

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

```
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 init
```

Then, you must customize `/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 migration:generate --name <ModelName>
```

Generate a model and its migration:

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

Run pending migrations:

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

Rollback migration:

```
npx dotenv sequelize db:migrate:undo      # rollback one migration  
  
npx dotenv sequelize db:migrate:undo:all # rollback all migrations  
  
npx dotenv -- sequelize db:migrate:undo:all --to <MigrationFileName>  
# rollback to specific migration
```

Generate a new seed file

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

Run all pending seeds

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

Rollback seeds

```
npx dotenv sequelize db:seed:undo      # rollback one seed  
  
npx dotenv sequelize 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");
```

Adding a Column

```
await queryInterface.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', // model name referencing the join table  
  foreignKey: 'studentId',  
  otherKey: 'lessonId'  
};  
  
Student.belongsToMany(models.Lesson, columnMapping);
```

lesson.js

```
const columnMapping = {  
  through: 'StudentLesson', // model name referencing the join table  
  foreignKey: 'lessonId',  
  otherKey: 'studentId'  
};  
  
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, // Destroys all the pets where the petType is 1
  },
});
```

Query Methods

findOne Method

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

findAll Method

```
await <Model>.findAll({
  where: {
    <columnName>: {
      [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

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 handle nested associations. Takes an object with keys, model and include. 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 }
```

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, PG only)
[Op.notILike]: '%hat' // NOT ILIKE '%hat' (PG 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/PG only)
[Op.notRegexp]: '^h|a|t' // NOT REGEXP/!~ '^h|a|t' (MySQL/PG only)
[Op.iRegexp]: '^h|a|t' // ~\* '^h|a|t' (PG only)
[Op.notIRegexp]: '^h|a|t' // !~\* '^h|a|t' (PG only)
[Op.like]: { [Op.any]: ['cat', 'hat'] } (PG only)
// LIKE ANY (ARRAY['cat', 'hat'])
```