PHASE 1

WEEK 3

DAY 3

ELBRUS/BOOTCAMP

План

- Sequelize CLI
- Модели
- Миграции
- Методы моделей

Sequelize CLI

Sequelize CLI

CLI (command line interface) — интерфейс командной строки.

Sequelize CLI — программа для работы с Sequelize ORM из терминала.

```
npm i sequelize-cli
```

Sequelize CLI

```
npx sequelize-cli init
```

Можно использовать краткую запись:

```
npx sequelize init
```

Sequelize CLI

Sequelize CLI создаёт 4 директории в корневой папке:

- `config/`
- `models/`
- `migrations/`
- `seeders/`

Sequelize CLI

`config/`

`config.json`

Реквизиты подключения к БД — имя и пароль пользователя БД, название базы, хост и порт, etc.

`models/`

`index.js` — само подключение к базе данных.

\+ модели для работы с БД.

Sequelize CLI

`migrations/`

Миграции для изменения структуры БД.

`seeders/`

Сидеры для начального заполнения данными.

.sequelizerc

Это специальный файл конфигурации, который должен находиться в корне папки с проектом. Он позволяет вам указывать различные параметры, которые вы обычно передаете в качестве аргументов в CLI.

Некоторые сценарии, в которых вы можете его использовать:

- вы хотите отменить заданные по умолчанию пути к `migrations`, `models`, `seeders` или `config` папке.
- вы хотите переименовать `config.json` во что-то другое, например `database.json`

Модели

Модель

Класс с описанием *таблицы* в базе данных.

Экземпляры этого класса описывают *записи* в таблице.

Создать модель можно через Sequelize CLI командой: `model:generate`

Создание модели

Команда `model:generate` требует следующие флаги:

`--name` — название модели.

`--attributes` — описание столбцов для таблицы.

ID прописывать не нужно — Sequelize CLI опишет его самостоятельно.

Создание модели

```
npx sequelize model:generate --name User --attributes
username:text,firstName:text
```

Эта команда создаст модель `User` с полями `username` и `firstName`.

Список типов данных Sequelize:
https://sequelize.org/v5/manual/data-types.html

Структура модели

```
class User extends Model { /* ... */ }
User.init(
   {
      // атрибуты модели — каждый ключ описывает колонку в таблице
      username: DataTypes.TEXT,
      firstName: DataTypes.TEXT,
   },
   {
      // дополнительные опции модели
   }
);
```

Структура модели

Созданную модель можно и порой нужно редактировать.

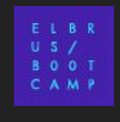
Чем точнее описаны ограничения (например, `references`, `notNull`, `unique`, etc.), тем лучше.

Список доступных настроек в документации по методу `init`:
https://sequelize.org/master/class/lib/model.js~Model.html#static-method-init

Структура модели

```
User.init(
   {
      // атрибут, описанный детально
      username: {
        type: DataTypes.TEXT,
        unique: true,
        allowNull: false
      },
   // ...
```

Миграции



Миграции

Миграция — способ описать изменения в структуре БД.

Файл миграции содержит две функции:

- up
  Код для изменения структуры БД.
- down
  Код для возврата к предыдущей структуре до этих изменений.

Миграции

Sequelize CLI автоматически генерирует миграцию при создании модели и сохраняет её в папку `migrations/`

Важно!

Если вы редактировали модель после создания, необходимо продублировать эти изменения в файле миграции.

Применение миграций

Чтобы описанные в миграции изменения были выполнены, необходимо её применить — то есть, вызвать функцию up.

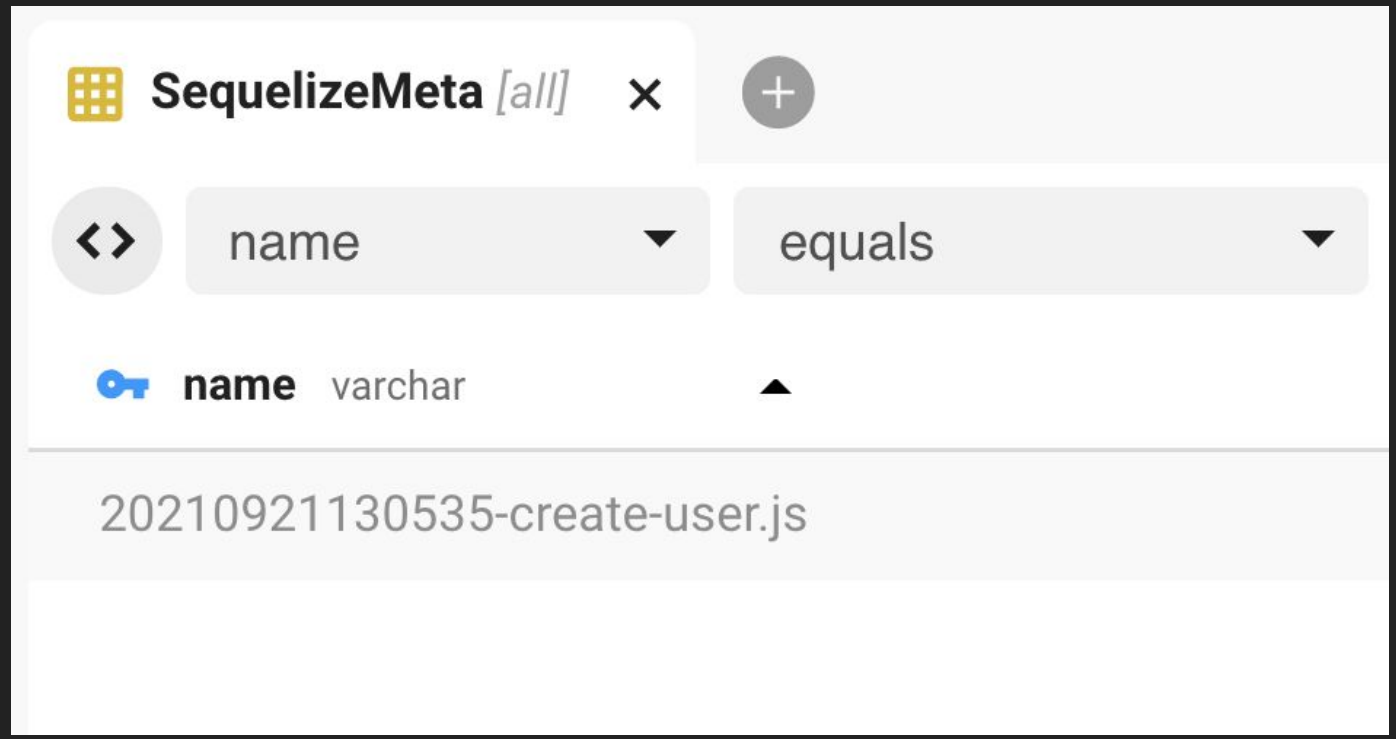
Команда `db:migrate` запустит все неприменённые миграции по очереди:

```
npx sequelize db:migrate
```

Применение миграций

Список уже применённых миграций можно проверить в таблице `SequelizeMeta`.

Эту таблицу Sequelize CLI создаёт автоматически.



Отмена миграций

Отмена миграции — это вызов функции down из файла миграции.

`db:migrate:undo`

Отменяет последнюю миграцию.

`db:migrate:undo:all`

Отменяет все миграции по очереди — от самой поздней к самой ранней.

Отмена миграций

Чтобы откатиться до конкретной миграции, можно использовать флаг `--to` и название этой миграции:

```
npx sequelize db:migrate:undo:all --to
XXXXXXXXXXXXXX-create-posts.js
```

Изменение миграции

Миграцию можно менять только в следующих случаях:

- эта миграция ещё не была применена к какой-либо БД,
- либо эта миграция была отменена для всех БД, где её ранее применяли.

Если вы закоммитили миграцию и опубликовали её — менять эту миграцию не стоит.

Методы моделей

Методы моделей

У Sequelize-моделей есть методы для всех CRUD-операций с записями.

- `INSERT → Model.create()`
- `SELECT → Model.findAll()`
- `UPDATE → Model.update()`
- `DELETE → Model.destroy()`

Все эти методы возвращают промис и могут принимать объект с параметрами.

create

Метод `create` создает экземпляр модели и сохраняет его в БД.

```
const user = await User.create({
   name: "Igor",
   email: "in@mail.ru"
});
```

findAll

Метод `findAll()` выполняет команду SELECT и возвращает промис со всеми совпадениями.

```
const users = await User.findAll(); // SELECT * FROM users;
```

findAll — attributes

Можно выбирать конкретные столбцы через параметр `attributes`:

```
// SELECT name, email FROM users
const users = await User.findAll({
 attributes: ['name', 'email']
});
```

findAll — where

Можно задавать условия выборки через параметр `where`:

```
// SELECT * FROM users WHERE student = true
const students = await User.findAll({
 where: { student: true }
});
```

findAll — where

```
// SELECT * FROM users
// WHERE student = true AND status = 'active';
const activeStudents = await User.findAll({
 where: {
   student: true,
   status: 'active'
 }
});
```

update

Метод `update()` редактирует записи в таблице, подходящие по заданным параметрам.

```
// UPDATE users SET name = 'Rauf' WHERE id = 3
await User.update(
   { name: "Rauf" },
   { where: { id: 3 } }
);
```

destroy

Метод `destroy()` удаляет записи, подходящие по заданным параметрам:

```
// DELETE user WHERE name = 'Igor'
await User.destroy({
   where: { name: "Igor" }
});
```