Aula 10 - Acesso a dados com Sequelize

SECITECI
SECRETARIA DE ESTADO DE CIÊNCIA E TECNOLOGIA E INOVAÇÃO

GOVERNO DE MATO GROSSO

<Fic_Dev>
Programador de Sistemas

Roteiro

1. Desafio do Dia
2. Introdução
3. Sequelize
4. Conclusão

Desafio do Dia

Vamor criar um novo Modelo chamado aluno com as seguintes informações: Id, Nome, Sexo, Data de Nascimento, Curso, Cursa Ensino Superior?, Estagiando?
Use o readline-async e demais estruturas para possibilitar:

- Incluir
- Alterar (a pesquisar)
- Excluir (a pesquisar)
- Listar Todos
- Pesquisar pelo nome

Crie uma opção de sair para encerrar.

<Fic_Dev>
Programador de Sistemas

Acesso a dados com Sequelize

Agora, passaremos ao estudo da Persistência de Dados, que é o conceito de manter informações de forma duradoura e recuperá-las posteriormente. Envolve o armazenamento de dados em algum tipo de meio persistente, como bancos de dados, sistemas de arquivos ou armazenamento em nuvem, garantindo que os dados permaneçam disponíveis mesmo após o encerramento do programa ou falhas de hardware.
A persistência de dados é essencial em muitas aplicações, permitindo que os dados sejam armazenados, consultados, atualizados e excluídos ao longo do tempo. Isso inclui a capacidade de realizar operações de criação, leitura, atualização e exclusão (CRUD - Create, Read, Update, Delete) nos dados armazenados.

Acesso a dados com Sequelize

No início dos anos 90, quando os bancos de dados relacionais já estavam bem estabelecidos e a orientação a objetos estava se consolidando, surge uma tecnologia chamada ORM, que desempenhava um papel importante na simplificação da interação com bancos de dados relacionais, fornecendo uma abstração de nível mais alto que permite que os desenvolvedores trabalhem com objetos em vez de tabelas e consultas SQL diretamente.

Acesso a dados com Sequelize

- Uma das primeiras implementações notáveis de ORM foi o framework TopLink, desenvolvido pela Oracle Corporation em meados dos anos 1990. TopLink foi inicialmente lançado como uma ferramenta proprietária e, posteriormente, foi adquirido pela Oracle e incorporado ao seu produto de banco de dados.
- Outro framework de ORM notável é o Hibernate, que foi criado por Gavin King em 2001 como um projeto de código aberto em Java. O Hibernate se tornou um dos frameworks de ORM mais populares e influentes, introduzindo conceitos e padrões que foram adotados por muitos outros frameworks subsequentes.

Acesso a dados com Sequelize

- Desde então, surgiram vários frameworks de ORM em várias linguagens de programação, incluindo o Entity Framework para o .NET Framework/Microsoft, o Django ORM para Python, o Ruby on Rails ActiveRecord para Ruby, o Sequelize para Node.js, entre outros.

Short Challenge

Pesquise sobre: Atualmente, o Sequelize consegue conversar com quais SGBD's?
Obs.: haverá entrega no Classroom.

Curiosidade

Existem diversas frameworks de ORM e para diversas linguagens. Além da eficiência, outras características entram em jogo, como por exemplo a maturidade e a sofisticação do framework.
Em Hibernate (Java) e em NHibernate (.Net), existe o conceito de Consulta Futura ou Future Queries (Future<>) que visa diminuir as idas e vindas (roundtrip) ao banco de dados.

Curiosidade

```
using (ISession session = sessionFactory.OpenSession())
{
    // this executes the first query
    var categories = session.CreateCriteria(typeof(ProductCategory)).List();
    // this executes the second query
    var suppliers = session.CreateCriteria(typeof(Supplier)).List();

    foreach (var category in categories)
    {
        // do something
    }

    foreach (var supplier in suppliers)
    {
        // do something
    }
}
```

Figura: Acesso sem consultas futuras

<Fic_Dev>
Programador de Sistemas

Curiosidade

```
using (ISession session = sessionFactory.OpenSession())
{
    // this creates the first query
    var categories = session.CreateCriteria(typeof(ProductCategory)).Future<ProductCategory>();
    // this creates the second query
    var suppliers = session.CreateCriteria(typeof(Supplier)).Future<Supplier>();

    // this causes both queries to be sent in ONE roundtrip
    foreach (var category in categories)
    {
        // do something
    }

    // this doesn't do anything because the suppliers have already been loaded
    foreach (var supplier in suppliers)
    {
        // do something
    }
}
```

Figura: Acesso com consultas futuras

<Fic_Dev>
Programador de Sistemas

Short Challenge

Pesquise sobre: O Sequelize implementa o recurso de Consultas Futuras?
Obs.: haverá entrega no Classroom.

Instalação e Configuração

Antes de qualquer coisa: https://sequelize.org/
Vamos acessar e ver a documentação.

Instalação e Configuração

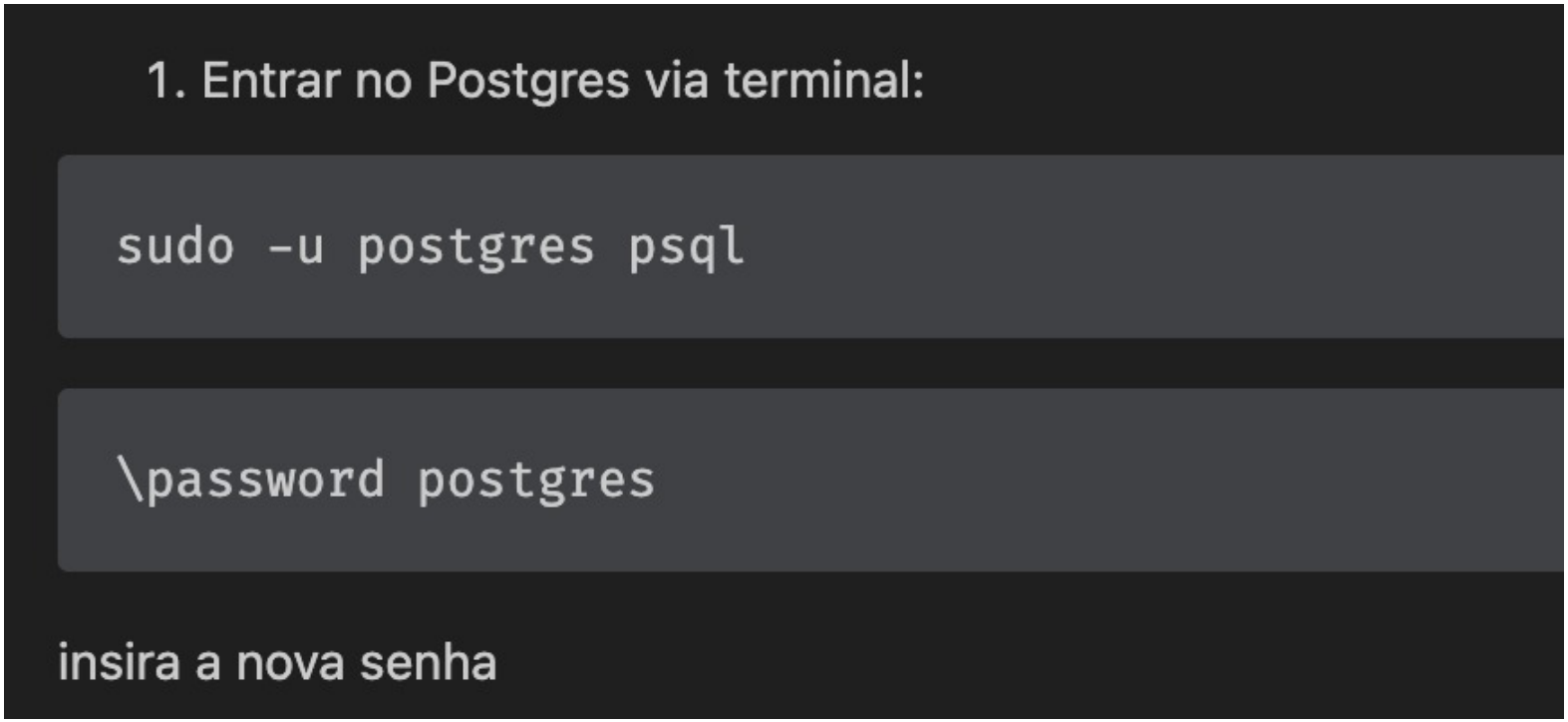

Figura: Ajuste da senha do Postgres

Vamos testar o Postgres criando um banco de dados: exemplosequelize1

Informações de acesso ao banco de dados

```
1  const Sequelize = require('sequelize')
2  const sequelize = new Sequelize('exemplo1sequelize', 'sa', 'sql2019', {
3    dialect: 'mssql',
4    host: 'localhost'
5  })
6
7  module.exports = sequelize;
```

Figura: Informações de acesso ao banco de dados

Criando um primeiro modelo

```
1  const Sequelize = require('sequelize');
2  const database = require('./db');
3
4  const Produto = database.define('produto', {
5    id: {
6      type: Sequelize.INTEGER,
7      autoIncrement: true,
8      allowNull: false,
9      primaryKey: true
10   },
11   nome: {
12     type: Sequelize.STRING,
13     allowNull: false
14   },
15   preco:{
16     type:Sequelize.DECIMAL,
17     allowNull: false
18   },
19   descricao:{
20     type:Sequelize.STRING,
21     allowNull: false
22   }
23 });
24
25 module.exports = Produto;
```

Figura: Criando um primeiro modelo

<Fic_Dev>
Programador de Sistemas

Criando um primeiro modelo

```
1  (async() => {
2  const Sequelize = require('sequelize');
3      const Op = Sequelize.Op;
4      const database = require('./db');
5      const Produto = require('./produto');
6      await database.sync();
7
8  })();
```

Figura: Criando uma Index.js

Criando um Produto

```
(async() => {
const Sequelize = require('sequelize');
  const Op = Sequelize.Op;
  const database = require('./db');
  const Produto = require('./produto');
  await database.sync();

  const produto = await Produto.create({
    nome: 'Coca-cola LT',
    preco: 5,
    descricao: 'Coca-cola LT e tals.'
  })

  console.log(produto);
})();
```

Figura: Criando um Prtoduto

Buscando todos os produtos

```
1  (async() => {
2  const Sequelize = require('sequelize');
3    const Op = Sequelize.Op;
4    const database = require('./db');
5    const Produto = require('./produto');
6    await database.sync();
7
8    const produtos = await Produto.findAll();
9    console.log(produtos);
10
11 })();
```

Figura: Buscando todos os produtos

Buscando pela Chave

```
1  (async() => {
2  const Sequelize = require('sequelize');
3    const Op = Sequelize.Op;
4    const database = require('./db');
5    const Produto = require('./produto');
6    await database.sync();
7
8    const produto = await Produto.findByPk(2);
9    console.log(produto);
10
11 })();
```

Figura: Buscando pela Chave

Buscando por Critérios

```
(async () => {
  const Sequelize = require('sequelize');
  const Op = Sequelize.Op;
  const database = require('./db');
  const Produto = require('./produto');
  await database.sync();

  const produtos = await Produto.findAll({
    where: {
      preco: {
        [Op.gte]:10
      }
    }
  });
  console.log(produtos);

})();
```

Figura: Buscando por Critérios

Conclusão

- Chegamos ao final dessa aula, que apenas introduziu o Sequelize.
- Nas próximas aulas teremos mais recursos a explorar.
- **Vamos ao Desafio do Dia!**