Sequelize Cheatsheet

In this article

Command Line

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

Init Project

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

You must create a database user, and update the `config/config.json` file to match your database settings to complete the initialization process.

Create Database

```
npx sequelize-cli db:create
```

Generate a model and its migration

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

Run pending migrations

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

Rollback one migration

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

Rollback all migrations

```
npx sequelize-cli db:migrate:undo:all
```

Generate a new seed file

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

Run all pending seeds

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

Rollback one seed

```
npx sequelize-cli db:seed:undo
```

Rollback all seeds

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

Migrations

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

```
// This uses the short form for references
return queryInterface.createTable(<TableName>, {
    <columnName>: {
        type: Sequelize.<type>,
        allowNull: <true|false>,
        unique: <true|false>,
        references: { model: <TableName> }, // This is the plural table name
                                            // that the column references.
    }
});
// This the longer form for references that is less confusing
return queryInterface.createTable(<TableName>, {
    <columnName>: {
        type: Sequelize.<type>,
        allowNull: <true|false>,
        unique: <true|false>,
        references: {
            model: {
                tableName: <TableName> // This is the plural table name
            }
        }
    }
});
```

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

```
return queryInterface.dropTable(<TableName>);
```

Adding a column

```
return queryInteface.addColumn(<TableName>, <columnName>: {
    type: Sequelize.<type>,
    allowNull: <true|false>,
    unique: <true|false>,
    references: { model: <TableName> }, // This is the plural table name
                                        // that the column references.
});
```

Removing a column

```
return 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 StudentLessons table

`student.js`

```
const columnMapping = {
    through: 'StudentLesson', // This is the model name referencing the join table.
    otherKey: 'lessonId',
    foreignKey: 'studentId'
}
Student.belongsToMany(models.Lesson, columnMapping);
```

`lesson.js`

```
const columnMapping = {
    through: 'StudentLesson', // This is the model name referencing the join table.
    otherKey: 'studentId',
    foreignKey: 'lessonId'
}
Lesson.belongsToMany(models.Student, columnMapping);
```

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
    }
});
```

Query Format

findOne

```
await <Model>.findOne({
    where: {
        <column>: {
            [Op.<operator>]: <value>
        }
    },
});
```

findAll

```
await <Model>.findAll({
    where: {
        <column>: {
            [Op.<operator>]: <value>
        }
    },
    include: <include_specifier>,
    offset: 10,
    limit: 2
});
```

findByPk

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

Eager loading associations 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,

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