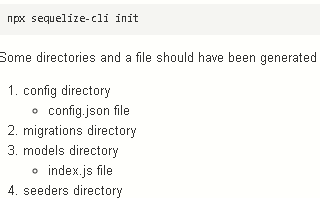
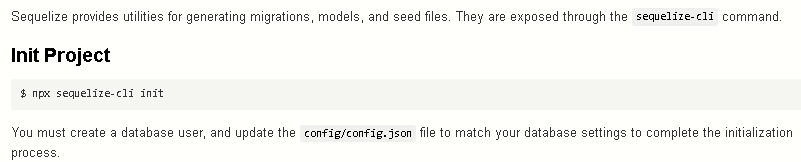
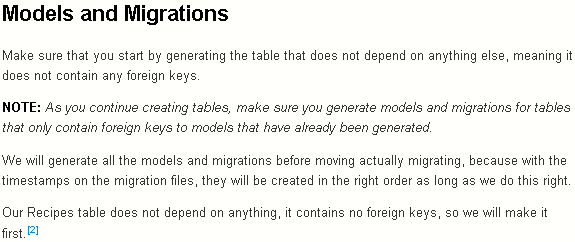


**create user recipe\_box\_app with createdb password 'password';**

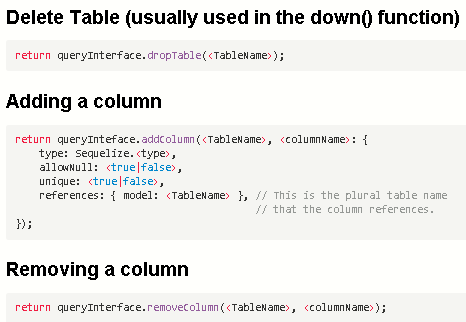


v



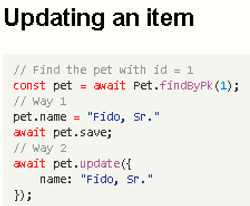


v





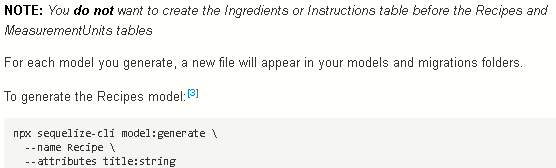
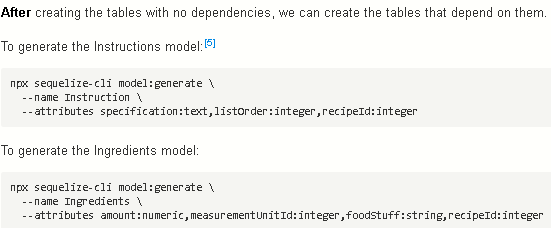


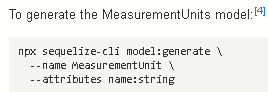


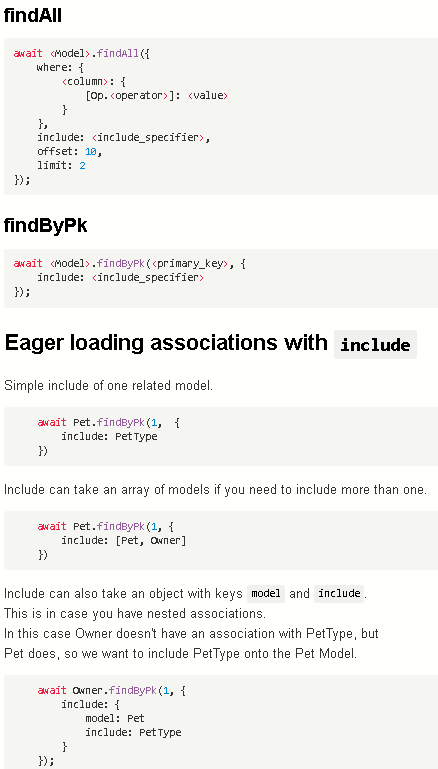
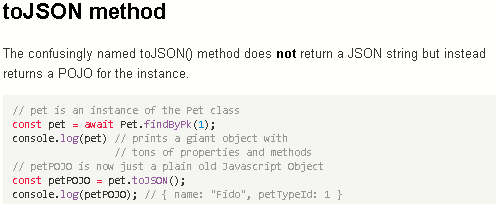
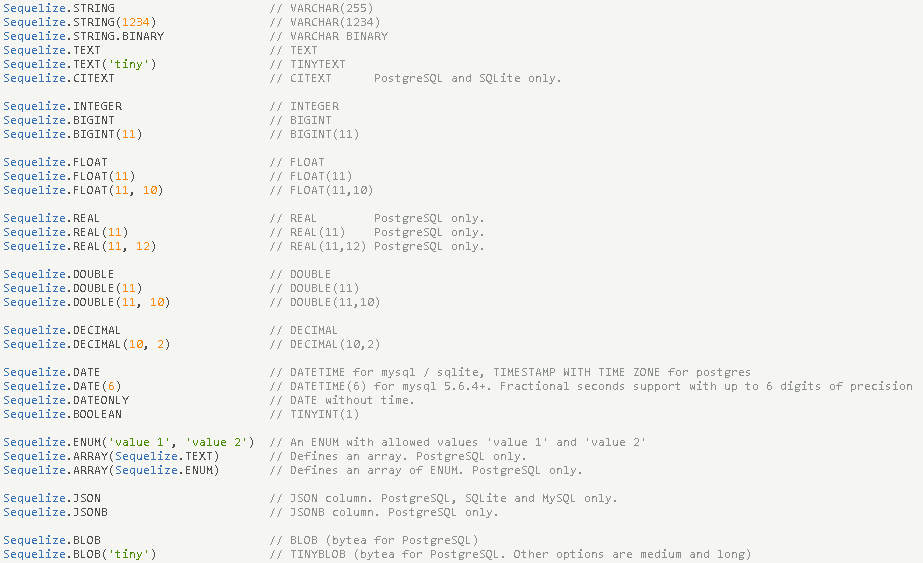




1. Change all of the usernames to the username from "root" to the username that you created before: "username": "recipe\_box\_app"
2. Change the password line to provide the password for the user you created for this app: "password": "password"
3. Change the database line for all three environments (development, test, and production) to "database": "recipe\_box\_development", "database": "recipe\_box\_test", and "database": "recipe\_box\_production".
4. Change the dialect line to "dialect": "postgres"
5. Remove the "operatorsAliases" line
6. Add the line: "seederStorage": "sequelize"

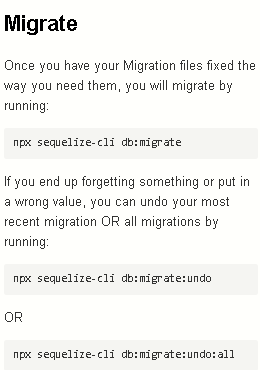


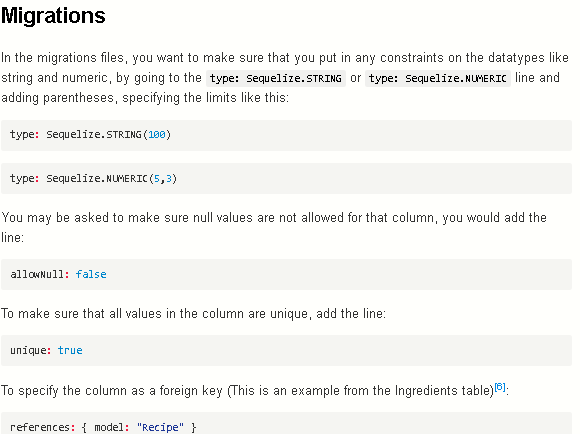




­







## Models and Associations

|  |
| --- |
| **In your model’s files, you will specify the associations. You can have one-to-one, one-to-many, or many-to-many.** |
| For our example, the Recipes and Instructions table have a one-to-many association, the Recipes and Instructions table have a one-to-many association, and the MeasurementUnits and Ingredients have a on-to-many relationship. |
| Let's associate the Recipes and Instructions first. We will go into the Recipes model file at models/recipe.js and in the Recipe.associate file. Since the Recipes table does not contain a foreign key, but is referenced by the Instructions table, we will call the hasMany function here: |
| Recipe.hasMany(models.Instruction, { foreignKey: 'recipeId' }); |
| In plain English, you can read the above code as: "Each recipe has many instructions, and each instruction references the recipe with the foreign key 'recipeId'." |
| Since the Instruction model contains a foreign key referencing the Recipe model, it *belongs to* the Recipe model. So, in the models/instruction.js we will associate the Instruction model to the Recipe model using the belongsTo function: |
| Instruction.belongsTo(models.Recipe, { foreignKey: 'recipeId' }); |
| In plain English, you can read the above code as: "Each Instruction belongs to a recipe that is referenced by the foreign key 'recipeId'." |
| To set up the Recipes to Ingredients association, |
| In the models/recipe.js file, define the association as: |
| Recipe.hasMany(models.Ingredient, { foreignKey: 'recipeId' }); |
| and in the models/ingredient.js file, define the association as: |
| Ingredient.belongsTo(models.Recipe, { foreignKey: 'recipeId' }); |
| Finally, for the Ingredients to MeasurementUnits associations, |
| In the models/ingredient.js file, define the association as: |
| Ingredient.belongsTo(models.MeasurementUnit, { foreignKey: 'measurementUnitId' }); |

and in the models/measurementunit.js file, define the association as:

MeasurementUnit.hasMany(models.Ingredient, { foreignKey: 'measurementUnitId' });

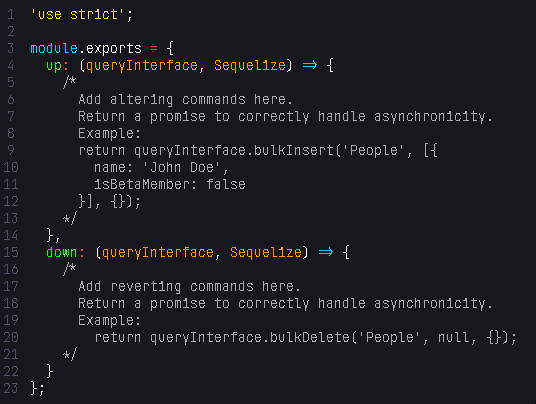
The many to many relationship is the only one that is ***really*** different

**Seeding Tables**

Now we need to seed the tables, so we will go back to the terminal and generate the seed files. You will run:

npx sequelize-cli seed:generate --name recipe-seeder

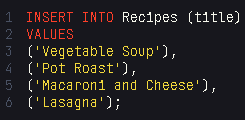
You can name the seeder file anything you would like. The file with the name you gave it will appear in your seeders/ folder. We need to open it up and put in information for each item that we want to seed into the table. **NOTE:** *Make sure you provide the createdAt and updatedAt values, otherwise they will be considered null, which is not allowed.*

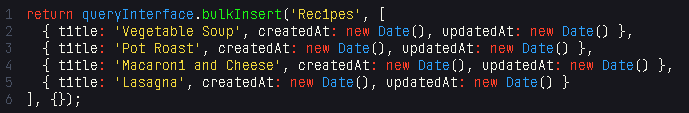


This will translate into SQL as:

Inside of the up and down methods here, you see a comment with an "Example" in it. These are extremely helpful so you don't have to memorize exactly what to do here. You can take these out of the comments and use them, but make sure you change 'People' to the table name, which should be \_plural\_ here.

You will place your seed data as an array of POJOs in the bulkInsert method in the up section. The up method of the Recipes seeder will look a little like this:





## Accessing the Data

You can access and query the data using the findByPk, findOne, and findAll methods. First, make sure you import the models in your JavaScript file. In this case, we are assuming your JavaScript file is in the root of your project and so is the models folder.

const { Recipe, Ingredient, Instruction, MeasurementUnit } = require('./models');

The models folder exports each of the models that you have created. We have these four in our data model, so we will destructure the models to access each table individually. The associations that you have defined in each of your models will allow you to access data of related tables when you query your database using the include option.

If you want to find all recipes, for the recipe list, you would use the findAll method. You need to await this, so make sure your function is async.

async function findAllRecipes() {

return await Recipe.findAll();

}

If you would like to include all the ingredients so you can create a shopping list for all the recipes, you would use include. This is possible because of the association you have defined in your Recipe and Ingredient models.

async function getShoppingList() {

return await Recipe.findAll({ include: [ Ingredient ] });

}

If you only want to find one where there is chicken in the ingredients list, you would use findOne and findByPk.

async function findAChickenRecipe() {

const chickenRecipe = await Ingredient.findOne({

where: {

foodStuff: 'chicken'

}

});

return await Recipe.findByPk(chickenRecipe.recipeId);

}

## Data Access to Create/Update/Delete Rows

You have two options when you want to create a row in a table (where you are saving one record into the table). You can either .build the row and then .save it, or you can .create it. Either way it does the same thing. Here are some examples:

Let's say we have a form that accepts the name of the recipe (for simplicity). When we get the results of the form, we can:

const newRecipe = await Recipe.build({ title: 'Chicken Noodle Soup' });

await newRecipe.save();

This just created our new recipe and added it to our Recipes table. You can do the same thing like this:

await Recipe.create({ title: 'Chicken Noodle Soup' });

If you want to modify an item in your table, you can use update. Let's say we want to change the chicken noodle soup to chicken noodle soup with extra veggies, first we need to get the recipe, then we can update it.

const modRecipe = await Recipe.findOne({ where: { title: 'Chicken Noodle Soup' } });

await modRecipe.update({ title: 'Chicken Noodle Soup with Extra Veggies' });

To delete an item from your table, you will do the same kind of process. Find the recipe you want to delete and destroy it, like this:

const deleteThis = await Recipe.findOne({ where: { title: 'Chicken Noodle Soup with Extra Veggies' } });

await deleteThis.destroy();

**NOTE:** If you do not await these, you will receive a promise, so you will need to use .then and .catch to do more with the items you are accessing and modifying.

## Documentation

For the data types and validations in your models, here are the official docs. The sequelize docs are hard to look at, so these are the specific sections with just the lists:  
**Sequelize Data Types:** [*https://sequelize.org/v5/manual/data-types.html*](https://sequelize.org/v5/manual/data-types.html)  
**Validations:** [*https://sequelize.org/v5/manual/models-definition.html#validations*](https://sequelize.org/v5/manual/models-definition.html#validations)  
When you access the data in your queries, here are the operators available, again because the docs are hard to navigate, this is the specific section with the list of operators.  
**Operators:** [*https://sequelize.org/v5/manual/querying.html#operators*](https://sequelize.org/v5/manual/querying.html#operators)  
The documentation for building, saving, creating, updating and destroying is linked here, it does a pretty good job of explaining in my opinion, it just has a title that we have not been using in this course. When they talk about an instance, they mean an item stored in your table.  
**Create/Update/Destroy:** [*https://sequelize.org/v5/manual/instances.html*](https://sequelize.org/v5/manual/instances.html)