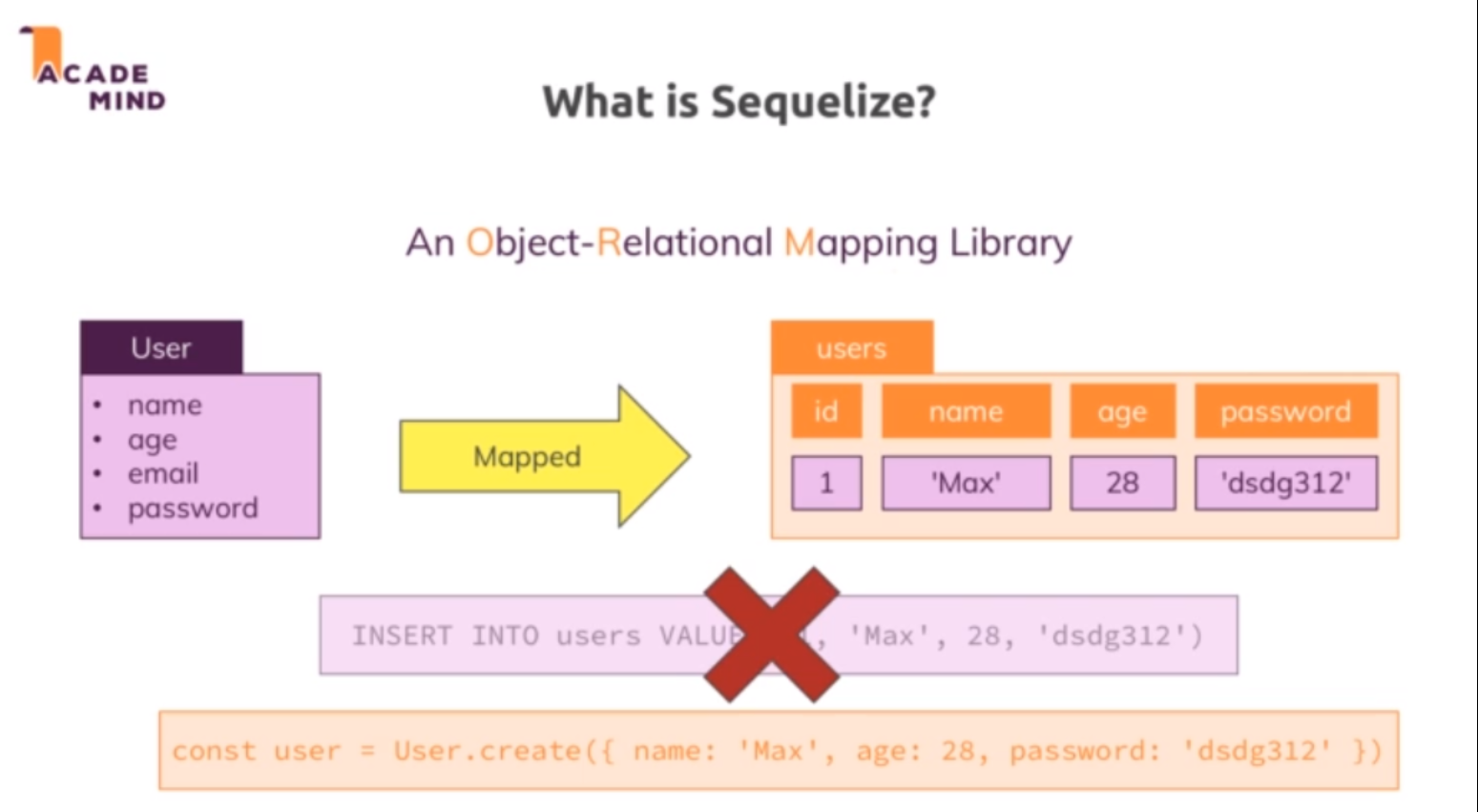
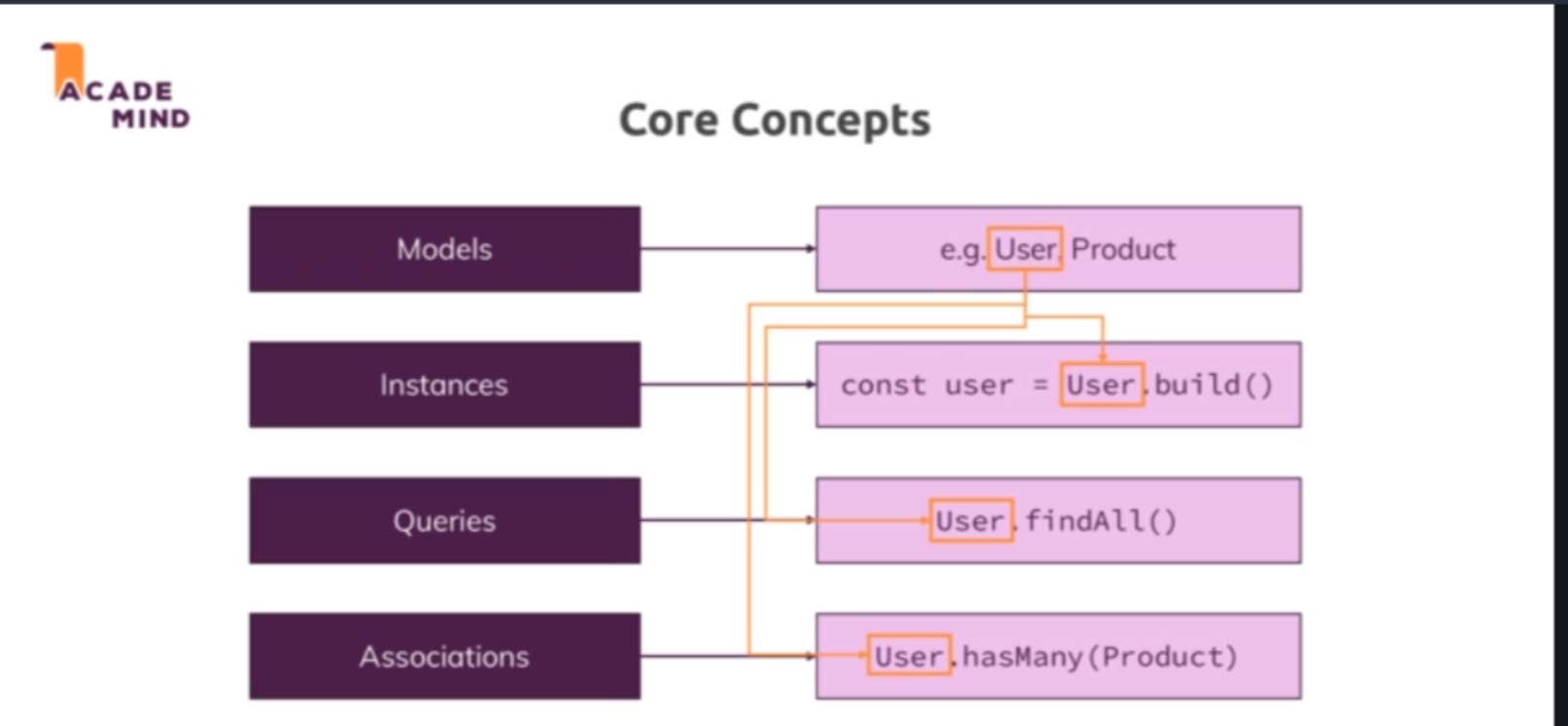
**Lecture 149**

**What is Sequelize?**





* Sequelize is a third party package, to be precise it's an object relational mapping library and this is a pretty long name which simply 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. It works like this, we got our object let's say a user with a name, age, email and password but of course this can be anything, could be a product, whatever you need and this is mapped to a database table by sequelize, so it automatically creates that table for us even, it automatically sets up relations and tables even for us, it does all that and when we create a new user for example, we simply call a method on that user javascript object and sequelize executes the SQL query or the SQL command that is required. So instead of writing this on our own, we simply create a javascript object and work with that and here would be one example using sequelize to create a new user which would behind the scenes execute the SQL code we don't have to write.
* 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. We can then instantiate these models, so these classes so to say, 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 here 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 and here again, this always relates 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.

**Lecture 150**

**Connecting to the database**

* Npm install –save sequelize.
* Sequelize requires mysql2 package.
* Refer code 01-defining-a-model , database.js
* , for example the dialect, we can set this to MySQL to make it clear that we connect to a MySQL database because different SQL engines or databases use slightly different SQL syntax and you can dive into the official sequelize docs to learn all about it

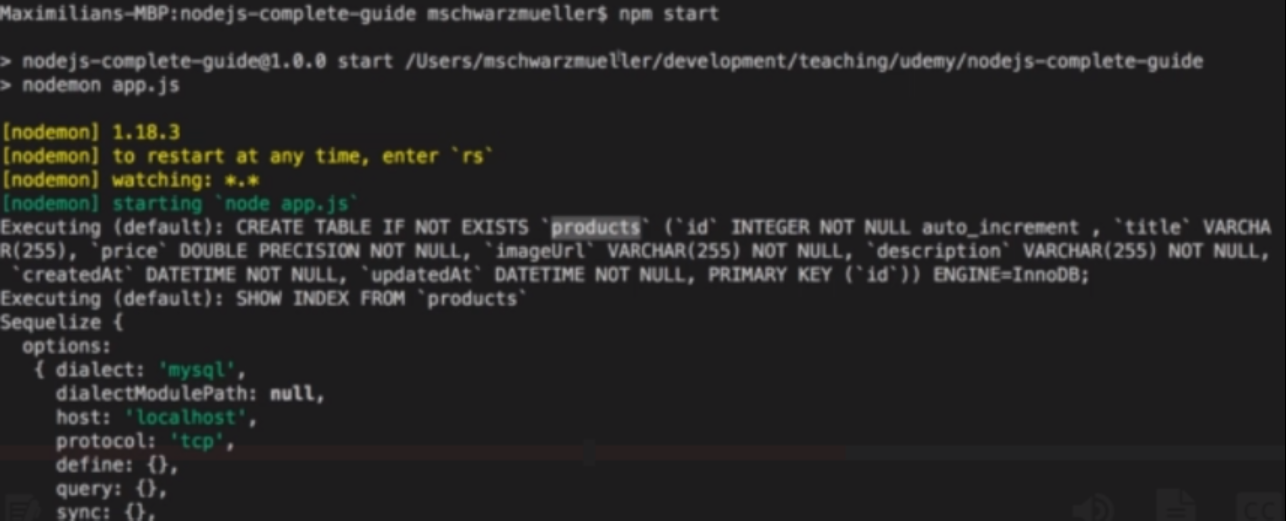
**Lecture 151**

**Defining a Model**

* Refer code 01-defining-a-model

**Lecture 152**

**Syncing JS Definitions to the Database**

* Refer code 02-inserting-data-and-creating-a-product.
* Sequelize.sync() creates/links to the tables based on the models we have created.
* 
* The table name will be the pluralised form of the model name. We created the model as ‘product’, so table name will be ‘products’.
* New table will be created only if not exists, otherwise it connect the model to the existing table.
* Sequelize automatically adds two new fields by default , createdAt and updatedAt. We can disable that behaviour.

**Lecture 153**

**Inserting Data & Creating a Product**

* Refer code 02-inserting-data-and-creating-a-product.
* Product.build creates only JS object.
* Product.create create the JS object and also save it to the database.

MUST READ: findById() in Sequelize 5

One quick note:

With Sequelize v5, findById() (which we'll use in this course) was replaced by findByPk().

You use it in the same way, so you can simply replace all occurrences of findById() with findByPk()

<https://sequelize.org/v5/>

**Lecture 155**

**Retrievning Data & Finding Products**

* Refer code 03-getting-a-single-product

**Lecture 156**

**Getting a Single Product with the ‘where’ condition**

* Refer code 03-getting-a-single-product

**Lecture 157**

**Fetching Admin Products**

* Refer code 04-updating-products.

**Lecture 158**

**Updating Products**

* Refer code 04-updating-products.
* product.save() updates the product in the database . If product doesnot exists it will create a new one.

**Lecture 159**

**Deleting Products**

* Refer code 05-deleting-products.

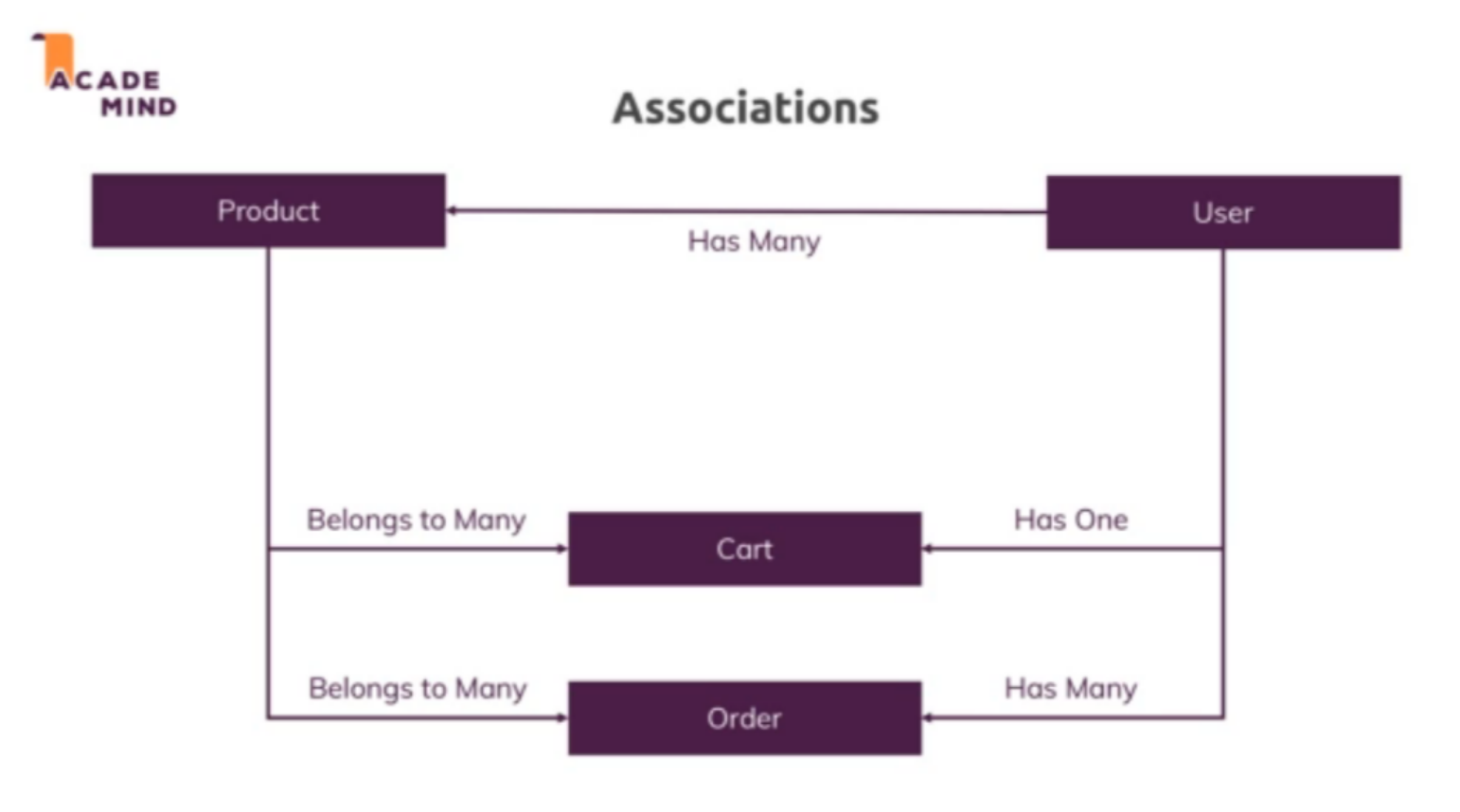
**Lecture 160**

**Creating a User Model**

* Refer code 06-adding-a-one-to-many-relationship 🡪 models/user.js

**Lecture 161**

**Adding a One To Many Relationship**

* Refer code 06-adding-a-one-to-many-relationship.
* App.js 🡪 Defining relationsa
* 

**Lecture 162**

**Creating and Managing a dummy user**

* Refer code 07-creating-and-managing-a-user 🡪 app.js.
* We are creating a dummy user and simulating login by putting the user data into the request object.
* Now onwards whenever we do some actions like create product , it will be created for that particular user.

**Lecture 163**

**Using Magic Association Methods**

* Refer code 08-fetching-related-products 🡪 controllers/admin.js
* Based on the associations we create , sequelize adds certain methods which we can use to do some db operations. E.g user object will have createProduct method which is automatically added by sequelize based on the one to many association b/w user and products table.

**Lecture 164**

**Fetching related products**

* Refer code 08-fetching-related-products 🡪
* Controllers/admin.js 🡪 getEditProduct method, getProducts method

**Lecture 165**

**One To Many and Many to Many Relations**

* Refer code 09-creating-and-fetching-a-cart
* Models/carts.js
* Models/cart-item.js
* App.js

**Lecture 166**

**Creating and Fetching a Cart**

* Refer code 10-adding-existig-products-and-retrieving
* Controllers/shop.js 🡪 getCart method
* App.js

**Lecture 167**

**Adding new products to the cart**

* Refer code 10-adding-existig-products-and-retrieving
* Controllers/shop.js 🡪 postCart method

**Lecture 168**

**Adding existing products and retrieving cart items**

* Refer code 10-adding-existig-products-and-retrieving
* Views/shop/cart.ejs 🡪 product.cartItem.quantity

Now the quantity is not part of that but of the related cart item you could say and conveniently, sequelize also gives us a cart item key for this which stores information about this in-between table and the entry that is related to this product there.

* Controllers/shop.js 🡪 postCart method.

**Lecture 169**

**Deleting related items and deleting cart**

* Refer code 11-deleting-related-items
* Controllers/shop.js 🡪 postCartDeleteProduct method

**Lecture 170**

**Adding an Order Model**

* Refer code 12-storing-cartitems-as-orderitems
* Let's start with the model and for this, I'll create a new order.js file and copy my cart item.js file, move it in there, import sequelize, rename it here to order, also here and now how should an order look like? Well an order is in the end just an in-between table between a user to which the order belongs and then multiple products that are part of the order and these products again do have a quantity attached to them. So just as we had cart items for the cart, I'll have order items for my order. So I can copy cart items again, move that into order item and rename cart item here to order item, starting with a lowercase o here in the string name definition and then it will have the same structure as an order, as in cart item here in order item and the order itself will not have anything but the ID because the order essentially is like the cart
* Models/order.js, order-item.js
* App.js 🡪 creating assocations

**Lecture 171**

**Storing CartItems as OrderItems**

* Refer code 12-storing-cartitems-as-orderitems.
* Cart.ejs 🡪 Added checkout button to create an order
* Controllers/shop.js 🡪 postOrder method
* Routes/shop.js 🡪 register new route /create-order
* So in post order here, we can now call request user and just as we create a cart for that user in app.js with create cart, we can now call create order for that user. Now this gives us an order but we don't just need the order, we also need to associate our products to that order, so here I'll return request user create order. And with the order created, and here I'll again do this nested, you can always restructure it to not use a nested promise here though if you want but here I will get my created order and now I want to associate my products to that order and that can be done easily by calling order add products and passing my products here. Now important, we need to specify true and set the quantity here correctly too but now which value would we assign there because we get different quantities for all the products? The approach is a little different, we don't pass it like this, we just pass products to add products but each product needs to have a special key, a special field which is then understood by sequelize.
* Now to assign that special field I'm talking of, the products we pass in here have to be modified and we can do this with the map method, a default javascript method that runs on an array and returns a new array with slightly modified elements. We add a function here that is executed for every element in the array and takes the element as an input and returns the modified version. I'll return products here in the end but before I do so, I do edit it slightly, a new property which sequelize will look for named order item. Now the name here is important to get this right. If in your order item model, you define this name, that is the name you have to use, if you chose a different name, you have to use the different name. So here I have order item with a lower case o and a capital I and therefore here, I have product order item written in the same way.
* This now stores a javascript object where I configure the value for this in-between table, so here I simply define quantity which is the value I need to store in between and I set this equal to product cart item, this is the related table I have due to my cart, quantity. So I get the quantity from the cart and store that for the order item, this then gets returned here, so now in the end I have an array of products with all the old product data but also this new information regarding the quantity for my order and add products will pick this up and add the products to the order with that quantity. This is what's happening here, now we can return order add products here and add a new then block here where I get any result and in here, I will then redirect to orders. With that set up, let's go back and reload our cart page, 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, there is one order and order items also has the respective elements that belong to the order with the right quantities.

**Lecture 172**

**Resetting the Cart and Fetching and Outputting Orders**

* Refer code 13-resetting-the-cart-and-fetching.
* Controllers/shop.js 🡪 postOrder method
* Views/shop/orders.ejs