# Java na WEB – Banco de Dados

**Objetivo:**
Ao final da aula o aluno será capaz de:

- Criar aplicações JSP com acesso ao Banco de Dados MySql
- Configurar o TomCat para acessar o MySql.

**Não é Objetivo:**
Ao final da aula o aluno não será capaz de:

- Desenvolver e implementar estruturas de Banco de Dados.
- Desenvolver instruções SQL complexas.

---

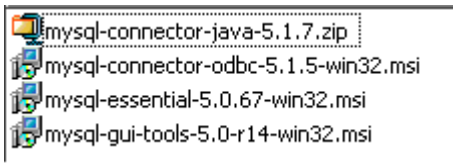
## Sumário



Prof. Vida
vida@utfpr.edu.br

# 9. O Banco de Dados MYSQL:

O servidor de Banco de Dados MySql é um servidor que pode ser utilizado para o desenvolvimento de aplicações para a web. O programa de instalação pode ser obtido a partir do web site da *MySQL AB*. Da página da *MySQL AB*, os seguintes arquivos necessitam ser baixados e instalados:




## *9.1 A instalação do MySQL.*

Recomenda-se que a instalação do MySQL seja feita no diretório `Arquivos de Programas` ou `Program Files`.

Durante a instalação do MySql, algumas perguntas são feitas. Pode-se utilizar os valores recomendados pelo programa de instalação.

As próximas figuras ilustram algumas telas que serão apresentadas durante o processo de instalação.

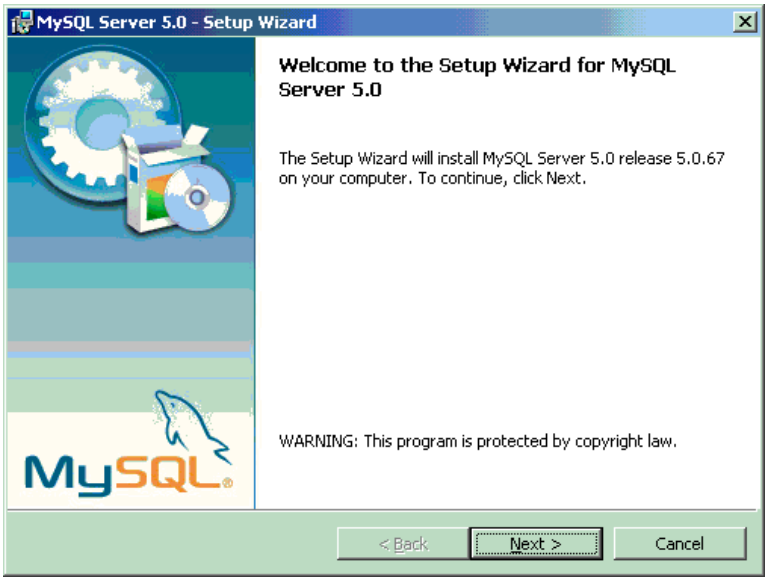



Figura 1-Ilustração da Tela Inicial da Instalação do MySQL

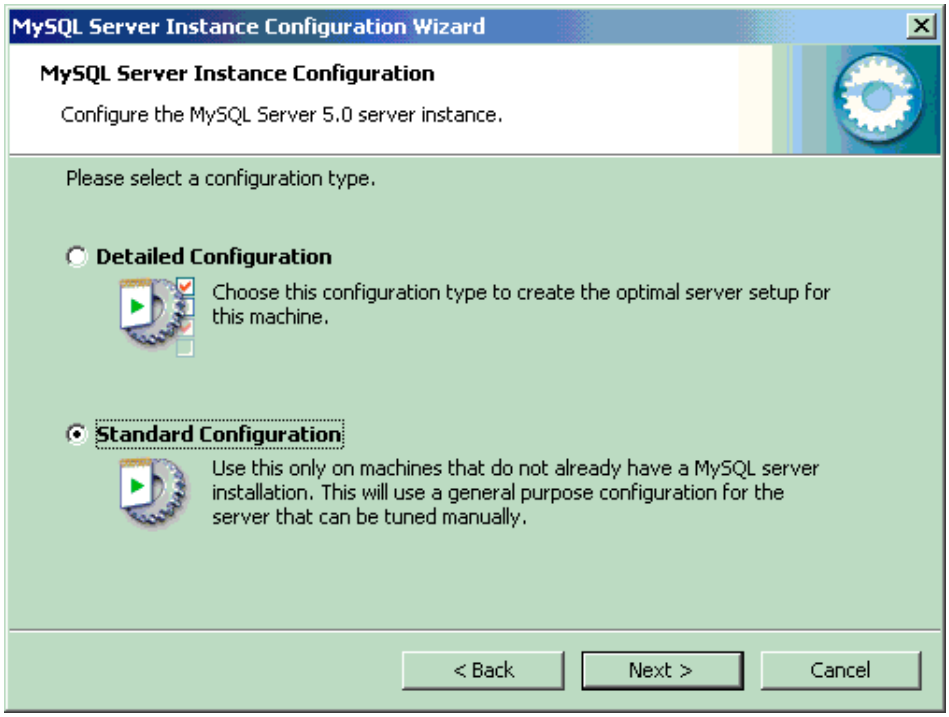



Figura 2-Interface que permite selecionar o tipo de Configuração

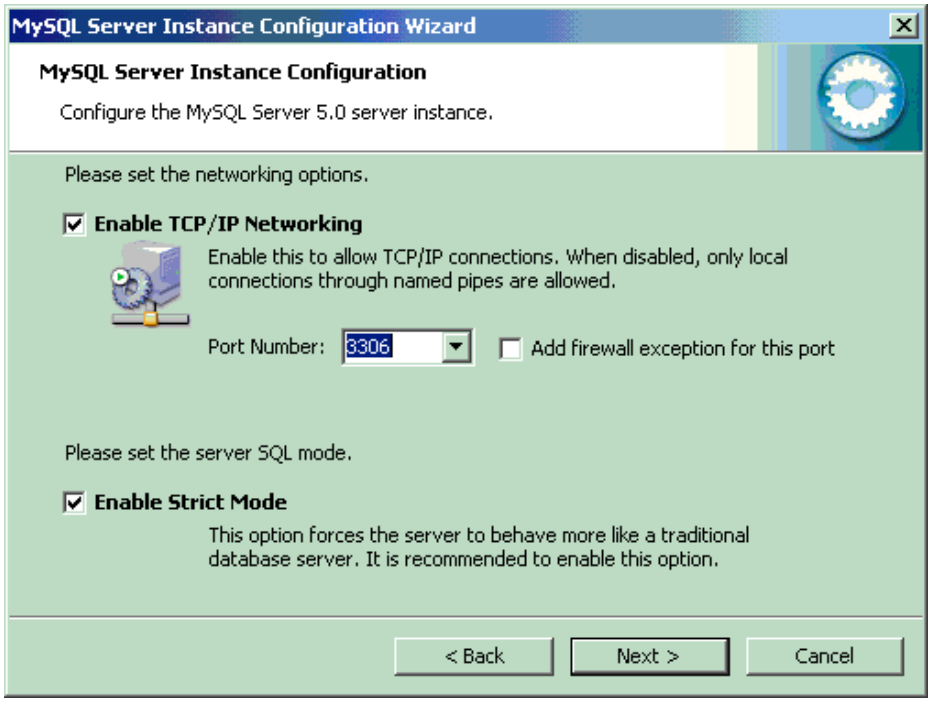



Figura 3-Interface que permite especificar qual a porta que será Monitorada pelo Servidor. O valor padrão é a 3306 que, se possível, deverá ser mantido.

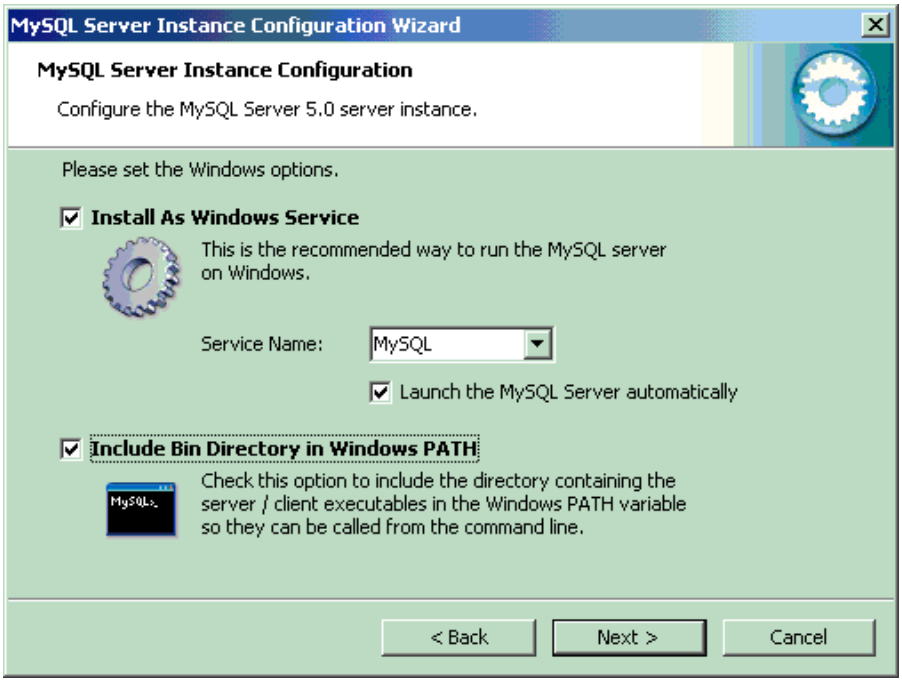



Figura 4-Interface que permite especificar se o serivodr será instalado como um serviço no Windows

O MySQL cria um usuário root cuja senha necessitará ser definida durante a instalação. Como sugestão, recomenda-se que **a senha também seja "root"**.

### *9.2 Ferramentas de Apoio*

Após ser feita a instalação do MySQL, recomenda-se a instalação das ferramentas de apoio ao MySQL (**mysql-gui-tools**) também seja feita no `Arquivos de Programas\MySql` ou `Program Files\MySql`.

Após as duas instalações, a seguinte estrutura deverá estar disponível:

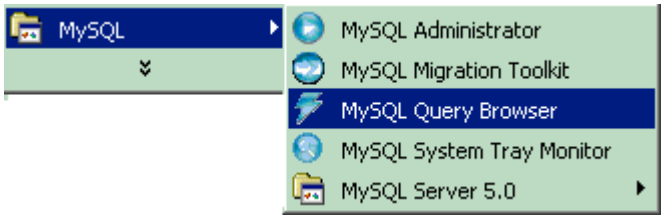
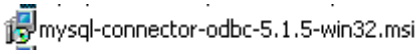
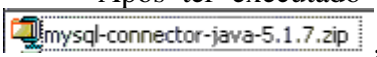



Figura 5-Esturutuda das Aplicaçãoes no Gerenciador de Programas.

### *9.3 Modificações no TomCat.*

Após ter executado o arquivo mysql-conector 

e 

, os conectores de acesso ao banco, o banco MySQL e as ferramentas de apoio estarão disponíveis na pasta `Arquivos de Programas\MySQL`, conforme ilustra a Figura 6.

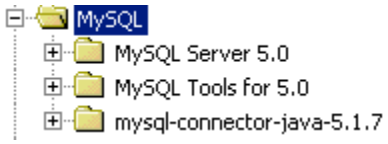



Figura 6-Sugestão de Estrutura de Diretórios

Um arquivo precisa ser transportado da pasta **mysql-connector-java-5.1.7** para a pasta commom\lib do TomCat.

Esse arquivo é `mysql-connector-java-5.1.7-bin.jar`, conforme destacado na figura a seguir.

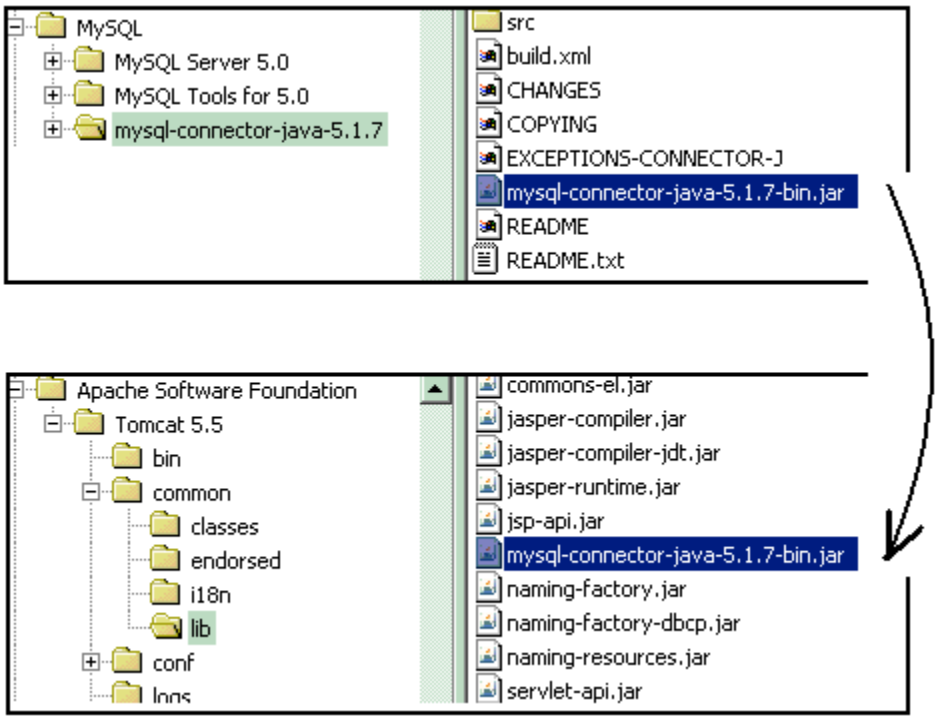



Figura 7-Local que o arquivo mysql-connector-java-5.1.7-bin.jar deve ser armazenado.

Esse arquivo possui as informações necessárias para o TomCat acessar o Banco de Dados MySql e deve estar localizado na pasta commom\lib.

### *9.4 Definição de Esquema*

O MySQL QueryBrowser permite a criação de esquemas, tabelas entre outras atividade do banco de dados. Com o botão direito do mouse, crie um novo esquema. Para manter a compatibilidade com o resto deste material de apoio, defina o nome do esquema para **lojinha**, conforme ilustrado a seguir.

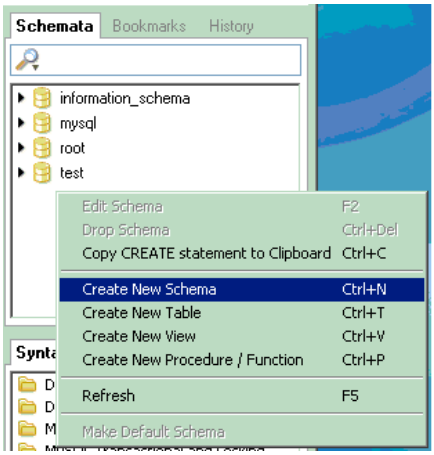



Figura 8-Definição do Esquema na Ferramenta MySQL QueryBrowser

### *9.5 Criação de Tabela*

Após ter criado o esquema, crie uma tabela conforme ilustrado a seguir:

```
create table produto

(codigo int unsigned not null auto_increment,

 nome varchar(100) not null,

 valor double ,

 primary key(codigo)

) engine=innodb default charset=latin1;
```

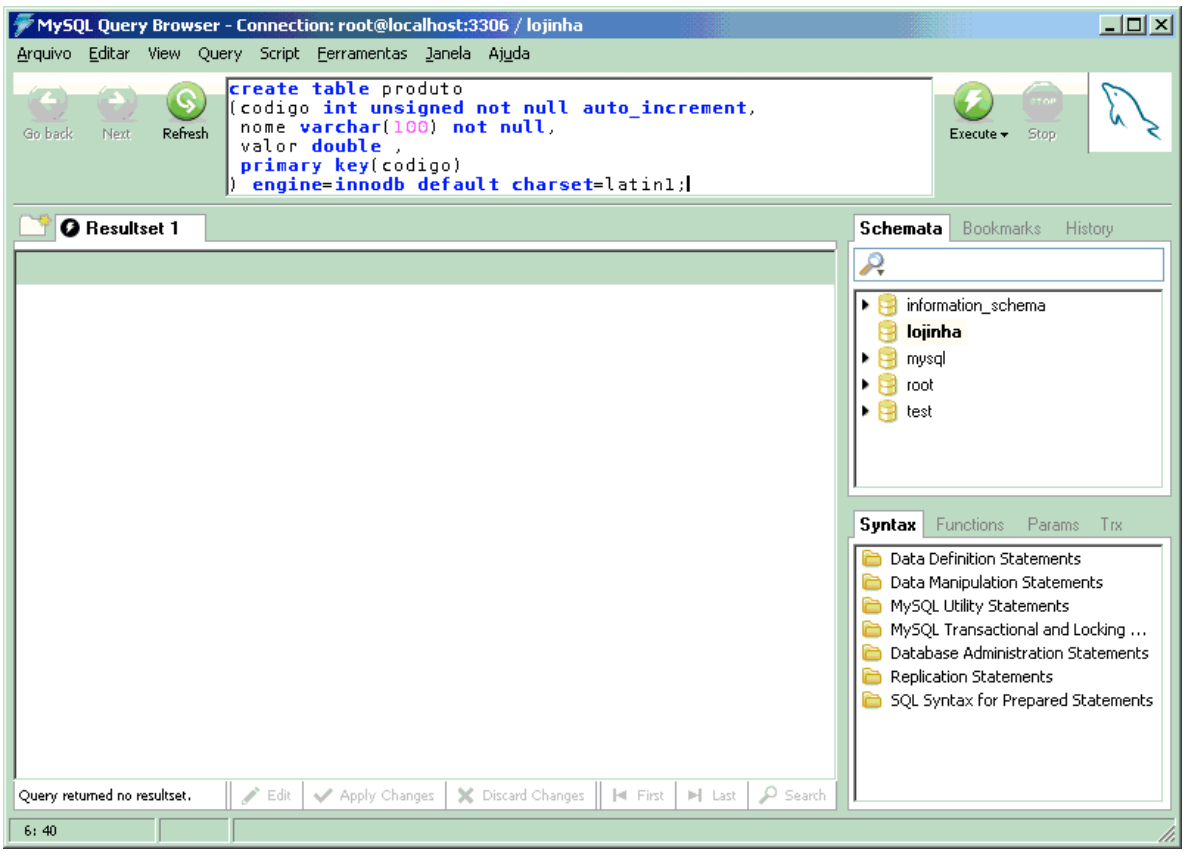



Figura 9-Definição de Tabela no MySQL QueryBrowser

Essa tabela será utilizada durante o transcorrer deste material de apoio ao estudo.

## 10. Acesso ao Banco de Dados.

Neste material será apresentado como páginas JSP podem acessar informações de um banco de dados. Para esse acesso, dois aspectos são importantes, a saber:

- **Classes para criação de Conexões:** `java.sql.DriverManager`, `java.sql.Driver` e `java.sql.Connection`.
- **Classes de Acesso ao Banco:** `java.sql.Statement` ou `java.sql.PrepareStatement` e `java.sql.ResultSet`.

Exemplos dessas classes são apresentados a seguir.

### *10. 1 Como acessar o banco de dados ?*

O acesso ao banco de dados pode ser realizado por uma classe Java específica, conforme o exemplo a seguir:

```
/*01*/ package javaNaWeb.BD;
/*02*/ import java.sql.*;
/*03*/ public class Conectar
/*04*/ {
/*05*/     private static Connection con;
/*06*/     private static Statement  stm;
/*07*/     private static String bd=
/*08*/             "jdbc:mysql://localhost:3306/lojinha";
/*09*/     private static String driver=
/*10*/             "com.mysql.jdbc.Driver";
/*11*/     private static String usuario="root";
/*12*/     private static String senha="root";

/*13*/     public static void open() throws Exception
/*14*/     {
/*15*/        Class.forName(driver);
/*16*/        con=
/*17*/        DriverManager.getConnection(bd,usuario,senha);
/*18*/        stm=con.createStatement();
/*19*/     }

/*20*/     public static void close() throws Exception
/*21*/     { if (closed()==false)
/*22*/         { stm.close();
/*23*/           con.close();}
/*24*/     }
```

```
/*25*/     public static boolean closed() throws Exception
/*26*/     {
/*27*/          return ((stm == null) && (con == null));
/*28*/     }

/*29*/     public static ResultSet executeQuery(String sql)
/*30*/                                  throws Exception
/*31*/     {
/*32*/          if (closed() ) open();
/*33*/          return stm.executeQuery(sql);
/*34*/     }
/*35*/     public static void setBD(String v){bd=v;}
/*36*/     public static String getBD(){return bd;}

/*37*/     public static void executeInsert(int codigo
/*38*/         , String nome, double valor)  throws Exception
/*39*/     {
/*40*/          String comando=
/*41*/    "insert into produto (codigo, nome, valor) values ("
/*42*/                          + codigo
/*43*/                          + ", \'" + nome + "\'"
/*44*/                          + "," + valor + ")";
/*45*/          executeMe(comando);
/*46*/     }

/*47*/     public static void executeUpdate(int codigo
/*48*/        , String nome, double valor)  throws Exception
/*49*/     {
/*50*/          String comando=
/*51*/           "update produto set nome=\'" + nome + "\'"
/*52*/                          + ", valor=" + valor
/*53*/                          + " where codigo=" + codigo;
/*54*/          executeMe(comando);
/*55*/     }

/*56*/     public static void delete(int codigo)
/*57*/                                    throws Exception
/*58*/     {
```

```
/*59*/           String comando="delete from produto where codigo="
/*60*/                          + codigo;
/*61*/           executeMe(comando);
/*62*/      }

/*63*/      private static void executeMe(String comando)
/*64*/                                    throws Exception
/*65*/      {
/*66*/          if (closed() ) open();
/*67*/            stm.executeUpdate(comando);
/*68*/       }
/*69*/ }
```

**Essa classe deverá ser compilada e armazenada na pasta `WEB-INF\classes\javaNaWeb\BD` do TomCat**.

### *10.2 Como emitir Relatórios em páginas JSP?*

A estratégia, nesse caso, é simples. Na página JSP, inserir na diretiva import a referência para a classe específica criada em 10.1 (`javaNaWeb.BD.*`) e para o pacote de acesso a bando de dados (`java.sql.*`).

Os métodos da classe `javaNaWeb.BD.Conectar` foram definidos com o atributo `static`, ou seja, não há a necessidade de instanciar essa classe para poder utilizar os seus métodos. Por exemplo, na linha /*09*/ do código a seguir, o método `open()` utilizado para abrir uma conexão com o banco, é chamado diretamente. Não há a necessidade de criar uma instância da classe `javaNaWeb.BD.Conectar` para depois chamar o método `open()`.

O objeto `ResultSet` , definido na linha /*10*/ do código a seguir, corresponde a uma tabela de dados representando um conjunto de resultados de uma pesquisa realizada no banco de dados. Por exemplo, essa pesquisa poderia ser realizada através da instrução genérica `select * from nomeDaTabela`.

Esse objeto `ResultSet` mantém um cursor apontando para a linha atual dos dados. Inicialmente, o cursor aponta para o (antes do) primeiro registro. O método `next()`

move o cursor para o próximo registro retornando `true` e esse método retorna `false` quando não existem mais registros.

Um dado pode ser acessado através da instrução get<tipo>(int) e o número da coluna. As linhas /*17*/, /*18*/ e /*19*/ do código a seguir acessam as informações da coluna cujo nome é "codigo" e das colunas 2 e 3 no formato String.

Os dados das colunas de uma tabela também poderiam ter sido acessados nos seguintes formatos entre outros:

- getArray(int ) / getArray(String)
- getBlob(int) / getBlob(String)
- getBoolean(int)/getBoolean(String)
- getByte(int) / getBytes(String)
- getDate(int) /getDate(String)
- getDouble(int) / getDouble(String)
- getFloat(int) / getFloat(String)
- getInt(int) / getInt(String)
- getLong(int) / getLong(String)
- getObject(int) / getObject(String)

```
<!--   arquivo relatorioBD.jsp -->
<!--01--> <%@ page import="java.sql.*"  %>
<!--02--> <%@ page import="java.util.*"  %>
<!--03--> <%@ page import="java.text.*"  %>
<!--04--> <%@ page import="javaNaWeb.BD.*"  %>


<!--05--> <html>
<!--06--> <body>

<!--07--> <%
/*08*/    String pesquisa = "select * from produto";
/*09*/    Conectar.open();
/*10*/    ResultSet rec;
/*11*/    rec = Conectar.executeQuery(pesquisa);

/*12*/    out.println("<center><table border=1><tr>"
/*13*/     + "<th> Codigo </th><th> Nome </th>"
/*14*/     + "<th> Valor </th></tr>");

/*15*/    while(rec.next())
/*16*/    {
/*17*/     out.println("<tr><td>"  + rec.getString("codigo")
/*18*/               + "</td><td>" + rec.getString(2)
/*19*/               + "</td><td>" + rec.getString(3)
```

```
/*20*/                 + "</td></tr>");
/*21*/     }
/*22*/     out.println("</table>");
/*23*/     Conectar.close();
/*24*/ %>
<!--25--> </body>
<!--26--> </html>
```

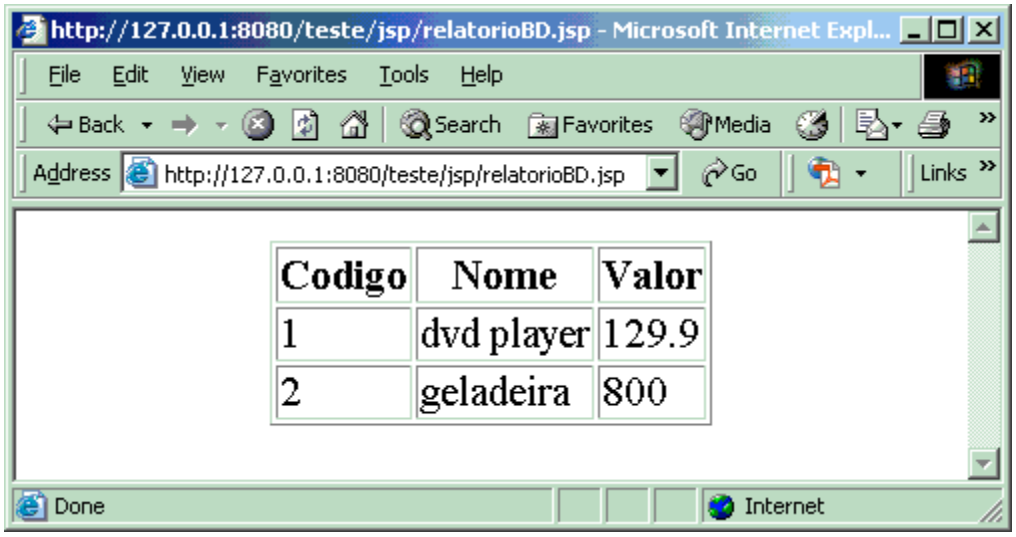



Figura 10-Exemplo de Execução do Arquivo relatorioBD.jsp

Outra opção de relatório é a definição de todo o código na página JSP. No exemplo a seguir, a classe `Statement` foi substituída pela `PreparedStatement` na linha 11.

A `PreparedStatement` é uma classe muito similar com a classe `Statement`. A instrução SQL fornecida nas linhas 20 e 21, **`pstm = con.prepareStatement("select * from produto where codigo = ?");`**, define uma variável com o símbolo "*?*".

A definição do valor dessa variável é definida na linha 22 do código a seguir. A instrução **`pstm.setInt(1,Integer.parseInt(codigo));`** define que a primeira variável deve ser setada como um valor inteiro e o seu valor é obtido da instrução `Integer.parseInt(codigo)`.

```
<!--   arquivo relatorioBDModificado.jsp -->
<!--01--> <%@ page import="java.sql.*"  %>
<!--02--> <%@ page import="java.io.*"  %>
          <%@ page import="java.util.*"  %>
          <%@ page import="java.text.*"  %>

<!--03--> <html><body><center>
<!--04--> <%!
/*05*/    private String bd="jdbc:mysql://localhost:3306/lojinha";
/*07*/    private String driver="com.mysql.jdbc.Driver";
/*08*/    private String usuario="root";
/*09*/    private String senha="root";
```

```
/*10*/    private Connection con;
/*11*/    private PreparedStatement  pstm;
/*12*/   %>
<!--13--> <%
/*14*/      String codigo = request.getParameter("codigo");
/*15*/      out.println("<h3> Dados do Produto #" + codigo + "</h3>");
/*16*/      if (codigo != null)
/*17*/      {
/*18*/        Class.forName(driver);
/*19*/        con=DriverManager.getConnection(bd,usuario,senha);

/*20*/        pstm= con.prepareStatement(
/*21*/                "select * from produto where codigo = ?");
/*22*/        pstm.setInt(1,Integer.parseInt(codigo));

/*23*/        ResultSet rs=pstm.executeQuery();
/*24*/        out.println("<center><table border=1>");
/*25*/        while(rs.next())
/*26*/        {
/*27*/         out.println("<tr><th>Codigo</th><td>"  + rs.getString(1)
/*28*/                 +"</td></tr>"
/*29*/                 + "<tr><th>Nome</th><td>" + rs.getString(2)
/*30*/                 + "</td></tr>"
/*31*/                 + "<tr><th>Valor</th><td>" + rs.getString(3)
/*32*/                 + "</td></tr>");
/*33*/        }
/*34*/        out.println("</table>");
/*35*/       pstm.close();con.close();
/*36*/      }
/*37*/      else
/*38*/      {
/*39*/       out.println("<h3> Faltou informar o código. </h3>");
/*40*/      }
/*41*/   %>
<!--42--> </body></html>
```

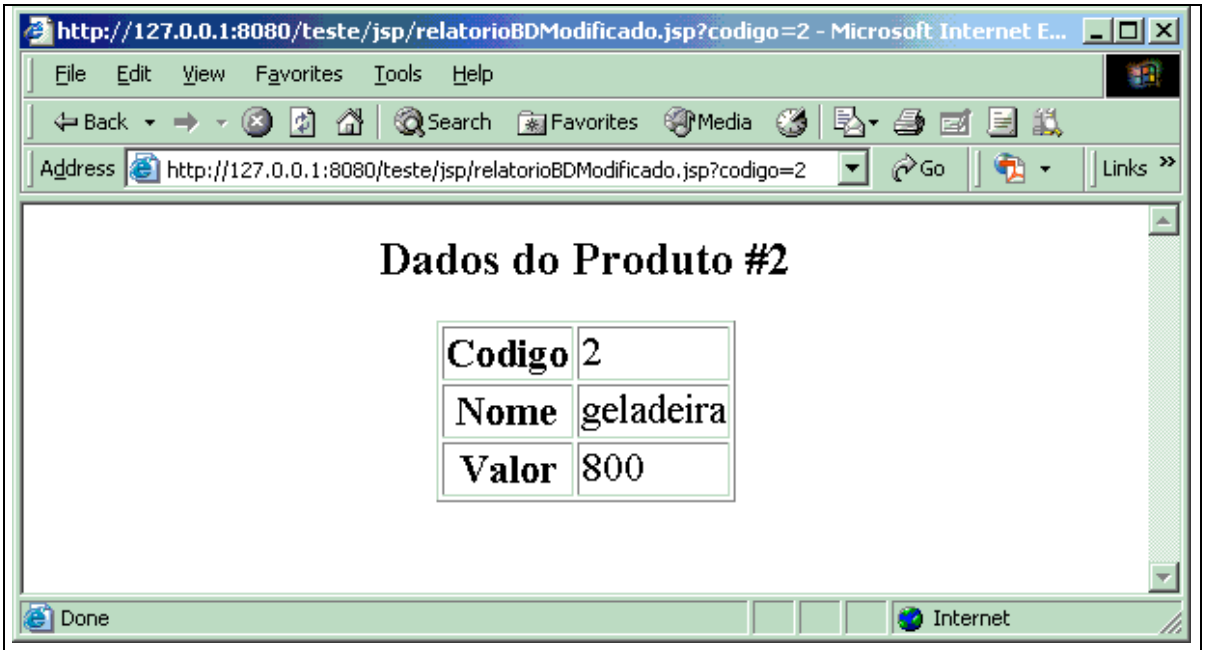



Figura 11-Exemplo de Execução do Arquivo relatorioBDModificado.jsp

### *10.3 Inserção de Registro*

A seguir, é apresentado um arquivo que permite a inserção de registro na base de dados. As linhas /*13, /*14*/ e /*15*/ chamam o método estático `executeInsert( )` definido na classe Conectar.

```
<!--- arquivo insereBD.jsp -->

<!-01--> <%@ page import="java.sql.*"  %>
<!-02--> <%@ page import="java.util.*"  %>
<!-03--> <%@ page import="java.text.*"  %>
<!-04--> <%@ page import="javaNaWeb.BD.*"  %>
<!-05--> <html>
<!-06--> <body>
<!-07--> <head><title> Resultado da Inserção</title></head>
<!-08--> <%
/*09*/    String codigo = request.getParameter("codigo");
/*10*/    String nome = request.getParameter("nome");
/*11*/    String valor = request.getParameter("valor");
/*12*/    Conectar.open();
/*13*/    Conectar.executeInsert(Integer.parseInt(codigo)
/*14*/                           , nome
/*15*/                           , Double.parseDouble(valor));
/*16*/    ResultSet rec;
/*17*/    rec = Conectar.executeQuery("select * from produto");

/*18*/    out.println("<center><table border=1>"
/*19*/     "<tr><th>Codigo</th><th>Nome</th><th>Valor </th></tr>"
/*20*/              );
/*21*/    while(rec.next())
/*22*/    {
/*23*/     out.println("<tr><td>"  + rec.getString("codigo")
```

```
/*24*/                 + "</td><td>" + rec.getString(2)
/*25*/                 + "</td><td>" + rec.getString(3)
/*26*/                 + "</td></tr>");
/*27*/     }
/*28*/     out.println("</table>");
/*29*/     Conectar.close();
/*30*/ %>
<!-31--> </body></html>
```

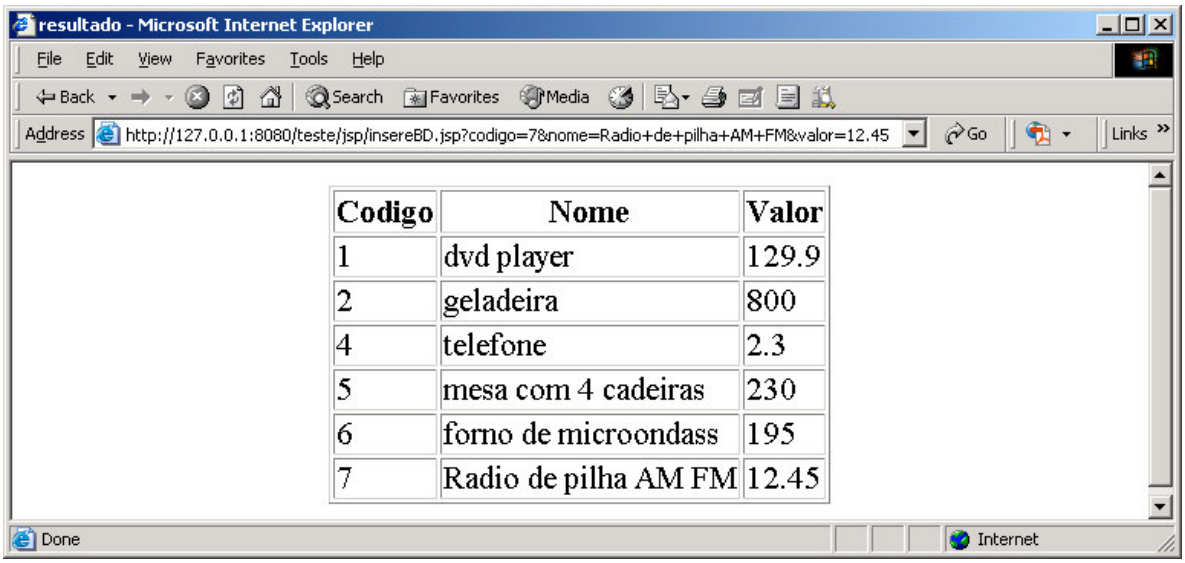



Figura 12-Exemplo de Execução do Arquivo insereBD.jsp

### *10.4 Exercício:*

Desenvolver os formulários para edição e deleção dos registros da tabela Produto.

**Obs:**

**Exemplo de Código de Edição:**

```
update produto set ( valor = 234 ) where codigo = 1
```

**Exemplo de Código de Exclusão:**

```
delete from produto  where codigo = 1
```