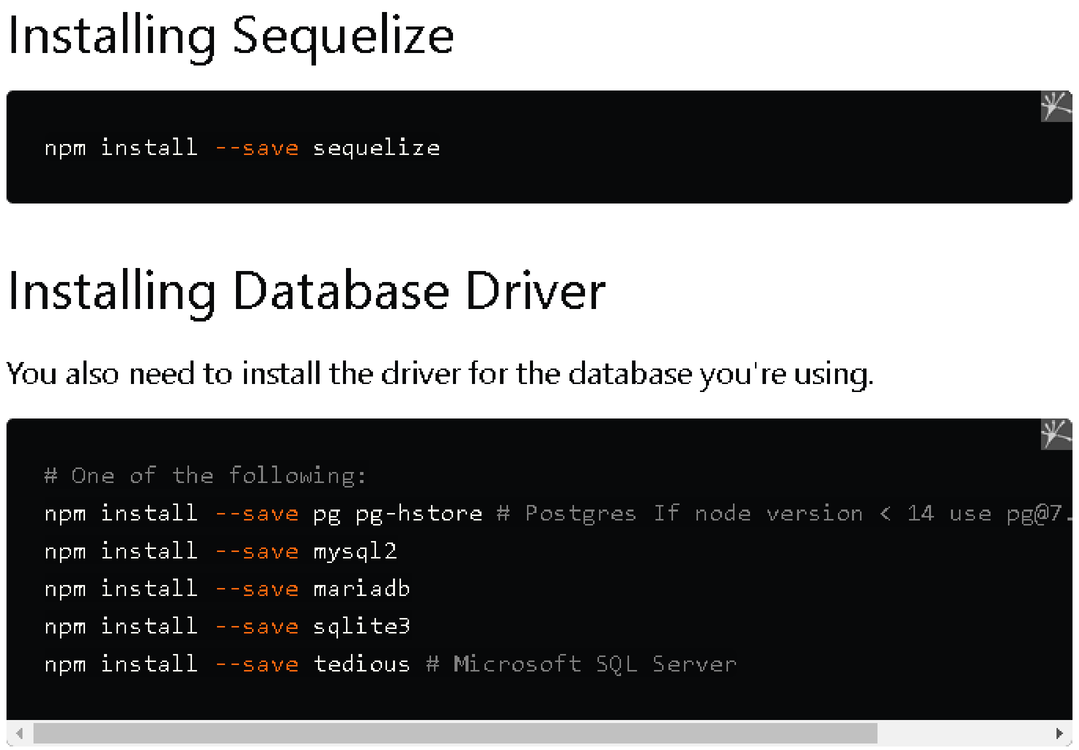


Installing Sequelize

```
npm install --save sequelize
```

Installing Database Driver

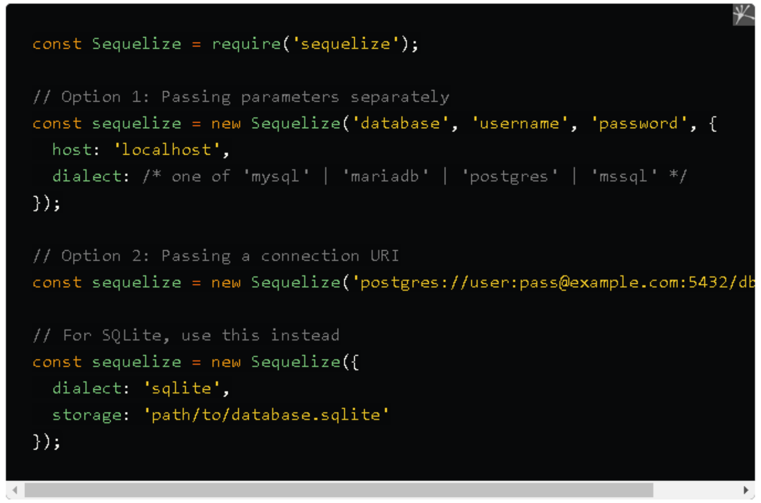
You also need to install the driver for the database you're using.

```
# One of the following:
npm install --save pg pg-hstore # Postgres If node version < 14 use pg@7.
npm install --save mysql2
npm install --save mariadb
npm install --save sqlite3
npm install --save tedious # Microsoft SQL Server
```

Setting up a Connection

A Sequelize instance must be created to connect to the database. By default, this connection is kept open and used for all the queries but can be closed explicitly.

Instance Creation



```
const Sequelize = require('sequelize');

// Option 1: Passing parameters separately
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: /* one of 'mysql' | 'mariadb' | 'postgres' | 'mssql' */
});

// Option 2: Passing a connection URI
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/db

// For SQLite, use this instead
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'path/to/database.sqlite'
});
```

Testing Connection

`.authenticate()` can be used with the created instance to check whether the connection is working.

```
sequelize
  .authenticate()
  .then(() => {
    console.log("Connection has been established successfully.");
  })
  .catch((err) => {
    console.error("Unable to connect to the database:", err);
  });
```

Closing Connection

```
sequelize.close();
```

Closing Connection

```
sequelize.close();
```

Defining Models

Basic Definition

To define mappings between Model and Table, we can use the `.define()` method
To set up a basic model with only attributes and their datatypes

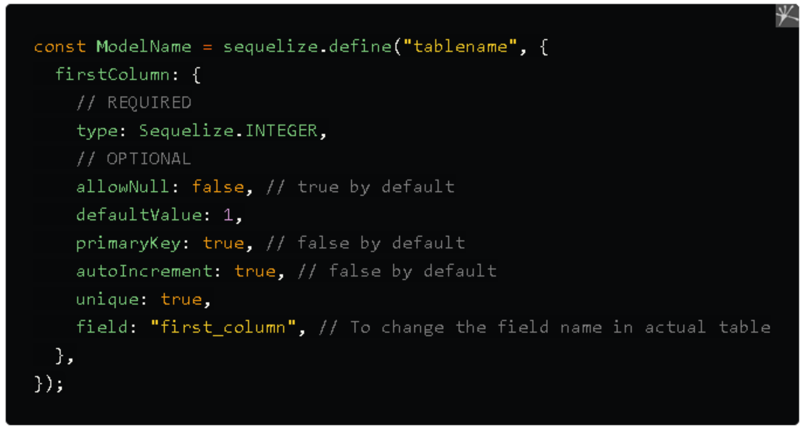
```
const ModelName = sequelize.define("tablename", {
  // s will be appended automatically to the tablename
  firstColumn: Sequelize.INTEGER,
  secondColumn: Sequelize.STRING,
});
```

For getting a list of all the data types supported by Sequelize, check out the official docs

Extending Column Definition

Basic Extentions

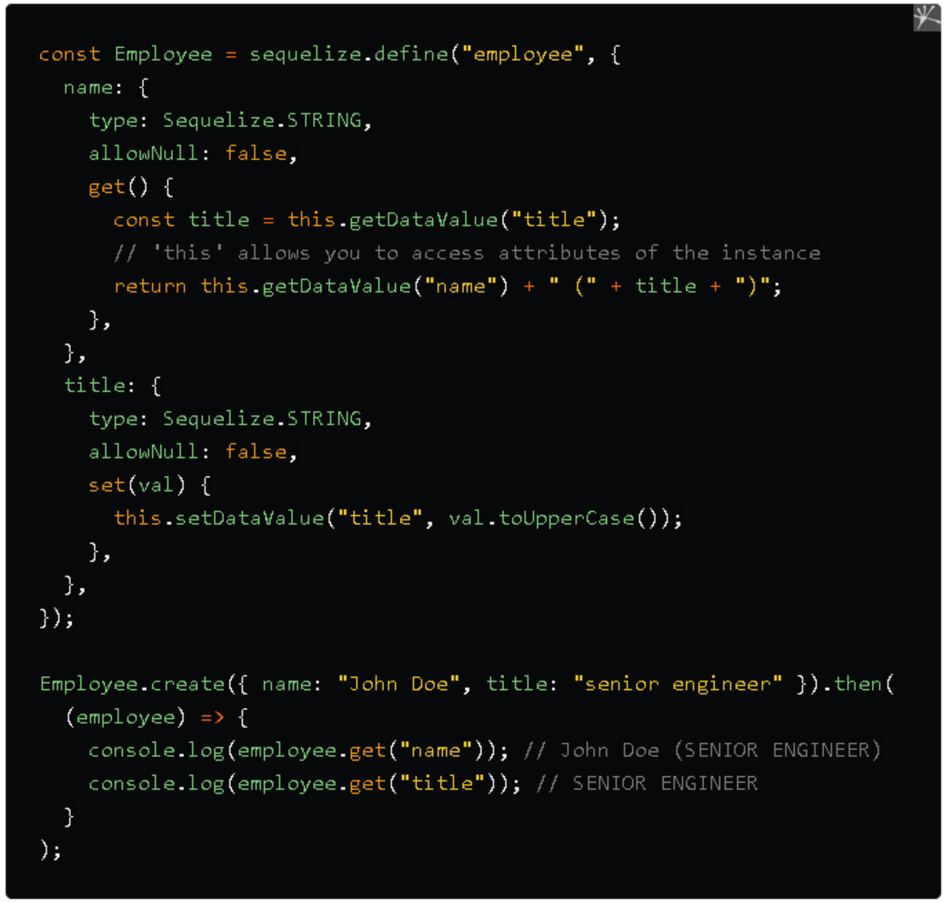
Apart from datatypes, many other options can also be set on each column



```
const ModelName = sequelize.define("tablename", {
  firstColumn: {
    // REQUIRED
    type: Sequelize.INTEGER,
    // OPTIONAL
    allowNull: false, // true by default
    defaultValue: 1,
    primaryKey: true, // false by default
    autoIncrement: true, // false by default
    unique: true,
    field: "first_column", // To change the field name in actual table
  },
});
```

Getters and Setters

Getters can be used to get the value of the column after some processing. Setters can be used to process the value before saving it into the table.

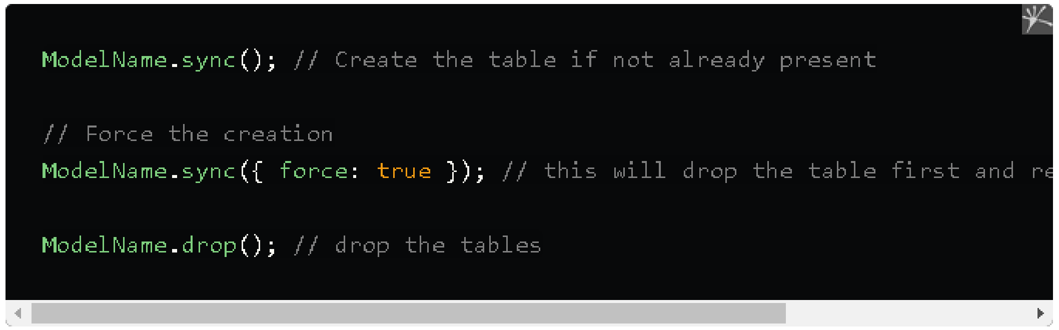


```
const Employee = sequelize.define("employee", {
  name: {
    type: Sequelize.STRING,
    allowNull: false,
    get() {
      const title = this.getDataValue("title");
      // 'this' allows you to access attributes of the instance
      return this.getDataValue("name") + " (" + title + ")";
    },
  },
  title: {
    type: Sequelize.STRING,
    allowNull: false,
    set(val) {
      this.setDataValue("title", val.toUpperCase());
    },
  },
});

Employee.create({ name: "John Doe", title: "senior engineer" }).then(
  (employee) => {
    console.log(employee.get("name")); // John Doe (SENIOR ENGINEER)
    console.log(employee.get("title")); // SENIOR ENGINEER
  }
);
```

For more in-depth information about Getters and Setters, check out the official docs

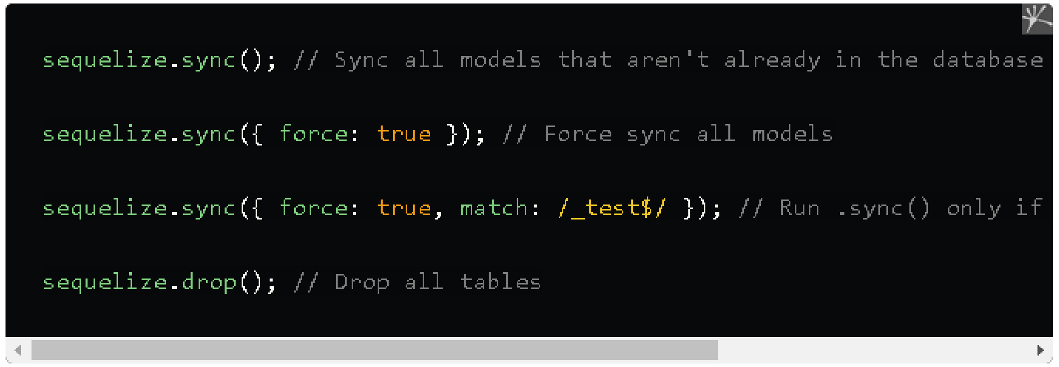
Database Synchronization

Sequelize can automatically create the tables, relations and constraints as defined in the models



```
ModelName.sync(); // Create the table if not already present

// Force the creation
ModelName.sync({ force: true }); // this will drop the table first and re

ModelName.drop(); // drop the tables
```

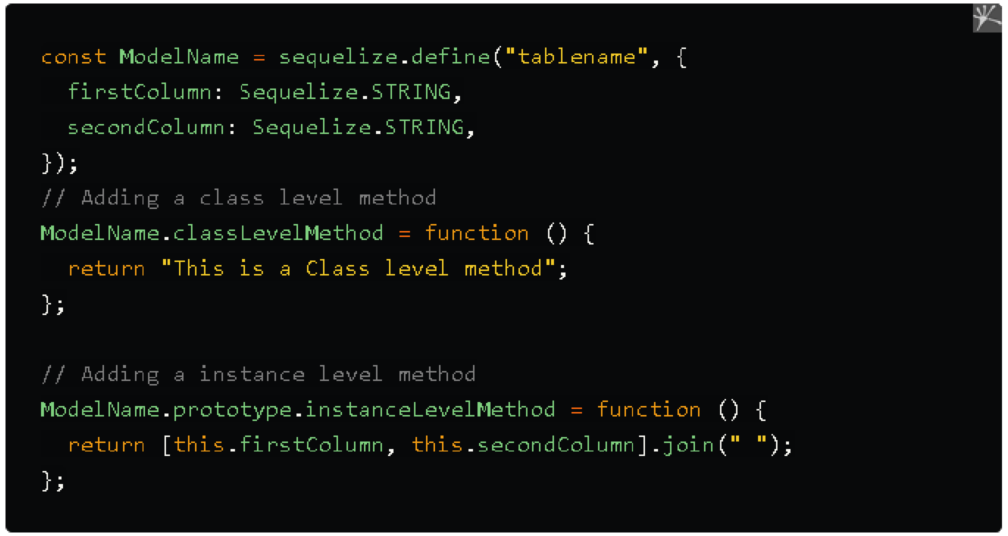
You can manage all models at once using sequelize instead



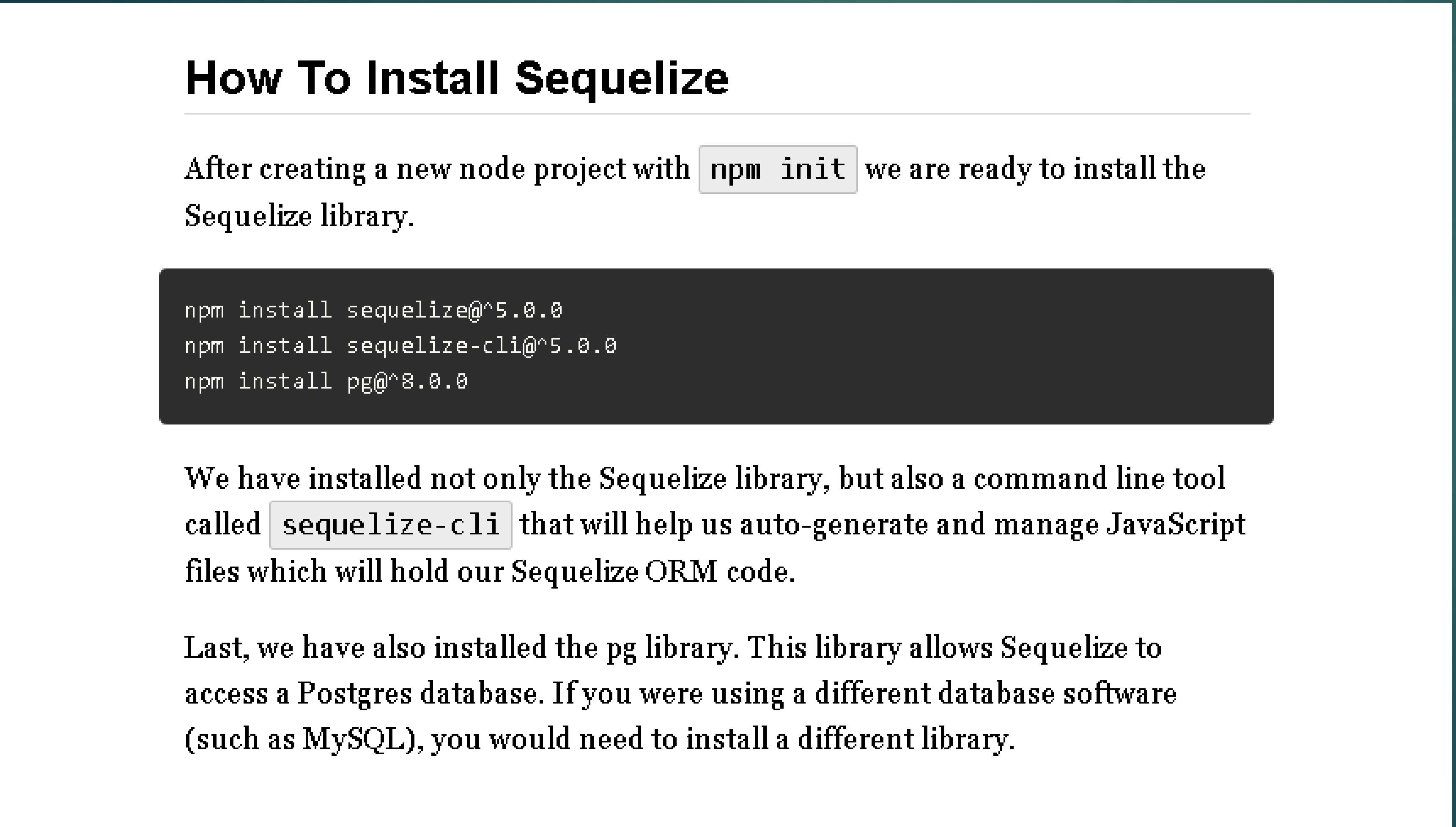
```
sequelize.sync(); // Sync all models that aren't already in the database

sequelize.sync({ force: true }); // Force sync all models

sequelize.sync({ force: true, match: /_test$/ }); // Run .sync() only if

sequelize.drop(); // Drop all tables
```

Expansion of Models

Sequelize Models are ES6 classes. We can easily add custom instance or class level methods.



```
const ModelName = sequelize.define("tablename", {
  firstColumn: Sequelize.STRING,
  secondColumn: Sequelize.STRING,
});
// Adding a class level method
ModelName.classLevelMethod = function () {
  return "This is a Class level method";
};

// Adding a instance level method
ModelName.prototype.instanceLevelMethod = function () {
  return [this.firstColumn, this.secondColumn].join(" ");
};
```



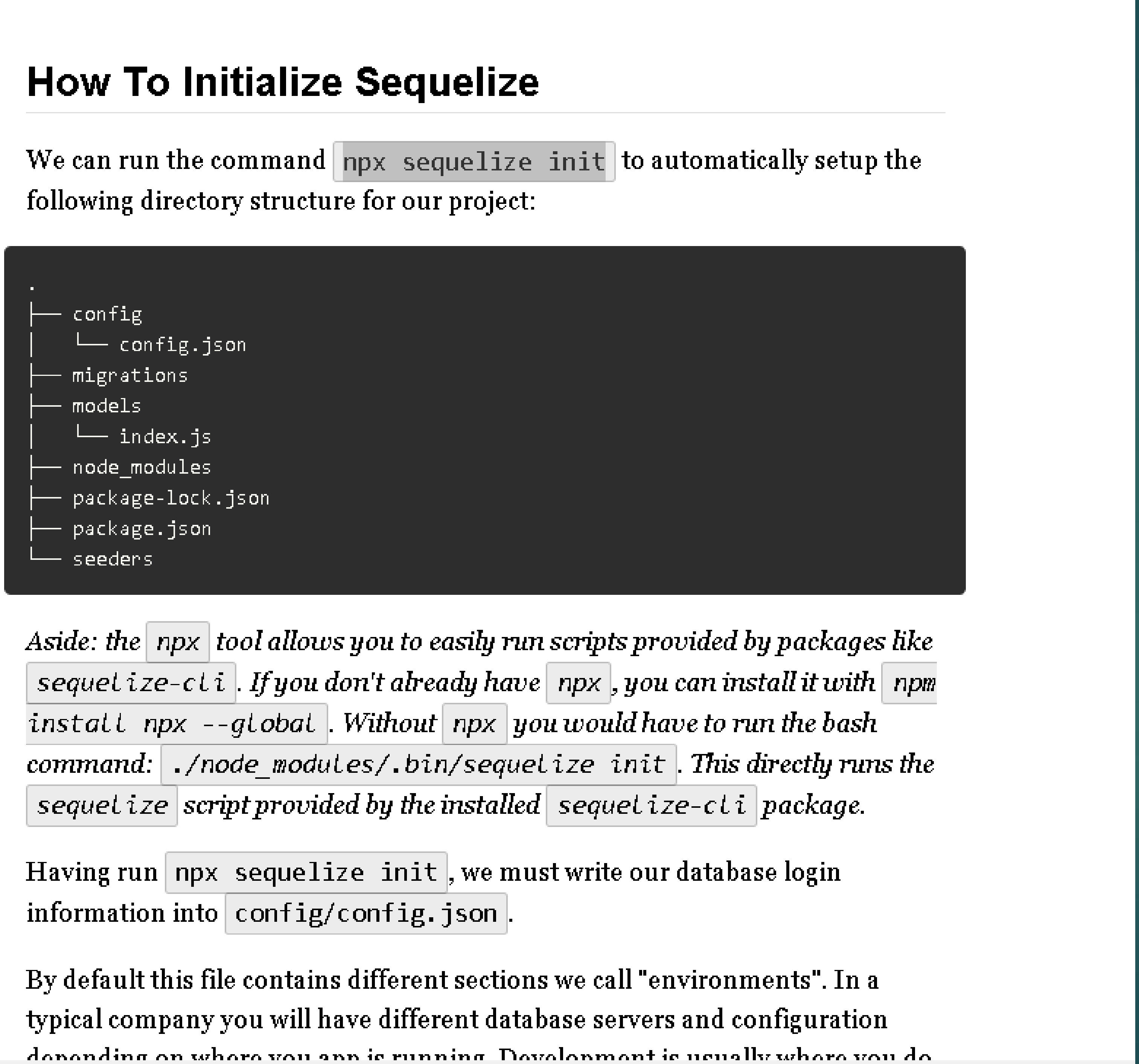
How To Install Sequelize

After creating a new node project with `npm init` we are ready to install the Sequelize library.

```
npm install sequelize@^5.0.0
npm install sequelize-cli@^5.0.0
npm install pg@^8.0.0
```

We have installed not only the Sequelize library, but also a command line tool called `sequelize-cli` that will help us auto-generate and manage JavaScript files which will hold our Sequelize ORM code.

Last, we have also installed the pg library. This library allows Sequelize to access a Postgres database. If you were using a different database software (such as MySQL), you would need to install a different library.



How To Initialize Sequelize

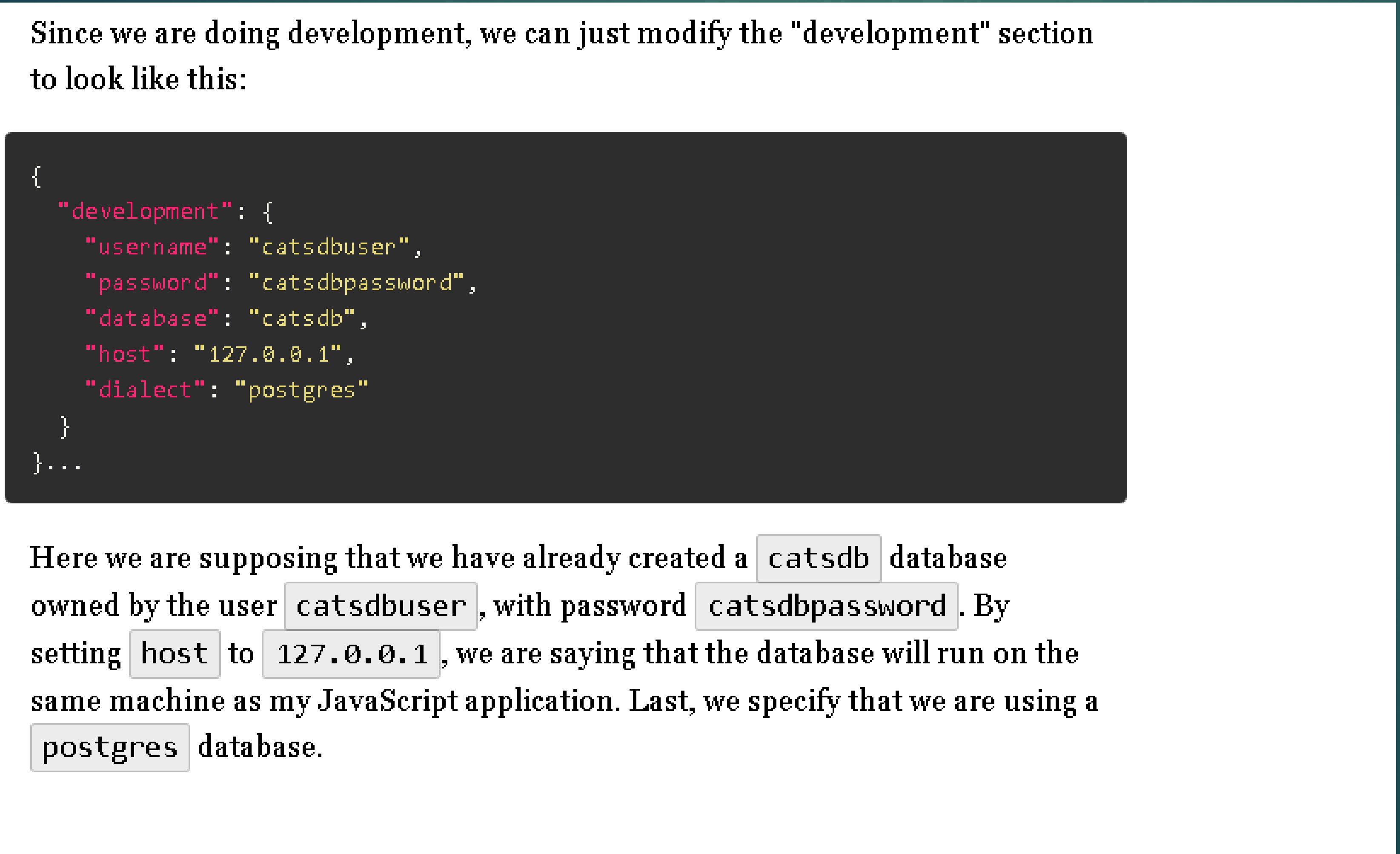
We can run the command `npx sequelize init` to automatically setup the following directory structure for our project:

```
.
├── config
│   └── config.json
├── migrations
├── models
│   └── index.js
├── node_modules
├── package-lock.json
├── package.json
└── seeders
```

Aside: the `npx` tool allows you to easily run scripts provided by packages like `sequelize-cli`. If you don't already have `npx`, you can install it with `npm install npx --global`. Without `npx` you would have to run the bash command: `./node_modules/.bin/sequelize init`. This directly runs the `sequelize` script provided by the installed `sequelize-cli` package.

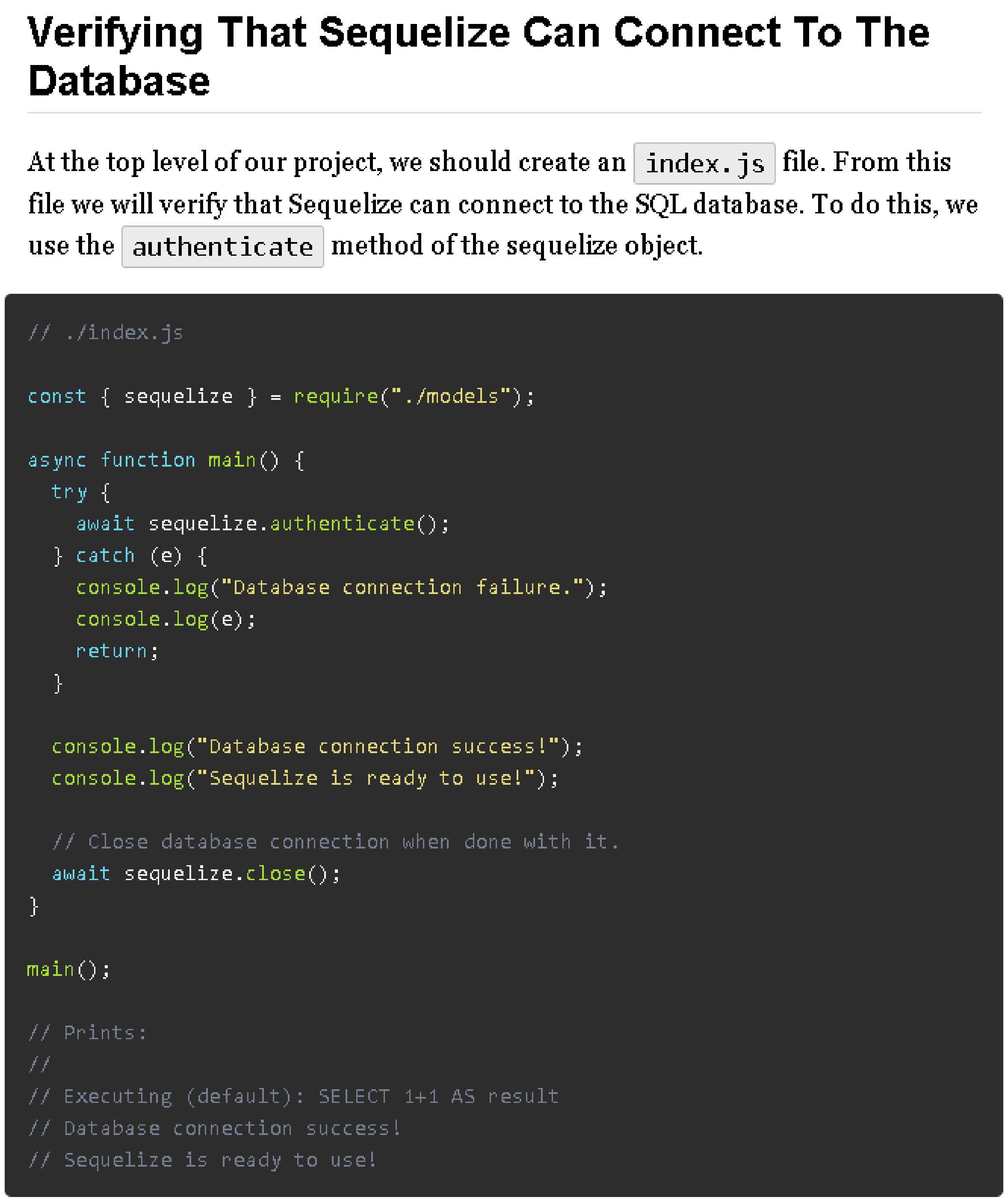
Having run `npx sequelize init`, we must write our database login information into `config/config.json`.

By default this file contains different sections we call "environments". In a typical company you will have different database servers and configuration depending on where you app is running. Development is usually where you do


Since we are doing development, we can just modify the "development" section to look like this:

```
{
  "development": {
    "username": "catsdbuser",
    "password": "catsdbpassword",
    "database": "catsdb",
    "host": "127.0.0.1",
    "dialect": "postgres"
  }
}...
```

Here we are supposing that we have already created a `catsdb` database owned by the user `catsdbuser`, with password `catsdbpassword`. By setting `host` to `127.0.0.1`, we are saying that the database will run on the same machine as my JavaScript application. Last, we specify that we are using a `postgres` database.



Verifying That Sequelize Can Connect To The Database

At the top level of our project, we should create an `index.js` file. From this file we will verify that Sequelize can connect to the SQL database. To do this, we use the `authenticate` method of the sequelize object.

```javascript
// ./index.js

const { sequelize } = require("./models");

async function main() {
  try {
    await sequelize.authenticate();
  } catch (e) {
    console.log("Database connection failure.");
    console.log(e);
    return;
  }

  console.log("Database connection success!");
  console.log("Sequelize is ready to use!");

  // Close database connection when done with it.
  await sequelize.close();
}

main();

// Prints:
//
// Executing (default): SELECT 1+1 AS result
// Database connection success!
// Sequelize is ready to use!
```

You may observe that the authenticate method returns a JavaScript Promise object. We use await to wait for the database connection to be established. If authenticate fails to connect, the Promise will be rejected. Since we use await, an exception will be thrown.

Many Sequelize methods return Promises. Using async and await lets us use Sequelize methods as if they were synchronous. This helps reduce code complexity significantly.

Note that I call sequelize.close(). This closes the connection to the database. A Node.js JavaScript program will not terminate until all open files and database connections are closed. Thus, to make sure the Node.js program doesn't "hang" at the end, we close the database connection. Otherwise we will be forced to kill the Node.js program with CTRL-C, which is somewhat annoying.

Using Sequelize To Generate The Model File

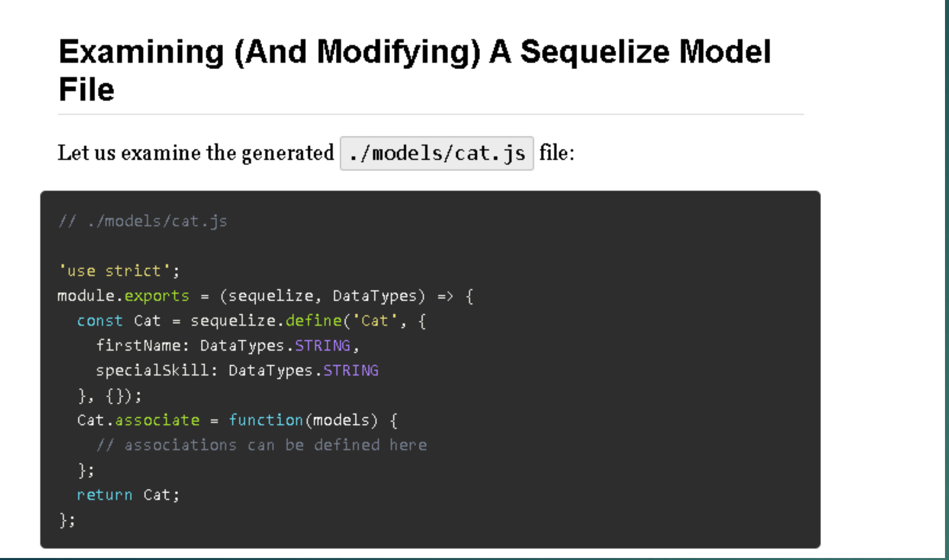
We will configure Sequelize to access the Cats table via a JavaScript class called Cat. To do this, we first use our trusty Sequelize CLI:

```
# Oops, forgot age:integer! (Don't worry we'll fix it later)
npx sequelize model:generate --name Cat --attributes
"firstName:string,specialSkill:string"
```

This command generates two files: a model file (./models/cat.js) and a migration file (./migrations/20200203211508-Cat.js). We will ignore the migration file for now, and focus on the model file.

When using Sequelize's model:generate command, we specify two things. First: we specify the singular form of the Cats table name (Cat). Second: we list the columns of the Cats table after the --attributes flag: firstName and specialSkill. We tell Sequelize that these are both string columns (Sequelize calls SQL character varying(255) columns strings).

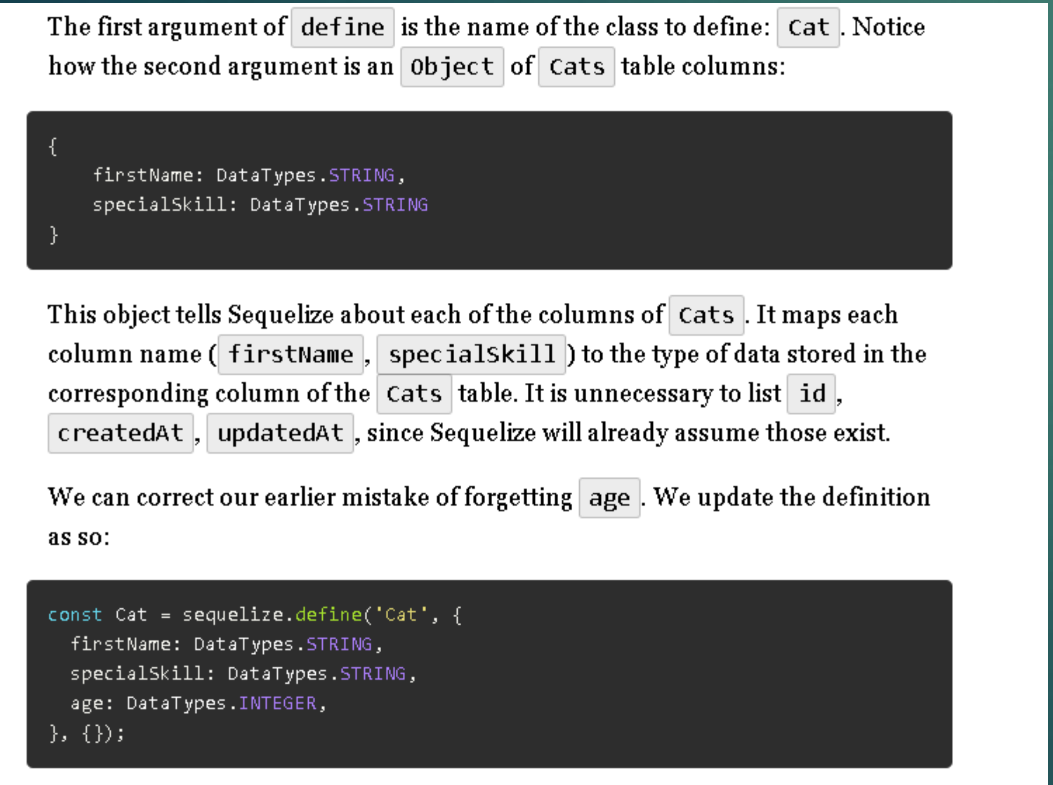
We do not need to list id, createdAt, or updatedAt. Sequelize will always presume those exist. Notice that we have forgotten to list age:integer



Examining (And Modifying) A Sequelize Model File

Let us examine the generated `./models/cat.js` file:

```
// ./models/cat.js

'use strict';
module.exports = (sequelize, DataTypes) => {
  const Cat = sequelize.define('Cat', {
    firstName: DataTypes.STRING,
    specialSkill: DataTypes.STRING
  }, {});
  Cat.associate = function(models) {
    // associations can be defined here
  };
  return Cat;
};
```



The first argument of `define` is the name of the class to define: `Cat`. Notice how the second argument is an `Object` of `Cats` table columns:

```
{
    firstName: DataTypes.STRING,
    specialSkill: DataTypes.STRING
}
```

This object tells Sequelize about each of the columns of `Cats`. It maps each column name (`firstName`, `specialSkill`) to the type of data stored in the corresponding column of the `Cats` table. It is unnecessary to list `id`, `createdAt`, `updatedAt`, since Sequelize will already assume those exist.

We can correct our earlier mistake of forgetting `age`. We update the definition as so:

```
const Cat = sequelize.define('Cat', {
  firstName: DataTypes.STRING,
  specialSkill: DataTypes.STRING,
  age: DataTypes.INTEGER,
}, {});
```

This file exports a function that defines a Cat class. When you use Sequelize to query the Cats table, each row retrieved will be transformed by Sequelize into an instance of the Cat class. A JavaScript class like Cat that corresponds to a SQL table is called a model class.

The ./models/cat.js will not be loaded by us directly. Sequelize will load this file and call the exported function to define the Cat class. The exported function uses Sequelize's define method to auto-generate a new class (called Cat).

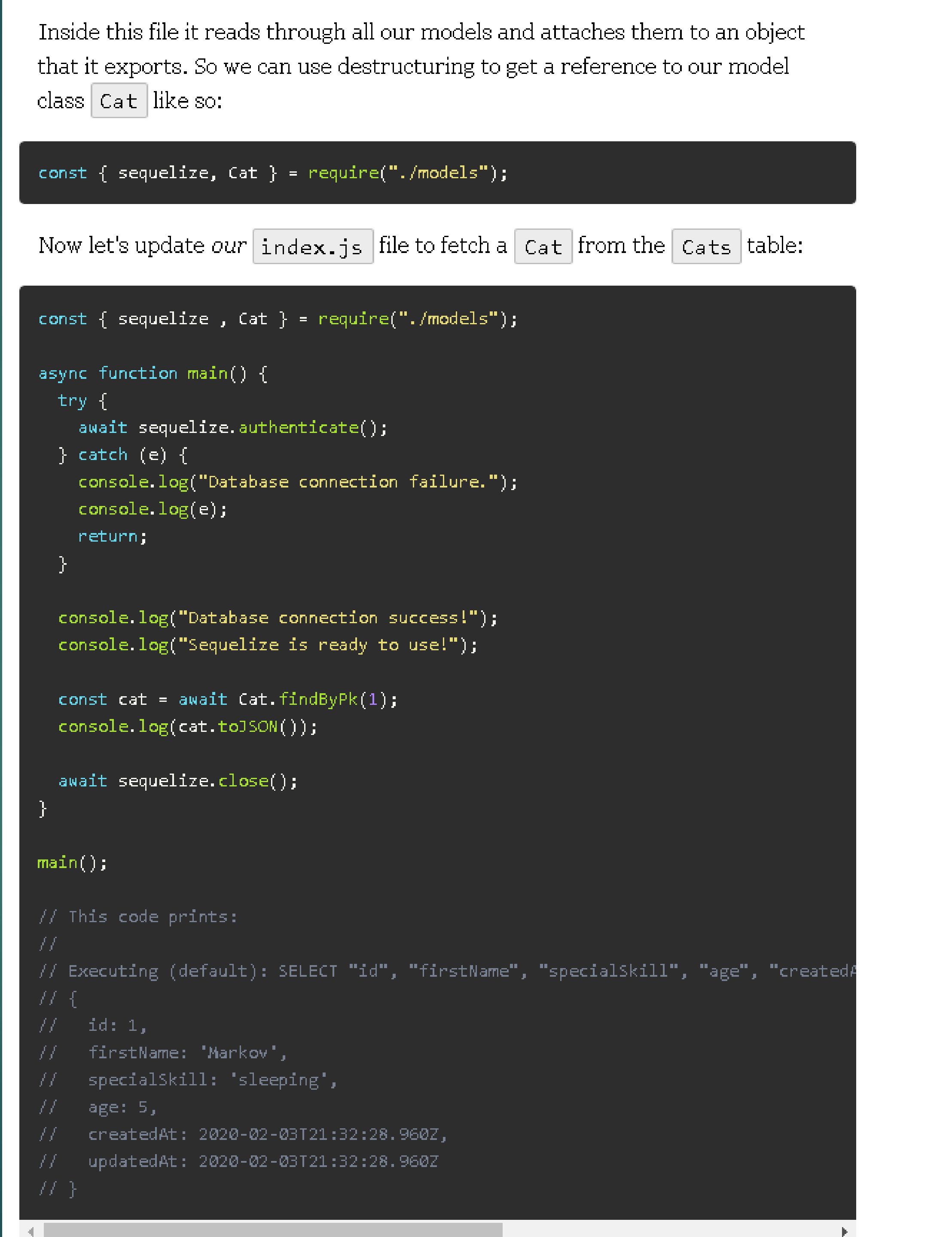
Note: You may notice we aren't using the JavaScript's class keyword to define the Cat class. With Sequelize, it is going to do all that for us with the define method. This is because Sequelize was around way before the class keyword was added to JavaScript. It is possible to use the class keyword with Sequelize,

Using The Cat Model To Fetch And Update SQL Data

We are now ready to use our Cat model class. When Sequelize defines the Cat class, it will generate instance and class methods needed to interact with the Cats SQL table.

As we mentioned before we don't require our cats.js file directly. Instead we require ./models which loads the file ./models/index.js.

Inside this file it reads through all our models and attaches them to an object that it exports. So we can use destructuring to get a reference to our model class Cat like so:

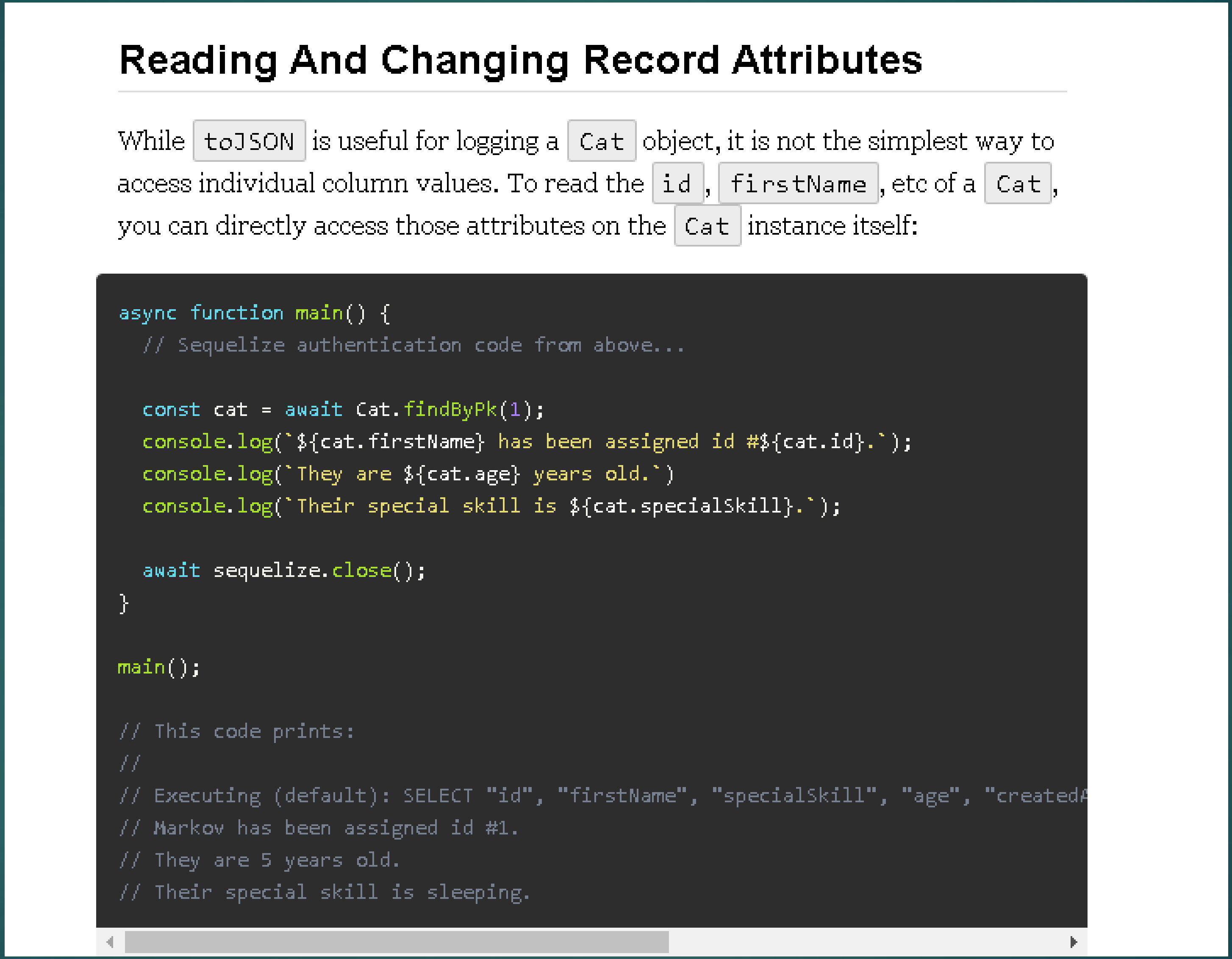


Inside this file it reads through all our models and attaches them to an object that it exports. So we can use destructuring to get a reference to our model class `Cat` like so:

```
const { sequelize, Cat } = require("./models");
```

Now let's update *our* `index.js` file to fetch a `Cat` from the `Cats` table:

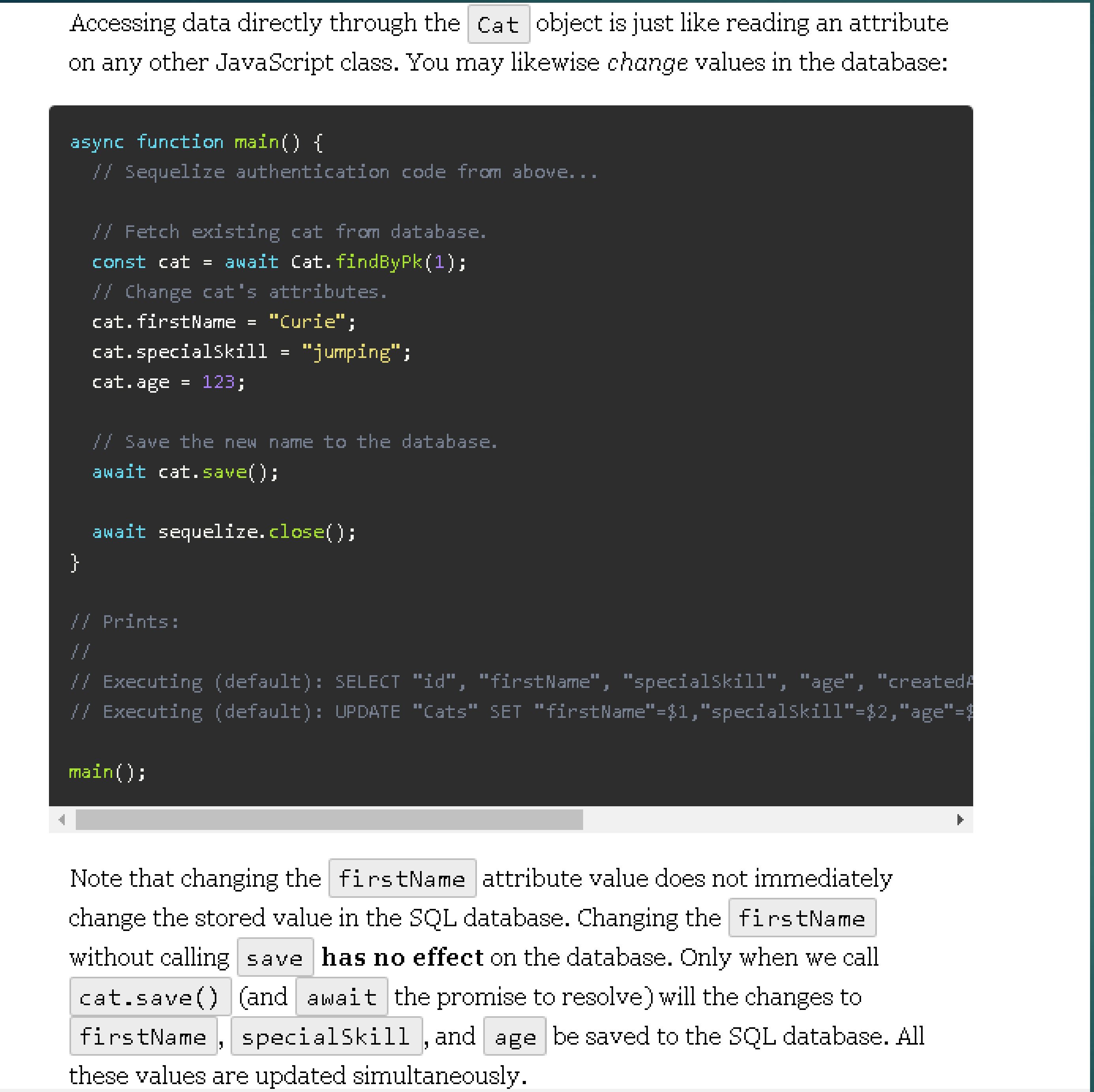
```
const { sequelize , Cat } = require("./models");

async function main() {
  try {
    await sequelize.authenticate();
  } catch (e) {
    console.log("Database connection failure.");
    console.log(e);
    return;
  }

  console.log("Database connection success!");
  console.log("Sequelize is ready to use!");

  const cat = await Cat.findByPk(1);
  console.log(cat.toJSON());

  await sequelize.close();
}

main();

// This code prints:
//
// Executing (default): SELECT "id", "firstName", "specialSkill", "age", "createdA
// {
//   id: 1,
//   firstName: 'Markov',
//   specialSkill: 'sleeping',
//   age: 5,
//   createdAt: 2020-02-03T21:32:28.960Z,
//   updatedAt: 2020-02-03T21:32:28.960Z
// }
```



Reading And Changing Record Attributes

While `toJSON` is useful for logging a `Cat` object, it is not the simplest way to access individual column values. To read the `id`, `firstName`, etc of a `Cat`, you can directly access those attributes on the `Cat` instance itself:

```
async function main() {
  // Sequelize authentication code from above...

  const cat = await Cat.findByPk(1);
  console.log(`${cat.firstName} has been assigned id #${cat.id}.`);
  console.log(`They are ${cat.age} years old.`)
  console.log(`Their special skill is ${cat.specialSkill}.`);

  await sequelize.close();
}

main();

// This code prints:
//
// Executing (default): SELECT "id", "firstName", "specialSkill", "age", "createdA
// Markov has been assigned id #1.
// They are 5 years old.
// Their special skill is sleeping.
```



Accessing data directly through the `Cat` object is just like reading an attribute on any other JavaScript class. You may likewise *change* values in the database:

```
async function main() {
  // Sequelize authentication code from above...

  // Fetch existing cat from database.
  const cat = await Cat.findByPk(1);
  // Change cat's attributes.
  cat.firstName = "Curie";
  cat.specialSkill = "jumping";
  cat.age = 123;

  // Save the new name to the database.
  await cat.save();

  await sequelize.close();
}

// Prints:
//
// Executing (default): SELECT "id", "firstName", "specialSkill", "age", "createdA
// Executing (default): UPDATE "Cats" SET "firstName"=$1,"specialSkill"=$2,"age"=$

main();
```

Note that changing the `firstName` attribute value does not immediately change the stored value in the SQL database. Changing the `firstName` without calling `save` **has no effect** on the database. Only when we call `cat.save()` (and `await` the promise to resolve) will the changes to `firstName`, `specialSkill`, and `age` be saved to the SQL database. All these values are updated simultaneously.

Database Migrations

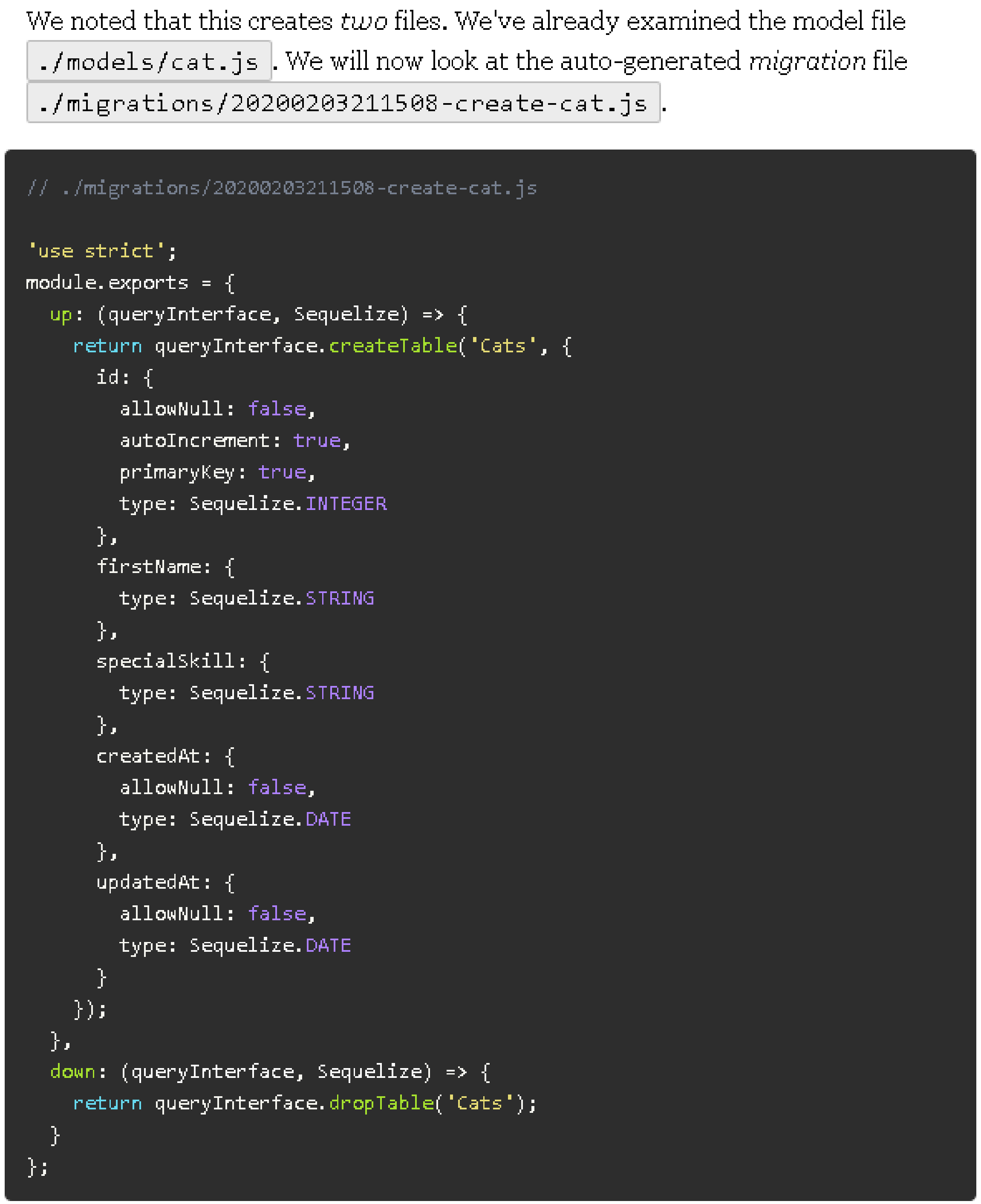
Sequelize Migration Files

In the prior reading we assumed that a Cats table already existed in our catsdb database. In this reading, we will presume that the Cats table does not exist, and that we have to create the table ourselves. This is the typical case when you aren't merely interacting with a preexisting database. When you develop your own application, the database will start out empty and with a blank schema.

We previously used the Sequelize CLI tool to autogenerate a Cat model file like so:

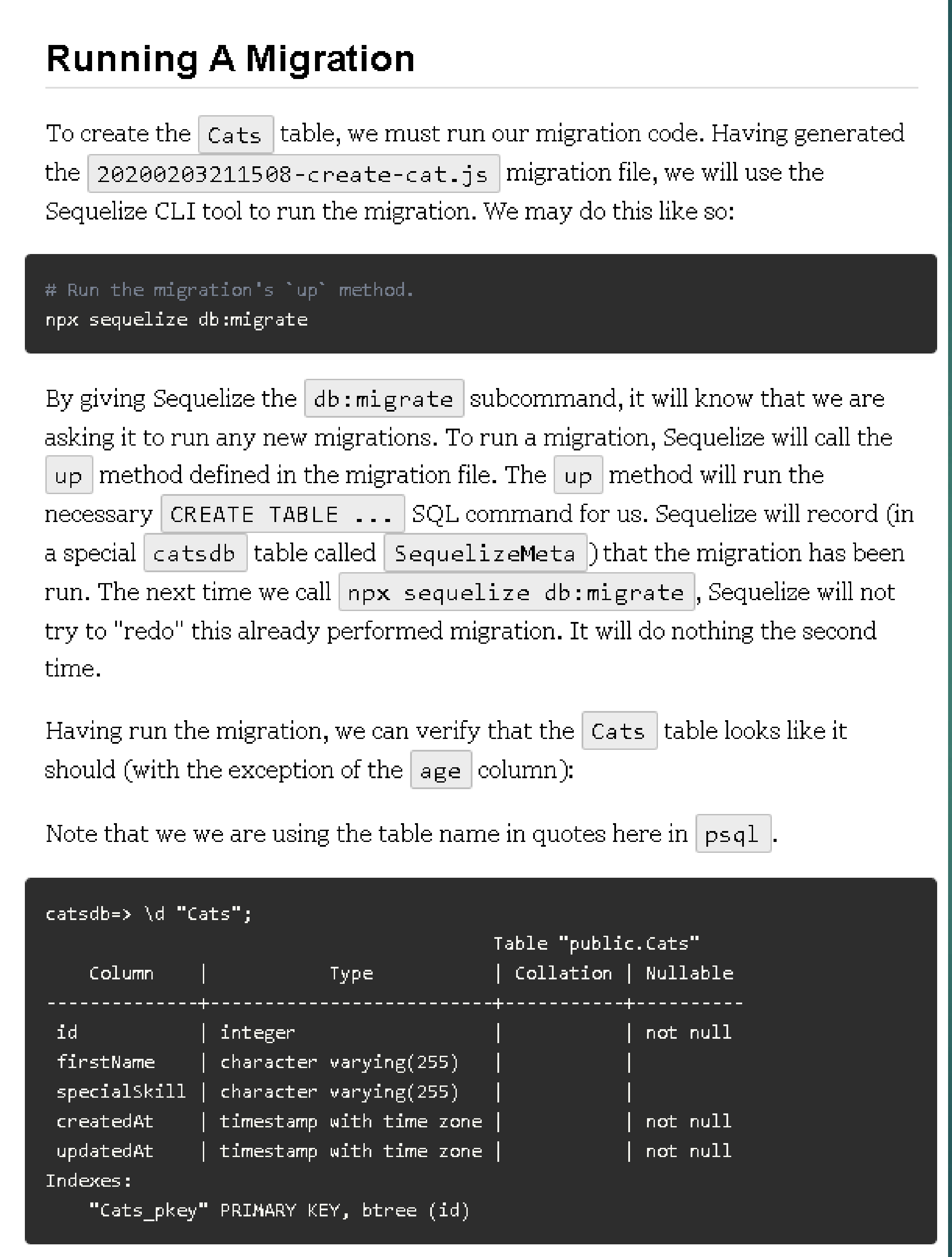
```
# Oops, forgot age:integer!
npx sequelize model:generate --name Cat --attributes "firstName:string,specialSkill:string"
```

We noted that this creates two files. We've already examined the model file ./models/cat.js. We will now look at the auto-generated migration file ./migrations/20200203211508-create-cat.js.



We noted that this creates *two* files. We've already examined the model file `./models/cat.js`. We will now look at the auto-generated *migration* file `./migrations/20200203211508-create-cat.js`.

```javascript
// ./migrations/20200203211508-create-cat.js

'use strict';
module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('Cats', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      firstName: {
        type: Sequelize.STRING
      },
      specialSkill: {
        type: Sequelize.STRING
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE
      }
    });
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('Cats');
  }
};
```



Running A Migration

To create the `Cats` table, we must run our migration code. Having generated the `20200203211508-create-cat.js` migration file, we will use the Sequelize CLI tool to run the migration. We may do this like so:

```
# Run the migration's `up` method.
npx sequelize db:migrate
```

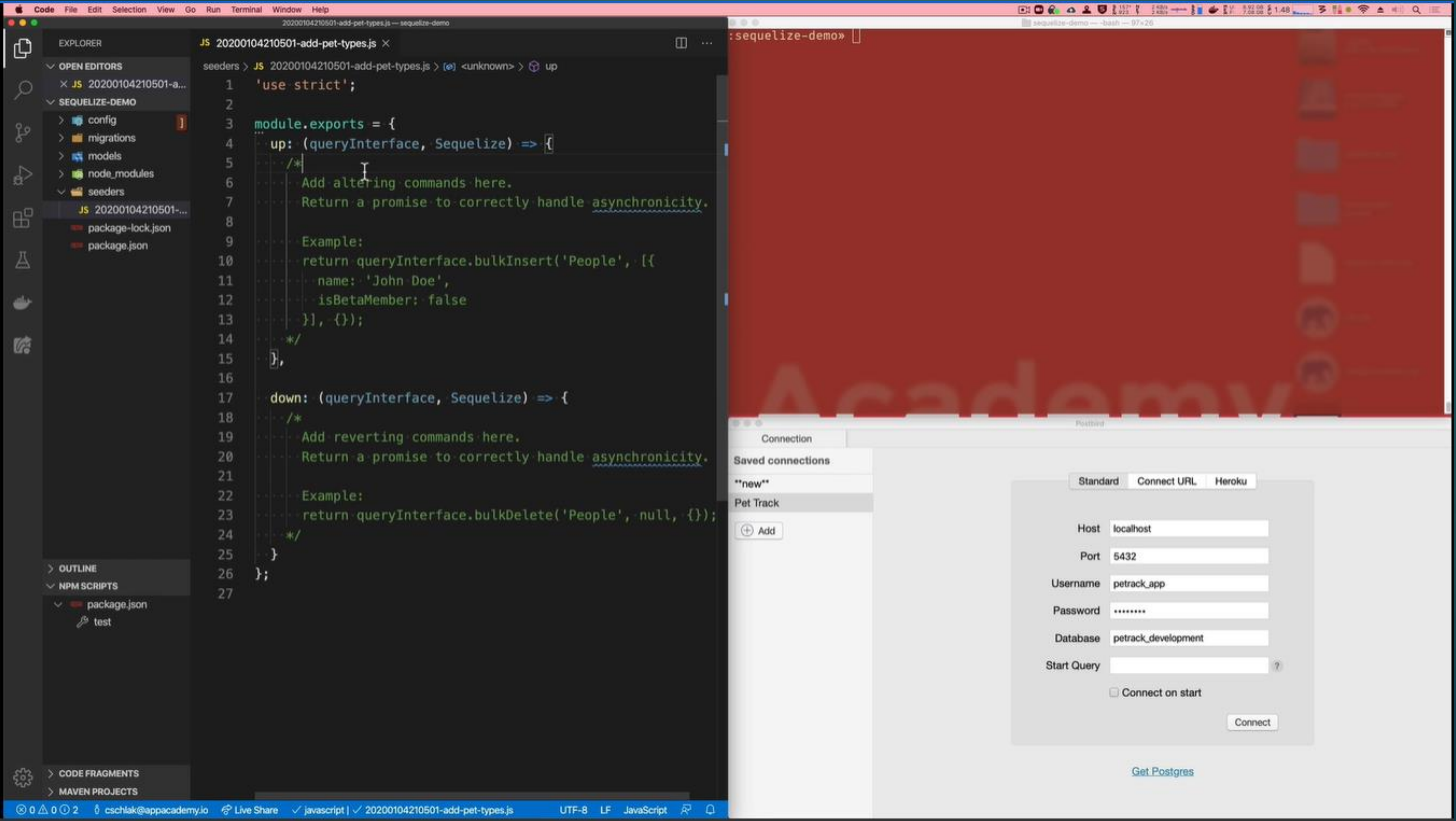
By giving Sequelize the `db:migrate` subcommand, it will know that we are asking it to run any new migrations. To run a migration, Sequelize will call the `up` method defined in the migration file. The `up` method will run the necessary `CREATE TABLE ...` SQL command for us. Sequelize will record (in a special `catsdb` table called `SequelizeMeta`) that the migration has been run. The next time we call `npx sequelize db:migrate`, Sequelize will not try to "redo" this already performed migration. It will do nothing the second time.

Having run the migration, we can verify that the `Cats` table looks like it should (with the exception of the `age` column):

Note that we we are using the table name in quotes here in `psql`.

```
catsdb=> \d "Cats";
                                         Table "public.Cats"
    Column    |           Type           | Collation | Nullable
--------------+--------------------------+-----------+----------
 id           | integer                  |           | not null
 firstName    | character varying(255)   |           |
 specialSkill | character varying(255)   |           |
 createdAt    | timestamp with time zone |           | not null
 updatedAt    | timestamp with time zone |           | not null
Indexes:
    "Cats_pkey" PRIMARY KEY, btree (id)
```

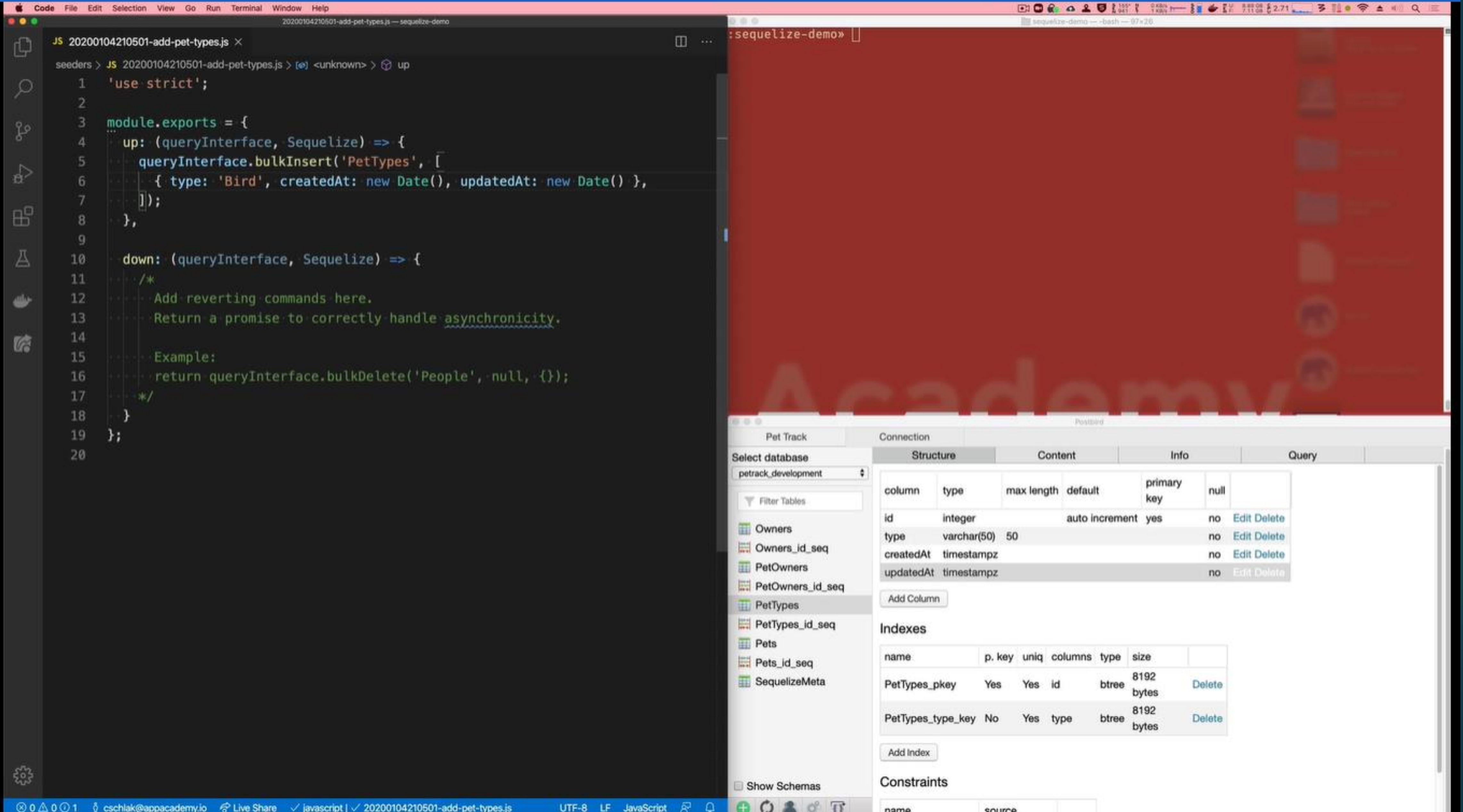
Inserting Data With Sequelize



```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    /*
      Add altering commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.bulkInsert('People', [{
        name: 'John Doe',
        isBetaMember: false
      }], {});
    */
  },

  down: (queryInterface, Sequelize) => {
    /*
      Add reverting commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.bulkDelete('People', null, {});
    */
  }
};
```

```
:sequelize-demo»
```

Host	localhost
Port	5432
Username	petrack_app
Password	••••••••
Database	petrack_development
Start Query	


Code File Edit Selection View Go Run Terminal Window Help
20200104210501-add-pet-types.js
seeders > JS 20200104210501-add-pet-types.js > <unknown> > up

```javascript
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    queryInterface.bulkInsert('PetTypes', [
      { type: 'Bird', createdAt: new Date(), updatedAt: new Date() },
    ]);
  },

  down: (queryInterface, Sequelize) => {
    /*
      Add reverting commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.bulkDelete('People', null, {});
    */
  }
};
```

```
:sequelize-demo»
```

column	type	max length	default	primary key	null	
id	integer		auto increment	yes	no	Edit Delete
type	varchar(50)	50			no	Edit Delete
createdAt	timestampz				no	Edit Delete
updatedAt	timestampz				no	Edit Delete

Add Column

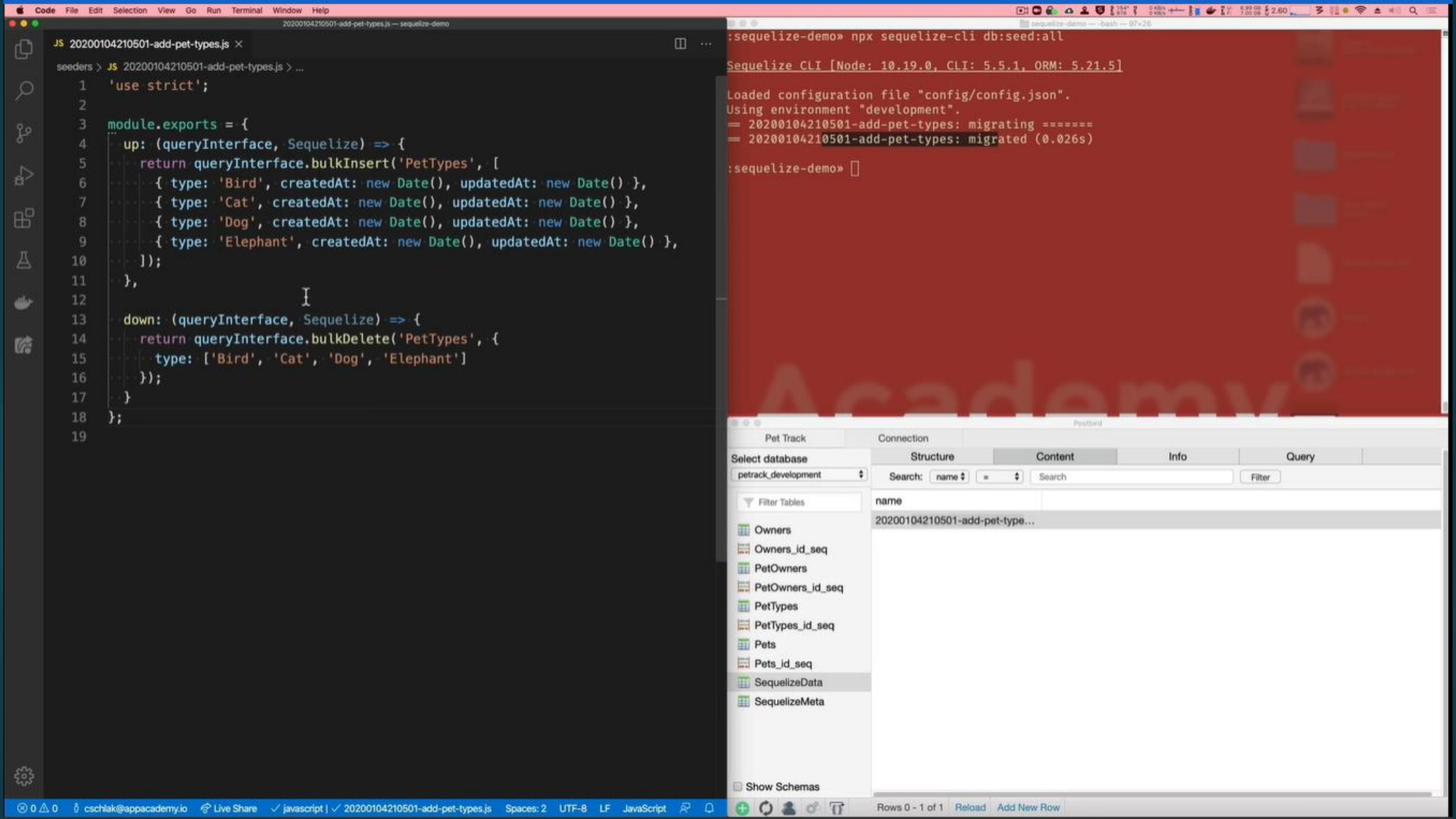
Indexes

name	p. key	uniq	columns	type	size	
PetTypes_pkey	Yes	Yes	id	btree	8192 bytes	Delete
PetTypes_type_key	No	Yes	type	btree	8192 bytes	Delete

Add Index

Constraints

⊗ 0 ⚠ 0 ⓘ 1 cschlak@appacademy.io Live Share ✓ javascript | ✓ 20200104210501-add-pet-types.js UTF-8 LF JavaScript



```javascript
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.bulkInsert('PetTypes', [
      { type: 'Bird', createdAt: new Date(), updatedAt: new Date() },
      { type: 'Cat', createdAt: new Date(), updatedAt: new Date() },
      { type: 'Dog', createdAt: new Date(), updatedAt: new Date() },
      { type: 'Elephant', createdAt: new Date(), updatedAt: new Date() },
    ]);
  },

  down: (queryInterface, Sequelize) => {
    return queryInterface.bulkDelete('PetTypes', {
      type: ['Bird', 'Cat', 'Dog', 'Elephant']
    });
  }
};
```

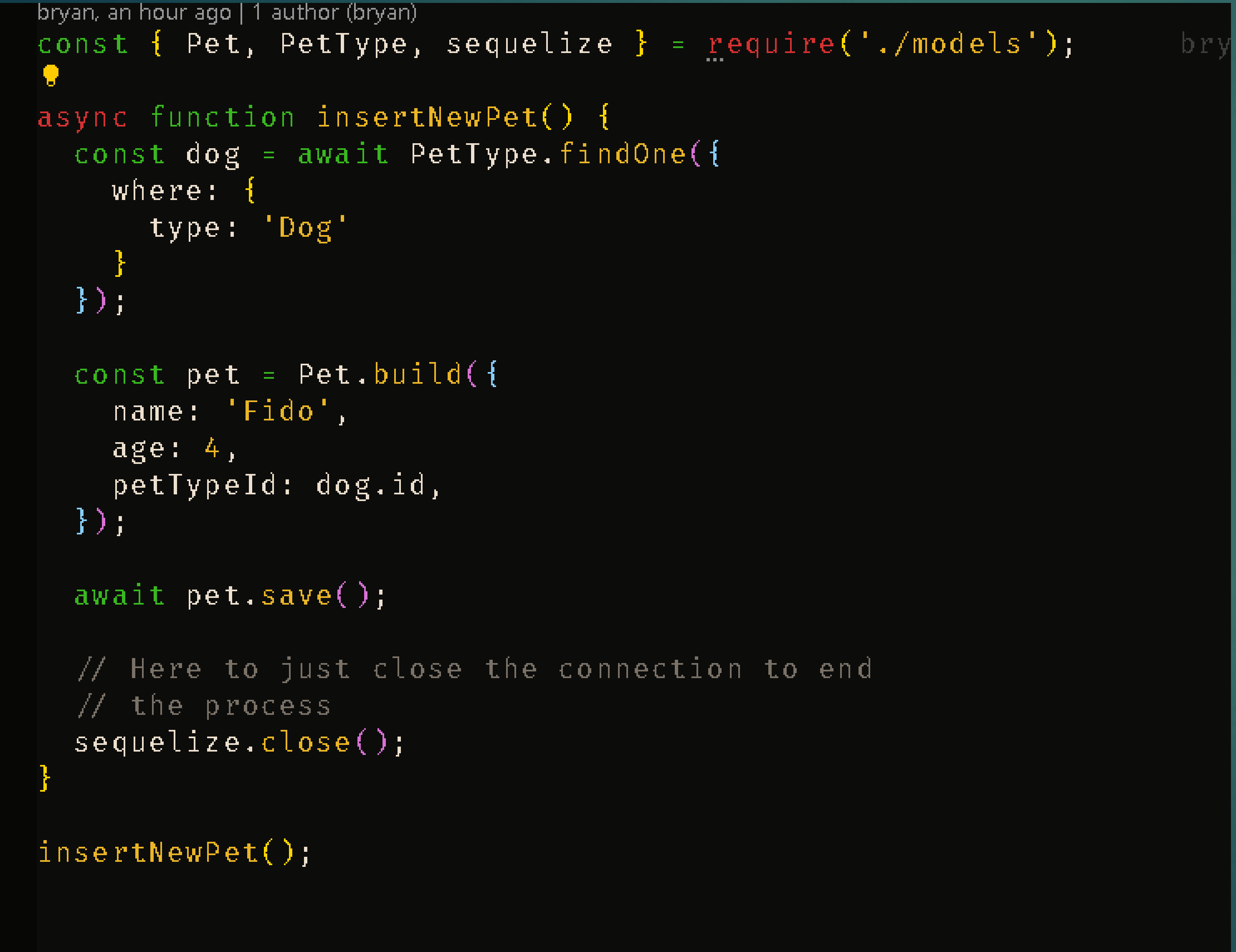
```
:sequelize-demo» npx sequelize-cli db:seed:all

Sequelize CLI [Node: 10.19.0, CLI: 5.5.1, ORM: 5.21.5]

Loaded configuration file "config/config.json".
Using environment "development".
== 20200104210501-add-pet-types: migrating =======
== 20200104210501-add-pet-types: migrated (0.026s)

:sequelize-demo» 
```

Select database: petrack_development

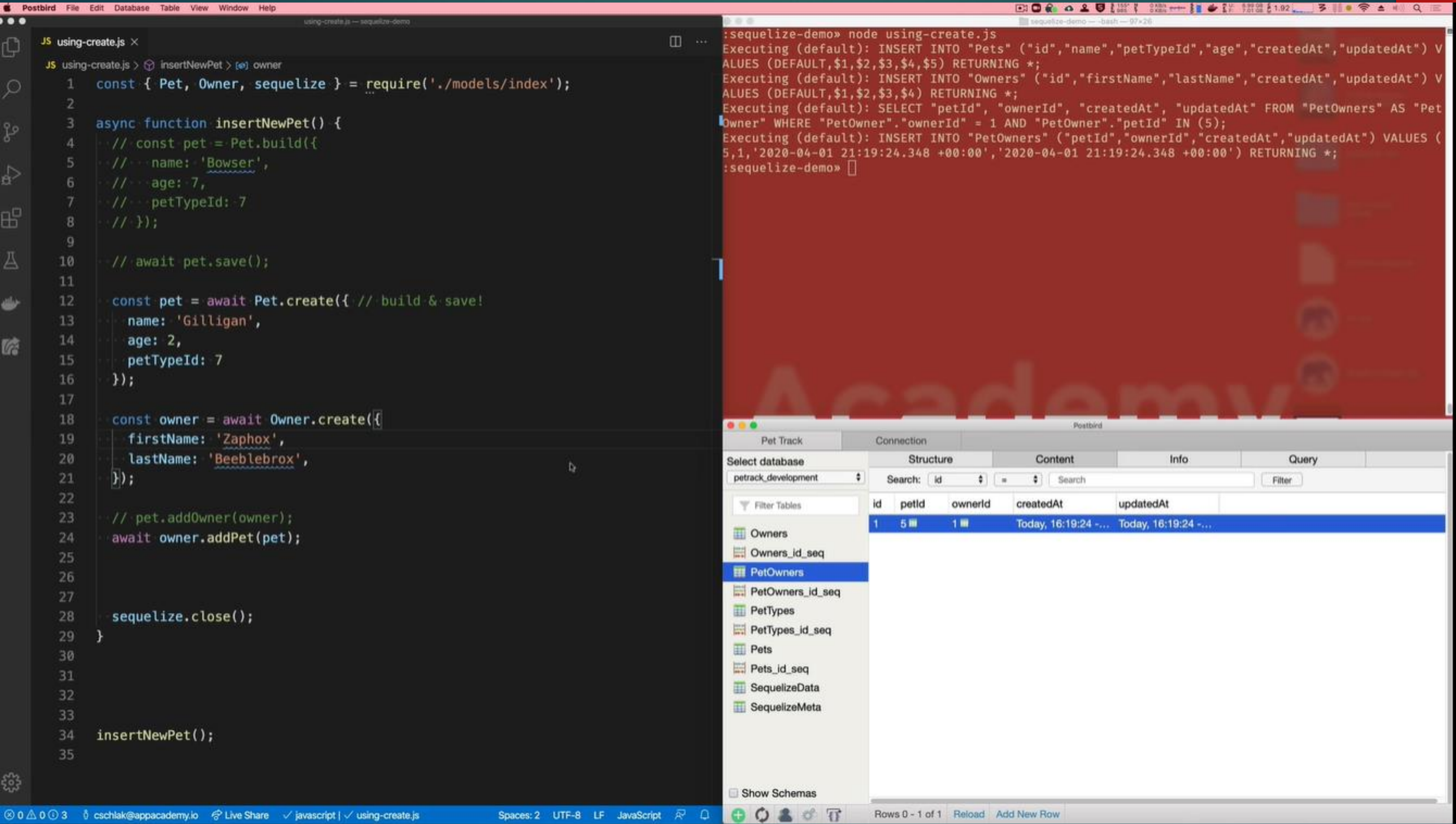
Tables: Owners, Owners_id_seq, PetOwners, PetOwners_id_seq, PetTypes, PetTypes_id_seq, Pets, Pets_id_seq, SequelizeData, SequelizeMeta

name
20200104210501-add-pet-type...

Rows 0 - 1 of 1


bryan, an hour ago | 1 author (bryan)

```
const { Pet, PetType, sequelize } = require('./models');      bry

async function insertNewPet() {
  const dog = await PetType.findOne({
    where: {
      type: 'Dog'
    }
  });

  const pet = Pet.build({
    name: 'Fido',
    age: 4,
    petTypeId: dog.id,
  });

  await pet.save();

  // Here to just close the connection to end
  // the process
  sequelize.close();
}

insertNewPet();
```



```javascript
const { Pet, Owner, sequelize } = require('./models/index');

async function insertNewPet() {
  // const pet = Pet.build({
  //   name: 'Bowser',
  //   age: 7,
  //   petTypeId: 7
  // });

  // await pet.save();

  const pet = await Pet.create({ // build & save!
    name: 'Gilligan',
    age: 2,
    petTypeId: 7
  });

  const owner = await Owner.create({
    firstName: 'Zaphox',
    lastName: 'Beeblebrox',
  });

  // pet.addOwner(owner);
  await owner.addPet(pet);



  sequelize.close();
}




insertNewPet();
```

```
:sequelize-demo» node using-create.js
Executing (default): INSERT INTO "Pets" ("id","name","petTypeId","age","createdAt","updatedAt") V
ALUES (DEFAULT,$1,$2,$3,$4,$5) RETURNING *;
Executing (default): INSERT INTO "Owners" ("id","firstName","lastName","createdAt","updatedAt") V
ALUES (DEFAULT,$1,$2,$3,$4) RETURNING *;
Executing (default): SELECT "petId", "ownerId", "createdAt", "updatedAt" FROM "PetOwners" AS "Pet
Owner" WHERE "PetOwner"."ownerId" = 1 AND "PetOwner"."petId" IN (5);
Executing (default): INSERT INTO "PetOwners" ("petId","ownerId","createdAt","updatedAt") VALUES (
5,1,'2020-04-01 21:19:24.348 +00:00','2020-04-01 21:19:24.348 +00:00') RETURNING *;
:sequelize-demo» 
```

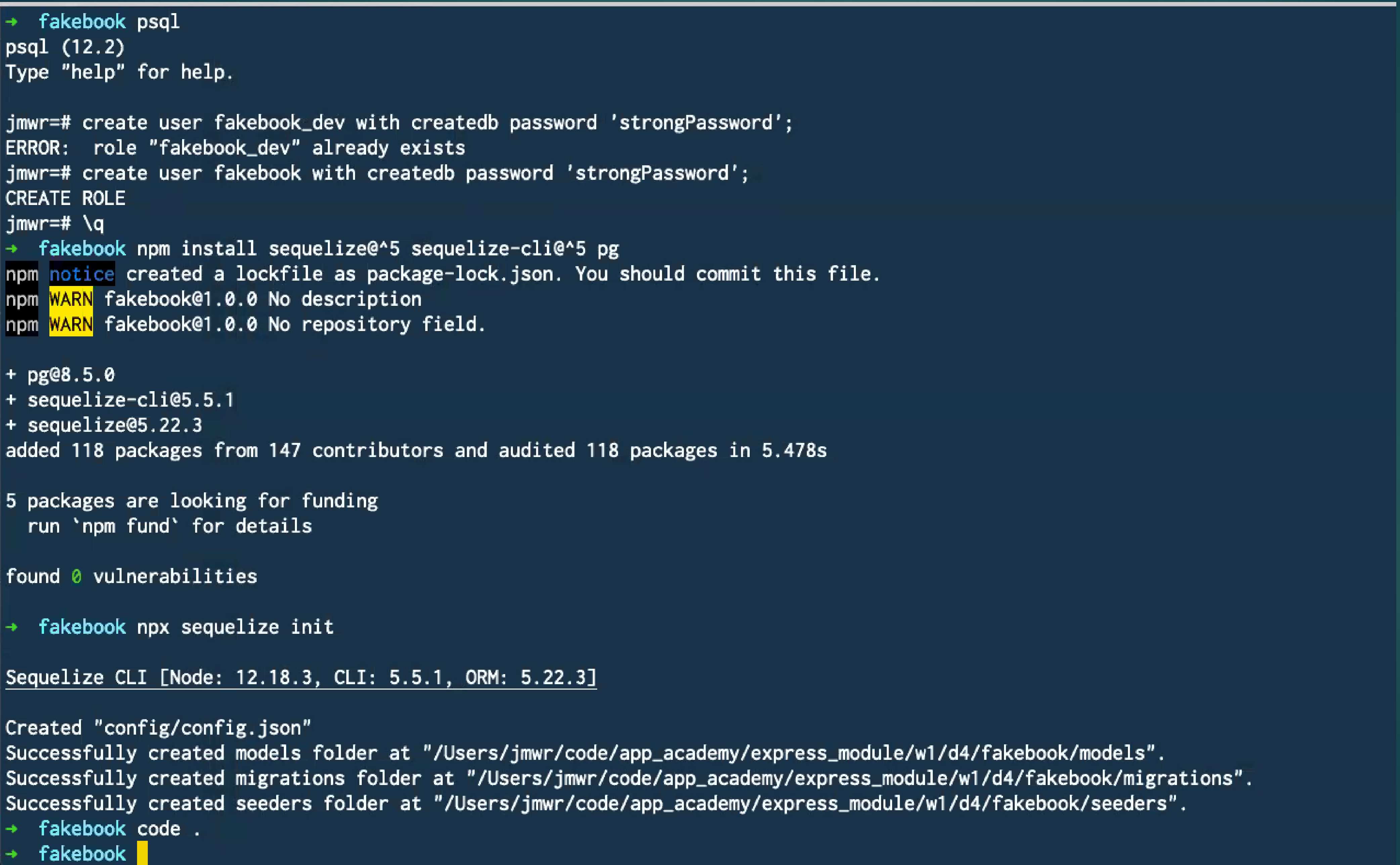
Select database: petrack_development

Owners
Owners_id_seq
PetOwners
PetOwners_id_seq
PetTypes
PetTypes_id_seq
Pets
Pets_id_seq
SequelizeData
SequelizeMeta

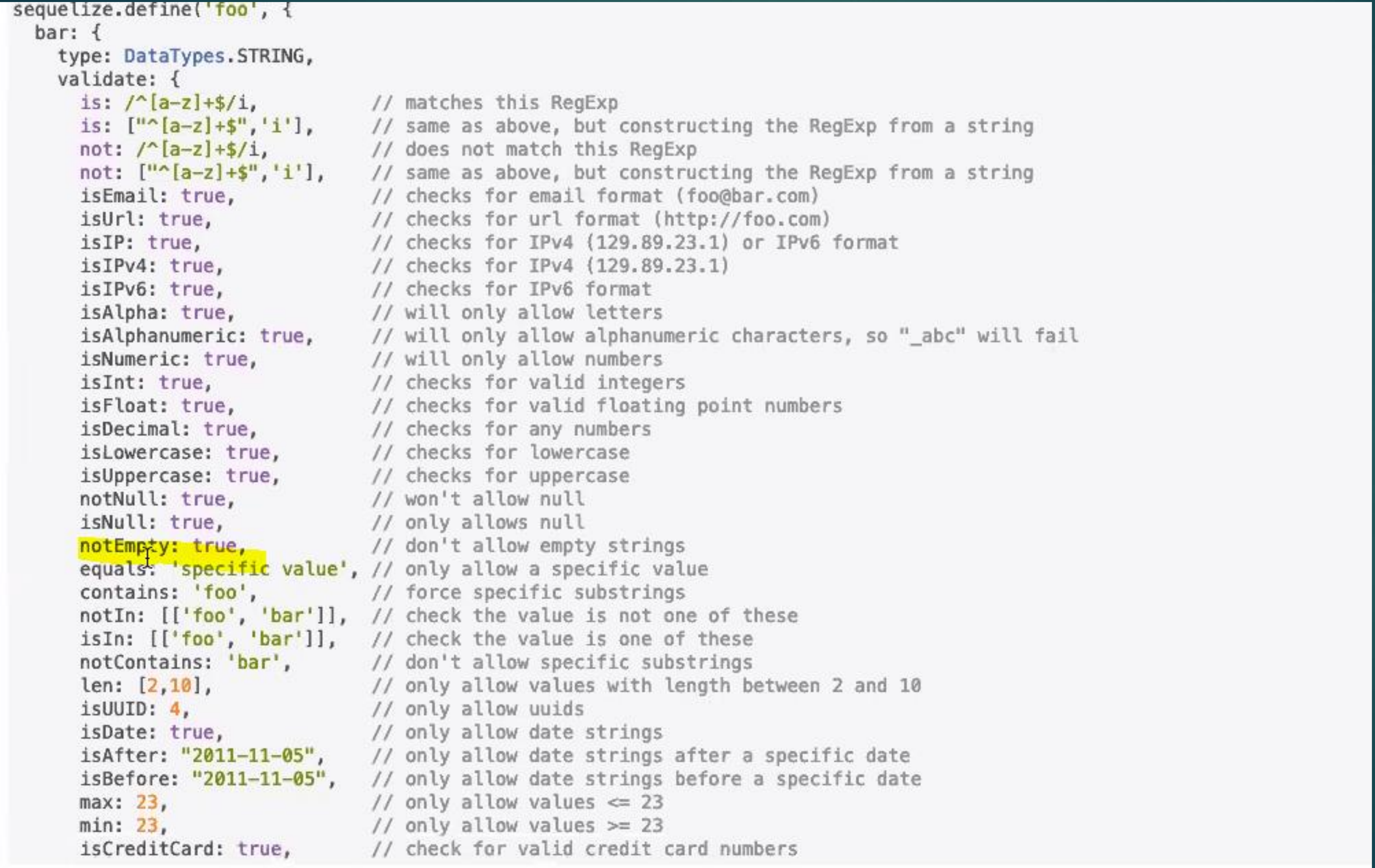
id	petId	ownerId	createdAt	updatedAt
1	5	1	Today, 16:19:24 -...	Today, 16:19:24 -...

Rows 0 - 1 of 1

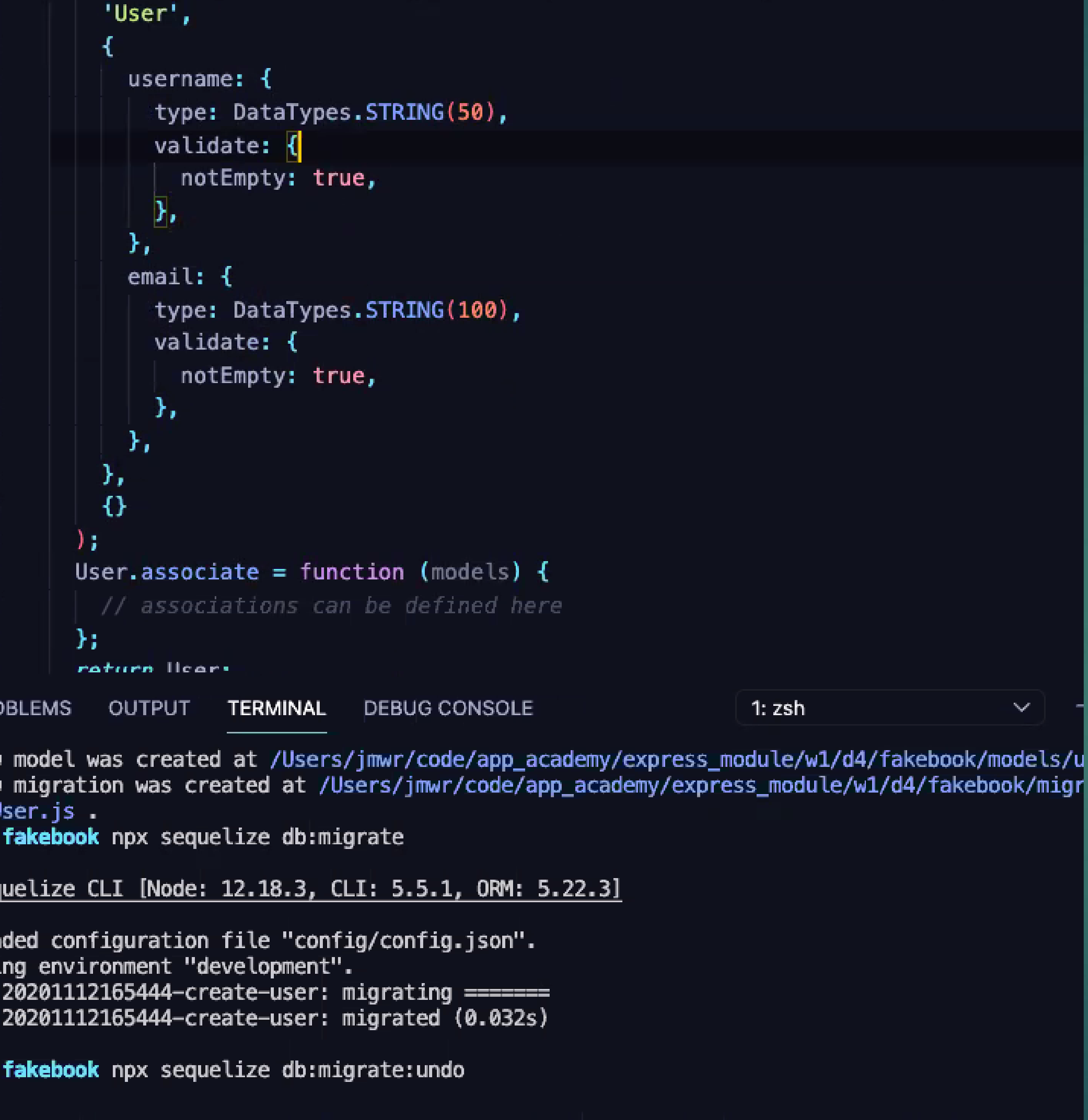
UPDATING DATA



```
➜  fakebook psql
psql (12.2)
Type "help" for help.

jmwr=# create user fakebook_dev with createdb password 'strongPassword';
ERROR:  role "fakebook_dev" already exists
jmwr=# create user fakebook with createdb password 'strongPassword';
CREATE ROLE
jmwr=# \q
➜  fakebook npm install sequelize@^5 sequelize-cli@^5 pg
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN fakebook@1.0.0 No description
npm WARN fakebook@1.0.0 No repository field.

+ pg@8.5.0
+ sequelize-cli@5.5.1
+ sequelize@5.22.3
added 118 packages from 147 contributors and audited 118 packages in 5.478s

5 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

➜  fakebook npx sequelize init

Sequelize CLI [Node: 12.18.3, CLI: 5.5.1, ORM: 5.22.3]

Created "config/config.json"
Successfully created models folder at "/Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/models".
Successfully created migrations folder at "/Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/migrations".
Successfully created seeders folder at "/Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/seeders".
➜  fakebook code .
➜  fakebook 
```

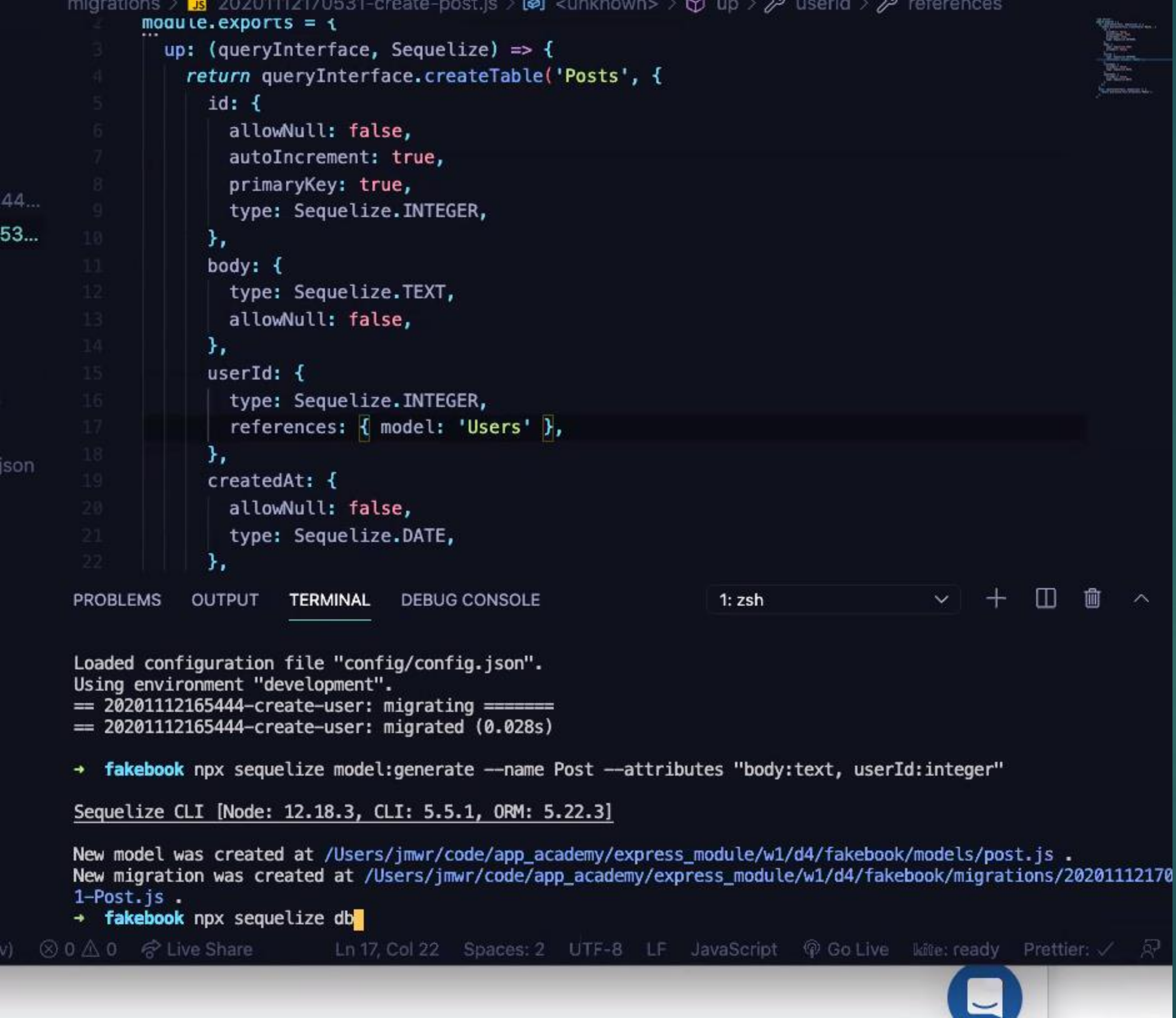


```
sequelize.define('foo', {
  bar: {
    type: DataTypes.STRING,
    validate: {
      is: /^[a-z]+$/i,          // matches this RegExp
      is: ["^[a-z]+$",'i'],     // same as above, but constructing the RegExp from a string
      not: /^[a-z]+$/i,         // does not match this RegExp
      not: ["^[a-z]+$",'i'],    // same as above, but constructing the RegExp from a string
      isEmail: true,            // checks for email format (foo@bar.com)
      isUrl: true,              // checks for url format (http://foo.com)
      isIP: true,               // checks for IPv4 (129.89.23.1) or IPv6 format
      isIPv4: true,             // checks for IPv4 (129.89.23.1)
      isIPv6: true,             // checks for IPv6 format
      isAlpha: true,            // will only allow letters
      isAlphanumeric: true,     // will only allow alphanumeric characters, so "_abc" will fail
      isNumeric: true,          // will only allow numbers
      isInt: true,              // checks for valid integers
      isFloat: true,            // checks for valid floating point numbers
      isDecimal: true,          // checks for any numbers
      isLowercase: true,        // checks for lowercase
      isUppercase: true,        // checks for uppercase
      notNull: true,            // won't allow null
      isNull: true,             // only allows null
      notEmpty: true,           // don't allow empty strings
      equals: 'specific value', // only allow a specific value
      contains: 'foo',          // force specific substrings
      notIn: [['foo', 'bar']],  // check the value is not one of these
      isIn: [['foo', 'bar']],   // check the value is one of these
      notContains: 'bar',       // don't allow specific substrings
      len: [2,10],              // only allow values with length between 2 and 10
      isUUID: 4,                // only allow uuids
      isDate: true,             // only allow date strings
      isAfter: "2011-11-05",    // only allow date strings after a specific date
      isBefore: "2011-11-05",   // only allow date strings before a specific date
      max: 23,                  // only allow values <= 23
      min: 23,                  // only allow values >= 23
      isCreditCard: true,       // check for valid credit card numbers
```



```
    'User',
    {
      username: {
        type: DataTypes.STRING(50),
        validate: {
          notEmpty: true,
        },
      },
      email: {
        type: DataTypes.STRING(100),
        validate: {
          notEmpty: true,
        },
      },
    },
    {}
  );
  User.associate = function (models) {
    // associations can be defined here
  };
  return User;
```

```
 model was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/models/u
 migration was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/migr
User.js .
 fakebook npx sequelize db:migrate

quelize CLI [Node: 12.18.3, CLI: 5.5.1, ORM: 5.22.3]

aded configuration file "config/config.json".
ing environment "development".
 20201112165444-create-user: migrating =======
 20201112165444-create-user: migrated (0.032s)

 fakebook npx sequelize db:migrate:undo
```

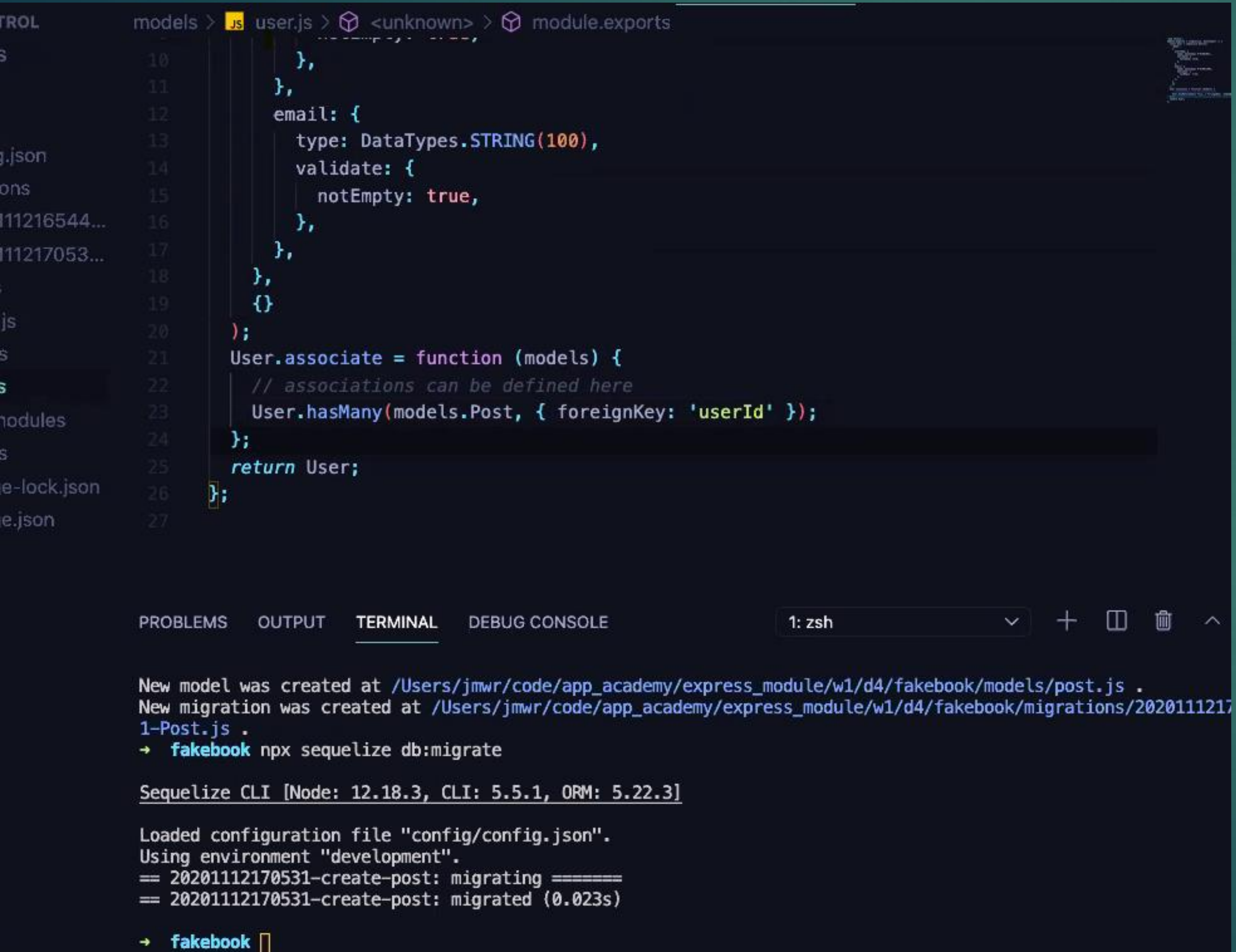

migrations > JS 20201112170531-create-post.js > <unknown> > up > userId > references

```javascript
module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('Posts', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER,
      },
      body: {
        type: Sequelize.TEXT,
        allowNull: false,
      },
      userId: {
        type: Sequelize.INTEGER,
        references: { model: 'Users' },
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
      },
```

PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
1: zsh

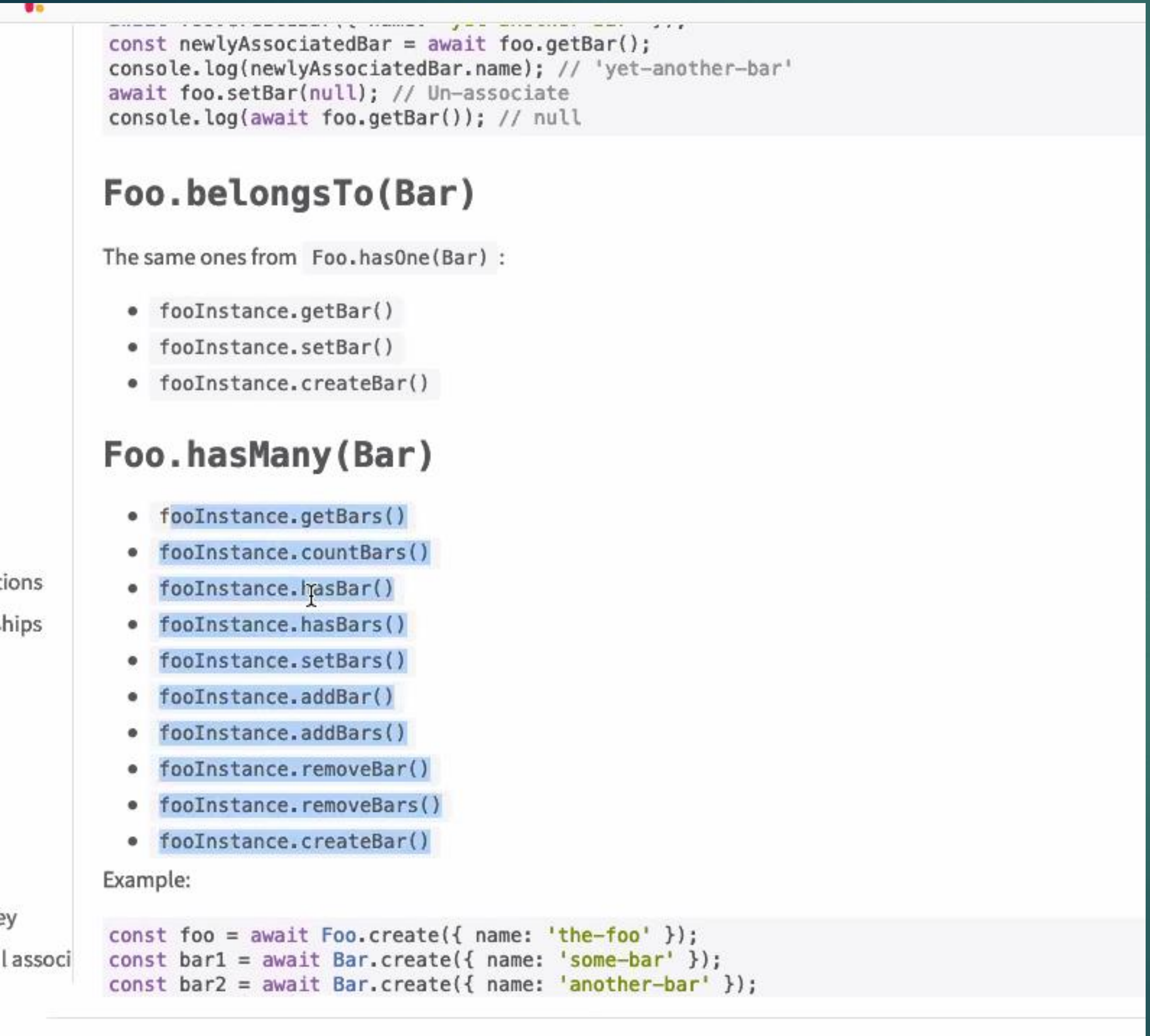
```
Loaded configuration file "config/config.json".
Using environment "development".
== 20201112165444-create-user: migrating =======
== 20201112165444-create-user: migrated (0.028s)

➜ fakebook npx sequelize model:generate --name Post --attributes "body:text, userId:integer"

Sequelize CLI [Node: 12.18.3, CLI: 5.5.1, ORM: 5.22.3]

New model was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/models/post.js .
New migration was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/migrations/20201112170
1-Post.js .
➜ fakebook npx sequelize db
```

⊗ 0 ⚠ 0 Live Share Ln 17, Col 22 Spaces: 2 UTF-8 LF JavaScript Go Live kite: ready Prettier: ✓



```
      },
    },
    email: {
      type: DataTypes.STRING(100),
      validate: {
        notEmpty: true,
      },
    },
  },
  {}
);
User.associate = function (models) {
  // associations can be defined here
  User.hasMany(models.Post, { foreignKey: 'userId' });
};
return User;
};
```

```
New model was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/models/post.js .
New migration was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/migrations/2020111217
1-Post.js .
➜  fakebook npx sequelize db:migrate

Sequelize CLI [Node: 12.18.3, CLI: 5.5.1, ORM: 5.22.3]

Loaded configuration file "config/config.json".
Using environment "development".
== 20201112170531-create-post: migrating =======
== 20201112170531-create-post: migrated (0.023s)

➜  fakebook
```



```
const newlyAssociatedBar = await foo.getBar();
console.log(newlyAssociatedBar.name); // 'yet-another-bar'
await foo.setBar(null); // Un-associate
console.log(await foo.getBar()); // null
```

Foo.belongsTo(Bar)

The same ones from `Foo.hasOne(Bar)` :

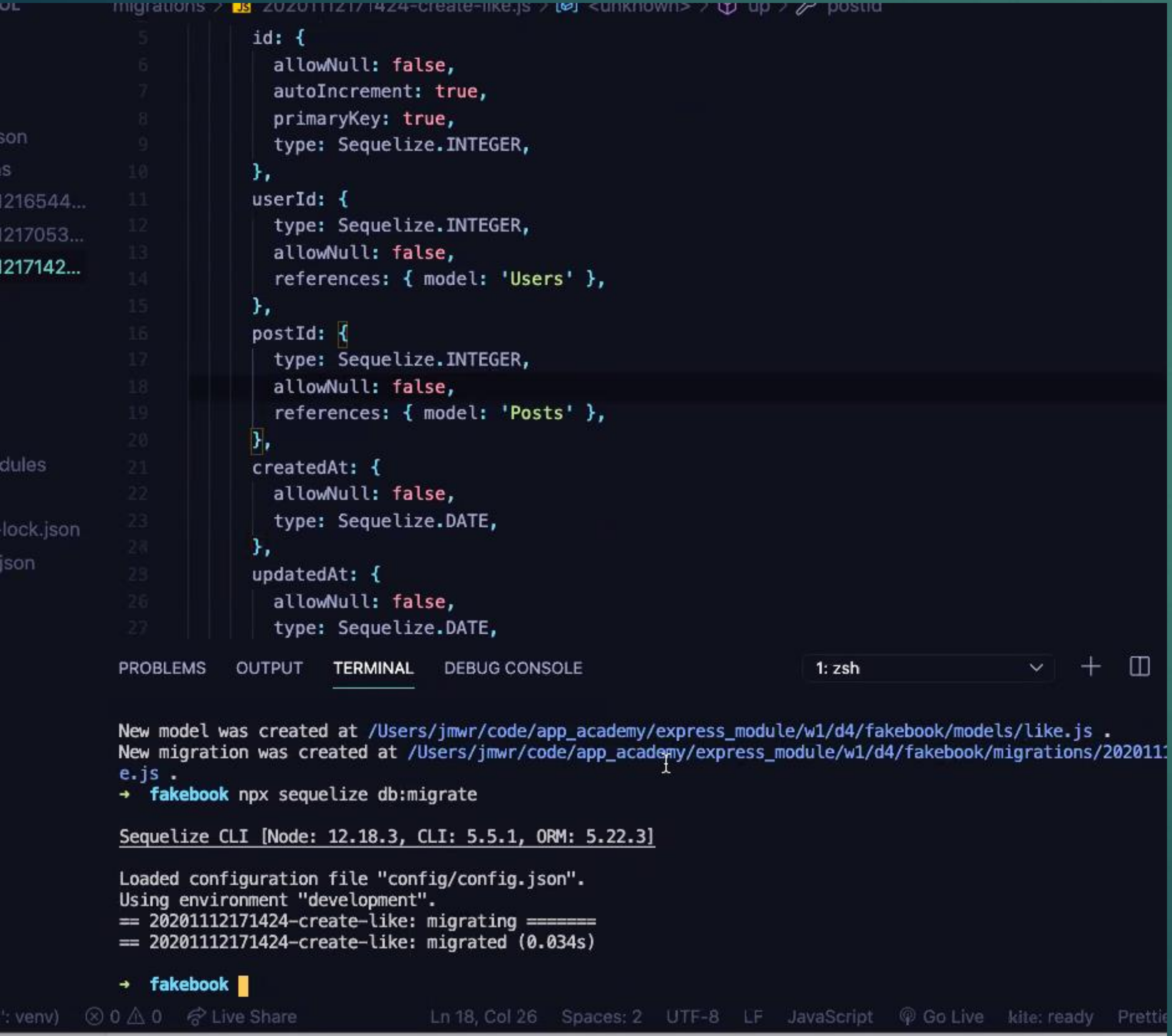
- `fooInstance.getBar()`
- `fooInstance.setBar()`
- `fooInstance.createBar()`

Foo.hasMany(Bar)

- `fooInstance.getBars()`
- `fooInstance.countBars()`
- `fooInstance.hasBar()`
- `fooInstance.hasBars()`
- `fooInstance.setBars()`
- `fooInstance.addBar()`
- `fooInstance.addBars()`
- `fooInstance.removeBar()`
- `fooInstance.removeBars()`
- `fooInstance.createBar()`

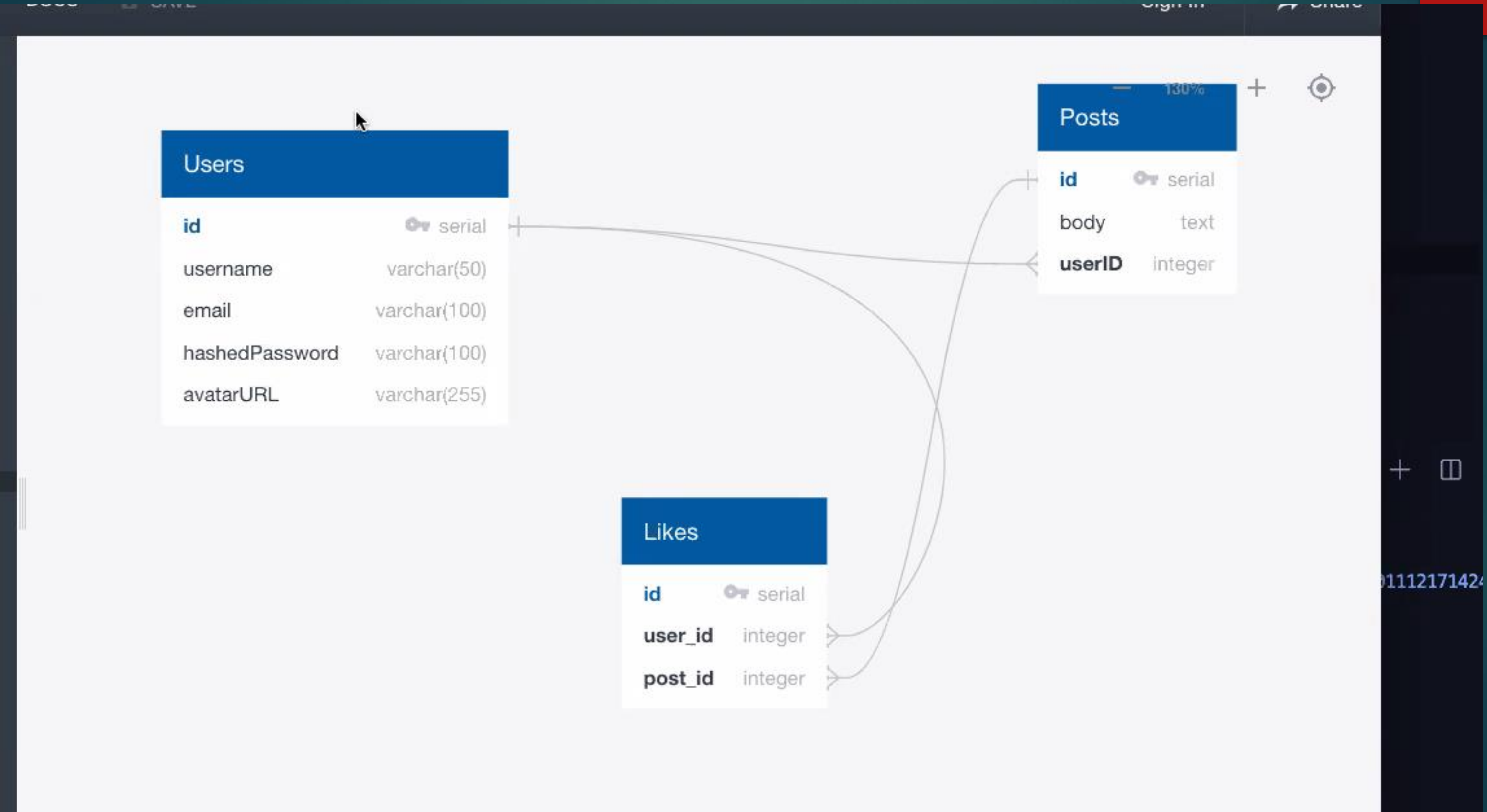
Example:

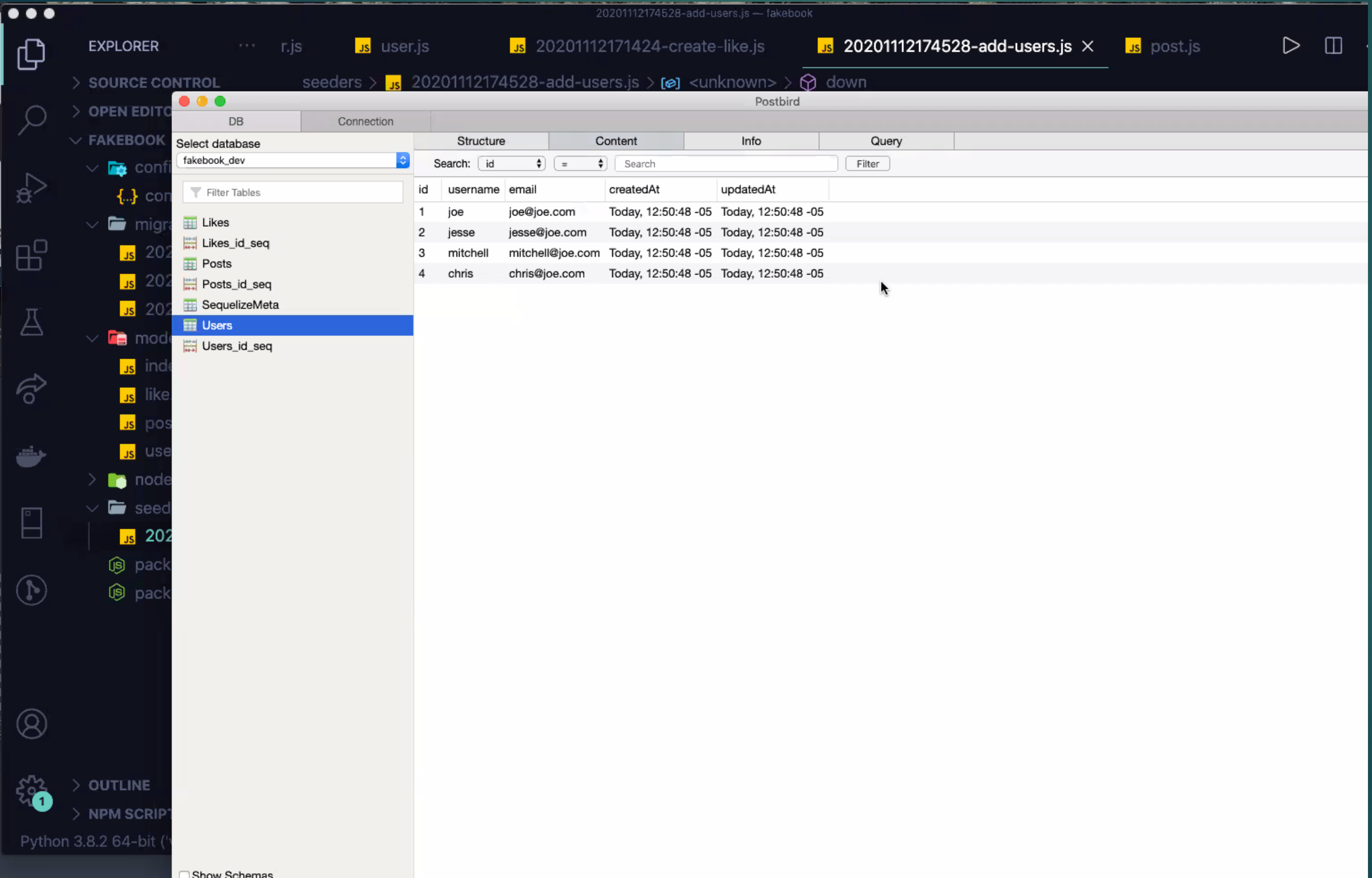
```
const foo = await Foo.create({ name: 'the-foo' });
const bar1 = await Bar.create({ name: 'some-bar' });
const bar2 = await Bar.create({ name: 'another-bar' });
```



```
migrations > 20201112171424-create-like.js > <unknown> > up > postId
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: Sequelize.INTEGER,
    },
    userId: {
      type: Sequelize.INTEGER,
      allowNull: false,
      references: { model: 'Users' },
    },
    postId: {
      type: Sequelize.INTEGER,
      allowNull: false,
      references: { model: 'Posts' },
    },
    createdAt: {
      allowNull: false,
      type: Sequelize.DATE,
    },
    updatedAt: {
      allowNull: false,
      type: Sequelize.DATE,
```

```
New model was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/models/like.js .
New migration was created at /Users/jmwr/code/app_academy/express_module/w1/d4/fakebook/migrations/202011
e.js .
→ fakebook npx sequelize db:migrate

Sequelize CLI [Node: 12.18.3, CLI: 5.5.1, ORM: 5.22.3]

Loaded configuration file "config/config.json".
Using environment "development".
== 20201112171424-create-like: migrating =======
== 20201112171424-create-like: migrated (0.034s)

→ fakebook
```


Users
id serial
username varchar(50)
email varchar(100)
hashedPassword varchar(100)
avatarURL varchar(255)
Posts
id serial
body text
userID integer
Likes
id serial
user_id integer
post_id integer
130%



20201112174528-add-users.js — fakebook

id	username	email	createdAt	updatedAt
1	joe	joe@joe.com	Today, 12:50:48 -05	Today, 12:50:48 -05
2	jesse	jesse@joe.com	Today, 12:50:48 -05	Today, 12:50:48 -05
3	mitchell	mitchell@joe.com	Today, 12:50:48 -05	Today, 12:50:48 -05
4	chris	chris@joe.com	Today, 12:50:48 -05	Today, 12:50:48 -05