Desenvolvimento Backend

Aula 06 - Banco de dados relacional (parte 1)

Material Didático do Instituto Metrópole Digital - IMD

Termo de uso

Apresentação

Nesta aula, você verá como utilizar um banco de dados para gerenciar os dados da aplicação através do uso de um ORM.

Objetivos

- Conhecer o conceito de ORM;
- Aprender a configurar o Sequelize;
- Aprender a criar modelos com o Sequelize;
- Entender como aplicar migrações;
- Utilizar operações básicas no banco com o Sequelize.

ORM Sequelize

Link do video da aula: https://youtu.be/5ARp-bkH8pA

Nesta aula, veremos como utilizar banco de dados com Node.js. Até aqui nossos dados estavam sendo salvos em variáveis e eram perdidos sempre que a aplicação era reiniciada.

No entanto, normalmente queremos manter os dados a salvo. Para isso, a partir desta aula veremos como integrar o Node.js com banco de dados.

Integração com banco de dados

Atualmente existem vários fornecedores de banco de dados como, por exemplo: Postgresql, MySQL, Mariadb, SQL Server, Oracle, MongoDDB etc. Cada um desses possui uma forma diferente de acesso com protocolos de comunicação distintos. Para resolver esse problema, normalmente os fornecedores disponibilizam *drivers* de acesso que implementam os protocolos e tornam esse acesso mais simples.

Apesar de funcional, o uso dos *drivers* diretamente deixa as aplicações dependentes dos fornecedores dos bancos de dados. Esse problema é abordado por fornecedores de ORM (*Object Relational Mapper*), que são ferramentas que buscam abstrair do desenvolvedor os aspectos internos de cada banco de dados. Para isso, os ORMs fornecem uma camada de alto nível para acesso aos dados permitindo que a decisão de qual banco utilizar possa ser tomada a *posteriori*.

Neste curso, utilizaremos o ORM Sequelize, que nos auxiliará em toda comunicação com os bancos de dados. Nas próximas videoaulas, veremos na prática como utilizá-lo.

Criando modelos e migrações no Sequelize

Link do video da aula: https://youtu.be/T7b-UMDZ-RA

Instalando o Sequelize

Para instalar o Sequelize, execute o seguinte comando:

```
$ npm install sequelize
```

Configurando o Sequelize

Após a instalação, precisamos criar os arquivos de configuração do Sequelize. Para isso, podemos utilizar uma ferramenta do próprio Sequelize que facilita esse processo.

Para iniciar, execute:

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

Feita a inicialização, devemos configurar o acesso ao banco de dados no arquivo `config/config.json`. Esse arquivo contém todos os dados de acesso aos bancos de dados. Para o ambiente de desenvolvimento, utilizaremos o banco de dados SQLite. Para configurá-lo, é necessário alterar a seção "development" desse arquivo. Após a alteração, o trecho "*development*" deve ficar assim:

```
{
  "development": {
    "username": "root",
    "password": null,
    "database": "bend",
    "host": "127.0.0.1",
    "dialect": "sqlite",
    "storage": "./db/dev.sqlite"
  }
}
```

Além do Sequelize, é necessário instalar o dialeto do banco que será utilizado. Cada banco de dados possui uma biblioteca própria que deve ser instalada em conjunto com o Sequelize. Para o SQLite, o dialeto se chama sqlite3 e deve ser instalado com

o seguinte comando:

```
$ npm install sqlite3
```

Criando os modelos

Os modelos são representações dos dados que serão manipulados no banco de dados. Para nosso exemplo, vamos criar um modelo para registrar dados dos usuários e postagens de um blog. Para criar o modelo do Usuário, execute:

```
$ npx sequelize-cli model:generate --name Usuario --attributes
email:string, senha:string
```

Além do modelo do Usuário, vamos criar um modelo para armazenar postagens. Para isso, execute:

```
$ npx sequelize-cli model:generate --name Post --attributes
titulo:string, texto:string
```

Criando estrutura do banco de dados

Link do video da aula: https://youtu.be/6PyCQmFRhi8

Aplicando as migrações

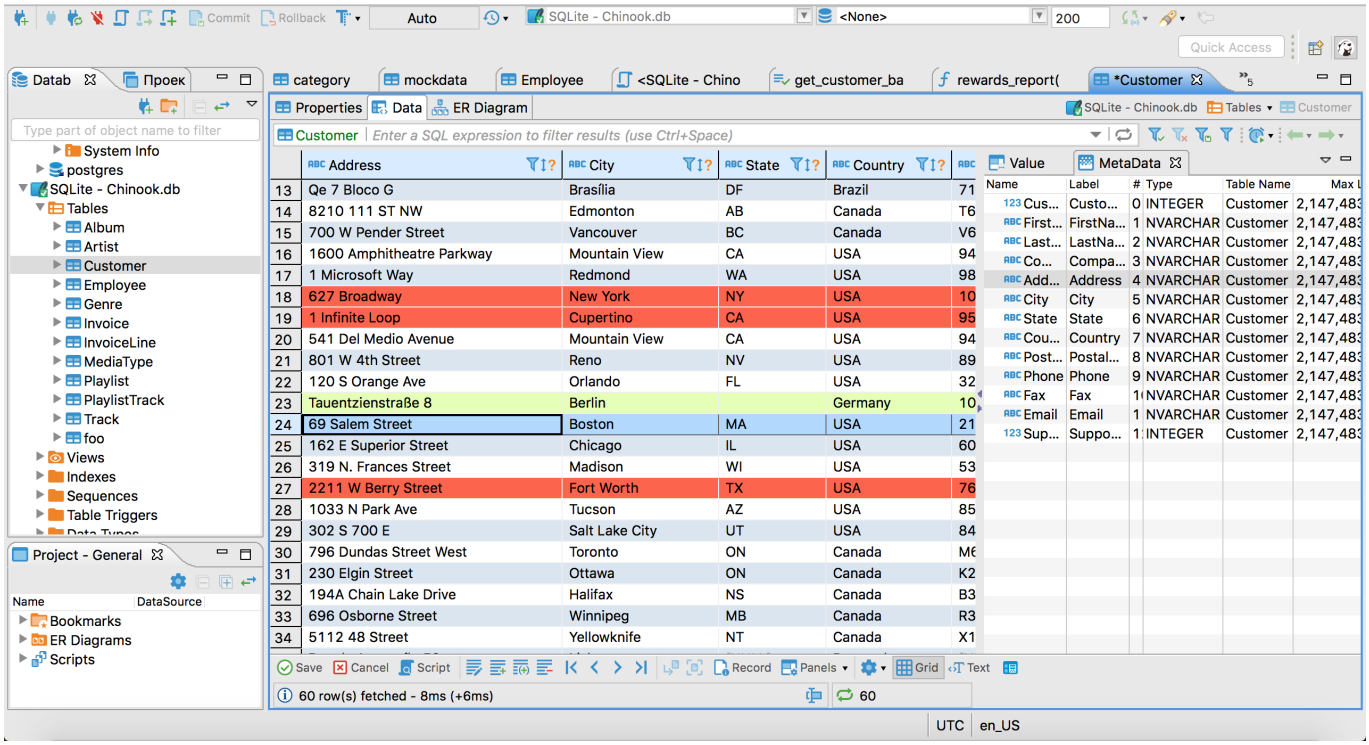
Dado que os modelos já foram criados, precisamos agora criar a estrutura de tabela no banco de dados. Para isso, é necessário executar o comando que aplica todas as migrações pendentes. As migrações permitem aplicar mudanças na estrutura, nos dados e ainda criar a estrutura inicial das tabelas.

Para executar todas as migrações pendentes, digite o seguinte comando:

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

Em seguida, vamos verificar se a operação deu realmente certo. Para isso, iremos utilizar o software chamado de DBeaver Community, que permite acesso fácil ao SQLite (e outros bancos).

Figura 1 - Interface do DBeaver



Realize a instalação da ferramenta com procedimento encontrado no site do fornecedor e em seguida acompanhe a videoaula para testar se a criação da estrutura no banco de dados ocorreu conforme esperado.

Manipulando dados no Banco de dados

Link do video da aula: https://youtu.be/RWEIvCmMIt8

Agora que os modelos e a estrutura já estão criados, chegou a hora de realizar as operações diretamente no banco de dados.

Para isso, acompanhe a videoaula para explicação detalhada de como executar as operações de criação, edição, busca e remoção de dados.

Abaixo é apresentado o conteúdo final do arquivo de rota (`posts.rota.js`), com as operações realizadas com auxílio do ORM Sequelize.

```
const express = require('express')
const router = express.Router()
const postMid = require('../middleware/validarPost.middleware')
const { Post } = require('../models')

router.post('/', postMid)
router.put('/', postMid)
```

```
router.get('/', async (req, res) => {
    const posts = await Post.findAll()
    res.json({posts: posts})
})

router.get('/:id', async (req, res) => {
    const post = await Post.findByPk(req.params.id)
    res.json({posts: post})
})

router.post('/', async (req, res) => {
    const post = await Post.create(req.body)
    res.json({msg: "Post adicionado com sucesso!"})
})

router.delete('/', async (req, res) => {
    const id = req.query.id
    const post = await Post.findByPk(id)
    if (post){
        await post.destroy()
        res.json({msg: "Post deletado com sucesso!"})
    }else{
        res.status(400).json({msg: "Post não encontrado!"})
    }
})

router.put('/', async (req, res) => {
    const id = req.query.id
    const post = await Post.findByPk(id)
    if (post){
        post.titulo = req.body.titulo
        post.texto = req.body.texto
        await post.save()
        res.json({msg: "Post atualizado com sucesso!"})
    }else{
        res.status(400).json({msg: "Post não encontrado!"})
    }
})

module.exports = router
```

Testando API criada

Link do video da aula: https://youtu.be/h7RD9XoLPv4

Agora que a implementação já foi realizada, vamos testar se tudo está funcionando como esperado.

Acompanhe a videoaula para ver o teste do código construído nesta aula.

Resumo

Nesta aula, você viu o conceito de ORM e como ele pode auxiliar no gerenciamento de dados com os bancos de dados. Conheceu o ORM Sequelize e como configurá-lo. Em seguida aprendeu a criar modelos, aplicar migrações e realizar as principais operações de banco de dados (criação, remoção, atualização e busca).