PHASE 1

WEEK 3

DAY 4

ELBRUS/BOOTCAMP

План

- Связи / ассоциации в Sequelize
- Модели и методы (продолжение)
- Миграции (повторение)

Связи в Sequelize

Виды связей / ассоциаций:

Sequelize поддерживает стандартные ассоциации или отношения между моделями:

- один-к-одному (One-To-One)
- один-ко-многим (One-To-Many)
- многие-ко-многим (Many-To-Many)

Типы связей / ассоциаций:

Существует 4 типа ассоциаций:

- HasOne
- BelongsTo
- HasMany
- BelongsToMany

Примеры использования связей:

```
const { Model_A, Model_B } = require('./db/models');

// означает, что между Model_A и Model_B существуют отношения один-к-одному,
при этом, внешний ключ (foreign key) определяется в целевой модели (Model_B)
Model_A.hasOne(Model_B, {});

// связь один-к-одному, внешний ключ определяется в источнике (Model_A)
Model_A.belongsTo(Model_B, {});

// связь один-ко-многим, внешний ключ определяется в целевой модели (Model_B).
Model_A.hasMany(Model_B, {});

// означает, что между Model_A и Model_B существуют связь многие-ко-многим,
таблица Model_C выступает в роли связующего звена между ними через внешние
ключи (например, Model_AId и Model_AId). Sequelize автоматически создает
модель Model_C при ее отсутствии, определяя в ней соответствующие ключи.
Model_A.belongsToMany(Model_B, { through: 'Model_C' });
```

One-To-Many с моделями User и Card:

В модели User:

```
static associate(models) {
 this.hasMany(models.Card, {
   foreignKey: 'author',
 });
}
```

В модели Card:

```
static associate(models) {
 this.belongsTo(models.User, {
   foreignKey: 'author',
 });
}
```

One-To-Many с моделями User и Card:

В миграции create-card:

```
author: {
    type: Sequelize.INTEGER,
    allowNull: false,
    references: {
        model: {tableName: 'Users'},
        key: 'id',
    },
},
```

Модели и методы

Модели

Модель — специальный класс Sequelize, позволяющий работать с данными в базе данных как с обычными JavaScript-объектами.

Создать модель можно с помощью метода sequelize.define

Модели

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', {
 // Атрибуты (колонки) модели описываются здесь
 firstName: {
   type: DataTypes.STRING,
   allowNull: false
 },
 lastName: {
   type: DataTypes.STRING
   // allowNull по умолчанию true
 }
}, {
 // Другие настройки модели пишутся тут
});

console.log(User === sequelize.models.User);
```

Методы моделей: findAll()

Метод findAll() создает простой SELECT на выборку всех записей из таблицы

```
Student.findAll();
// SELECT * FROM students;
Student.findAll({
 attributes: ['name', 'age'],
});
// SELECT name, age FROM students;
```

Методы моделей: findByPk()

Метод findByPk() выбирает из таблицы одну запись по первичному ключу

```
const project = await Project.findByPk(123);
if (project === null) {
 console.log('Not found!');
} else {
 console.log(project instanceof Project); // true
 console.log(project);
}
```

Методы моделей: findOne()

Метод findOne выбирает из таблицы одну запись по любому условию

```
const project = await Project.findOne({
 where: { title: 'ToDo' },
});

console.log(project);
```

Методы моделей: findOrCreate()

Метод findOrCreate() создаст запись в таблице, если не сможет найти ее по условию.

В любом случае вернется экземпляр и логическое значение (created), указывающее, был этот экземпляр создан или уже существовал

```
const [book, created] = await Book.findOrCreate({
 where: { author: 'Fyodor Mikhailovich Dostoevsky' },
 defaults: { title: 'Notes from Underground' },
});
console.log(book.author); // 'Fyodor Mikhailovich Dostoevsky'
console.log(created); // false если запись найдена или true если создана
```

Методы моделей: findAndCountAll()

Метод findAndCountAll() объединяет в себе два метода: findAll() и count(). В результате возвращается объект с двумя свойствами: count и rows - количество строк и массив объектов с записями

```
const { count, rows } = await Project.findAndCountAll({
 where: { title: 'ToDo' },
});
console.log(count);
console.log(rows);
```

Миграции

Что такое миграции?

Миграции — способ версионирования структуры базы данных. В Sequelize есть встроенная система миграций.

По возможности не помещайте в миграцию изменение данных в БД! Миграции должны работать со структурой.

```
npm install --save-dev sequelize-cli
```

Создание миграции

```
npx sequelize-cli migration:generate --name migration-skeleton
```

```
module.exports = {
 up: (queryInterface, Sequelize) => {
   // логика для перевода БД в новое состояние
 },
 down: (queryInterface, Sequelize) => {
   // логика для отмены действий из функции up
 },
};
```

Запуск и откат миграции

- Запуск всех ещё не запускавшихся миграций:

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

- Отмена (откат) последней миграции:

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