Sequelize Reference Sheet

Command Line

Sequelize provides utilities for generating migrations, models, and seed files. They are exposed through the sequelize-cli command.

Initialize Project

First, create `.sequelizerc` file, which configures how Sequelize will initialize itself in your application.
`server/.sequelizerc`

```
const path = require("path");
module.exports = {
    config: path.resolve("config", "database.js"),
    "models-path": path.resolve("db", "models"),
    "seeders-path": path.resolve("db", "seeders"),
    "migrations-path": path.resolve("db", "migrations"),
};
```

Let's initialize sequelize by typing below in your terminal.

```
$ npx sequelize-cli init
```

Then, you must customize the config/database.js file to match your database settings to complete the initialization process.

server/config/database.js

```
module.exports = {
    development: {
        storage: process.env.DB * FILE,
        dialect: "sqlite",
        seederStorage: "sequelize",
        benchmark: true,
        logQueryParameters: true,
        typeValidation: true,
        // logging: false
    },
};
```

Create a `.env` file in order to store environment variables.

server/.env

```
DB_FILE=db/dev.db
```

We are ready to create our database. In your terminal run

```
npx dotenv sequelize db:migrate
```

If `dev.db` file got created in the `server/db` directory, that means your project is initialized.

Commonly Used Terminal Commands

Generate a migration

```
npx sequelize-cli migration:generate --name <ModelName>
```

Generate a model and its migration

```
npx sequelize-cli model:generate --name <ModelName> --attributes <column1>:<type>,
<column2>:<type>,...
```

Run pending migrations

```
npx dotenv sequelize-cli db:migrate
```

Rollback migration

```
npx dotenv sequelize-cli db:migrate:undo # rollback one migration npx
dotenv sequelize-cli db:migrate:undo:all # rollback all migrations
```

Generate a new seed file

```
npx sequelize-cli seed:generate --name <descriptiveName>
```

Run all pending seeds

```
npx dotenv sequelize-cli db:seed:all
```

Rollback seeds

```
npx dotenv sequelize-cli db:seed:undo # rollback one seed npx
dotenv sequelize-cli db:seed:undo:all # rollback all seeds
```

Migrations

Create Table (usually used in the `up()` method)

```
// This uses the short form for references
await queryInterface.createTable('TableName', {
    columnName: {
        type: Sequelize.<type>,
        allowNull: true // false,
        unique: true // false,
        references: { model: 'TableName' }, // plural form
    }
});
```

Delete Table (usually used in the `down()` function)

```
await queryInterface.dropTable("tableName");
```

continued on next page

Adding a column

```
await queryInteface.addColumn('TableName', 'columnName', {
    type: Sequelize.<type>,
    allowNull: true // false,
    unique: true // false,
    references: { model: 'TableName' }, // plural form
});
```

Removing a column

```
await queryInterface.removeColumn("TableName", "columnName");
```

Model Associations

One-to-One between `Student` and `Scholarship`

student.js

```
Student.hasOne(models.Scholarship, { foreignKey: "studentId" });
```

scholarship.js

```
Scholarship.belongsTo(models.Student, { foreignKey: "studentId" });
```

One-to-Many between `Student` and `Class`

student.js

```
Student.belongsTo(models.Class, { foreignKey: "classId" });
```

class.js

```
Class.hasMany(models.Student, { foreignKey: "classId" });
```

Many-to-Many between `Student` and `Lesson` through `StudentLesson`

student.js

```
const columnMapping = {
    through: 'StudentLesson', // This is the model name referencing the join
    table.
    otherKey: 'lessonId',
    foreignKey: 'studentId'
}

Student.belongsToMany(models.Lesson, columnMapping);
```

class.js

```
const columnMapping = {
    through: 'StudentLesson', // This is the model name referencing the join
    table.
    otherKey: 'studentId',
    foreignKey: 'lessonId'
}

Lesson.belongsToMany(models.Student, columnMapping);
```

CRUD Operations

Inserting a new item

```
// Way 1 - With build and save
const pet = Pet.build({
    name: "Fido",
    petTypeId: 1,
});

await pet.save();

// Way 2 - With create
const pet = await Pet.create({
    name: "Fido",
    petTypeId: 1,
});
```

Updating an item

```
// Find the pet with id = 1
const pet = await Pet.findByPk(1);

// Way 1
pet.name = "Fido, Sr.";
await pet.save;

// Way 2
await pet.update({
    name: "Fido, Sr.",
});
```

Deleting a single item

```
// Find the pet with id = 1
const pet = await Pet.findByPk(1);

// Notice this is an instance method
pet.destroy();
```

Deleting multiple items

```
// Notice this is a static class method
await Pet.destroy({
    where: {
        petTypeId: 1, // Destorys all the pets where the petType is 1 }
    },
});
```

continued on next page

Query Methods

`findOne` Method

```
await Model.findOne({
    where: {
        column: {
            [Op.<operator>]: <value> // refer to "Common Where Operators" below
        }
    },
});
```

`findAll` Method

```
await <Model>.findAll({
    where: {
        <column>: {
            [Op.<operator>]: <value> // refer to "Common Where Operators" below
        }
    },
    include: <include_specifier>,
    offset: 10,
    limit: 2
});
```

`FindByPk` Method

```
await <Model>.findByPk(<primary_key>, {
    include: <include_specifier>
});
```

Eager loading association with `include`

simple include of one related model:

```
await Pet.findByPk(1, {
    include: PetType,
});
```

Include can take an array of models if you need to include more than one:

```
await Pet.findByPk(1, {
    include: [Pet, Owner],
});
```

Include can also take an object with keys model and include. This is in case you have nested associations. In this case Owner doesn't have an association with PetType, but Pet does, so we want to include PetType onto the Pet Model:

```
await Owner.findByPk(1, {
    include: {
        model: Pet
        include: PetType
    }
});
```

`toJSON` method

The confusingly named toJSON() method does not return a JSON string but instead returns a POJO for the instance:

```
// pet is an instance of the Pet class
const pet = await Pet.findByPk(1);

console.log(pet); // prints a giant object with tons of properties and methods.

const petPOJO = pet.toJSON(); // petPOJO is now just a plain old Javascript Object

console.log(petPOJO); // { name: "Fido", petTypeId: 1 }
```

continued on next page

Common Where Operators

```
const Op = Sequelize.Op

[Op.and]: [{a: 5}, {b: 6}] // (a = 5) AND (b = 6)
[Op.or]: [{a: 5}, {a: 6}] // (a = 5 OR a = 6)
[Op.gt]: 6, // > 6
[Op.gte]: 6, // >= 6
[Op.lt]: 10, // < 10
[Op.lte]: 10, // <=10
[Op.ne]: 20, // != 20
[Op.eq]: 3, // =3
[Op.is]: null // IS NULL
[Op.not]: true, //IS NOT TRUE
[Op.between]: [6, 10], // BETWEEN 6 AND 10
[Op.notBetween]: [11, 15], // NOT BETWEEN 11 AND 15
[Op.in]: [1, 2], // IN [1, 2]
[Op.notIn]: [1, 2], // NOT IN [1, 2]
[Op.like]: '%hat', // LIKE '%hat'
[Op.notLike]: '%hat' // NOT LIKE '%hat'
[Op.iLike]: '%hat' // ILIKE '%hat' (case insensitive, Postgres only)
[Op.notILike]: '%hat' // NOT ILIKE '%hat' (Postgres only)
[Op.startsWith]: 'hat' // LIKE 'hat%'
[Op.endsWith]: 'hat' // LIKE '%hat'
[Op.substring]: 'hat' // LIKE '%hat%'
[Op.regexp]: '^[h|a|t]' // REGEXP/~ '^[h|a|t]' (MySQL/Postgres only)
[Op.notRegexp]: '^[h|a|t]' // NOT REGEXP/!~ '^[h|a|t]' (MySQL/Postgres only)
[Op.iRegexp]: '^[h|a|t]' // ~\* '^[h|a|t]' (Postgres only)
[Op.notIRegexp]: '^[h|a|t]' // !~\_ '^[h|a|t]' (Postgres only)
[Op.like]: { [Op.any]: ['cat', 'hat'] }
```