

Prática 2 - Banco de Dados H2

 moodle.utfpr.edu.br/pluginfile.php/3341177/mod_resource/content/2/Prática 2 - Banco de Dados H2.html

- 1) Objetivo

O que é o H2?

H2 é um banco de dados SQL escrito em Java. É muito rápido e requer menos recursos em comparação com outros bancos de dados SQL.

Nesta prática vamos:

Configurar o H2 para manter os dados de produtos.

Configurar projeto para acessar o H2 (Spring JPA)

Alterar o código do projeto para armazenar e buscar dados do H2

- 2) Adicionar Dependências ao Projeto

O projeto precisa receber duas nova dependências (H2 e Spring JPA)

No arquivo `pom.xml`, adicione a dependências:

```
<dependencies>
```

```
...
```

```
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

```
...
```

```
</dependencies>
```

Atualize o maven por meio do IntelliJ, para fazer o download do h2 e configurar o projeto.

Clique no ícone de atualizar (duas setas em círculo).

▼ Explicação JPA (Java Persistence API)

O JPA (Java Persistence API) é uma especificação da linguagem Java que define como os objetos Java podem ser mapeados para tabelas em um banco de dados relacional, permitindo a persistência de dados.

Ele abstrai a lógica de acesso ao banco de dados, facilitando o trabalho com operações de CRUD (criação, leitura, atualização e exclusão) sem que o desenvolvedor precise escrever consultas SQL diretamente.

O JPA utiliza anotações para configurar entidades e suas relações, e frameworks como Hibernate implementam essa especificação.



• 3) Configurar Banco de Dados H2

Passo 1:

No arquivo `src/main/resources/application.properties`, adicione as configurações para habilitar o H2 e definir o modo de funcionamento (por exemplo, em memória):

```
# Nome do banco de dados em memória
# spring.datasource.url=jdbc:h2:mem:produtos_db
# Nome do banco de dados em arquivo
spring.datasource.url=jdbc:h2:file:./produtos_db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.platform=h2
# Habilita o console do H2
spring.h2.console.enabled=true
# Caminho do console
spring.h2.console.path=/h2-console
# Atualiza o schema automaticamente (create, none)
spring.jpa.hibernate.ddl-auto=update
# Exibe SQL gerado
spring.jpa.show-sql=true
```

A configuração acima cria um arquivo `produtos_db.mv.db` no diretório raiz da aplicação.

Passo 2:

Iniciar a aplicação e acessar o console do H2 em <http://localhost:8080/h2-console>.

Use as seguintes credenciais:

JDBC URL: jdbc:h2:file:./produtos_db

Username: sa

Password: (deixe vazio)

- 4) Anotar Modelo para Usar JPA

Após adicionar as dependências e configurar o banco de dados, você deve anotar suas classes de modelo com

`@Entity`.

Acesse a classe Produto do pacote modelo e anote conforme o exemplo abaixo:

```
@Entity
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "produto_seq")
    @SequenceGenerator(name = "produto_seq", sequenceName = "produto_seq",
allocationSize = 1)
    private Long id;
    private String description;
    private Integer quantity;
    private Double price;
    private String category;

    public Produto(Long id, String description, Integer quantity, Double price,
String category) {
        this.id = id;
        this.description = description;
        this.quantity = quantity;
        this.price = price;
        this.category = category;
    }
    // Getter and Setters omitidos
}
```

▼ Explicação:

As estratégias de geração de IDs no JPA definem como o valor da chave primária será gerado para cada entidade persistida.

As principais estratégias são:

AUTO: O JPA escolhe automaticamente a estratégia mais apropriada com base no banco de dados que está sendo utilizado. Em bancos como o H2 ou PostgreSQL, por exemplo, pode ser usado um **sequence**.

IDENTITY: O banco de dados gera o valor do ID automaticamente, geralmente usando uma coluna auto-incremento. O JPA recupera o valor após a inserção.

SEQUENCE: Utiliza uma sequência de banco de dados para gerar os IDs. Essa estratégia é comum em bancos como o PostgreSQL ou Oracle, onde as sequências são geradas e mantidas separadamente.

TABLE: Usa uma tabela específica no banco de dados para controlar a geração dos IDs, armazenando os valores e incrementando-os manualmente. É menos comum e menos performática.

@SequenceGenerator vincula a sequência **produto_seq** ao campo **id**, usando o mesmo nome e **allocationSize** definidos pelo Hibernate.

- 5) Inserir Registro no H2

No console do H2 cole o comando abaixo e execute.

```
INSERT INTO produto (id, description, quantity, price, category) VALUES
(NEXT VALUE FOR produto_seq, 'Notebook', 10, 2500.00, 'Eletrônicos'),
(NEXT VALUE FOR produto_seq, 'Smartphone', 20, 1500.00, 'Eletrônicos'),
(NEXT VALUE FOR produto_seq, 'Cadeira de Escritório', 150, 450.00, 'Móveis'),
(NEXT VALUE FOR produto_seq, 'Mouse Gamer', 50, 120.00, 'Periféricos'),
(NEXT VALUE FOR produto_seq, 'Teclado Mecânico', 30, 300.00, 'Periféricos'),
(NEXT VALUE FOR produto_seq, 'Monitor 24 polegadas', 25, 850.00, 'Monitores'),
(NEXT VALUE FOR produto_seq, 'Impressora Multifuncional', 10, 1200.00,
'Impressoras'),
(NEXT VALUE FOR produto_seq, 'Headset', 40, 200.00, 'Periféricos'),
(NEXT VALUE FOR produto_seq, 'SSD 1TB', 70, 600.00, 'Armazenamento'),
(NEXT VALUE FOR produto_seq, 'Placa de Vídeo RTX 3060', 15, 2800.00, 'Hardware');
```

▼ Comandos para modificar a sequence gerada pelo Hibernate:

```
-- O padrão de sequence gerada pelo Hibernate tem incremento de 50 em 50.
-- Essa configuração é realizada ao anotar a classe de modelo (Produto)
-- Em Produto usamos configuração personalizada para gerar IDs.
-- Caso necessário alterar a sequence, siga o passo a passo abaixo:

-- Removendo o sequence gerada no H2 pelo Hibernate
DROP SEQUENCE produto_seq;

-- Criando a sequence, mas agora com incremento de 1 em 1
CREATE SEQUENCE produto_seq START WITH 1 INCREMENT BY 1;

-- Inserindo um registro usando a sintaxe do H2 para chamar a sequence
INSERT INTO produto (id, description, quantity, price, category)
VALUES (NEXT VALUE FOR produto_seq, 'Notebook', 10, 2500.00, 'Eletrônicos');

-- Para alterar o valor da sequence para um número específico
ALTER SEQUENCE produto_seq RESTART WITH 11;
```

• 6) Criar a Interface ProdutoRepository

Use a interface

JpaRepository para gerenciar a persistência dos seus dados.

Passo 1: Crie um novo pacote chamado repositories.

Passo 2: Crie a interface abaixo.

```
package br.edu.utfpr.aluno.api_produto.repositories;

import br.edu.utfpr.aluno.api_produto.model.Produto;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProdutoRepository extends JpaRepository<Produto, Long> { }
```

▼ Explicação:

JpaRepository é uma interface do Spring Data JPA que fornece operações CRUD (Create, Read, Update, Delete) básicas para gerenciar entidades no banco de dados.

Ela estende outras interfaces do Spring, como **CrudRepository** e **PagingAndSortingRepository**, oferecendo métodos prontos para uso, como **save()**, **findById()**, **findAll()**, **deleteById()**, entre outros.

Com o **JpaRepository**, você evita escrever código repetitivo para operações básicas de persistência e pode, se necessário, personalizar consultas utilizando a convenção de nomes ou a anotação **@Query**.

• 7) Modifique o Código do Controller para Acessar o H2

A versão atual da aplicação armazena os produtos em uma lista em memória.

Acesse o arquivo ProdutoController e faça as modificações abaixo.

▼ Injetar ProdutoRepository

```
// Adicionar atributo repository
private ProdutoRepository repository;

// Adicionar construtor com parâmetro ProdutoRepository
// Spring injetará uma instância repository
public ProdutoController(ProdutoRepository repository){
    this.repository = repository;
}
```

▼ GetMapping (getAll)

```
// Alterar a implementação de getAll conforme abaixo
@GetMapping
public ResponseEntity<List<Produto>> getAll(){
    // Agora os produtos estão sendo recuperados do H2
    return ResponseEntity.ok(this.repository.findAll());
}
```

▼ GetMapping (getOne)

```
// Alterar a implementação de getOne conforme abaixo
@GetMapping(path =("/{id}")
public ResponseEntity<Produto> getOne(@PathVariable(name = "id") Long idProduto){
    // Busca o produto no H2 a partir do Id
    Produto produtoEncontrado = this.repository.findById(idProduto).orElse(null);

    if (produtoEncontrado == null)
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
    else
        return ResponseEntity.status(HttpStatus.OK).body(produtoEncontrado);
}
```

▼ PostMapping (addOne)

```
// Alterar a implementação de addOne conforme abaixo
@PostMapping
public ResponseEntity<String> addOne(@RequestBody Produto produto) {
    if (produto.getDescription() == null || produto.getPrice() < 0){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Descrição ou Preço inválidos");
    } else {
        // Modificação: solicitar ao repository para inserir registro
do H2
        this.repository.save(produto);
        return ResponseEntity.status(HttpStatus.CREATED).body("Produto cadastrado com sucesso");
    }
}
```

▼ PutMapping (update)

```
// Alterar a implementação de update conforme abaixo
@PutMapping(path="/{id}")
public ResponseEntity<String> update(@PathVariable(name="id") Long idProduto,
@RequestBody Produto produto) {
    // Busca produto do H2, retorna null se não encontrar
    Produto produtoDB = this.repository.findById(idProduto).orElse(null);
    if (produtoDB != null){
        produtoDB.setDescription(produto.getDescription());
        produtoDB.setQuantity(produto.getQuantity());
        produtoDB.setPrice(produto.getPrice());
        produtoDB.setCategory(produto.getCategory());
        // Atualiza dados do produto no H2
        this.repository.save(produtoDB);
        return ResponseEntity.ok("Produto atualizado com sucesso.");
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Produto não encontrado.");
    }
}
```

▼ DeleteMapping (delete)

```
@DeleteMapping(path = "/{id}")
public ResponseEntity<String> delete(@PathVariable(name="id") Long idProduto){
    // Busca o produto no H2, por meio do Id
    Produto produtoDB = this.repository.findById(idProduto).orElse(null);
    if (produtoDB != null){
        // Se encontrou, remove o produto do H2
        this.repository.delete(produtoDB);
        return ResponseEntity.status(HttpStatus.OK).body("Produto removido com sucesso.");
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Produto não encontrado");
    }
}
```

• 8) Consultar Produtos por Categoria

Passo 1: Atualizar o **Repository**

Primeiro, você precisa adicionar um método de consulta por categoria no repositório do JPA (**ProdutoRepository**).

```
public interface ProdutoRepository extends JpaRepository<Produto, Long> {

    public List<Produto> findByCategory(String category);

}
```

▼ Explicação

Spring Data JPA permite criar consultas de forma automática com base nos nomes dos métodos na interface do repositório.

Para isso, o nome do método segue uma convenção que começa com um prefixo e inclui o nome dos campos da entidade.

O Spring Data JPA, então, gera a implementação da consulta automaticamente.

Consulta com base em dois campos:

```
List<Produto> findByCategoryAndPriceLessThan(String category, Double price);
```

Prefixos comuns:

findBy: Busca registros.

countBy: Conta registros.

deleteBy: Remove registros.

Passo 2: Criar um novo método no **Controller**

Adicione um endpoint no seu **Controller** para buscar os produtos com base na categoria.

```
@GetMapping("/categoria/{categoria}")
public ResponseEntity<List<Produto>> getByCategory(@PathVariable("categoria")
String category){
    // Chamando o método criado no Passo 1.
    // select * from produto where category = 'XXX'
    List<Produto> lista = this.repository.findByCategory(category);
    if (lista.isEmpty()){
        return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
    }
    return ResponseEntity.ok(lista);
}
```

Teste no Postman:

<http://localhost:8080/produtos/categoria/XXXX>