CRUD no Banco de Dados com Java e JDBC - parte 1

A maioria das aplicações web precisam guardar seus dados em um banco de dados. Se você está iniciando em java, esse tutorial vai explicar como realizar um CRUD (Create Read Update Delete) através de uma aplicação web de exemplo. Utilizaremos nesse exemplo alguns padrões de projeto como MVC, DAO e Front Controller. Explicarei o que cada um destes significa e os beneficios resultantes de adotá-los. Neste exemplo além de mostrar como realizar operações no banco mostrarei também como estruturar seu código corretamente para a realização de transações (uma transação são várias consultas executadas para um determinado objetivo. Se uma consulta falhar, a transação garante que as outras sejam canceladas/desfeitas).

Espera-se que você já possua algum conhecimento em java e orientação a objetos, bem como html e servlets/jsp.

Resumo da aplicação:
Esta aplicação exemplo será um sistema de estoque, onde poderemos cadastrar, listar, atualizar e deletar um produto.

Este tutorial está dividido em duas partes: a primeira é a criação do código que acessará o banco de dados, e a segunda parte será o código que utilizará a classe criada na primeira parte.

Primeiro passo: Criação do banco de dados.
Abra seu cliente sql favorito para o seu banco de dados e crie a tabela de produtos. Supondo que você esteja usando o mysql pode ser assim o scritp:

```
CREATE TABLE `produto` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `nome` text NOT NULL,
  `descricao` text NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB
```

Segundo Passo: Criando sua classe que represente essa tabela.
Voce precisa de uma Classe-Entidade para seu sistema, a fim de desfrutar dos benefícios da orientação a objetos.

```
public class Produto{
private int id;
private String nome;
private String descricao;

//getters e setters
public void setId(int id){
   this.id = id;
}

public int getId()
   return this.id;
}

public void setNome(String nome){
   this.nome = nome;
}

public String getNome(){
```

```
   return this.nome
}

public void setDescricao(String  descricao ){
   this. descricao =  descricao ;
}

public String getDescricao(){
   return this.descricao;
}

}//fim da classe Produto
```

Terceiro Passo: Criando seu DAO
DAO (Data Access Object) se refere a classe que irá fazer a comunicação de seu sistema com o banco de dados. Esse padrão é muito importante. Ele promove a reutilização de código, também promove a boa manutenção do código pois ele não deixa que o código de acesso ao banco de dados fique misturado com outras partes da aplicação que possuem outros objetivos.

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Connection;
import java.util.List;
import java.util.ArrayList;


public class ProdutoDAO{

private Connection con;//objeto connection que será usado nos métodos abaixo

/*
* Construtor que recebe como parametro uma conexao com o banco de dado.
*/
public ProdutoDAO(Connection con){
   this.con = con;
}

public void cadastrar(Produto produto) throws Exception {
   PreparedStatement p =
   con.prepareStatement("insert into produto (nome, descricao) values (?,?)");
   p.setString(1, produto.getNome());
   p.setString(2, produto.getDescricao());
   p.executeUpdate();
   p.close();
}

public void deletar(Produto produto) throws Exception {
   PreparedStatement p = con.prepareStatement("delete from produto where id = ?");
   p.setInt(1, produto.getId());
   p.executeUpdate();
   p.close();
}

public void update(Produto produto) throws Exception {
   PreparedStatement p =
   con.prepareStatement("update produto set nome = ?, descricao = ? where id = ?");
   p.setString(1, produto.getNome());
```

```
    p.setString(2, produto.getDescricao());
    p.setInt(3, produto.getId());
    p.executeUpdate();
    p.close();
  }

  public List<Produto> listarTodos() throws Exception{
    List<Produto> produtos = new ArrayList<Produto>();
    PreparedStatement p = con.prepareStatement("select * from produto");
    ResultSet rs = p.executeQuery();
    while(rs.next()){
      Produto produto = new Produto();
      produto.setId(rs.getInt("id"));
      produto.setNome(rs.getString("nome"));
      produto.setDescricao(rs.getString("descricao"));
      produtos.add(produto);
    }
    rs.close();
    p.close();
    return produtos;
  }

  }//fim da classe ProdutoDAO
```

Ok, está pronta toda nossa comunicação com o banco de dados.

Na próxima parte deste tutorial explicarei como criar um Servlet cliente de nosso DAO Quando eu digo cliente, quero dizer quem vai utilizar os serviços (métodos) providos pela classe ProdutoDAO. Esse cliente pode ser uma aplicação web ou desktop/swing. Vamos fazer um cliente web através de servlets e jsp com jstl.

Att
Gustavo Marques.

CRUD no Banco de Dados com Java e JDBC parte 2 - Criando seu Servlet

No post anterior expliquei como criar o seu DAO, ou seja, a camada de sua aplicação que irá acessar o banco de dados.

Vamos agora criar o nosso Servlet que irá utilizar os serviços oferecidos pelo nosso DAO.

Servlet

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
```

```
 *
 * @author User
 */
public class ListarProdutos extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //variável que controla para qual view rotear após a execução desse Servlet
        String view = "";
        //Vamos criar uma conexão com o banco de Dados para passar ao nosso Dao.
        //É esse servlet que irá gerenciar a conexão
        java.sql.Connection con = null;
        try {
            con = Conexao.fabricar();//padrão factory, classe descrita depois desse servlet.
            con.setAutoCommit(false);//permitindo transações
            ProdutoDao dao = new ProdutoDao(con);
            List<Produto> produtos = dao.listarTodos();
            con.commit();
            //manda para a view
            request.setAttribute("produtos",produtos);
            view = "listagem.jsp";
        } catch(Exception ex){
            view = "erro.jsp";
            try{
               con.rollback();
            }catch(Exception ex2){}
        }finally {
            try{
               con.close();
            }catch(Exception ex3){}
        }
        request.getRequestDispatcher(view).foward(request, response);
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the
left to edit the code.">
    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
```

```
    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Não esqueça de registrar seu Servlet no arquivo web.xml, logo abaixo da tag <web-app> inclua:

web.xml

```
  <servlet>
        <servlet-name>ListarProdutos</servlet-name>
        <servlet-class>servlets.ListarProdutos</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListarProdutos</servlet-name>
        <url-pattern>/ListarProdutos</url-pattern>
    </servlet-mapping>
```

Agora a classe Fábrica de Conexões:

Conexao.java

```
public class Conexao {

    public static Connection fabricar() throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        return DriverManager.getConnection("jdbc:mysql://localhost/banco" ,"usuario" ,"senha" );
    }
}
```

No próximo post implementaremos a View com JSP e JSTL.

Gustavo Marques.