Desenvolvimento Web II

Aula 11 - Projeto (Parte 2): Incorporando o Banco de Dados

Apresentação

Na aula anterior, você começou a desenvolver os requisitos necessários para poder incorporar um banco de dados em sua livraria virtual. Nesta aula, você completará essa tarefa, substituindo o que era feito somente em memória (útil apenas como aprendizado), para que seja feito usando banco de dados (solução real).Para iniciar esta aula, tenha certeza que você completou todos os requisitos da aula anterior. Nesta aula, você continuará, por conta própria, a implementar melhorias na sua livraria virtual da disciplina de Web I. O objetivo desta aula é realizar a integração com o banco de dados, portanto os recurso extras adicionados podem ser adicionados a vontade.

Boa aula!

Objetivos

- Substituir o armazenamento dos livros em memória por banco de dados.
- Implementar a funcionalidade de buscar produto (por palavra-chave).
- Cadastrar no banco de dados as informações da compra.

Substituir o Uso de Armazenamento em Memória

Como você deve se lembrar, a implementação que usamos como exemplo ao longo das aulas da disicplina de Web I utilizava uma estrutura de lista (em memória) para armazenar os livros e assim permitir operações como busca de um livro, listagem de livros, compra etc.

Como pode ser observado, isso era feito na classe Livraria onde o método popularLivros() era responsável por criar livros fictícios e armazená-los na lista, durante a inicialização do sistema (criação através do construtor).

Essa abordagem só é útil como exemplo de implementação de sistemas, pois não trata de maneira correta algumas situações bem comuns do dia a dia de um sistema, como as descritas a seguir.

Vídeo 02 - Inserir Banco de Dados

1. Necessidade de reiniciar o sistema, por problemas na máquina ou manutenções no *software*. Nesse caso, todos os dados de quantidade em estoque seriam perdidos e o sistema seria reinicializado com as quantidade originais.

2. Aquisição de novos produtos não é possível. Como a criação dos livros e o armazenamento são feitos na inicialização do sistema e estão fixos no código, novos produtos não podem ser incluídos no sistema sem que o código seja modificado e, consequentemente, reiniciado.

Como pode ser notado, a utilização somente de memória para armazenamento de informações de uma loja virtual não é nem um pouco adequada. Em sistemas reais, esses dados são armazenados em um meio de armazenamento não volátil.

Os bancos de dados servem como solução ideal para boa parte dos sistemas desse tipo, pois, além de evitar que os dados se percam, possuem diversas outras facilidades de acesso a dados, como segurança e integridade de dados.

O primeiro passo que você deve fazer para usar um banco de dados é, ao invés de armazenamento em memória, modificar a sua classe que representa a Livraria, para que ela use a classe RepositorioLivrosJDBC que desenvolvemos na aula passada.

O exemplo a seguir mostra parcialmente como seria a mudança no caso do nosso exemplo de Livraria.

```
1  ...
2  import livraria.bd.RepositorioLivrosJDBC;
3  ...
4  public class Livraria {
5
6      private RepositorioLivrosJDBC estoqueLivros;
7
8      public Livraria() {
9          estoqueLivros = new RepositorioLivrosJDBC();
10     }
11     ...
12 }
```

Como você deve ter notado, substituímos o tipo List<Livro> pelo RepositorioLivrosJDBC. Para completar a substituição, precisamos alterar os métodos da classe Livraria para que eles usem os métodos corretos da classe RepositorioLivrosJDBC.

Os dois primeiros métodos a serem modificados (getLivro() e getLivros()) podem ser vistos a seguir:

```
public List<Livro> getLivros() {
    List<Livro> livros;

    livros = estoqueLivros.getLivros();

    return livros;
}

public Livro getLivro(String idLivro) throws LivroNaoEncontradoException{
    Livro livro;

    livro = estoqueLivros.getLivro(idLivro);

    return livro;

}
```

Além dessa mudança, precisamos alterar o método comprarLivro(), pois ele altera a quantidade de livros do estoque acessando o objeto Livro e alterando o valor do seu atributo, e tudo isso só tem efeito em memória. Dessa forma, você precisará implementar um método para atualizar os dados de um livro no banco de dados, inclusive a sua quantidade em estoque. Esse método, como mostrado no código a seguir, tem assinatura void atualizarLivro(Livro):

```
public void comprarLivro(String idLivro, int qtdComprada) throws CompraException {
    Livro livroSelecionado;
    try {
        livroSelecionado = getLivro(idLivro);

        int qtdEstoque = livroSelecionado.getQuantidade();

        if ((qtdEstoque - qtdComprada) >= 0) {
            int novaQtd = qtdEstoque - qtdComprada;
            livroSelecionado.setQuantidade(novaQtd);
        } else {
                throw new CompraException("Livro " + idLivro
                + " sem estoque suficiente.");
        }
        estoqueLivros.atualizaLivro(idLivro, livroSelecionado);
    } catch (LivroNaoEncontradoException e) {

        throw new CompraException(e.getMessage());
    }
}
```

Vídeo 03 - Inserir Banco de Dados Servlet

Como você vem fazendo as mudanças no projeto base da Livraria então já é possível nesse momento reiniciar o servidor e acessar novamente o sistema pela URL <http://localhost:8080/livraria/livros/catalogo?Add=>e você verá a lista de livros, porém esses livros estão vindo do banco de dados. Para verificar basta fazer alguma alteração no título de um livro (por exemplo) no banco de dados e atualizar a página para ver a mudança.

Atividade 01

1. Realize os passos apresentados para integração do banco de dados no sistema de livraria virtual.

Implementar a Busca de Produtos

Na implementação atual da livraria, quando o usuário inicia as compras, todos os livros são listados e a partir daí ele pode selecionar qual dos livros ele deseja adicionar no seu carrinho de compras.

O problema dessa funcionalidade é que em sistemas reais existem milhares de produtos a serem vendidos e mostrar todos na página principal não é uma opção muito adequada. A solução para esse problema é implementar algo parecido com o que é feito em sites similares, tais como http://www.livrariacultura.com.br/ e http://www.amazon.com/. Nesses dois casos, como você pode perceber, existe um campo de texto onde o usuário pode digitar uma palavra-chave e buscar. A lista resultante só vai conter itens que contenham a palavra-chave informada pelo usuário.

Nessa parte do projeto, o objetivo é implementar essa mesma funcionalidade. Iremos sugerir alguns passos que devem ser feitos, mas não iremos indicar a solução completa. Por isso, usem os seus conhecimentos e um pouco de criatividade para implementar esse novo requisito da maneira como você julgar mais interessante.

Vídeo 04 - Inserir Banco de Dados JSP

Como sugestão, você pode se orientar pelos passos seguintes.

1. Adicionar um método public List<Livro> buscarLivros(String palavra) na sua classe RepositorioLivrosJDBC que realiza a consulta no banco de dados pela palavra-chave e retornar a lista de livros cujo título contém essa palavra (você pode incrementar esse recurso buscando por outros campos além do título):

```
public List<Livro> buscarLivros(String palavra) {
   List<Livro> livros = new ArrayList<Livro>();
      try {
         PreparedStatement st;
         st = this.conexaoBD.prepareStatement("SELECT * FROM livros WHERE titulo LIKE ?");
         st.setString(1, "%" + palavra + "%"); // Adicionando % no início e fim da palavra faz com esse t
         ResultSet result = st.executeQuery();

         while (result.next()) {
            Livro livro = new Livro();

            livro.setIdLivro(result.getString("idLivro"));
            livro.setTitulo(result.getString("titulo"));
            livro.setAutores(result.getString("autores"));
            livro.setAno(result.getInt("ano"));
            livro.setPreco(result.getDouble("preco"));
            livro.setQuantidade(result.getInt("quantidade"));
            livro.setDescricao(result.getString("descricao"));

            livros.add(livro);
         }
      } catch (SQLException e) {
         e.printStackTrace();
      }

   return livros;
}

```

2. Na classe Livraria, adicione um novo método que, simplesmente, chama o método criado no passo 1.

```
public List<Livro> buscarLivros(String palavra) {
   List<Livro> livros;

   livros = estoqueLivros.buscarLivros(palavra);

   return livros;
}

```

Repare que os livros resultantes da busca agora são adicionados no request scope (request.setAttribute("livros", livros);) portanto isso deve ser levando em conta no JSP que o vai exibir (catalogo.jsp)

3. No ServletControlador adicione o seguinte trecho de código relacionado ao tratamento da ação “/livros/catalogo”.

```
if (acaoSelecionada.equals("/livros/catalogo")) {
    idLivro = request.getParameter("Add");

    if (idLivro != null && !idLivro.equals("")) {
        try {
            livro = livraria.getLivro(idLivro);
            carrinho.adicionar(livro);
        } catch (LivroNaoEncontradoException ex) {
        }
    }
    else {

        String palavra = request.getParameter("buscar");

        if (palavra != null && !palavra.equals("")) {
            List<Livro> livros = livraria.buscarLivros(palavra);
            request.setAttribute("livros", livros);
        }
    }

}

```

Como pode ser visto, nessa ação modificamos o primeiro teste condicional, pois existirão situações em que o parâmetro Add será null. Além disso, incluímos uma segunda parte da condição indicando que, quando não for uma adição de um item ao carrinho, pode ser uma busca. Neste caso, fazemos a consulta na livraria (usando o método que foi criado no passo anterior) e depois adicionamos o resultado da busca como atributo do request para que o jsp possa recuperar essa lista ao montar a página.

4. No catalogo.jsp, modifique a linha do loop (<c:forEach>) de modo que, ao invés de recuperar todos os livros, ele acesse o objeto que foi adicionado no escopo do request.

```
<c:forEach var="livro" begin="0" items="${requestScope.livros}">
```

Implementadas essas quatro modificações, o requisito de filtrar os livros na página principal estará praticamente atendido. Dessa forma quando você simplesmente acessar o catálogo nenhum livro aparecerá (pois não foi feita uma busca), porém se você adicionar na URL buscar=ALGUMA_PALAVRA o resultado da busca será exibido. Teste por exemplo com a URL: http://localhost:8080/livraria/livros/catalogo?Add=&buscar=Desenvolvendo

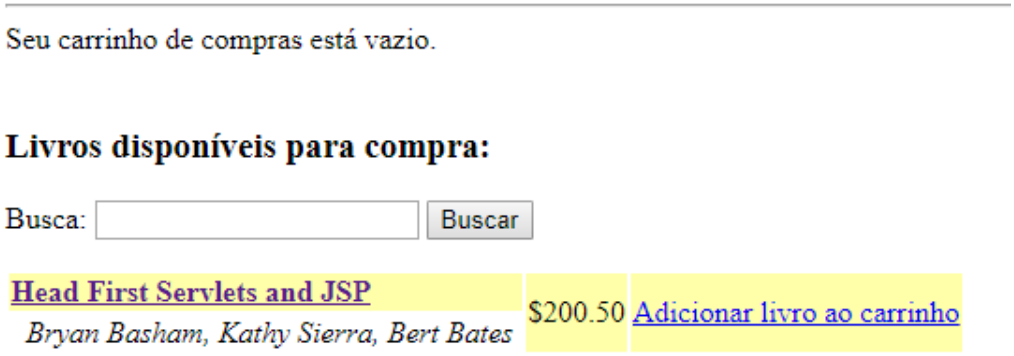
Você pode ainda alterar o catalog.jsp e adicionar um formulário de busca para que não seja necessário sempre digitar a palavra que se deseja na URL. Adicione antes da tabela que lista os livros o seguinte form:

```
<form method="GET" >
   <input type="hidden" name="Add" value="" />
   Busca: <input type="text" name="buscar" />
   <input type="submit" value="Buscar" />
</form>

```

Repare que o form tem o método “GET” e a omissão do seu action faz com que ele seja a própria página atual (catalogo.jsp) e além do campo “buscar” existe um campo escondido chamado “Add” sem valor para que o sistema funcione como o esperado, ou seja, sem adicionar nenhum livro no carrinho somente pelo fato de se realizar uma busca. O resultado da busca por “Head First” deve retornar somente 1 livro:

Figura 01 - Resultado da busca



Ainda são muitas as possibilidades que podem ser implementadas para melhorar o sistema da livraria. Vale lembrar estamos explorando uma integração básica de um sistema com banco de dados e que para se criar um sistema completo para o mercado muitos recursos precisam ser implementados utilizando os conhecimentos que você está aprendendo.

Atividade 02

1. Realize os passos apresentados para integração do banco de dados no sistema de livraria virtual.
2. Altere o código da livraria virtual para avisar ao usuário quando a sua busca não retornar nenhum resultado (diga que ele deve melhorar a palavra-chave utilizada).

Resumo

Na aula de hoje, você continuou a integração do seu sistema com um banco de dados e depois começou a implementar algumas funcionalidades para tornar o seu sistema mais interessante. Na próxima aula, você irá implementar mais funcionalidades, tornando a sua livraria ainda mais próxima de um sistema real de comércio eletrônico.

Até lá!

Referências

JAVASCRIPT and HTML DOM reference. Disponível em: <http://www.w3schools.com/jsref/>. Acesso em: 27 ago. 2012.

STATUS CODE DEFINITIONS. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>. Acesso em: 27 ago. 2012.

W3C. Disponível em: <http://www.w3.org/>. Acesso em: 27 ago.2012.

W3SCHOOL. Disponível: <http://www.w3schools.com/>. Acesso em: 27 ago.2012.

XMLHTTPREQUEST. Disponível em: <http://www.w3.org/TR/XMLHttpRequest/>. Acesso em: 27 ago. 2012.