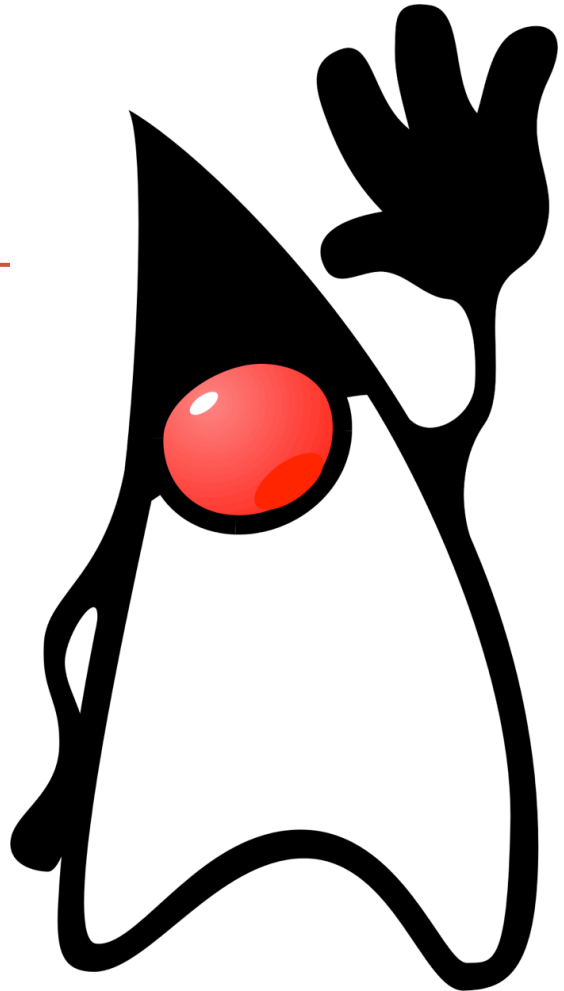
PROGRAMAÇÃO WEB

- Modelo Relacional e SQL
- JDBC para acesso a banco de dados com Java Server Pages



Prof. Luiz Carlos Querino Filho

luiz.querino@fatec.sp.gov.br

Fatec Garça – 2017

Parte 05

Modelo Relacional

- Um **banco de dados relacional** é, resumidamente, aquele em que os dados ficam armazenados em **tabelas**.
- As tabelas agrupam as informações em **linhas** (registros ou tuplas). Para cada linha, um ou mais dados são definidos em **colunas** (campos ou atributos).
- A **chave primária** de uma tabela é um (ou mais) campo(s) que garante(m) um valor único para cada linha. Ex.: o CPF ou um código de um cliente.
- O modelo relacional estabelece que essas tabelas podem estar associadas entre si por meio de **chaves estrangeiras**.
- Exemplo: a tabela de **Pedidos** possui registros relacionados com aqueles da tabela de **Clientes**.
- Como? Na tabela de **Pedidos**, a chave estrangeira *CodigoDoCliente* identifica o **Cliente** que fez o pedido.

Tabelas Clientes

Código	Nome	Telefone	Cidade
1	Luiz	3333-1111	Marília
2	Maria	2222-9898	Garça
3	Ana	4545-3222	Marília

Tabelas Pedidos

Numero	Cliente	Data	Valor
1001	1	12/01/2014	R$ 450,00
1002	1	20/01/2014	R$ 2000,00
1003	2	23/01/2014	R$ 670,00
1004	3	01/02/2014	R$ 125,80

SQL (Structured Query Language)

- A linguagem SQL é o padrão para **definição** e **manipulação** de bancos de dados relacionais modernos (como MySQL, Oracle, PostgreSQL, SQL Server, etc.)
- Os comandos para **definição** dos dados são usados para criar e mudar a **estrutura** de uma tabela.
- O comando `CREATE TABLE` é usado para **criar** uma tabela e definir seus **campos**.
- Os campos devem ser criados especificando seu nome e o tipo dos seus dados, como por exemplo:
  - `INT` para valores inteiros
  - `VARCHAR(tamanho máximo)` para valores alfanuméricos
  - `DECIMAL` para valores decimais

SQL (Structured Query Language)

- Sintaxe básica do comando de criação de uma tabela

```
CREATE TABLE nome_da_tabela (
        nome_campo_1 tipo opcoes,
        nome_campo_2 tipo opcoes,
        …
        nome_campo_N tipo opcoes,
        PRIMARY KEY (nome_campo_chave_primaria)
);
```

- As **opções** dependem do SGBD usado. Para o MySQL, as seguintes são usadas frequentemente:
  - `NOT NULL` para campos que não podem ficar vazios
  - `AUTO_INCREMENT` para campos do tipo `INT` com valor incrementado automaticamente.

SQL (Structured Query Language)

- Exemplo de `CREATE TABLE` para uma tabela de clientes (usando tipos SQL e opções do MySQL):

```
CREATE TABLE clientes (
    codigo INT NOT NULL AUTO_INCREMENT,
    nome VARCHAR(50),
    endereco VARCHAR(100),
    cidade VARCHAR(30),
    estado CHAR(2),
    email VARCHAR(30),
    telefone VARCHAR(15),
    PRIMARY KEY (codigo)
);
```

SQL (Structured Query Language)

- Para manipulação dos dados (consulta, inclusão, atualização e exclusão), são usados os comandos
  - `SELECT`
  - `INSERT`
  - `UPDATE`
  - `DELETE`
- Juntos, esses cinco comando permitem realizar o chamado CRUD no banco de dados:
- **C**reate (criar – inclusão com `INSERT`)
- **R**ead (ler – consultar com `SELECT`)
- **U**pdate (atualizar – alteração com `UPDATE`)
- **D**elete (excluir – exclusão com `DELETE`)

SQL (Structured Query Language)

- Usando **SELECT** para consultas:

```
SELECT * | campo1, …, campoN
FROM tabela
[WHERE condição]
[ORDER BY campo_para_ordenacao]
```

- Ex1: Retorna todos os registros da tabela de clientes:

```
SELECT * FROM clientes
```

- Ex2: Retorna todos os campos do cliente com codigo 101

```
SELECT * FROM clientes
WHERE codigo = 101
```

SQL (Structured Query Language)

- Incluindo registros com **INSERT**

```
INSERT INTO tabela (campo1, campo2, …, campoN)
VALUES (valor1, valor2, …, valorN)
```

- Não informamos o valor para campos auto-incrementados!
- Valores alfanuméricos (tipos CHAR e VARCHAR) devem obrigatoriamente ser colocados entre apóstrofo (').

```
INSERT INTO clientes (nome, endereco, telefone)
VALUES ('Maria', 'Rua dos Ipês', '3333-0101');
```

SQL (Structured Query Language)

- Alterando registros com **UPDATE**

```
UPDATE tabela
SET campo1 = valor1 [, campoN = valorN]
[WHERE condição]
```

- Cuidado: se você omitir a cláusula `WHERE` **TODOS** os registros de *tabela* serão modificados!
- Para alterar um registro específico, use a chave primária da tabela na cláusula `WHERE`:

```
UPDATE clientes
SET nome = 'Nome Correto'
WHERE codigo = 1965;
```

SQL (Structured Query Language)

- Excluindo registros com **DELETE**

```
DELETE FROM tabela
[WHERE condição];
```

- Caso a cláusula WHERE e a **condição** não sejam especificadas, **TODOS** os registros de tabela serão excluídos
- Para excluir um único registro especificamente, devemos usar o campo **chave primária** na condição:

```
DELETE FROM clientes
WHERE codigo = 1569;
```

JDBC (Java Database Connectivity)

- A **JDBC** é uma **API** (*Application Programming Interface*) para acesso a *bancos de dados relacionais* usando a linguagem Java.
- É mantida e desenvolvida pela Oracle. Portanto, é uma API oficial da linguagem Java.
- Uma das vantagens do uso da JDBC é que ela oferece uma **abstração** em relação ao banco de dados usado.
- A API do JDBC pode ser usada tanto em aplicativos Java tradicionais (desktop) quanto aplicativos Java Web.

JDBC (Java Database Connectivity)

- A **abstração** no uso do banco significa que, independentemente do SGBD escolhido (MySQL, Oracle, PostgreSQL etc.), o uso das classes básicas da API será sempre o mesmo.
- Esse fato facilita bastante a migração da aplicação para diferentes SGBDs, aumentando sua **portabilidade**.
- Para que seu programa (seja ele Java desktop ou Java Web) possa usar um determinado banco de dados por meio do JDBC, basta **adicionar a ele o driver JDBC para o banco** (um **arquivo .jar** que você pode baixar no site do banco de dados – ou que pode estar incluído na sua ferramenta de programação, como o NetBeans)

JDBC (Java Database Connectivity)

- Em um programa Java, usamos as classes da JDBC para realizar as seguintes etapas:
  1. Conectar ao banco de dados
  2. Enviar consultas (SELECT) e/ou comandos de modificação dos dados (INSERT, UPDATE e DELETE) ao banco
  3. Receber e utilizar os resultados.
- As classes da JDBC estão nos pacotes **`java.sql`** e **`javax.sql`**.
- **`Connection`**, **`InitialContext`** e **`DataSource`**
  - Essas três classes são usadas para se obter uma conexão ao banco de dados
- **`PreparedStatement`**
  - Após obtida a conexão, é com essa classe que executamos comandos SQL no banco.
- **`ResultSet`**
  - No caso de comandos de consulta (SELECT), usamos um objeto da classe ResultSet para acessar os dados retornados.

Preparando o banco de dados MySQL

- Para exemplificar o uso da JDBC em um aplicativo Java Web, vamos criar um cadastro de clientes.
- Antes de escrever o código Java, precisamos preparar o banco de dados e configurar o acesso ao mesmo.
- A primeira etapa consiste em se conectar ao banco e criar a estrutura básica da tabela de **clientes**.
- Vamos utilizar o banco de dados MySQL,que se encontra instalado nos laboratórios da Fatec Garça.
- Se você não possui o MySQL instalado no computador que está utilizando, acesse ***dev.mysql.com*** para baixá-lo.

Preparando o banco de dados MySQL

- O SGBD MySQL permite que tenhamos vários "**bancos de dados**" em uma mesma instalação, sendo cada um deles um conjunto de uma ou mais **tabelas**.
- Vamos criar um banco de dados chamado **clientes**, e dentro dele uma tabela de mesmo nome, **clientes**:

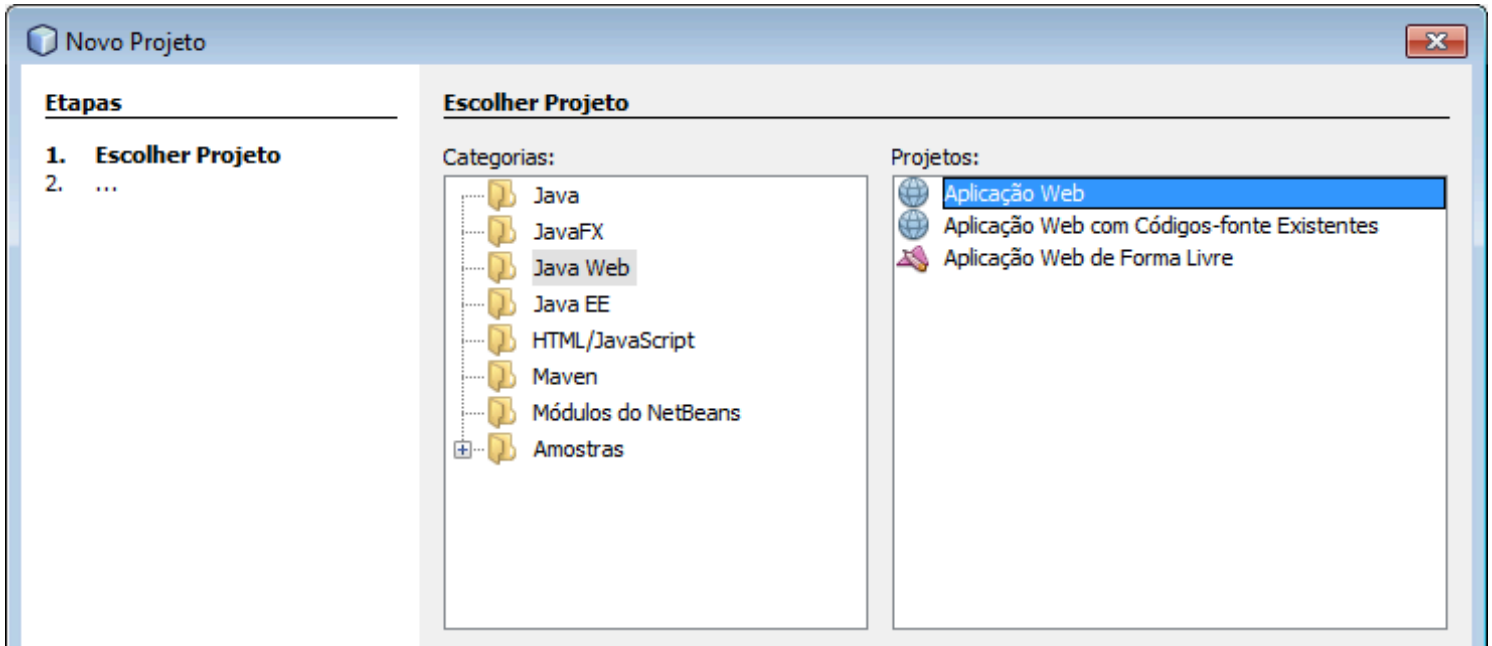
CP	Nome do Campo	TIpo	Opções
X	codigo	INT	NOT NULL AUTO_INCREMENT
	nome	VARCHAR(50)	
	endereco	VARCHAT(100)	
	cidade	VARCHAR(30)	
	estado	CHAR(2)	
	email	VARCHAR(30)	
	telefone	VARCHAR(15)	

Preparando o banco de dados MySQL

- Como criar o banco de dados e a tabela no MySQL?
- Uma das maneiras é usando o utilitário de linha de comando **`mysql`**, incluído em qualquer instalação do MySQL
- Outra possibilidade é usar um cliente gráfico, como o MySQL Query Browser ou o phpMyAdmin.
- Vamos fazer de uma maneira ainda mais simples: de dentro do próprio NetBeans!
- O NetBeans possui recursos básicos para criação e manutenção de qualquer banco de dados compatível com JDBC.

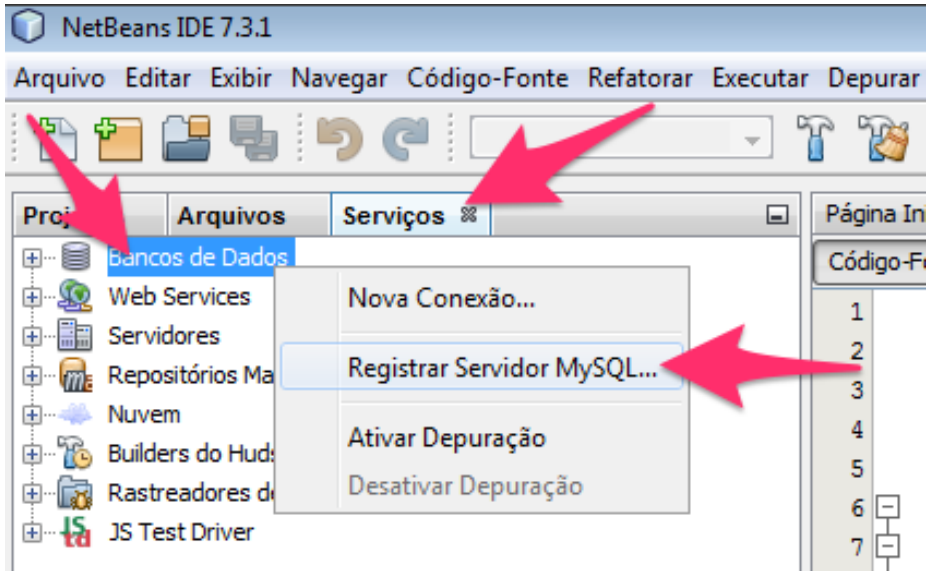
Criando o projeto no NetBeans

- Como vamos poder fazer todo o procedimento de criação do banco no NetBeans, você já pode abri-lo e criar um novo projeto Java Web, com as mesmas configurações de sempre.
- Vamos chamar este projeto de **cadastro**.



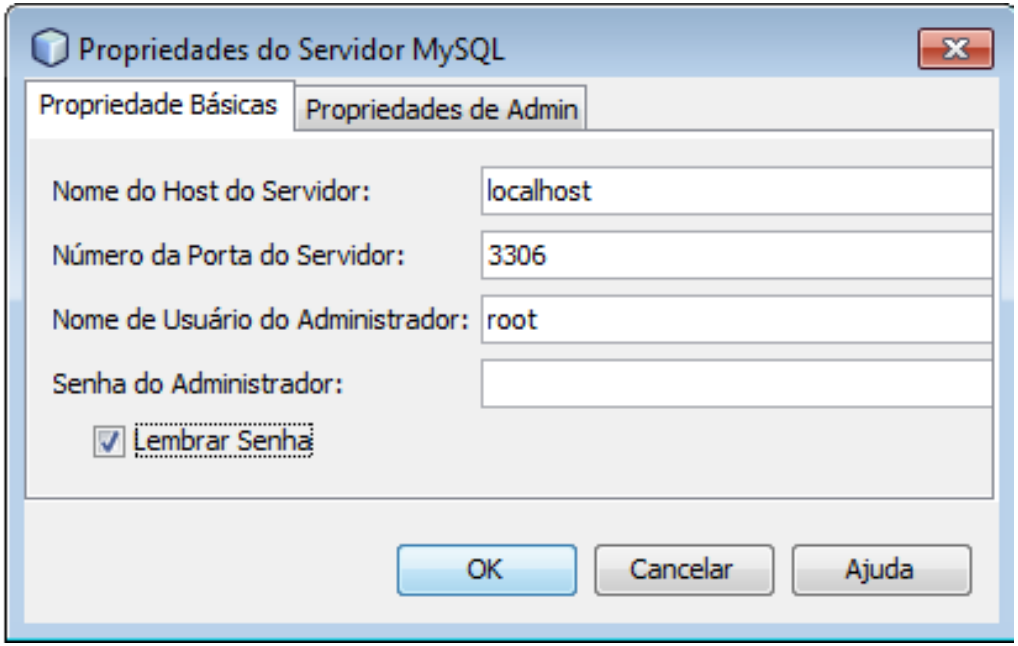
Criando o banco de dados no MySQL pelo NetBeans

- Depois de criado o projeto, a próxima etapa é fazer com que o NetBeans reconheça o MySQL instalado na máquina.
- Para que isso funcione, você precisa estar com o MySQL Server **instalado** e em **funcionamento** no se computador:
  - Abra a aba **Serviços**.
  - Clique com o botão direito em **Banco de Dados** e selecione **Registrar Servidor MySQL**:



Criando o banco de dados no MySQL pelo NetBeans

- Será exibida uma tela onde você deverá especificar a localização do servidor, o nome do usuário e a senha para acesso. Se o MySQL está na sua própria máquina, mantenha os valores default.
- Marque a opção **Lembrar Senha** para que não tenha que ficar digitando a mesma em todo acesso.
- Em novas instalações do MySQL, a senha do **root** está vazia.

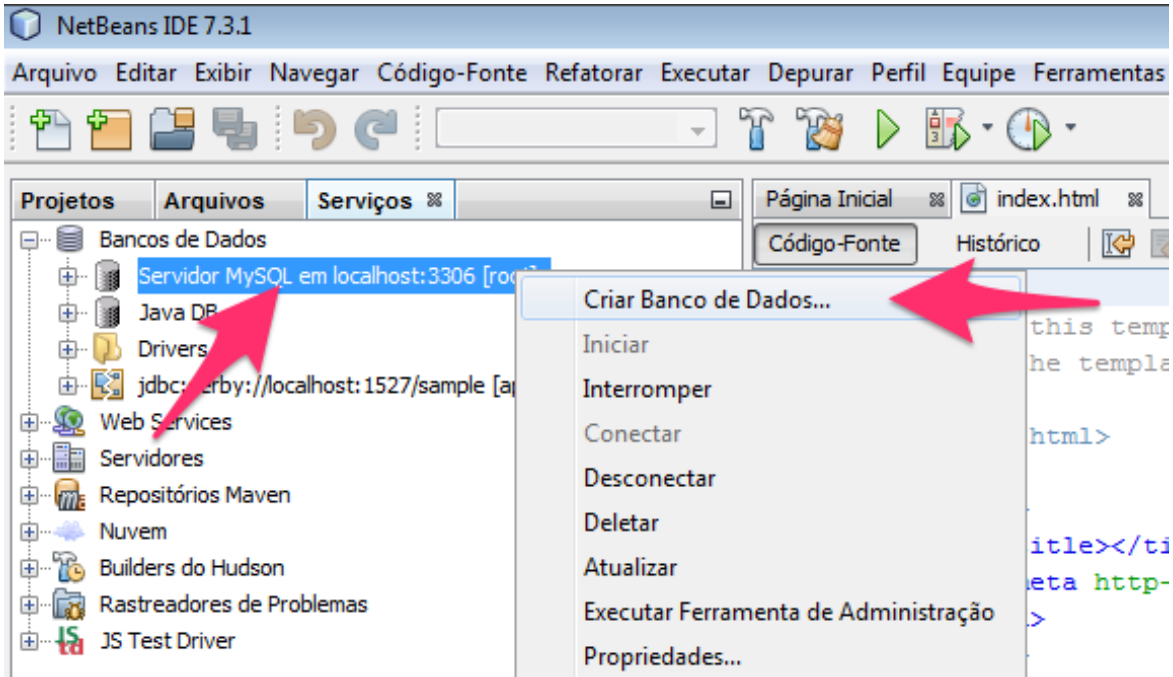


Acessar o MySQL como root…

- **…geralmente não é uma boa ideia.**
- Para facilitar as coisas, vamos configurar o acesso de nossa aplicação web ao banco MySQL utilizando o usuário `root`, administrador do banco MySQL.
- No nosso caso, em que não estamos em um ambiente de produção, não há problema.
- Porém, em um servidor real, isso NUNCA deve acontecer, pois pode causar SÉRIOS problemas de segurança.
- Em um ambiente de produção, crie no MySQL um usuário comum (usando o comando **`GRANT`**) e reconfigure seu aplicativo para acessar o banco com este usuário.

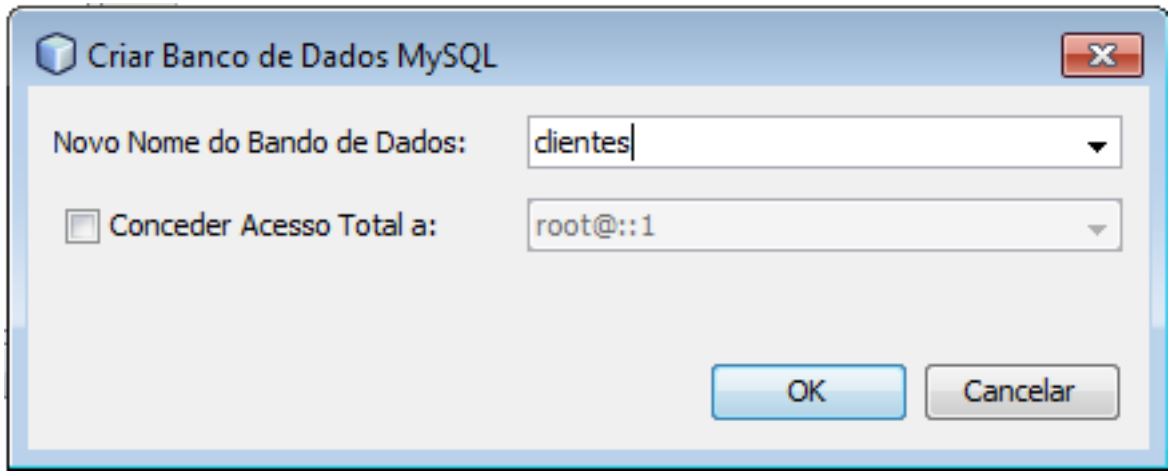
Conectando o MySQL pelo NetBeans e criando o banco de dados

- Abra a chave **Banco de Dados**.
- Você deverá ver o **Servidor MySQL** registrado na lista.
- Clique com o botão direito do mouse sobre ele e selecione **Criar Banco de Dados** (se esta opção não estiver ativa, clique antes em Conectar).



Conectando o MySQL pelo NetBeans e criando o banco de dados

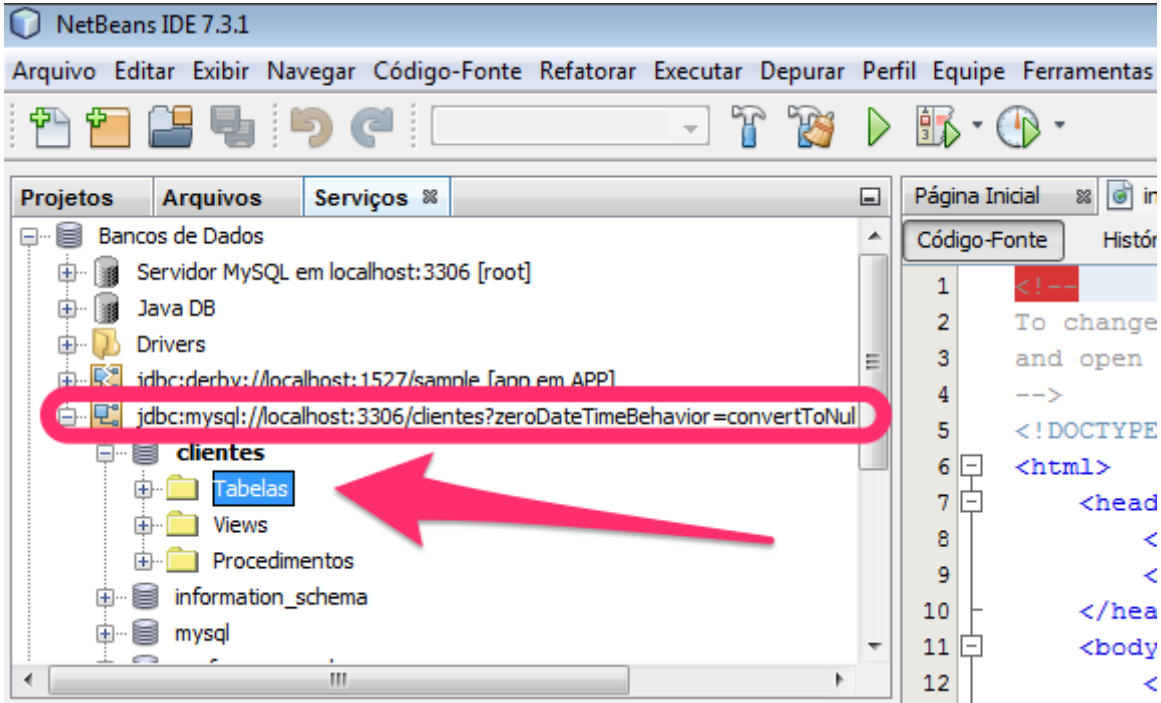
- Na janela que será exibida, informe o nome do banco (**clientes**) e clique em **OK**.



{
// I
// Code
// Java
}

Criando a tabela no banco

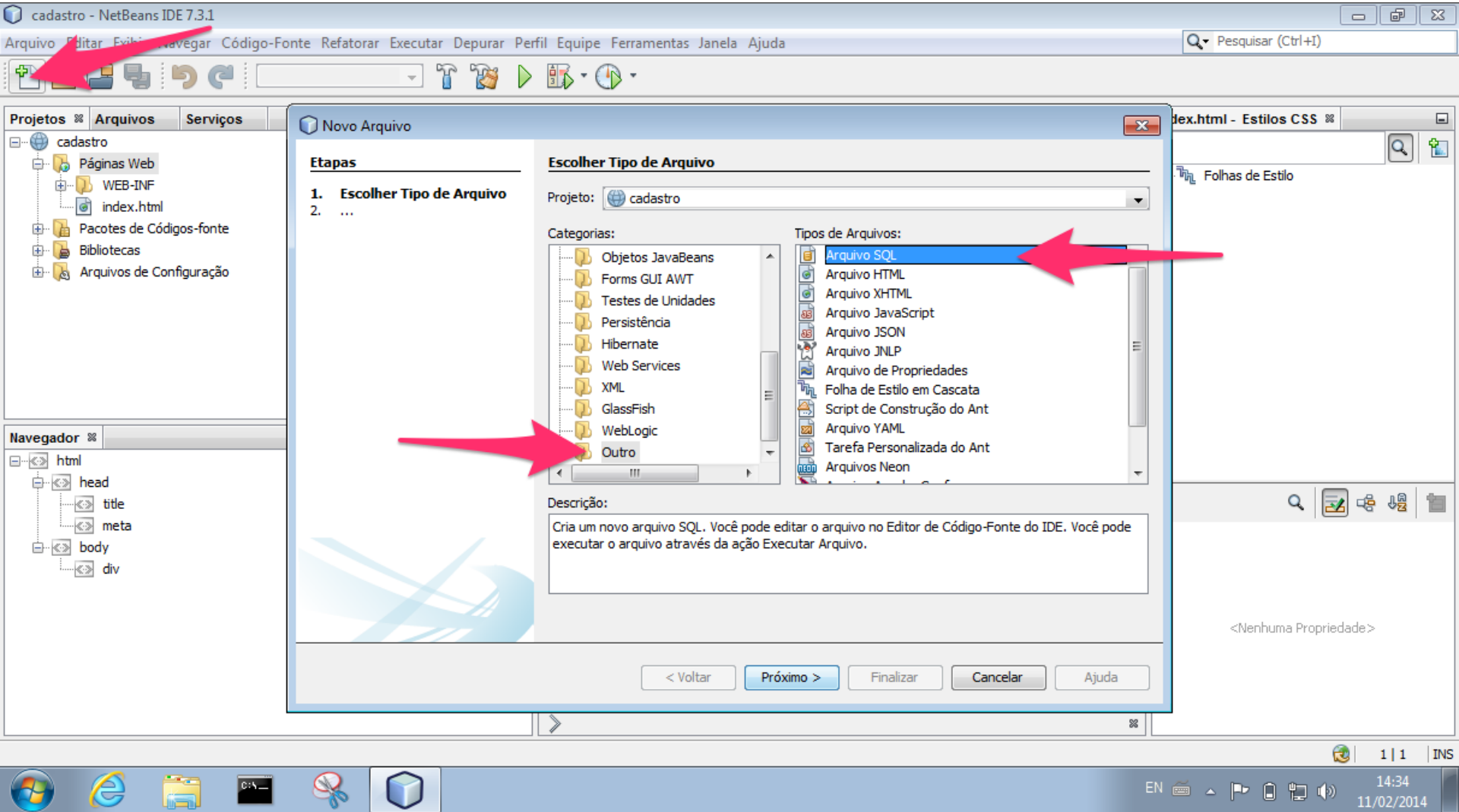
- Após criar o banco de dados, o NetBeans já cria uma nova conexão para ele.
- Por essa conexão, você pode visualizar as tabelas e os registros existentes nesse banco.



Criando a tabela no banco

- Vamos criar nossa tabela usando comandos SQL, ao invés de usar o editor visual de tabelas do NetBeans, que é bastante limitado.
- Adicione ao seu projeto um **Arquivo SQL**, onde vamos colocar os comandos necessários à criação da tabela (veja no próximo slide onde criá-lo)
- Dê a este arquivo o nome **cadastro.sql**.
- Fazendo dessa forma, teremos este arquivo sempre a mão, dentro do projeto, caso precisemos recriar as tabelas.
- **IMPORTANTE**: Caso vá implantar esta aplicação em um servidor real, remova este arquivo SQL do projeto final, por motivos de segurança!!!

Adicionando arquivo SQL ao projeto



Escreva os seguintes comandos SQL no seu arquivo:

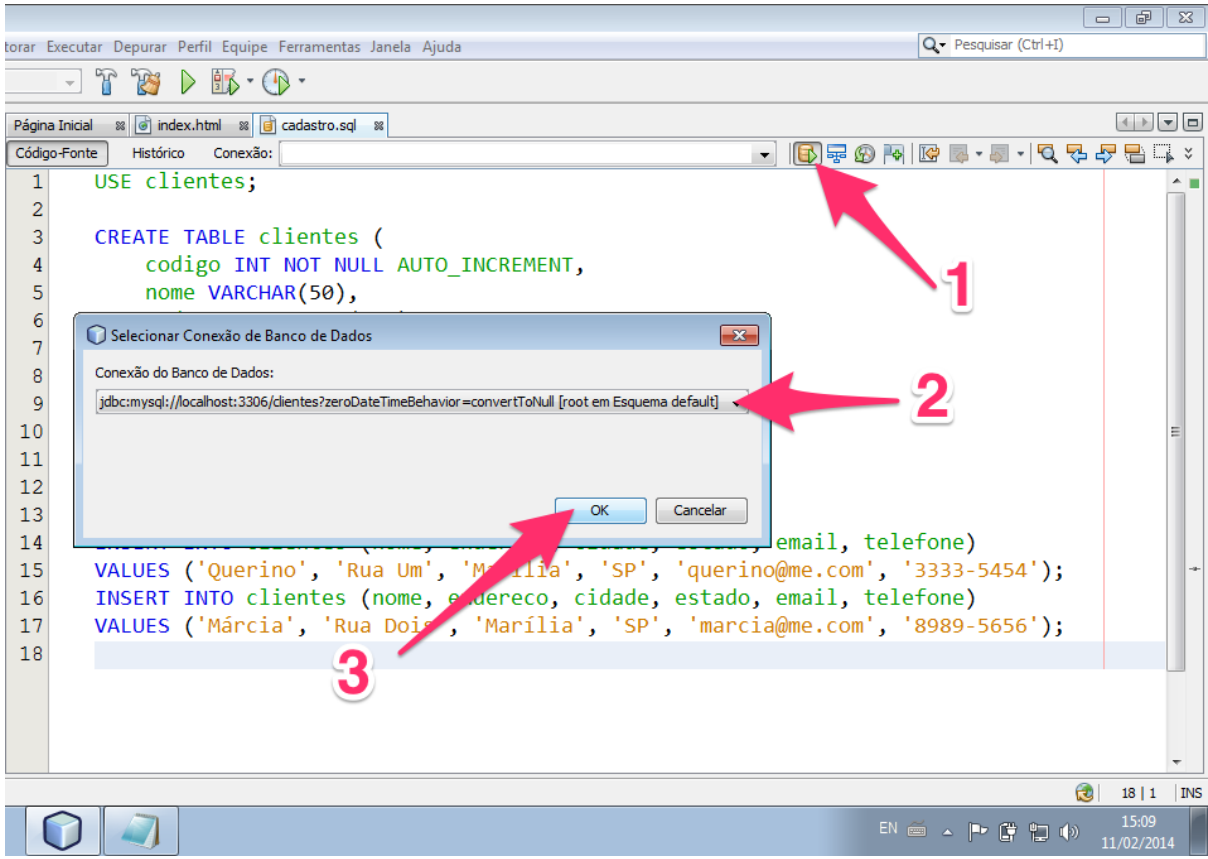
```sql
USE clientes;

CREATE TABLE clientes (
    codigo INT NOT NULL AUTO_INCREMENT,
    nome VARCHAR(50),
    endereco VARCHAR(100),
    cidade VARCHAR(30),
    estado CHAR(2),
    email VARCHAR(30),
    telefone VARCHAR(15),
    PRIMARY KEY (codigo)
);

INSERT INTO clientes (nome, endereco, cidade, estado, email, telefone)
VALUES ('Querino', 'Rua Um', 'Marília', 'SP', 'querino@me.com', '3333-5454');
INSERT INTO clientes (nome, endereco, cidade, estado, email, telefone)
VALUES ('Márcia', 'Rua Dois', 'Marília', 'SP', 'marcia@me.com', '8989-5656');
```

Executando os comandos do arquivo SQL

- Após terminar a digitação do conteúdo do arquivo SQL, salve-o e clique no **botão indicado** na figura para executá-lo (**1**).
- Na janela que será mostrada, selecione a **conexão com o MySQL** (**2**) e clique em **OK** (**3**).

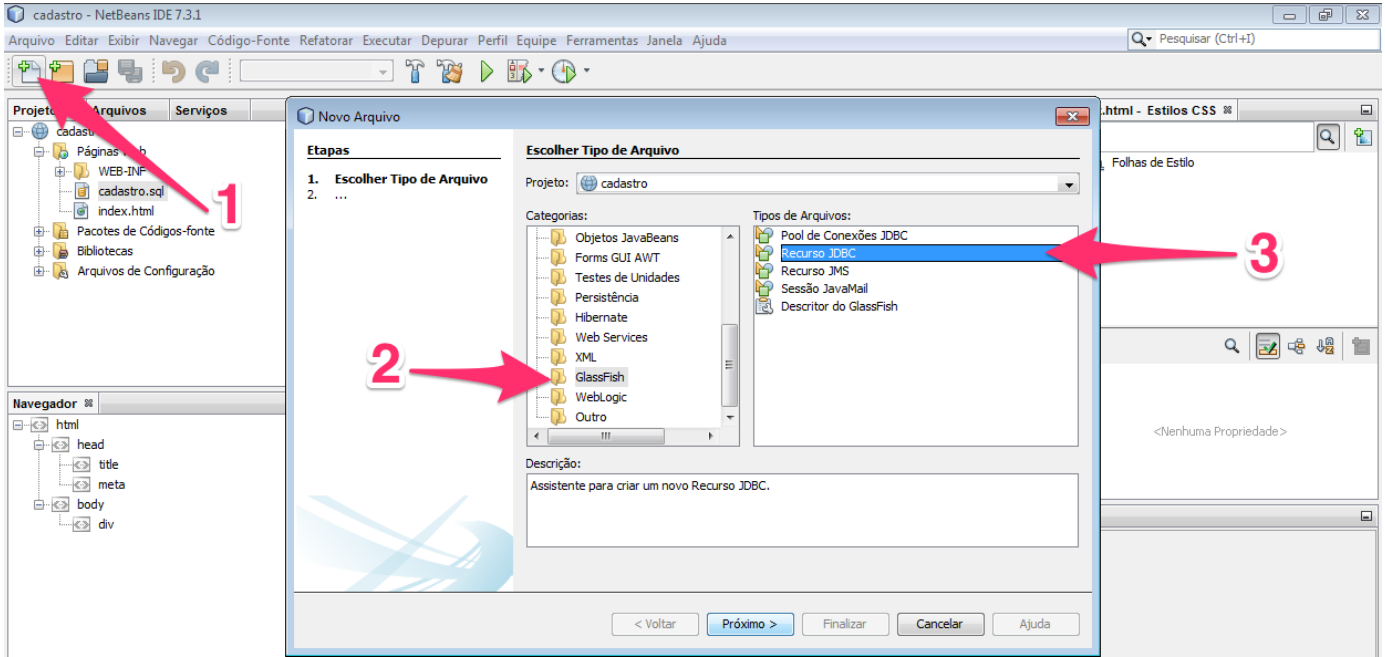


Configurando a conexão no GlassFish

- É o GlassFish quem gerencia as conexões ao banco de dados.
- Por isso, precisamos adicionar ao projeto um arquivo que "envia" ao GlassFish os dados sobre o banco de dados, como por exemplo:
  - Qual é o banco de dados usado?
  - Qual será o driver (arquivo JAR) para a conexão?
  - Onde esse banco está localizado (seu endereço IP)
  - Qual usuário e senha utilizar?
- Além disso, o GlassFish cria um grupo de conexões (***pool***) reutilizáveis para o servidor do banco, evitando que conexões adicionais desnecessárias sejam criadas.

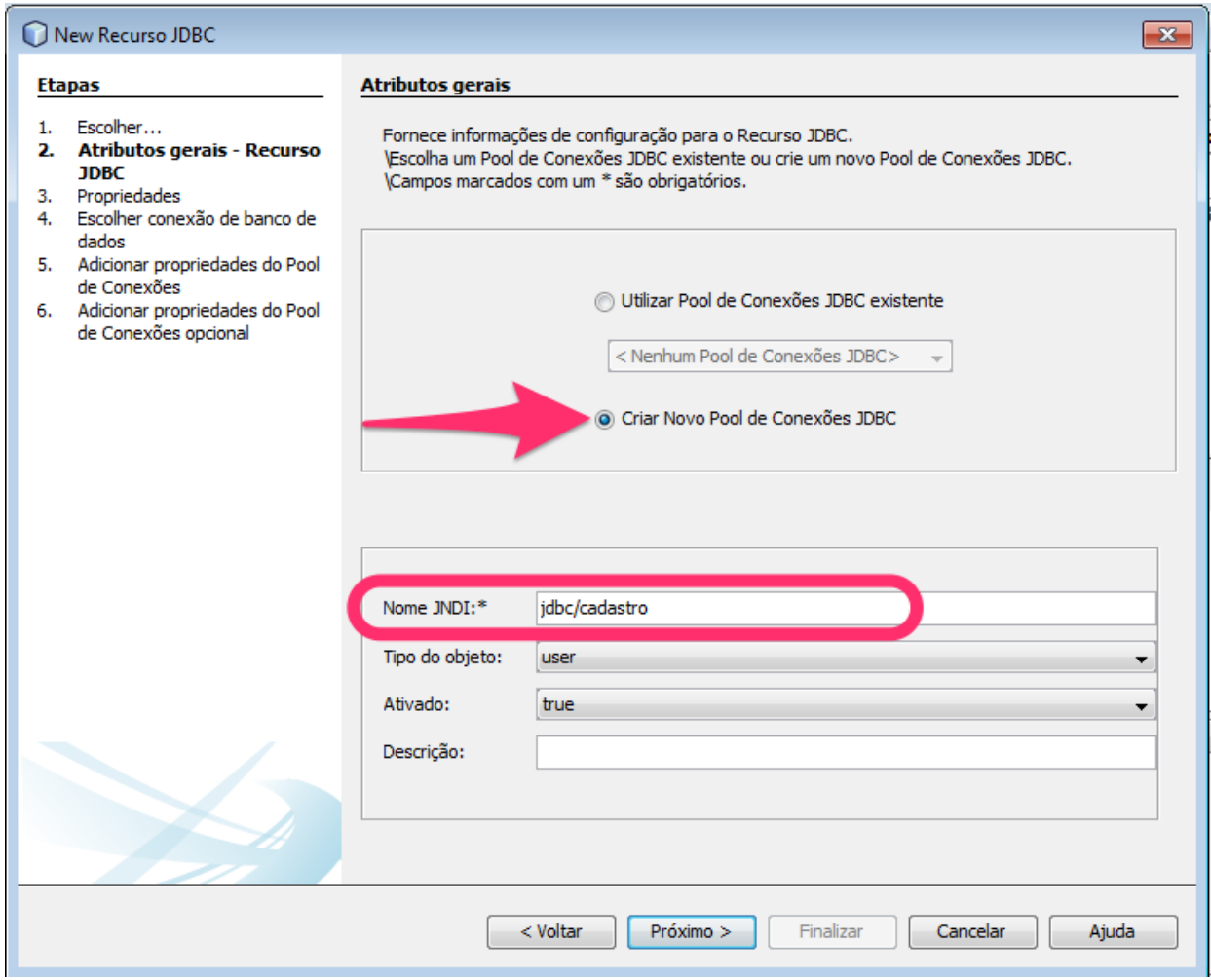
Criando a fonte de conexão para o GlassFish

- No NetBeans, clique no botão **Novo Arquivo...**
- Na janela que será aberta selecione a categoria **GlassFish** e, dentro dela, **Recurso JDBC**.
- Clique em **Próximo**



Criando a fonte de conexão para o GlassFish

- Na tela seguinte, especificamos que precisaremos **Criar Novo Pool de Conexões JDBC**.
- Definimos o **Nome JNDI** como **jdbc/cadastro** e, opcionalmente, podemos colocar uma **Descrição** para o recurso.
- Clique em **Próximo**

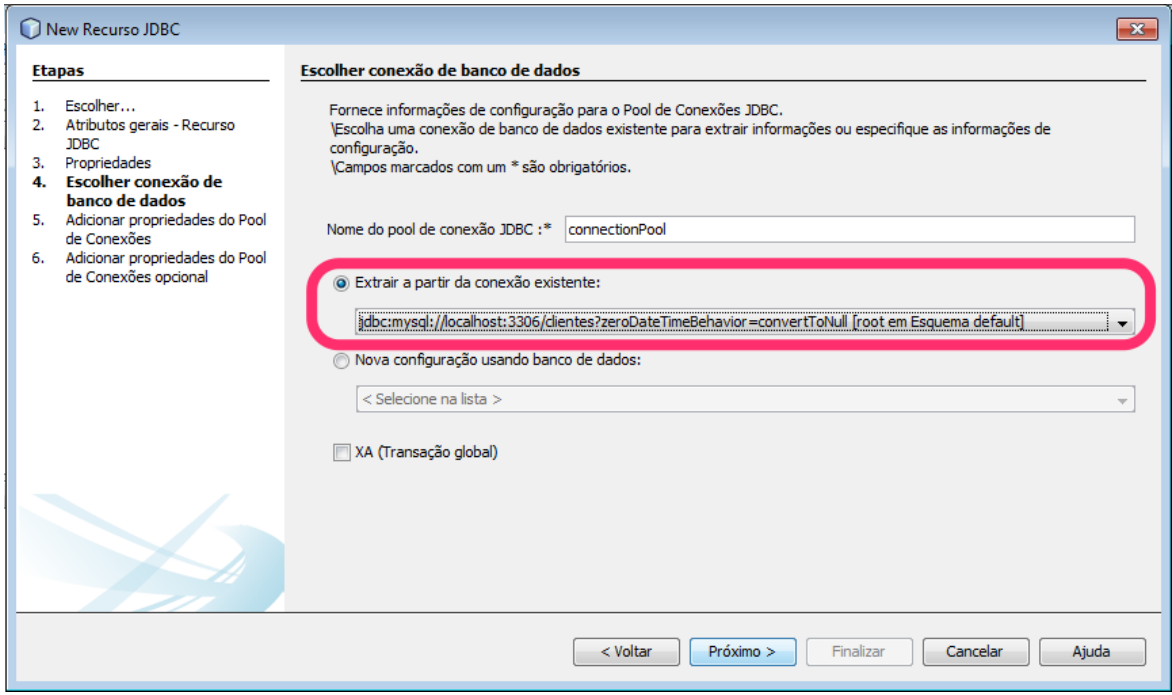


Criando a fonte de conexão para o GlassFish

- A próxima tela serve para definir configurações adicionais. Podemos passar direto por ela clicando em **Próximo**.

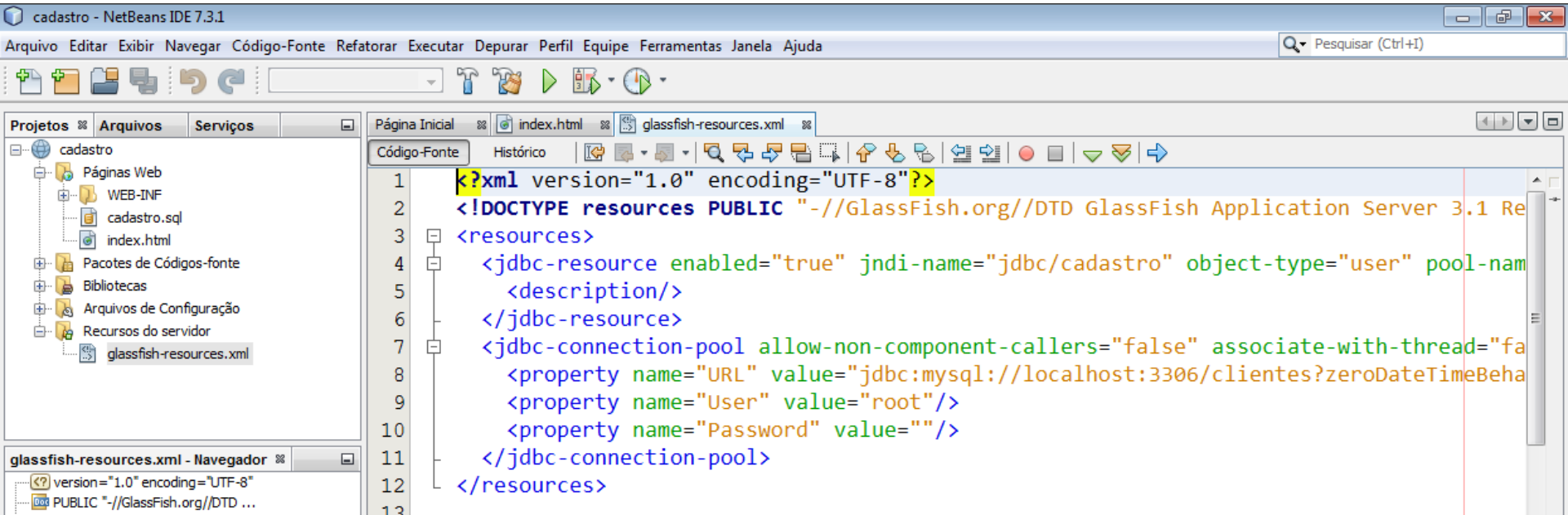
Criando a fonte de conexão para o GlassFish

- Nesta tela especificamos um **Nome do pool de conexão JDBC** (pode manter o padrão **connectionPool**).
- Na configuração **Extrair a partir da conexão existente:** escolha a conexão para o MySQL.
- Clique em **Próximo**.



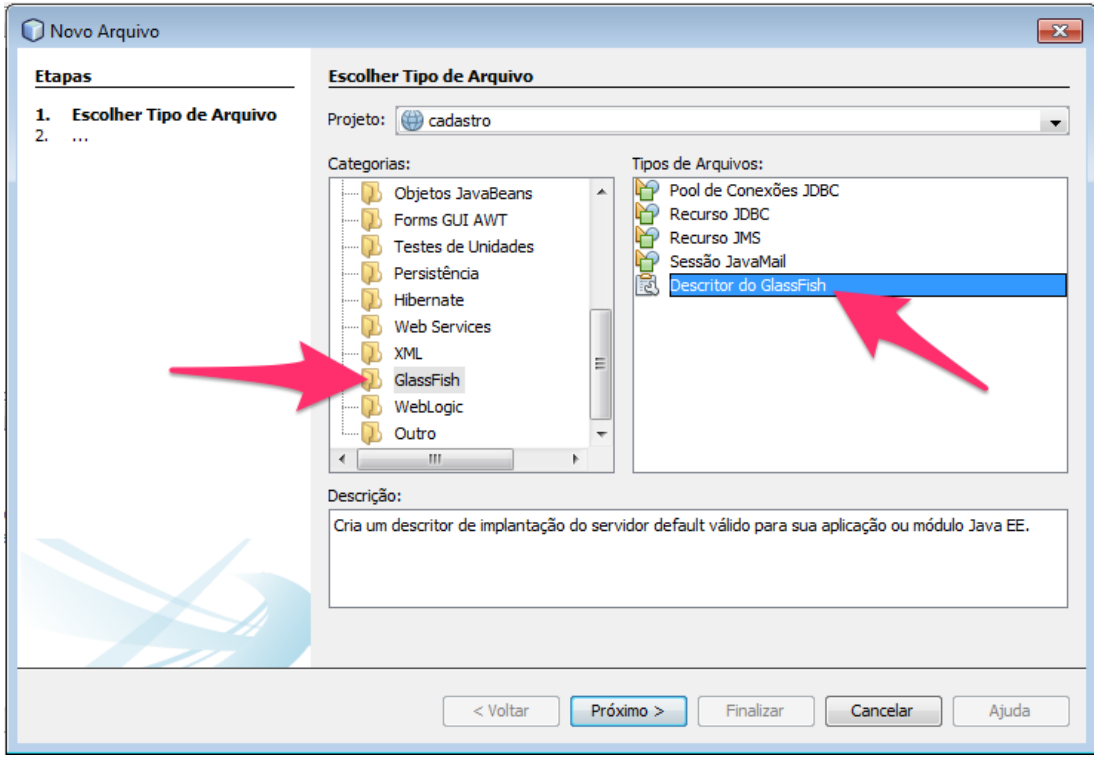
Criando a fonte de conexão para o GlassFish

- A última tela exibe os dados da conexão. Basta encerrar clicando em **Finalizar**.
- No seu projeto, será criado um arquivo chamado **`glassfish-resources.xml`**, dentro de um pasta **Recursos do servidor**. Este arquivo "enviará" ao GlassFish os dados da conexão.



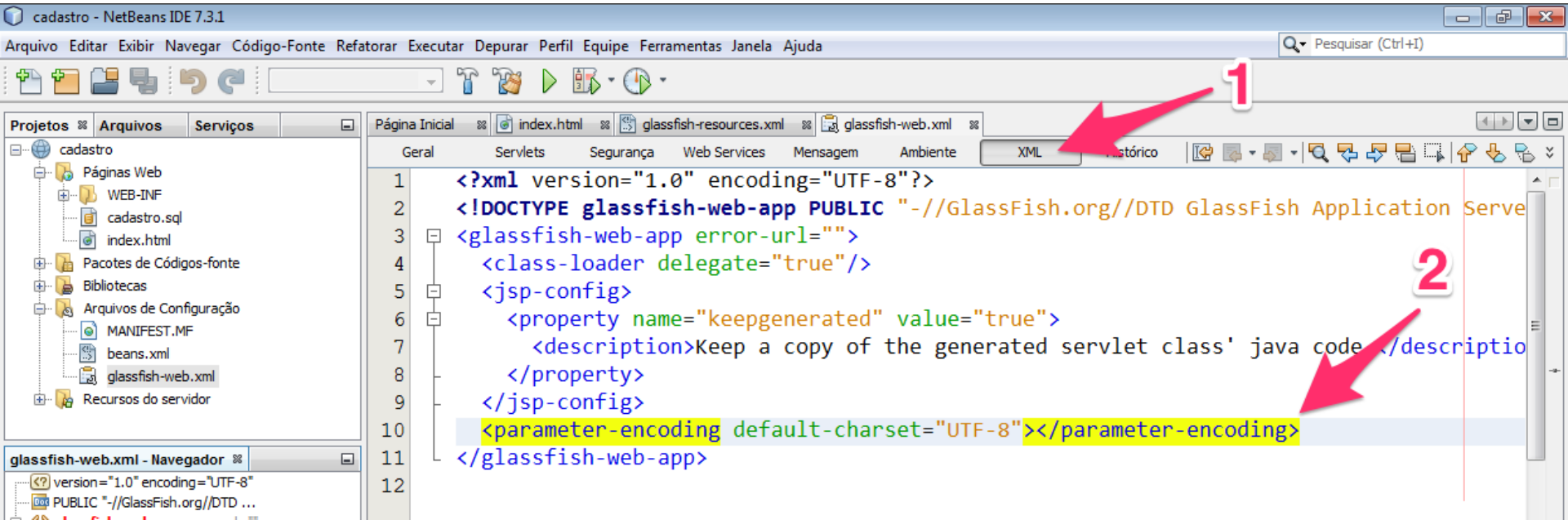
Configurar o GlassFish do Windows para UTF-8

- Em sistemas Windows, o GlassFish não vem configurado para reconhecer o padrão **UTF-8** de caracteres.
- Para que não apareça caracteres estranhos na sua aplicação no lugar de acentos e cedilha, adicione ao projeto um arquivo do tipo **GlassFish | Descritor do GlassFish**.



Configurar o GlassFish do Windows para UTF-8

- Abra o arquivo, alterne para a visualização do mesmo em XML (**1**) e adicione a linha em destaque exatamente no local indicado (**2**).



Implementando a listagem de clientes na página inicial

- Na página **`index`** da aplicação, vamos exibir ao usuário uma listagem dos clientes atualmente cadastrados, formatando-a com a tag `<TABLE>`.
- Vamos precisar usar código Java para tanto. Por isso, remova o arquivo **`index.html`** original do projeto e acrescente um **`index.jsp`**.
- Também vamos precisar de um arquivo HTML com nome **`form.html`**, que exibirá o formulário de cadastro de um cliente, e um outro arquivo JSP **`incluir`**.jsp, que receberá os dados do formulário e realizará a inclusão no MySQL, usando as classes do JDBC.
- Implemente o código desses arquivos de acordo com os próximos slides.

Código do arquivo `index.jsp`

```
16  <html>
17      <head>
18          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
19          <title>Listagem de Clientes</title>
20      </head>
21      <body>
22          <h1>Listagem de Clientes</h1>
23          <%
24              Connection conexao;
25
26              try {
27                  InitialContext contexto = new InitialContext();
28                  DataSource ds = (DataSource)contexto.lookup("jdbc/cadastro");
29                  conexao = ds.getConnection();
30                  PreparedStatement comando = conexao.prepareStatement("SELECT * "
31                          + "FROM clientes");
32                  ResultSet res = comando.executeQuery();
33                  out.println("<table border=1>");
34                  out.println("<tr>");
35                  out.println("<th>Código</th>");
```

Código do arquivo `index.jsp`

```
36                  out.println("<th>Nome</th>");
37                  out.println("<th>Endereço</th>");
38                  out.println("<th>Cidade</th>");
39                  out.println("<th>Estado</th>");
40                  out.println("<th>E-mail</th>");
41                  out.println("<th>Telefone</th>");
42
43                  while (res.next()) {
44                      out.println("<tr>");
45                      out.println("<td>" + res.getInt("codigo") + "</td>");
46                      out.println("<td>" + res.getString("nome") + "</td>");
47                      out.println("<td>" + res.getString("endereco") + "</td>");
48                      out.println("<td>" + res.getString("cidade") + "</td>");
49                      out.println("<td>" + res.getString("estado") + "</td>");
50                      out.println("<td>" + res.getString("email") + "</td>");
51                      out.println("<td>" + res.getString("telefone") + "</td>");
52                      out.println("</tr>");
53                  }
54                  out.println("</tr>");
```

Código do arquivo `index.jsp`

```
55                  out.println("</table>");
56                  conexao.close();
57              } catch (SQLException ex) {
58                  out.print("Não foi possível se conectar ao banco de dados.");
59                  System.out.println("Falha na conexão com o MySQL: " +
60                          ex.getMessage());
61              }
62
63          %>
64          <p>
65              <a href="form.html">Clique aqui para cadastrar um novo cliente</a>
66          </p>
67      </body>
68  </html>
```

Explicação sobre o código de `index.jsp`

- Para obter uma conexão, representada por uma instância da classe **`Connection`**, usamos objetos das classes **`InitialContext`** e **`DataSource`**, a forma recomendada atualmente pela Oracle.
- Passando o nome do recurso JDBC criado anteriormente (”`jdbc/cadastro`”) ao método **`lookup()`** do contexto, conseguimos obter uma referência à fonte de dados (**`DataSource`** – a conexão ao MySQL gerenciada pelo GlassFish).
- Finalmente, usamos o método **`getConnection()`**, existente no objeto **`DataSource`** para obter a conexão, que será efetivamente usada para a execução dos comandos.

Explicação sobre o código de `index.jsp`

- Por meio da conexão, criamos um **`PreparedStatement`**.
- Um **`PreparedStatement`** é objeto que representa um comando SQL que será executado de forma otimizada no servidor MySQL, melhorando o desempenho nas aplicações.
- Neste caso, criamos um comando SQL do tipo `SELECT`, que retornará todos os registros da tabela **clientes**.
- Após a execução do comando com o método **`executeQuery()`**, obtemos como retorno um objeto **`ResultSet`** que é, literalmente, um conjunto de registros.
- Precisamos percorrer esses registros, um a um, e exibi-los ao usuário em uma tabela HTML.

Explicação sobre o código de `index.jsp`

- Fazemos essa “varredura” linha-a-linha do resultado em um laço **`while`**. Na condição deste, colocamos a chamada ao método **`next()`** do objeto **`ResultSet`**.
- Este método retorna **`true`** se encontra um próximo registro dentro do conjunto do resultado.
- Assim, em cada passo do laço, estaremos posicionados em um registro do resultado, até o final do conjunto.
- Para obter os valores dos campos do registro atual, usamos o método **`getString()`** do objeto **`ResultSet`**, passando a esse o nome da coluna como parâmetro.
- Se o campo da tabela a ser lido fosse inteiro, usaríamos **`getInt()`**; se fosse decimal, usaríamos **`getDouble()`**, e assim por diante.

Explicação sobre o código de `index.jsp`

- Após o término do laço (que acontece automaticamente quando não há mais registros), imprimimos o fechamento da tabela HTML.
- Também fazemos o encerramento da conexão com o banco, executando o método `close()` do respectivo objeto.
- Repare que todo o processo de conexão e consulta ao banco ficou contido em um bloco `try…catch` do Java.
- Esse tipo de bloco existe para tratarmos erros (**exceções**) que podem acontecer com o programa.
- Nesse caso, previnimos com relação à `SQLException`, que acontece no caso de falha de conexão ao banco ou erro no comando.

Código do arquivo `form.html`

```
<h1>Formulário de Cadastro de Clientes</h1>
<form method="POST" action="incluir.jsp">
    <table border="1">
        <tr>
            <td>Nome:</td>
            <td><input type="text" name="nome"
                       size="50" maxlength="50"/></td>
        </tr>
        <tr>
            <td>Endereço:</td>
            <td><textarea name="endereco"
                          cols="30" rows="3"></textarea></td>
        </tr>
        <tr>
            <td>Cidade:</td>
            <td><input type="text" name="cidade"
                       size="30" maxlength="30"/></td>
        </tr>
```

Código do arquivo `form.html`

```
<tr>
    <td>Estado:</td>
    <td>
        <select name="estado">
            <option>MG</option>
            <option>PR</option>
            <option>RJ</option>
            <option>SP</option>
        </select>
    </td>
</tr>
<tr>
    <td>E-mail:</td>
    <td><input type="text" name="email"
               size="30" maxlength="30"/></td>
</tr>
<tr>
    <td>Telefone</td>
    <td><input type="text" name="telefone"
```

Código do arquivo `form.html`

```
                                size="15" maxlength="15"/></td>
                </tr>
                <tr>
                    <td></td>
                    <td><input type="submit"/></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

Código do arquivo `incluir.jsp`

```
18      <body>
19          <h1>Inclusão de Cliente</h1>
20          <%
21          String nome = request.getParameter("nome");
22          String endereco = request.getParameter("endereco");
23          String cidade = request.getParameter("cidade");
24          String estado = request.getParameter("estado");
25          String email = request.getParameter("email");
26          String telefone = request.getParameter("telefone");
27
28          Connection conexao;
29
30          InitialContext contexto = new InitialContext();
31          DataSource ds = (DataSource)contexto.lookup("jdbc/cadastro");
32          conexao = ds.getConnection();
33
34          String sql = "INSERT INTO clientes (nome, endereco, cidade,"
35                  + "estado, email, telefone) VALUES (?, ?, ?, ?, ?, ?)";
36
37          PreparedStatement stmt = conexao.prepareStatement(sql);
```

Código do arquivo `incluir.jsp`

```
38          stmt.setString(1, nome);
39          stmt.setString(2, endereco);
40          stmt.setString(3, cidade);
41          stmt.setString(4, estado);
42          stmt.setString(5, email);
43          stmt.setString(6, telefone);
44
45          if (stmt.executeUpdate() > 0) {
46          %>
47              <h2>Inclusão realizada com sucesso.</h2>
48          <%
49          } else {
50          %>
51              <font size="14" color="Red">Falha na inclusão</font>
52          <%
53          }
54
55          conexao.close();
56          %>
```

Código do arquivo `incluir.jsp`

```
57          <p>
58              <a href="index.jsp">Voltar</a>
59          </p>
60      </body>
61  </html>
```

Explicação sobre o código de `incluir.jsp`

- Começamos recebendo em variáveis String os parâmetros passados pelo formulário de cadastro.
- Em seguida, obtemos a conexão ao banco, usando exatamente o mesmo procedimento visto no index.jsp.
- Agora, criamos um PreparedStatement diferente: um comando SQL de inclusão (INSERT).
- Para evitar um problema de segurança gravíssimo denominado injeção de SQL (SQL Injection), montamos o comando INSERT usando parâmetros para os valores (repare nas interrogações nos locais onde seriam colocados os valores para os campos).

Explicação sobre o código de `incluir.jsp`

- Usando o método **`setString()`** do objeto **`PreparedStatement`** (chamado **`stmt`**), colocamos os valores no lugar das interrogações (em ordem sequencial, começando em 1 para o primeiro).
- Dessa maneira, além de colocar caracteres especiais que impedem a injeção de SQL, o JDBC automaticamente adiciona os apóstrofos necessários aos valores alfanuméricos.
- Se estivéssemos colocando valores inteiros, usaríamos `setInt()`, ou `setDouble()` para valores decimais.
- Após definir os valores de todos os parâmetros, executamos o comando com o método **executeUpdate()**.
- O retorno desse método indica se a inclusão foi bem sucedida: é um inteiro com a quantidade de registros criados (nesse exemplo, deve ser 1 para sucesso ou 0 para falha).

Adicionando o driver JDBC do MySQL ao projeto

- Para nossa sorte, a Oracle incluiu no NetBeans o driver JDBC para conexão com MySQL. Basta adicioná-lo ao seu projeto.
- Clique com o botão direito do mouse sobre o item **Bibliotecas** do seu projeto, e no menu que será exibido selecione **Adicionar Biblioteca...**
- Localize, na janela que será aberta, localize o **Driver JDBC do MySQL**
- Clique sobre ele e,em seguida, sobre o botão **Adicionar Biblioteca**.
- Pronto! Seu aplicativo já sabe como conversar com o MySQL.

Exercício 1

- Implemente a alteração e a exclusão para o cadastro de clientes. Para a alteração, você deverá usar o comando SQL **`UPDATE`** e para a exclusão, o comando **`DELETE`**.
- DICA: a exclusão pode ser facilmente implementada em um arquivo JSP que **recebe o código do cliente a ser excluído como parâmetro**. Você pode passar esse código de duas maneiras:
  - Criando um formulário com um campo para digitação direta do código ou…
  - **de uma forma ainda mais prática**: na listagem de clientes, crie um link de exclusão para cada um, onde o alvo será o arquivo JSP de exclusão, passando a ele o código do cliente diretamente pelo endereço (equivale a usar o método GET de um formulário):
  - `<a href="excluir.jsp?codigo=2">EXCLUIR</a>`

Exercício 1

- DICA 2: A alteração pode ser implementada seguindo o mesmo conceito de passagem do código como parâmetro, mas você deverá mostrar, em um formulário, os dados atuais antes de alterá-los com o comando SQL `UPDATE`.
- Para colocar um valor dentro de um campo de formulário (como em um `<input type=text />`), use a sua opção **`value`**:

- `<input type="text" value="<%= res.getString("cidade") %>" />`

Exercício 2

- Criar uma aplicação Web com um cadastro de usuários, contendo:
  - Login
  - Senha
  - Nome Completo
- Usar os dados do cadastro para que o usuário possa entrar na página principal.
- Com uso de `session`, controle se o usuário já realizou ou não o login no site.

Trabalho Bimestral

- Implemente um sistema de comentários para um site, onde o usuário poderá deixar sua opinião.
- Dados de cada comentário:
  - Nome
  - E-mail
  - Data
  - Opinião sobre o site (use um campo de formulário <select> com Ruim, Regular, Bom, Ótimo)
  - Texto do Comentario
- Mostrar todos os comentários no final da página principal do site.
- Elabore uma página para o adminstrador do site excluir comentários abusivos.

BIBLIOGRAFIA

- HEFFELFINGER, D. W. Java EE 6 Development with NetBeans 7. Birmingham: Packt Publishing, 2011.
- PANDA, D.; RAHMAN, R.; LANE, D. EJB 3 in Action. Greenwich: Manning Publications, 2007.
- BASHAM, Bryan. Use A Cabeça! Servlets e JSP. Alta Books, 2008.
- KURNIAWAN, B. Java para Web com Servlets, JSP e EJB. São Paulo: Ciência Moderna, 2002.