

FIA/P GRADUAÇÃO

SISTEMAS DE INFORMAÇÃO

MICROSERVICE AND WEB ENGINEERING

PROF. PEDRO IVO CORREIA
PROF. LUCAS FURLANETO

Agenda

Aula anterior:

- Implementação do CRUD de Categoria

Aula de Hoje:

- Adaptar o fluxo de CRUD do produto para receber a categoria.
- Finalizar o conteúdo de JDBC Template.

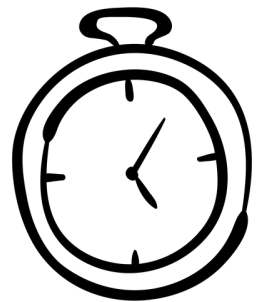
Objetivo

- Entender todo o funcionamento de um CRUD com relacionamento one-to-many.
- Saber quando utilizar o BeanPropertyRowMapper, RowMapper e ResultsetExtractor.



Time Box

Atividade	Duração
Introdução	10 min
Revisão Produto vs Categoria	20 min
Dúvidas	5 min
Exercício Categoria no CRUD de Produtos	100 min
Conclusão JDBC Template	35 min



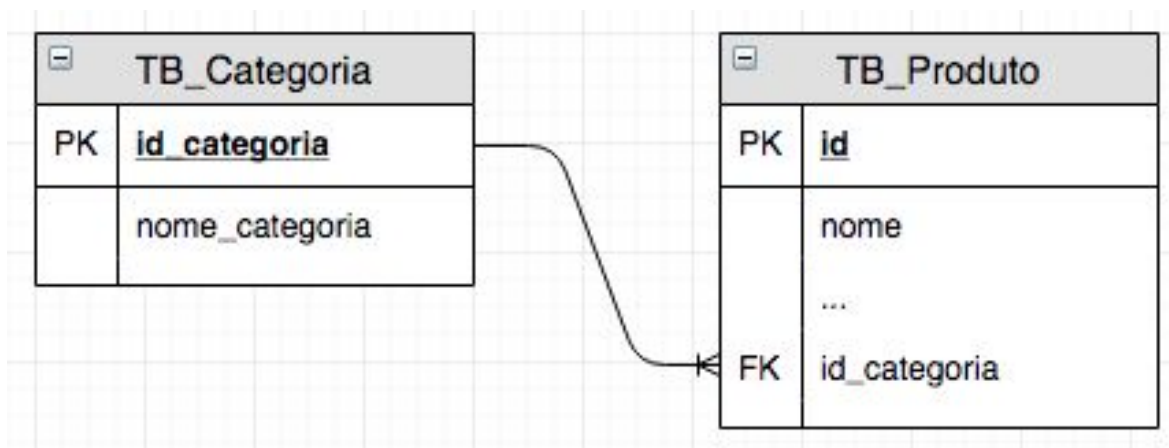
Revisão

Produtos vs. Categorias



Spring + Jdbc Template

- Relacionamento




- Depois de alterado nosso modelo, qualquer produto em nossa base de dados pode ter uma categoria.
- Podemos ter categorias sem produto, mas o ideal é não ter mais produtos sem categoria.

Spring + Jdbc Template

- Relacionamento

- Vamos tornar a categoria obrigatória dentro dos produtos em nosso modelo de dados.
- Adicione a opção de **NOT NULL** no id da categoria de produto e execute novamente os scripts:

```
CREATE TABLE tb_produto (  
    id                NUMBER(10) NOT NULL PRIMARY KEY,  
    nome              VARCHAR2(256),  
    sku               VARCHAR2(256),  
    descricao         VARCHAR2(256),  
    caracteristicas   VARCHAR2(256),  
    preco             NUMBER(10, 2),  
    id_categoria      NUMBER(10) NOT NULL,  
    CONSTRAINT FK_ID_CATEGORIA FOREIGN KEY (id_categoria)  
);
```



Spring + Jdbc Template

- Edição de Produto - Protótipo



Produto

Nome:

SKU:

Categoria:

Descrição:



**Dúvida: O que acham da nossa experiência para o usuário?
Ficou bem fácil alterar a categoria de um produto, não?**

Spring + Jdbc Template

- Edição de Produto - Protótipo

Produto

Nome:

SKU:

Categoria:

Descrição:



Spring + Jdbc Template

- Edição de Produto - Protótipo

Produto

Nome:

Nome do produto 1

SKU:

sku-0001

Categoria:

✓ Notebook
Smartphone
TV
Tablet



Spring + Jdbc Template

Select ou Drop-down

- Exemplo de implementação com Spring:

Atributo do objeto modelo que receberá o valor

`<form:select path="categoria.idCategoria">`

`<form:options items="${categorias}" ← Lista de informações
itemValue="idCategoria" itemLabel="nomeCategoria" />`

`</form:select>`

Valor do item exibido

Valor exibido na tela





Exercício

Inserir Categoria no CRUD de Produtos



Spring + Jdbc Template

- Adicione a Categoria na classe de Produto:

```
public class ProdutoModel {  
  
    private long id;  
    private String nome;  
    private String sku;  
    private String descricao;  
    private double preco;  
    private String caracteristicas;  
    ➔ private CategoriaModel categoria;
```

- Não esqueça de atualizar o construtor, gets e sets!

Spring + Jdbc Template

- Altere as queries de consulta de Produto relacionando com categoria:

```
private static final String FIND_ALL = "SELECT * "  
    + "FROM TB_PRODUTO "  
    + "P INNER JOIN TB_CATEGORIA C "  
    + "ON C.ID_CATEGORIA = P.ID_CATEGORIA "  
    + "ORDER BY P.ID";  
  
private static final String FIND_BY_ID = "SELECT * "  
    + "FROM TB_PRODUTO "  
    + "P INNER JOIN TB_CATEGORIA C "  
    + "ON C.ID_CATEGORIA = P.ID_CATEGORIA "  
    + "WHERE P.ID = ?";
```


Spring + Jdbc Template

- Insira uma coluna de categoria na listagem de produtos:

Produtos

Novo Produto

Nome	Categoria	Preço	Lançamento	Ações
Nome do produto 2		2.99	22/04/2019	<div>Detalhes</div> <div>Editar</div> <div>Excluir</div>
Nome do produto 3		3.99	22/04/2019	<div>Detalhes</div> <div>Editar</div> <div>Excluir</div>

Spring + Jdbc Template

```
<tr>
  <th data-field="name">Nome</th>
  ➔ <th data-field="categoria">Categoria</th>
  <th data-field="preco">Preço</th>
  <th data-field="lançamento">Lançamento</th>
  <th class="actions" width="220">Ações</th>
</tr>
</thead>
<tbody>

<c:forEach items="{produtos}" var="produto">
  <tr>
    <td>${produto.nome}</td>
    ➔ <td>${produto.categoriaModel.nomeCategoria}</td>
    <td>${produto.preco}</td>
```

Spring + Jdbc Template



**Execute a aplicação,
perceba que as categorias
não aparecem na listagem,
MAS POR QUE?**

Spring + Jdbc Template

- Executando a consulta em modo debug, o objeto de **CategoriaModel** não foi carregado pelo **BeanPropertyRowMapper**:

```
▼ 🔍 "repository.findAll()"= ArrayList<E> (id=173)
  ▼ 🔍 elementData= Object[10] (id=175)
    ▼ 🔍 [0]= ProdutoModel (id=176)
      ▶ 📦 características= "Características do produto 1" (id=180)
      ➡ 📦 categoria= null
      ▶ 📦 descricao= "Descrição do Produto 1" (id=181)
      ▶ 📦 id= 1
      ▶ 📦 nome= "Nome do produto 1" (id=182)
      ▶ 📦 preco= 1.99
      ▶ 📦 sku= "sku-0001" (id=183)
```



Spring + Jdbc Template

- O **BeanPropertyRowMapper** não mapeia objetos complexos, como no caso de uma **CategoriaModel** dentro de **ProdutoModel** : (
- Para solucionar, podemos utilizar o **RowMapper** do JdbcTemplate!

RowMapper:

- A interface **org.springframework.jdbc.core.RowMapper <T>** é utilizada pelo JdbcTemplate para mapear as linhas de um ResultSet individualmente.
- As implementações dessa interface executam o trabalho de mapear cada linha para um objeto de resultado.

Spring + Jdbc Template





- Crie uma classe **ProdutoRowMapper** e informe no método de consulta:

```
public List<ProdutoModel> findAll() {  
    List<ProdutoModel> produtos = this.jdbcTemplate.query(FIND_ALL, new ProdutoRowMapper());  
    return produtos;  
}
```



- Implementação do **ProdutoRowMapper**:

```
public class ProdutoRowMapper implements RowMapper<ProdutoModel> {  
    @Override  
    public ProdutoModel mapRow(ResultSet rs, int rowNum) throws SQLException {  
        ProdutoModel produto =  
            ➡ new BeanPropertyRowMapper<>(ProdutoModel.class).mapRow(rs, rowNum);  
  
        CategoriaModel categoria =  
            ➡ new BeanPropertyRowMapper<>(CategoriaModel.class).mapRow(rs, rowNum);  
  
        ➡ produto.setCategoriaModel(categoria);  
        return produto;  
    }  
}
```



Spring + Jdbc Template

- Execute a aplicação novamente e a categoria aparecerá!

Produtos

Novo Produto



Nome	Categoria	Preço	Lançamento	Ações		
Nome do produto 2	Smartphone	2.99	22/04/2019	Detalhes	Editar	Excluir
Nome do produto 3	TV	3.99	22/04/2019	Detalhes	Editar	Excluir



Spring + Jdbc Template

Agora é com você! Complete os demais itens do CRUD !



Spring + Jdbc Template

Produto

Nome: Nome do produto 2

Categoria: **Smartphone** ←

SKU: **sku-0002**

Descrição: **Descrição do Produto 2**

Preço: **2.99**

Características: **Características do produto 2**

Data de Lançamento: **22/04/2019**

Voltar

Produto

Nome:

Categoria: **Notebook** ▼ ←

SKU:

Produto

Nome:

Nome do produto 3

Categoria: **TV** ▼ ←

SKU:

sku-0002



Spring + Jdbc Template

Dicas:

- Na **ProdutoController**, utilize o método de listar todas as categorias para carregar o combo de categorias nas páginas de **inserção** e **edição**.
- **Identifique** em cada action do **ProdutoController** onde deve ser implementada as informações de categorias.
- Faça **passo a passo e teste (baby steps)**, implemente a ação de detalhe, depois o cadastro e a edição!
- **Revise o Repository, Controller e JSP** em cada implementação.
- **Utilize o debug sempre que estiver com problemas!**

Spring + Jdbc Template

O **JdbcTemplate** também oferece a opção de recuperar o resultSet completo da query, e o desenvolvedor controlar o de / para em seu objeto.

```
public List<CategoriaModel> getProductsByCategories() {  
    List<CategoriaModel> categorias = jdbcTemplate.query(GET_CATEGORIAS_PRODUTOS,  
        new ResultSetExtractor<List<CategoriaModel>>() {  
  
        public List<CategoriaModel> extractData(ResultSet rs) throws SQLException {  
            Map<Long, CategoriaModel> categorias = new HashMap<Long, CategoriaModel>();  
  
            while (rs.next()) {  
                Long categoriaId = rs.getLong("ID_CATEGORIA");  
  
                CategoriaModel categoriaModel = categorias.get(categoriaId);  
                if (categoriaModel == null) {  
                    categoriaModel = new CategoriaModel(  
                        categoriaId, rs.getString("NOME_CATEGORIA"), new ArrayList<ProdutoModel>());  
                    categorias.put(categoriaId, categoriaModel);  
                }  
  
                ProdutoModel produtoModel = new ProdutoModel(rs.getLong("id"), rs.getString("nome"));  
                categoriaModel.getProdutos().add(produtoModel);  
            }  
  
            return new ArrayList<CategoriaModel>(categorias.values());  
        }  
    });  
    return categorias;  
}
```

Spring + Jdbc Template

Aprendemos 3 formas de tratar o ResultSet com JDBC Template:

BeanPropertyRowMapper:

- Converte o ResultSet para objetos simples de forma automática.
- Desenvolvedor sem controle da implementação.



Trade OFF:

- **Maior produtividade**
- **Maior facilidade de implementação**
- **Menor autonomia do funcionamento**
- **Atende casos simples**

Spring + Jdbc Template

Aprendemos 3 formas de tratar o ResultSet com JDBC Template:

RowMapper:

- Converte o ResultSet linha por linha para objetos mais complexos.
- Desenvolvedor com controle parcial da implementação.



Trade OFF:

- Média produtividade
- Média facilidade de implementação
- Média autonomia de funcionamento
- Atende casos medianos

Spring + Jdbc Template

Aprendemos 3 formas de tratar o ResultSet com JDBC Template:

ResultSetExtractor:

- Converte o ResultSet inteiro para objetos mais complexos.
- Desenvolvedor com controle total da implementação.



Trade OFF:

- **Maior autonomia do funcionamento**
- **Atende casos complexos**
- **Menor produtividade**
- **Menor facilidade de implementação**



Spring + Jdbc Template

1 - Finalizar o CRUD !!!



HOMEWORK

Copyright © 2020

Profº. Pedro Ivo Correia e Profº. Lucas Furlaneto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).