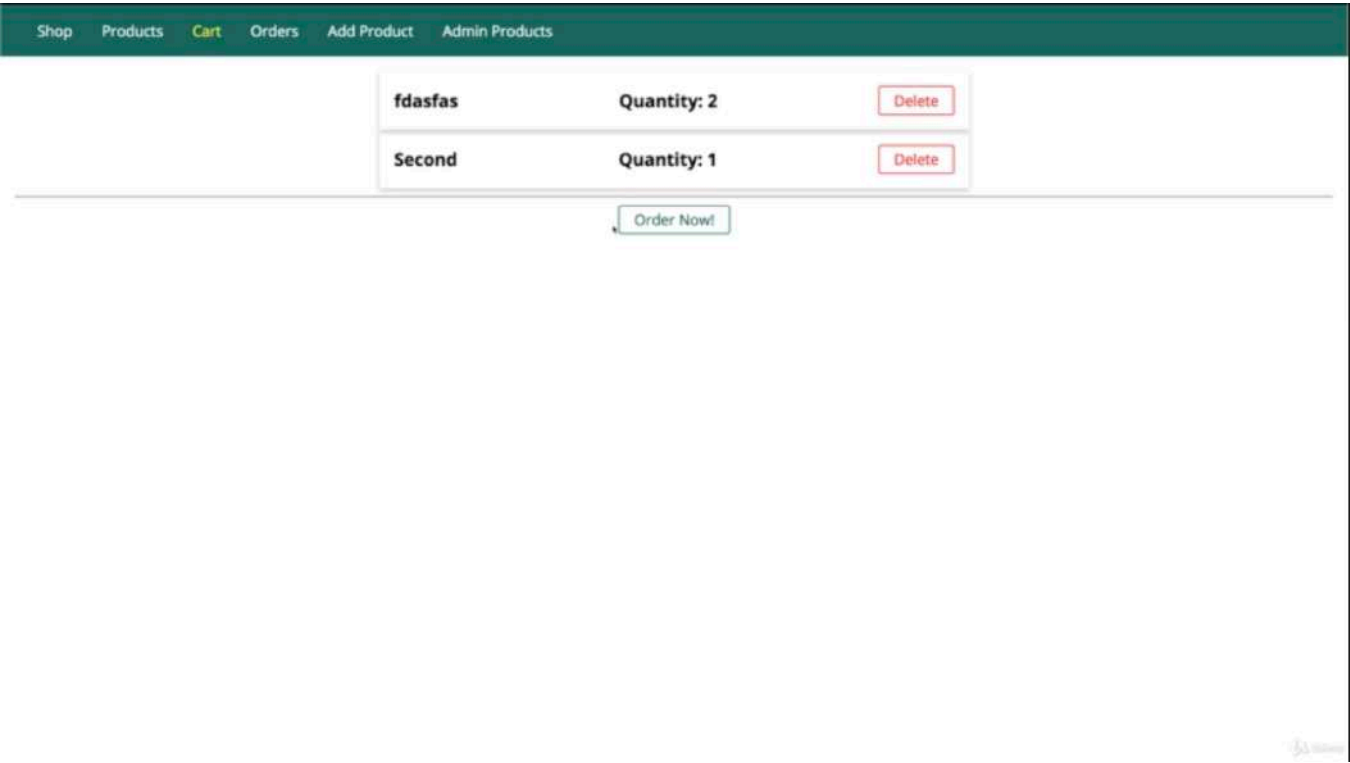
Second

\$ 22

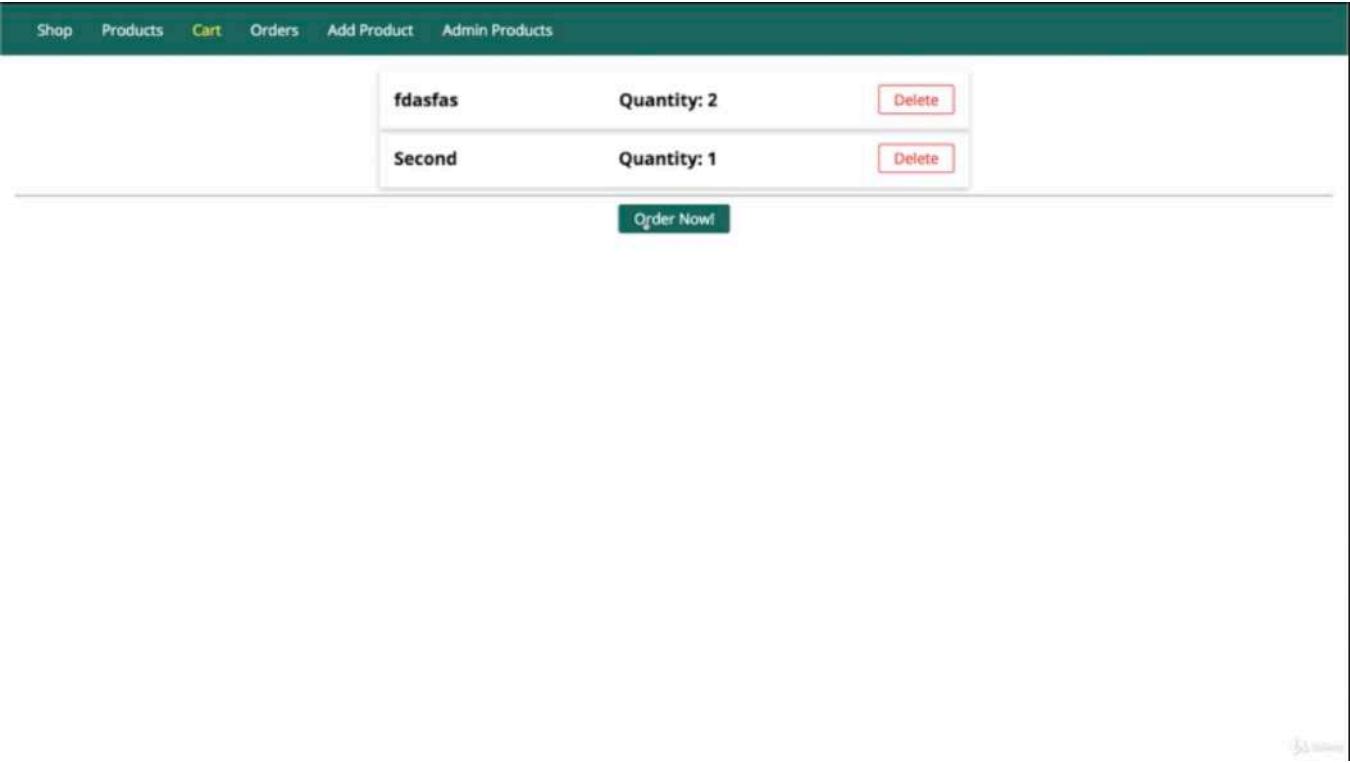
fsfsfs

Details

Add to Cart



- now order now should create a new order with these 2 items and clear the cart  
  

```
111     return cart.getProducts({ where: { id: prodId } });
112   })
113   .then(products => {
114     const product = products[0];
115     return product.cartItem.destroy();
116   })
117   .then(result => {
118     res.redirect('/cart');
119   })
120   .catch(err => console.log(err));
121   });
122
123   exports.postOrder = (req, res, next) => {
124     Id` AND `cartItem`.`cartId` = 1;
125     [ product {
126       dataValues:
127         { id: 1,
128           title: 'fdasfas',
129           price: 12,
130           imageUrl: 'fadsfsa',
131           description: 'fdsafas',
132           createdAt: 2018-09-06T08:24:56.000Z,
133           updatedAt: 2018-09-06T08:24:56.000Z,
134           userId: 1,
135           cartItem: [cartItem] },
136       _previousDataValues:
137         { id: 1,
138           title: 'fdasfas',
139           price: 12,
```

- if we now click this button, it won't do anything here.
- but in the console, we can see the products that were retrieved and we see that the products that were retrieved are the products here which also have that cartItem attribute which in turn gives us information about the cart item in this in-between table.



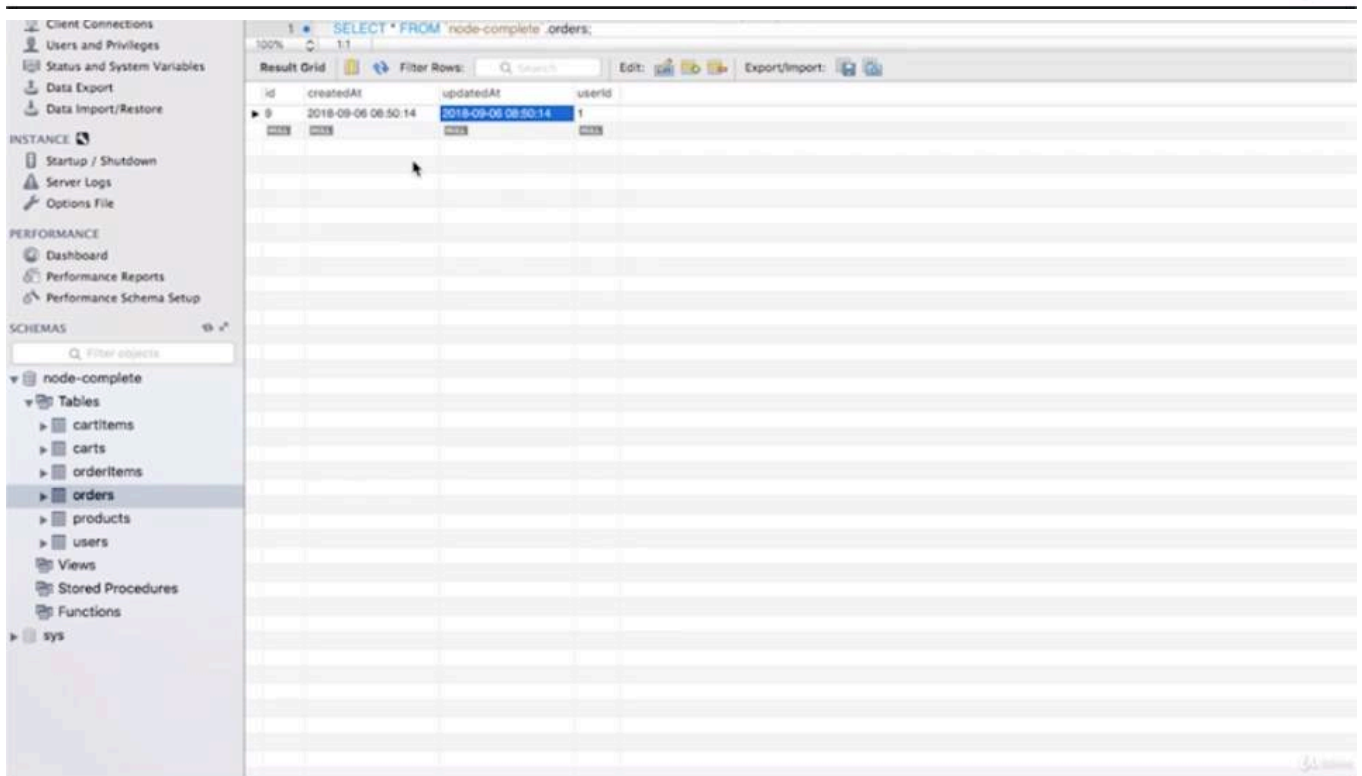




fdasfas	Quantity: 2	Delete
Second	Quantity: 1	Delete

Order Now!

Nothing there!



- if we click 'order now', i'm on the orders page where we never display anything. but we should be able to see some data if we load the orders table in workbench. there's one order





The screenshot shows a database management interface. On the left, there's a sidebar with categories like 'Client Connections', 'Users and Privileges', 'Status and System Variables', 'Data Export', 'Data Import/Restore', 'INSTANCE', 'PERFORMANCE', and 'SCHEMAS'. Under 'SCHEMAS', 'node-complete' is expanded, showing tables like 'cartItems', 'carts', 'orderItems', 'orders', 'products', 'users', 'Views', 'Stored Procedures', and 'Functions'. The main window displays a query result for 'SELECT \* FROM node-complete.orderItems;'. The result grid shows two rows of data.

id	quantity	createdAt	updatedAt	orderid	productid
9	2	2018-09-06 08:50:14	2018-09-06 08:50:14	9	1
10	1	2018-09-06 08:50:14	2018-09-06 08:50:14	9	2

- and in orderItems also has the respective elements that belongs to the order with the right quantitles

```

1  //./models/order-item.js
2
3  const Sequelize = require('sequelize');
4
5  const sequelize = require('../util/database');
6
7  /**you define this name 'orderItem'
8   * that is the name you have to use to ./controllers/shop.js
9   */
10 const OrderItem = sequelize.define('orderItem', {
11   id: {
12     type: Sequelize.INTEGER,
13     autoIncrement: true,
14     allowNull: false,
15     primaryKey: true
16   }
17 });
18
19 module.exports = OrderItem;

```

```

1  //./controllers/shop.js
2
3  const Product = require('../models/product');
4  /**we don't need the cart import
5   * because we never directly use the cart model.
6   * we always do so through the user model
7   * but we will need the order model
8   *
9   * or do we?
10  * just as a cart is related to user,
11  * so is an order
12  *
13  * so we don't even need that import,
14  * we can clear both 'Cart' and 'Order'

```

```

15  * because we will create a new order as an order
16  * that is associated to a user
17  */
18  //const Cart = require('../models/cart');
19  //const Order = require('../models/order')
20
21  exports.getProducts = (req, res, next) => {
22    Product.findAll()
23      .then(products => {
24        res.render('shop/product-list', {
25          prods: products,
26          pageTitle: 'All Products',
27          path: '/products'
28        });
29      })
30      .catch(err => {
31        console.log(err);
32      });
33  };
34
35  exports.getProduct = (req, res, next) => {
36    const prodId = req.params.productId;
37    // Product.findAll({ where: { id: prodId } })
38    // .then(products => {
39    //   res.render('shop/product-detail', {
40    //     product: products[0],
41    //     pageTitle: products[0].title,
42    //     path: '/products'
43    //   });
44    // })
45    // .catch(err => console.log(err));
46    Product.findByPk(prodId)
47      .then(product => {
48        res.render('shop/product-detail', {
49          product: product,
50          pageTitle: product.title,
51          path: '/products'
52        });
53      })
54      .catch(err => console.log(err));
55  };
56
57  exports.getIndex = (req, res, next) => {
58    Product.findAll()
59      .then(products => {
60        res.render('shop/index', {
61          prods: products,
62          pageTitle: 'Shop',
63          path: '/'
64        });
65      })
66      .catch(err => {
67        console.log(err);
68      });
69  };
70

```

```

71 exports.getCart = (req, res, next) => {
72   req.user
73     .getCart()
74     .then(cart => {
75       return cart
76         .getProducts()
77         .then(products => {
78           res.render('shop/cart', {
79             path: '/cart',
80             pageTitle: 'Your Cart',
81             products: products
82           });
83         })
84         .catch(err => console.log(err));
85     })
86     .catch(err => console.log(err));
87 };
88
89 exports.postCart = (req, res, next) => {
90   const prodId = req.body.productId;
91   let fetchedCart
92   let newQuantity = 1
93   req.user
94     .getCart()
95     .then(cart => {
96       fetchedCart = cart
97       return cart.getProducts({ where: { id: prodId } })
98     })
99     .then(products => {
100       let product
101       if(products.length > 0){
102         product = products[0]
103       }
104       if(product){
105         const oldQuantity = product.cartItem.quantity
106         newQuantity = oldQuantity + 1;
107         return product
108       }
109       return Product.findByPk(prodId)
110     })
111     .then(product => {
112       return fetchedCart.addProduct(product, {
113         through: { quantity: newQuantity }
114       })
115     })
116     .then(() => {
117       res.redirect('/cart')
118     })
119     .catch(err => console.log(err))
120 };
121
122 exports.postCartDeleteProduct = (req, res, next) => {
123   const prodId = req.body.productId;
124   req.user
125     .getCart()
126     .then(cart => {

```

```

127     return cart.getProducts({ where: { id: prodId } })
128   })
129   .then(products => {
130     const product = products[0]
131     return product.cartItem.destroy()
132   })
133   .then(result => {
134     res.redirect('/cart');
135   })
136   .catch(err => console.log(err))
137 };
138
139 /**i will also create the new route for this to handle a POST request to this order route
140  * so a POST request to '/create-order'
141  *
142  */
143 exports.postOrder = (req, res, next) => {
144   req.user.getCart()
145     .then(cart => {
146       return cart.getProducts()
147     })
148     .then(products => {
149 /**we don't need the cart import
150  * because we never directly use the cart model.
151  * we always do so through the user model
152  * but we will need the order model
153  *
154  * or do we?
155  * just as a cart is related to user,
156  * so is an order
157  *
158  * so we don't even need that import,
159  * we can clear both 'Cart' and 'Order'
160  * because we will create a new order as an order
161  * that is associated to a user
162  *
163  * so in postOrder here,
164  * we can now call 'req.user'
165  * and just as we create a Cart for that user in app.js
166  * with 'createCart()', we can call 'createOrder()' for that user
167  *
168  * now this gives us an order
169  * but we don't need the order
170  * we also need to associate our products to that order
171  * so i will return 'req.user.createOrder()'
172  */
173       return req.user
174         .createOrder()
175 /**i will get my created order
176  * and i wanna associate my products to that order
177  * and that can be done by calling 'order.addProducts(products)'
178  * and we need to specify 'through'
179  * and set the quantity correctly too
180  *
181  * each product needs to have a special key field which is then understood by sequelize
182  * to assign that special field,

```

```

183 * the products we pass in have to be modified
184 * and we can do that with 'map()' method which is a default javascript method that runs on
    array
185 * and returns a new array with slightly modified elements.
186 * we add a function that is executed for every element in the array
187 * and takes the element as an input
188 * and returns the modified version
189 * i will return products in the end
190 * but before i do so,
191 * i edit it slightly, a new property
192 * which sequelize will look for named 'orderItem'
193 * Now the name here 'orderItem' is important to get this right
194 * go to ./models/order-item.js file.
195 * you define this name 'orderItem' that is the name you have to use
196 * if you choose a different name, you have to use different name.
197 */
198
199 /**this stores a javascript object where i configure the value for this in-between table
200  * so here i define 'quantity' which is the value i need to store in-between table
201  * and i set this equal to 'product.cartItem'
202  * which is the related table i have due to my cart, quantity.
203  */
204     .then(order => {
205         return order.addProducts(
206             products.map(product => {
207                 product.orderItem = { quantity: product.cartItem.quantity }
208             })
209         )
210     })
211     .catch(err => console.log(err))
212 })
213     .then(result => {
214         res.redirect('/orders')
215     })
216     .catch(err => console.log(err))
217 }
218
219 exports.getOrders = (req, res, next) => {
220     res.render('shop/orders', {
221         path: '/orders',
222         pageTitle: 'Your Orders'
223     });
224 };
225
226 exports.getCheckout = (req, res, next) => {
227     res.render('shop/checkout', {
228         path: '/checkout',
229         pageTitle: 'Checkout'
230     });
231 };
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');

```

```

6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 router.get('/products/:productId', shopController.getProduct);
16
17 router.get('/cart', shopController.getCart);
18
19 router.post('/cart', shopController.postCart);
20
21 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
22
23 router.post('/create-order', shopController.postOrder);
24
25 router.get('/orders', shopController.getOrders);
26
27 router.get('/checkout', shopController.getCheckout);
28
29 module.exports = router;
30

```

```

1 <!--./views/shop/cart.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/cart.css">
5   </head>
6
7   <body>
8     <%- include('../includes/navigation.ejs') %>
9     <main>
10       <% if (products.length > 0) { %>
11         <ul class="cart__item-list">
12           <% products.forEach(p => { %>
13             <li class="cart__item">
14               <h1><%= p.title %></h1>
15               <h2>Quantity: <%= p.cartItem.quantity %></h2>
16               <form action="/cart-delete-item" method="POST">
17                 <input type="hidden" value="<%= p.id %>" name="productId">
18                 <button class="btn danger" type="submit">Delete</button>
19               </form>
20             </li>
21           <% }) %>
22         </ul>
23         <hr>
24         <div class="centered">
25           <form action="/create-order" method="POST">
26             <button type="submit" class="btn">Order Now!</button>
27           </form>
28         </div>
29       <% } else { %>
30         <h1>No Products in Cart!</h1>
31       <% } %>

```

32  
33

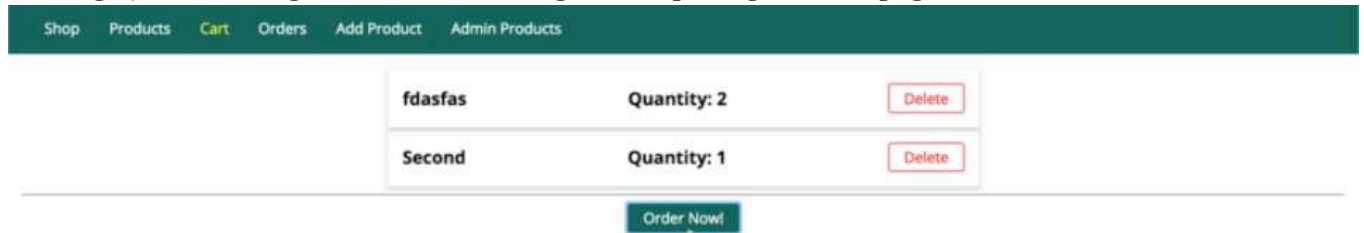
```
</main>  
<%- include('../includes/end.ejs') %>
```

## \* Chapter 169: Resetting The Cart & Fetching And Outputting Orders

- 1. update
  - ./controllers/shop.js
  - ./views/orders.ejs
  - ./routes/shop.js







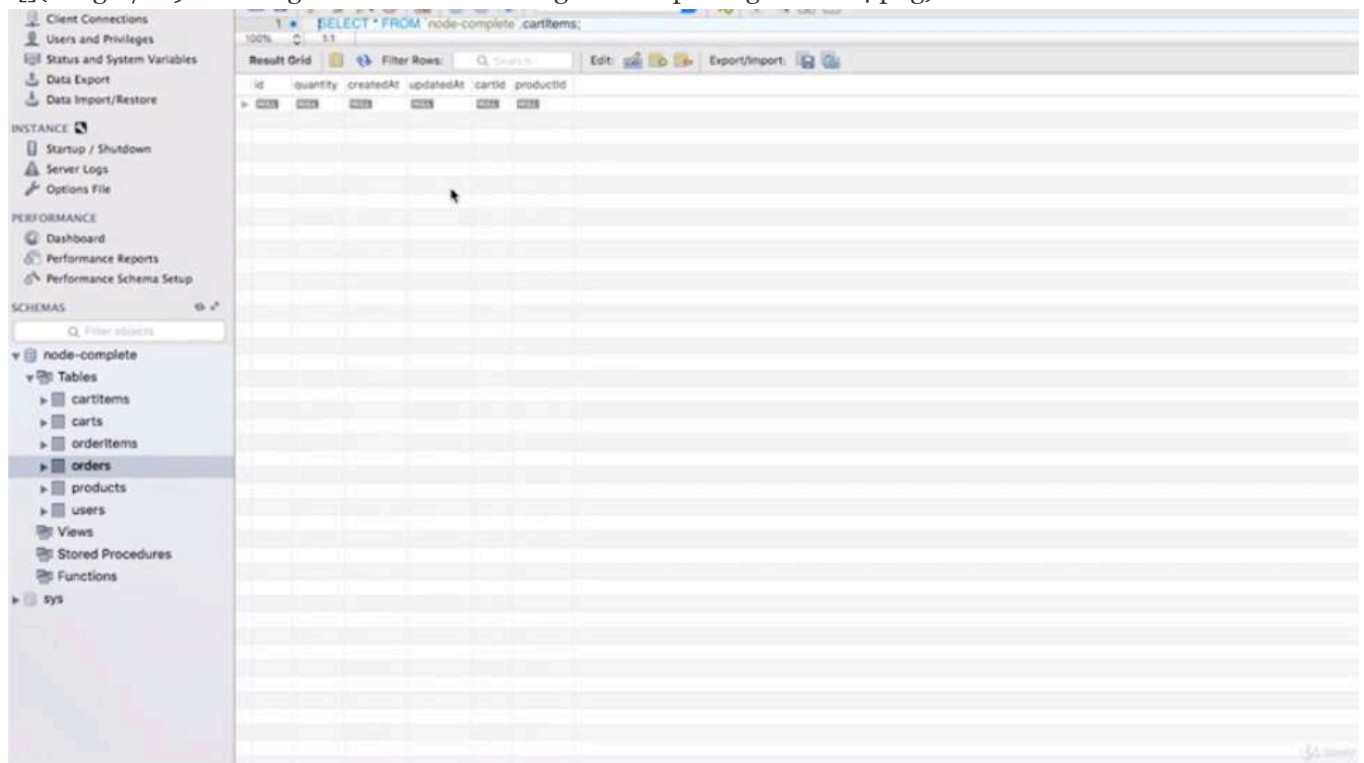
- reload the cart page, click 'order now', we are on the orders page.





- go back to the cart and we got no products in the cart





because if we go into the workbench, we see that the cartitems are empty because we dropped all the items in the cart by setting them to null





• # 9

- fdfasfas (2)
- Second (1)

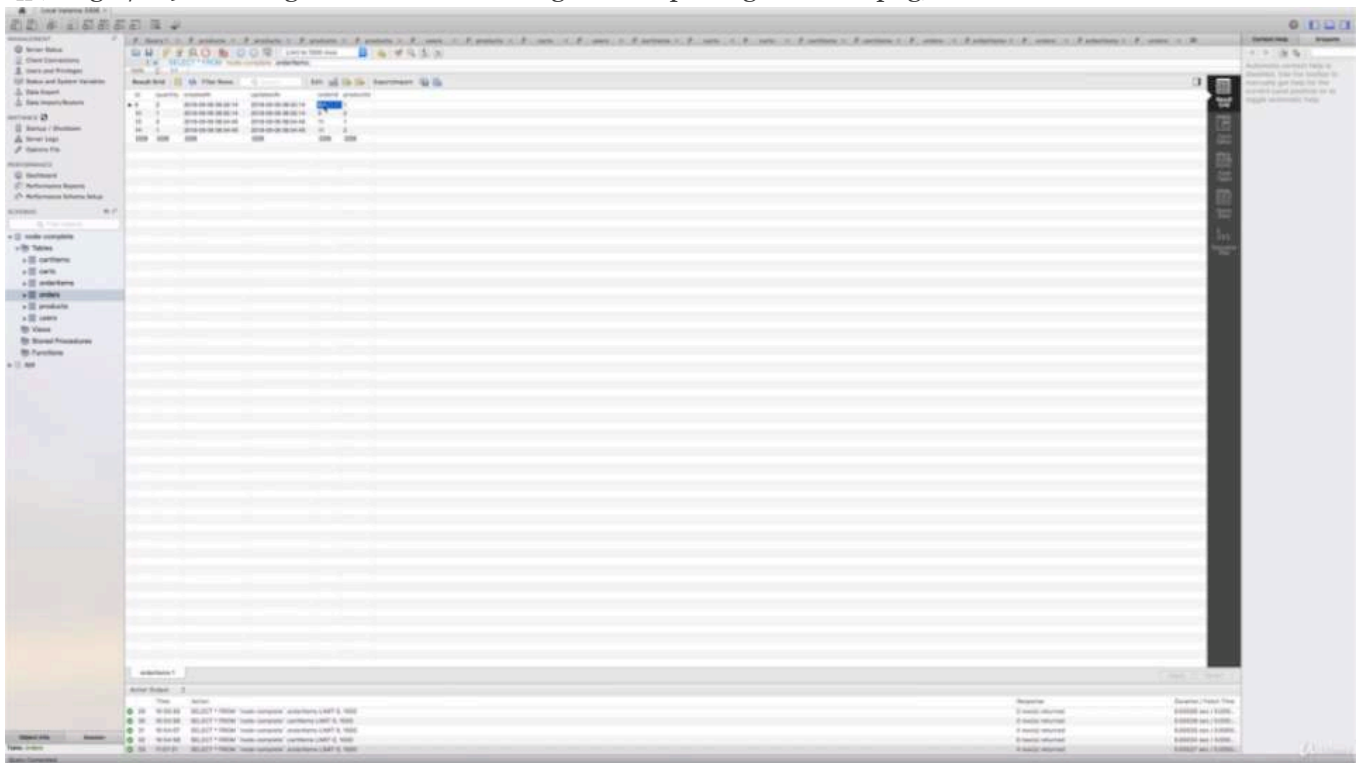
• # 10

• # 11

- fdfasfas (2)
- Second (1)

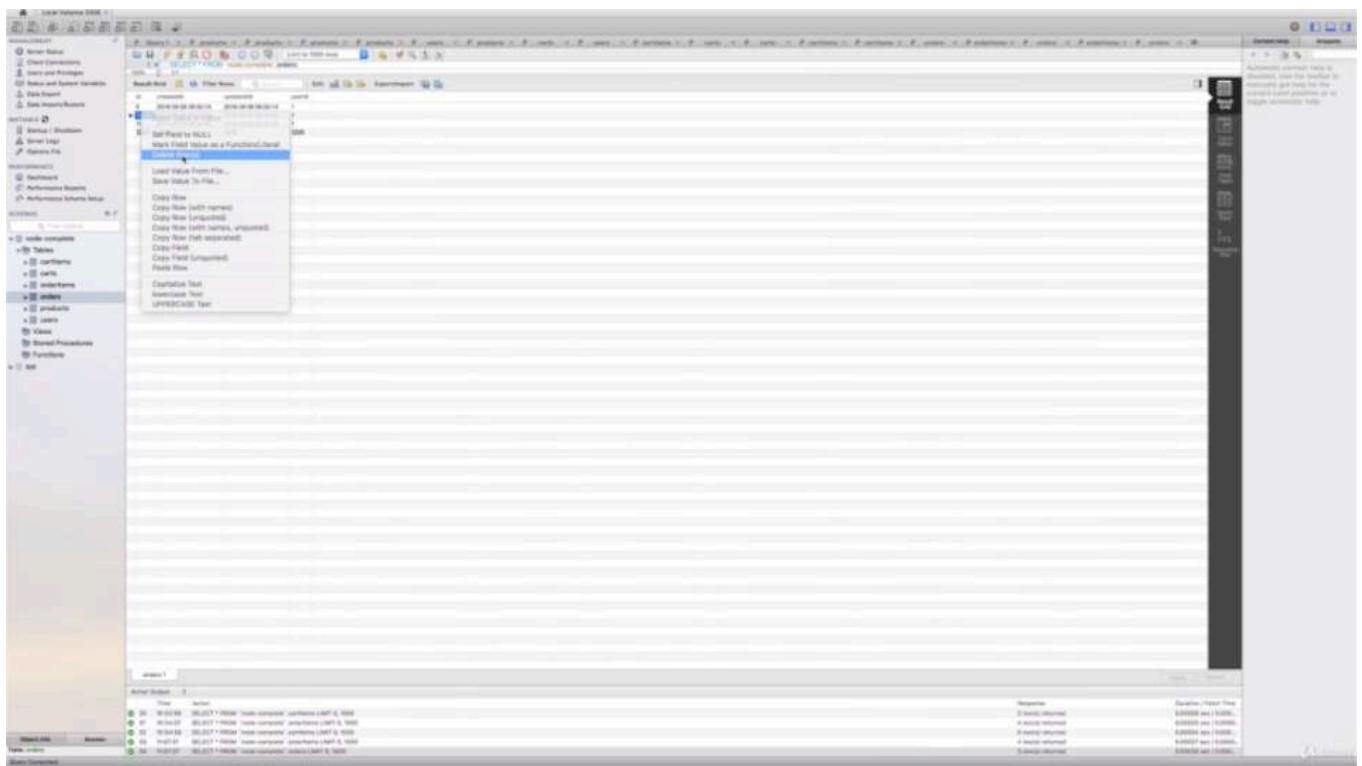
- if we reload this page, we do see our orders with the nested products in there.





- we can always verify this by looking into the database, we get 4 order items related to orders with the ID 9 and 11





- in the orders, we got 9, 10 and 11 so indeed there's the order with the ID 10 which has no items  

- right clicking 'delete' now and 'apply' and you can see #10 is gone

```

1  //./controllers/shop.js
2
3  const Product = require('../models/product');
4
5  exports.getProducts = (req, res, next) => {
6    Product.findAll()
7      .then(products => {
8        res.render('shop/product-list', {
9          prods: products,
10         pageTitle: 'All Products',
11         path: '/products'
12       });
13     })
14    .catch(err => {
15      console.log(err);
16    });
17  };
18
19  exports.getProduct = (req, res, next) => {
20    const prodId = req.params.productId;
21    // Product.findAll({ where: { id: prodId } })
22    // .then(products => {
23    //   res.render('shop/product-detail', {
24    //     product: products[0],
25    //     pageTitle: products[0].title,
26    //     path: '/products'
27    //   });
28    // })
29    // .catch(err => console.log(err));
30    Product.findById(prodId)
31      .then(product => {

```

```

32     res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36     });
37 })
38 .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42     Product.findAll()
43     .then(products => {
44         res.render('shop/index', {
45             prods: products,
46             pageTitle: 'Shop',
47             path: '/'
48         });
49     })
50     .catch(err => {
51         console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56     req.user
57     .getCart()
58     .then(cart => {
59         return cart
60             .getProducts()
61             .then(products => {
62                 res.render('shop/cart', {
63                     path: '/cart',
64                     pageTitle: 'Your Cart',
65                     products: products
66                 });
67             })
68             .catch(err => console.log(err));
69     })
70     .catch(err => console.log(err));
71 };
72
73 exports.postCart = (req, res, next) => {
74     const prodId = req.body.productId;
75     let fetchedCart;
76     let newQuantity = 1;
77     req.user
78     .getCart()
79     .then(cart => {
80         fetchedCart = cart;
81         return cart.getProducts({ where: { id: prodId } });
82     })
83     .then(products => {
84         let product;
85         if (products.length > 0) {
86             product = products[0];
87         }

```

```

88
89     if (product) {
90         const oldQuantity = product.cartItem.quantity;
91         newQuantity = oldQuantity + 1;
92         return product;
93     }
94     return Product.findByPk(prodId);
95 })
96 .then(product => {
97     return fetchedCart.addProduct(product, {
98         through: { quantity: newQuantity }
99     });
100 })
101 .then(() => {
102     res.redirect('/cart');
103 })
104 .catch(err => console.log(err));
105 };
106
107 exports.postCartDeleteProduct = (req, res, next) => {
108     const prodId = req.body.productId;
109     req.user
110         .getCart()
111         .then(cart => {
112             return cart.getProducts({ where: { id: prodId } });
113         })
114         .then(products => {
115             const product = products[0];
116             return product.cartItem.destroy();
117         })
118         .then(result => {
119             res.redirect('/cart');
120         })
121         .catch(err => console.log(err));
122 };
123
124 /**i will also create the new route for this to handle a POST request to this order route
125  * so a POST request to '/create-order'
126  *
127  */
128
129 exports.postOrder = (req, res, next) => {
130     /**a magic method added by sequelize
131     * thanks to our association.
132     */
133     let fetchedCart;
134     req.user
135         /**if we wanna fetch the related products to an order,
136         * we have to pass an object
137         * where we set 'include' to an array with the field 'products' or the element
138         'products' as a string
139         * why 'products'?
140         * because in app.js file, we associate an order to many 'Product',
141         * and if we have a look at the ./models/product.js file,
142         * the 'product' model has the name 'product'
143         * sequelize pluralize this

```

```

143     * and then we can use a concept called 'eager loading'
144     * where we instruct sequelize like
145     * hey if you are fetching all the orders,
146     * please also fetch all related 'products' already
147     * and give me back one array of orders that also includes the 'products' per order
148     * this only works because we do have a relation between orders and products as set up
in app.js 'Order.belongsToMany(Product, { through: OrderItem })'
149     * we can load both together.
150     *
151     * this will still not make our template work immediately
152     * but now we got orders with more data in them
153     * each order will now have a products array
154     * and with that in mind, we can go back to ./views/shop/orders.ejs file
155     * we can loop through the orders
156     * and every order will have an ID
157     */
158     .getCart()
159     .then(cart => {
160         fetchedCart = cart;
161         return cart.getProducts();
162     })
163     .then(products => {
164         return req.user
165             .createOrder()
166             .then(order => {
167                 return order.addProducts(
168                     products.map(product => {
169                         product.orderItem = { quantity: product.cartItem.quantity };
170                         return product;
171                     })
172                 );
173             })
174             .catch(err => console.log(err));
175     })
176     .then(result => {
177         return fetchedCart.setProducts(null);
178     })
179     .then(result => {
180         res.redirect('/orders');
181     })
182     .catch(err => console.log(err));
183 };
184
185 exports.getOrders = (req, res, next) => {
186     req.user
187         .getOrders({include: ['products']})
188         .then(orders => {
189             res.render('shop/orders', {
190                 path: '/orders',
191                 pageTitle: 'Your Orders',
192                 orders: orders
193             });
194         })
195         .catch(err => console.log(err));
196 };
197

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 router.get('/products/:productId', shopController.getProduct);
16
17 router.get('/cart', shopController.getCart);
18
19 router.post('/cart', shopController.postCart);
20
21 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
22
23 router.post('/create-order', shopController.postOrder);
24
25 router.get('/orders', shopController.getOrders);
26
27 module.exports = router;
28

```

```

1 <!--./views/shop/orders.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4 </head>
5
6 <body>
7   <%- include('../includes/navigation.ejs') %>
8   <main>
9     <% if (orders.length <= 0) { %>
10      <h1>Nothing there!</h1>
11    <% } else { %>
12      <ul>
13        <% orders.forEach(order => { %>
14          <li>
15            <h1># <%= order.id %></h1>
16            <ul>
17              <% order.products.forEach(product => { %>
18                <li><%= product.title %> (<%= product.orderItem.quantity
19              %></li>
20            <% }); %>
21          </ul>
22        </li>
23      <% }); %>
24    </ul>
25  <% } %>
26  </main>
27  <%- include('../includes/end.ejs') %>
28

```

# \* Chapter 170: Wrap Up





## Module Summary

### SQL

- SQL uses strict data schemas and relations
- You can connect your Node.js app via packages like mysql2
- Writing SQL queries is not directly related to Node.js and something you have to learn in addition to Node.js

### Sequelize

- Instead of writing SQL queries manually, you can use packages (ORMs) like Sequelize to focus on the Node.js code and work with native JS objects
- Sequelize allows you define models and interact with the database through them
- You can also easily set up relations ("Associations") and interact with your related models through them