05

Completando o CRUD

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/JSF/livraria-capitulo9.zip) do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.

Nesse capítulo, vamos completar a nossa aplicação e implementar as funcionalidades de alteração e remoção do livro. Ou seja, faremos um famoso **CRUD** (**C**reate, **R**ead, **U**pdate, **D**elete).

Implementando a exclusão

Vamos começar com a exclusão de um livro. Na página `livro.xhtml` já temos a lista de livros. Nela, adicionaremos uma nova coluna, onde ficará o link para remover:

```
<h:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">
    <h:column>
        <f:facet name="header">Título</f:facet>
        <h:outputText value="#{livro.titulo}" />
    </h:column>
    <h:column>
        <f:facet name="header">ISBN</f:facet>
        <h:outputText value="#{livro.isbn}" />
    </h:column>
    <h:column>
        <f:facet name="header">Preço</f:facet>
        <h:outputText value="#{livro.preco}">
            <f:convertNumber type="currency" pattern="R$ #0.00"
                currencySymbol="R$" locale="pt_BR" />
        </h:outputText>
    </h:column>
    <h:column>
        <f:facet name="header">Data</f:facet>
        <h:outputText value="#{livro.dataLancamento.time}">
            <f:convertDateTime pattern="dd/MM/yyyy"
                timeZone="America/Sao_Paulo" />
        </h:outputText>
    </h:column>
    <!-- NOVA COLUNA AQUI -->
    <h:column>
        <f:facet name="header">Remover</f:facet>
        <h:commandLink value="Remover" />
    </h:column>
</h:dataTable>
```

Acessamos a página e... Obtivemos o seguinte erro: **Remover: Este link está desativado, pois não está aninhado em um formulário JSF!** O que exatamente aconteceu?

Acontece que, novamente usamos o componente `<h:commandLink>` e, como qualquer outro comando (`UICommand`), ele deve ficar dentro de um `<h:form>` . Vamos colocar `<h:form>` no início e final da tabela:

```
<h:form id="formTabelaLivros">
    <h:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">

        <!-- outras colunas omitidas -->

        <h:column>
            <f:facet name="header">Remover</f:facet>
            <h:commandLink value="Remover" />
        </h:column>
    </h:dataTable>
</h:form>
```

Mas antes, devemos fazer uma pequena alteração no botão "Gravar", já que agora, para referenciar a tabela `tabelaLivros`, precisamos dizer que agora esta tabela está dentro do formulário `formTabelaLivros`:

```
<h:commandButton value="Gravar" action="#{livroBean.gravar}">
    <f:ajax execute="@form" render="@form :formTabelaLivros:tabelaLivros" />
</h:commandButton>
```

Agora, com o "Remover" visível, falta chamar um método no `h:commandLink` que realmente apague o livro no banco de dados, através do atributo `action`:

```
<h:form id="formTabelaLivros">
    <h:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">

        <!-- outras colunas omitidas -->

        <h:column>
            <f:facet name="header">Remover</f:facet>
            <h:commandLink value="Remover" action="#{livroBean.remover(livro)}" />
        </h:column>
    </h:dataTable>
</h:form>
```

Mas o método `remover` ainda não existe, vamos implementá-lo na classe `LivroBean`:

```
public void remover(Livro livro) {
    System.out.println("Removendo livro " + livro.getTitulo());
    new DAO<Livro>(Livro.class).remove(livro);
}
```

Ótimo! Já conseguimos remover um livro da lista de livros cadastrados, mas não podemos esquecer (como o instrutor no video) remover o livro da lista de livros guardada no bean:

```
public void remover(Livro livro) {
    System.out.println("Removendo livro " + livro.getTitulo());
    new DAO<Livro>(Livro.class).remove(livro);
    this.livros.remove(livro); //removendo da lista
}
```

E a alteração?

Com a remoção implementada, o próximo passo é fazer a alteração dos dados de um livro. Para isso, devemos ter uma nova coluna com um novo link, semelhante à coluna "Remover", que, ao ser clicado, carregue os dados do livro em questão no formulário. Após isso nós gravamos o formulário e salvamos as alterações.

Vamos dividir esse processo em dois passos, o primeiro é criar a coluna com o link que carregue os dados do livro no formulário; o segundo é como iremos gravar essas alterações do livro.

Começando com o primeiro passo, criamos mais uma coluna em `livro.xhtml` :

```
<h:form id="formTabelaLivros">
    <h:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">

        <!-- outras colunas omitidas -->

        <h:column>
            <f:facet name="header">Remover</f:facet>
            <h:commandLink value="Remover" action="#{livroBean.remover(livro)}" />
        </h:column>
        <h:column>
            <f:facet name="header">Alterar</f:facet>
            <h:commandLink value="Alterar" action="#{livroBean.carregar(livro)}" />
        </h:column>
    </h:dataTable>
</h:form>
```

Mas mais uma vez precisamos implementar o método `carregar` em `LivroBean` , já que o mesmo ainda não existe:

```
public void carregar(Livro livro) {
    System.out.println("Carregando livro " + livro.getTitulo());
    // O que fazer?
}
```

Mas como mostrar no formulário os dados do livro passado por parâmetro no método `carregar` ? Vamos lembrar que na classe `LivroBean` temos um atributo `livro` , e esse atributo é o livro do nosso formulário! Logo, basta fazer com que esse atributo da classe receba o livro passado por parâmetro, assim os dados desse livro serão colocados no formulário:

```
public void carregar(Livro livro) {
    System.out.println("Carregando livro " + livro.getTitulo());
    this.livro = livro;
}
```

Agora podemos recarregar a página `livro.xhtml` . Mas ao fazer isso recebemos outro erro! Uma exceção do tipo `LazyInitializationException` . Como resolver essa exceção?

Ajustando o relacionamento

O problema está no relacionamento dos livros com os seus autores. Um livro pode ter muitos autores, e um autor pode escrever vários livros, por isso a relação deles é de `many-to-many` (muitos para muitos). No Hibernate, qualquer relação

`*ToMany` é **LAZY**, isso porque essas relações `*toMany` são provavelmente mais custosas, trazendo mais objetos para a memória. Ok, mas o que isso interfere no nosso projeto?

Isso significa que quando carregamos os livros, os autores não são carregados ao mesmo tempo. Ou seja, quando clicamos em "Alterar", conseguimos carregar os dados do livro, mas não os do autor! Por isso ocorre a exceção.

Para resolver isso, podemos dizer para o Hibernate para, quando carregar um livro, automaticamente carregar os seus autores, ou seja, ao invés de **LAZY**, queremos que a relação seja **EAGER**. Então lá na classe `Livro`, adicionamos essa opção na anotação:

```
@ManyToMany(fetch=FetchType.EAGER)
private List<Autor> autores = new ArrayList<Autor>();
```

Ótimo, com isso conseguimos carregar todos os dados de um livro, primeiro passo concluído! Agora vamos alterá-lo e salvá-lo, o que acontece? Nada!!!

Salvando as alterações de um livro

Mas afinal, o que aconteceu? Visualizando a estrutura da página `livro.xhtml`, vemos que o botão "Gravar" chama o método de mesmo nome, da classe `LivroBean`. Vamos dar uma olhada nele:

```
public void gravar() {
    System.out.println("Gravando livro " + this.livro.getTitulo());

    if (livro.getAutores().isEmpty()) {
        FacesContext.getCurrentInstance().addMessage("autor",
                new FacesMessage("Livro deve ter pelo menos um Autor."));
        return;
    }

    new DAO<Livro>(Livro.class).adiciona(this.livro);

    this.livro = new Livro();
}
```

Por sua vez, o método `gravar` chama o método `adiciona`, da classe `DAO`:

```
public void adiciona(T t) {

    // consegue a entity manager
    EntityManager em = new JPAUtil().getEntityManager();

    // abre transacao
    em.getTransaction().begin();

    // persiste o objeto
    em.persist(t);

    // commita a transacao
    em.getTransaction().commit();
```

```
        // fecha a entity manager
        em.close();
    }
```

Se queremos alterar um livro, não deveríamos chamar o método `adiciona`, certo? E é exatamente isso que está ocorrendo. O método `adiciona` chama o método `persist` que **não funciona** com livros já cadastrados, com livros que já estejam no banco.

Para resolver isso, vamos voltar à classe `LivroBean` e modificar o método `gravar`. Se o livro não existir no banco, ou seja, se ele não tiver um `id` (for nulo), ele será adicionado no banco. Se o livro tiver um `id`, significa que ele já existe no banco, aí iremos **atualizá-lo**, chamando o método `atualiza` do `DAO`:

```
public void gravar() {
    System.out.println("Gravando livro " + this.livro.getTitulo());

    if (livro.getAutores().isEmpty()) {
        FacesContext.getCurrentInstance().addMessage("autor",
                new FacesMessage("Livro deve ter pelo menos um Autor."));
        return;
    }

    if (this.livro.getId() == null) {
        new DAO<Livro>(Livro.class).adiciona(this.livro);
    } else {
        new DAO<Livro>(Livro.class).atualiza(this.livro);
    }

    this.livro = new Livro();
}
```

Agora experimente alterar o preço de um livro, por exemplo, e tente salvar essa alterações. Funcionou? Segundo passo concluído!

Excluindo autores do livro

Para finalizar, falta um último detalhe no nosso formulário. Caso adicionemos um autor erradamente, temos como excluí-lo? Atualmente não. Vamos então implementar essa funcionalidade, a remoção de autor.

A primeira coisa que devemos fazer é mexer no formulário, adicionando um "X" ao lado do autor, um link para removê-lo. Esse link chama o método `removerAutorDoLivro`, da classe `LivroBean`:

```
<h:dataTable value="#{livroBean.autoresDoLivro}" var="autor" id="tabelaAutores">
    <h:column>
        <h:outputText value="#{autor.nome}" />
    </h:column>
    <h:column>
        <h:commandLink value="X" action="#{livroBean.removerAutorDoLivro(autor)}" />
    </h:column>
</h:dataTable>
```

Como o método `removerAutorDoLivro` ainda não existe, vamos implementá-lo, lá na classe `LivroBean`. A classe `Livro` tem um método que retorna todos os autores, então vamos acessar esse método e excluir o autor passado por parâmetro, através do método `remove`:

```
public void removerAutorDoLivro(Autor autor) {
    this.livro.getAutores().remove(autor);
}
```

Por questões de boas práticas, queremos que o método `removerAutorDoLivro` só remova o autor, sem precisar acessar a lista de autores do livro, vamos destinar essa responsabilidade de acessar a lista de autores para a classe `Livro`:

```
public void removeAutor(Autor autor) {
    this.autores.remove(autor);
}
```

E o método `removerAutorDoLivro`, da classe `LivroBean`, passará a chamar este método:

```
public void removerAutorDoLivro(Autor autor) {
    this.livro.removeAutor(autor);
}
```

O que aprendemos

- O que é CRUD;
- Remover o livro através do h:commandLink;
- Relacionamento EAGER;
- Alterar o livro e exclusão do autor;