INSTITUTO FEDERAL GOIANO - CAMPUS MORRINHOS
TÉCNICO EM INFORMÁTICA
LINGUAGEM DE PROGRAMAÇÃO
Prof. Dr. Jesmmer Alves

Notas de Aula - Banco de Dados 01

---

1 INTRODUÇÃO

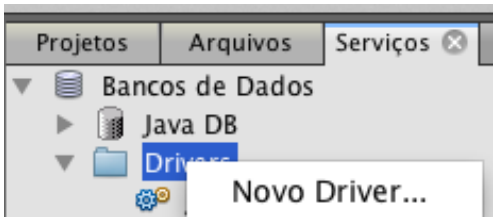
O objetivo desta atividade é criar uma aplicação Java para manipulação de uma tabela em um banco de dados. Para isto, utilizaremos uma classe para conexão com o banco de dados, uma classe para gerenciar as manipulações em registros e um formulário para inserir os dados.

2 O BANCO DE DADOS

Primeiro é necessário criar o banco de dados. Abra `Xampp` e inicie os servidores. No navegador, localhost, acesso o link para o `phpMyAdmin` (no canto superior direito). Agora crie o usuário `aulaUS` com senha `123`, o banco de dados `aulaBD` e a tabela `Clientes`. Todos os campos são `VARCHAR`, exceto o campo `id`, que é `INTEGER` e auto incremento. Veja figura abaixo:

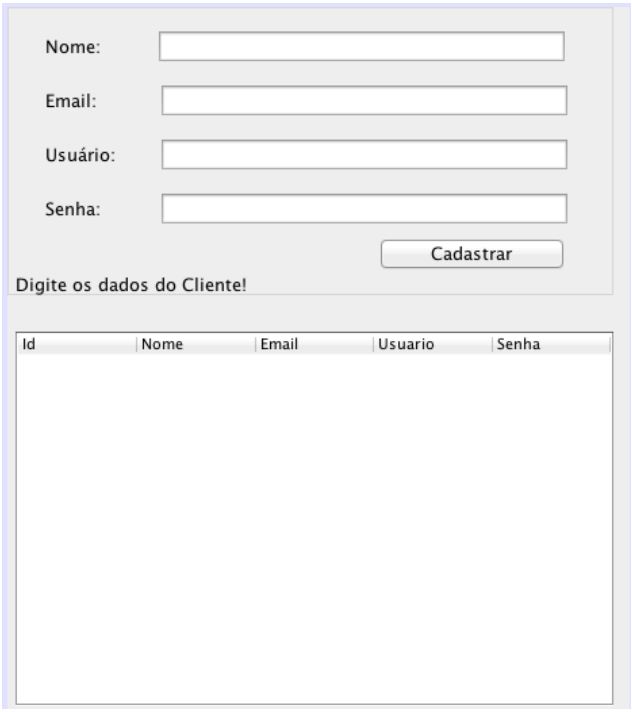
3 DESENVOLVENDO A APLICAÇÃO

Para que o NetBeans consiga fazer a conexão do Sistema com o Banco de Dados é necessário obter um arquivo de driver `JDBC` para o `MySQL`. Faça o download deste arquivo no endereço:
`https://dev.mysql.com/downloads/connector/j/`
Agora, no NetBeans, acesse a paleta `Serviços` -> `Banco de Dados` -> `Drivers`. Se o `Driver connector` para `MySQL` não estiver instalado, instale um novo driver (clique com o botão direito do mouse sobre `Drivers`) selecionando o arquivo `.jar` que foi baixado. Após instalar o driver, faça a conexão (clique com o botão direito do mouse em `MySQL Driver`) utilizando o usuário e senha criados anteriormente no `phpMyAdmin`.

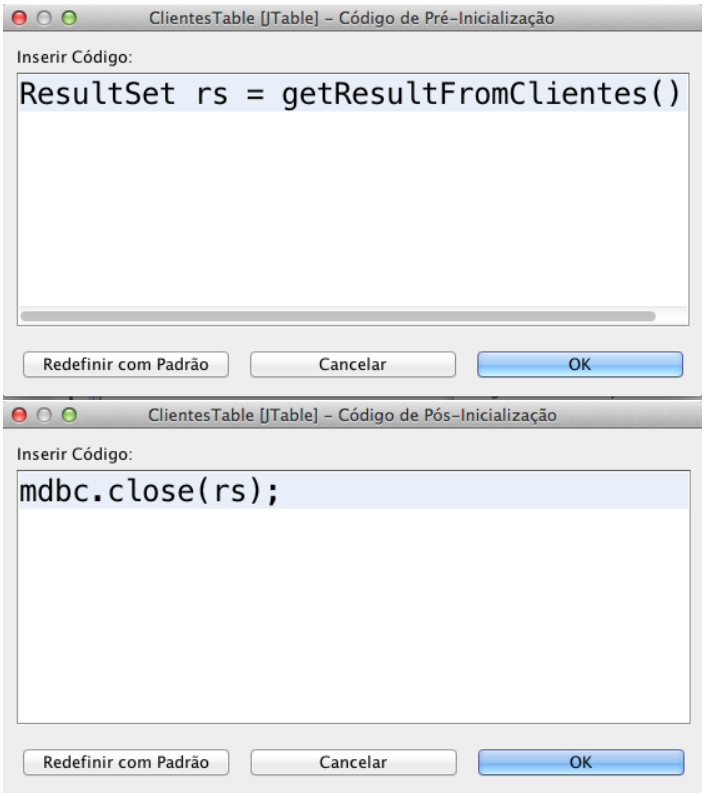


O próximo passo é criar um novo projeto Java (com o nome aulaBD01, sem a classe principal) e inserir as classes. As classes `Conexao` e `ClientesModelo` estão definidas no Apêndice (no final deste artigo). Crie as duas classes e faça as importações necessárias. Obs.: após copiar talvez seja necessário digitar novamente as aspas.

A Classe principal vai ser o formulário `CadastroClientes`. Insira um `JFrame Form` na sua aplicação e crie o seguinte formulário:



A personalização da `jTable` é feita da seguinte forma: clique com o botão direito sobre a tabela e acesse `vincular -> elements`. Selecione a tabela `Clientes` na opção `importar dados para o Form`. Ainda com a tabela selecionada, acesse `Propriedades -> Código` e insira o código de pré-inicialização e pós-inicialização, conforme figuras a seguir:



O método denominado `getResultFromClientes()` e a declaração das variáveis `mdbc` e `stmt` devem ser criados manualmente no código fonte do formulário. Obs.: Faça as devidas importações.

```
public ResultSet getResultFromClientes() {
        ResultSet rs = null;
        try{
            rs = stmt.executeQuery("select * from Clientes");
        }
        catch(SQLException e){}
        return rs;
}
private Conexao mdbc;
private java.sql.Statement stmt;
```

Agora modifique o `Construtor` e o método `Main` conforme definido abaixo.

```
    public CadastroClientes() throws SQLException {
         mdbc = new Conexao();
         mdbc.init();
         Connection conn = mdbc.getMyConnection();
         stmt= conn.createStatement();

         initComponents();
    }

     public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                try{
                    new CadastroClientes().setVisible(true);
                }catch(Exception e){}
            }
        });
    }
```

Após esta última modificação podemos executar a aplicação e verificar como ficou o formulário. Até aqui, o formulário ainda não fará o cadastro dos clientes, no entanto a tabela deve mostrar os registros cadastrados na tabela `Clientes`.

O cadastro dos clientes será feito através da execução de uma instrução SQL, após pressionar o botão Cadastrar. Portanto, dê dois cliques sobre o botão `Cadastrar` e insira o código abaixo (obs.: verifique se as aspas foram inseridas corretamente). Note que, será necessário modificar o nome das variáveis no formulário. Por exemplo, `jTNome` é o nome da variável relacionada ao campo `Nome` no formulário). Para modificar o nome da variável, clique com o botão direito do mouse sobre o elemento e acesse `Alterar nome da variável`.

```
private void jBCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
        String nome = jTNome.getText();
        String email = jTEmail.getText();
        String usuario = jTUsuario.getText();
        String senha = jTSenha.getText();
        String insertStr = "";
        try{
            insertStr = "insert into Clientes (nome, email, usuario, senha) values('"
                    +nome+"','"
                    +email+"','"
                    +usuario+"','"
                    +senha+"')";
            System.out.println(insertStr);
            int done = stmt.executeUpdate(insertStr);
            getContentPane().removeAll();
            initComponents();
            jLComentarios.setText("Um registro inserido!");
        }
        catch(Exception e){
            jLComentarios.setText("Erro ao inserir registro!");
        }
}
```

4 Explicando o Código

Vamos descrever as três classes java para trabalhar com banco de dados. A classe `Conexao` é utilizada para fazer a conexão com o banco de dados. A classe `ClientesModelo` é usada para gerenciar as informações de manipulações na tabela `Clientes`. A classe `CadastroClientes` é um formulário usado para inserir e apagar registros na tabela `Clientes`.

5 A classe Conexao

A classe `Conexao` tem como objetivo fazer a conexão com o banco de dados. Para isto, o método `init()` determina o driver de conexão JDBC e a função `getConnection()` faz a conexão com o banco de dados. Os parâmetros em `getConnection()` são: endereço e nome do banco de dados; nome do usuário; e senha. No nosso exemplo é `aulaBD`, `aulaUS` e 123, respectivamente.

Todo o processo é envolvido com o bloco `try catch`, que identifica se algum problema aconteceu durante a conexão e mostra uma mensagem caso isto aconteça. O método `getMyConnection()` retorna a conexão criada pelo método `init`.

```
public void init(){
    try{
        Class.forName("com.mysql.jdbc.Driver");
        myConnection = DriverManager.getConnection("jdbc:mysql://localhost/aulaBD","aulaUS", "123")
    }
    catch(Exception e){
        System.out.println("Falha ao fazer a conexão!");
        e.printStackTrace();
    }
}

public Connection getMyConnection(){
    return myConnection;
}
```

Esta classe ainda tem outros métodos importantes para finalizar a conexão com o banco de dados e não perder informações. O método `close` é usados para fechar as instâncias usadas na coexão e o método `destroy` para finalizar a conexão `myConnection`.

```
public void close(ResultSet rs){
    if(rs != null){
        try{
            rs.close();
        }
            catch(Exception e){}
    }
}
public void close(java.sql.Statement stmt){
    if(stmt != null){
        try{
            stmt.close();
        }
        catch(Exception e){}
    }
}
public void destroy(){
    if(myConnection != null){
        try{
            myConnection.close();
        }
        catch(Exception e){}
    }
}
```

6 A CLASSE CLIENTESMODELO

A classe `ClientesModelo` possui algumas variáveis que guardam informações específicas sobre a tabela no banco de dados, neste caso, a tabela é `Clientes` com 5 colunas `id`, `Nome`, `email`, `usuario`, `senha`. A variável `colnum` define o número de colunas, `rownum` o número de linhas, `colNames` os nomes das colunas e `ResultSets` é um vetor que vai receber informações sobre registros nesta tabela.

```
private int colnum = 5;
private int rownum;
private String[] colNames = {"id","nome","email","usuario", "senha"};
private  ArrayList<String[]> ResultSets;
```

O array de Strings ResultSets será usado para armazenar os registros da tabela, identificar as linhas e as colunas e permitir que os mesmos sejam acessados separadamente. O método Construtor ClientesModelo será executado quando uma instância desta classe for criada. Este método recebe como parâmetro rs, que é o "pacote" com todas as informações (registros) da tabela Clientes. Desta forma, através do comando de repetição while(rs.next()), todas as informações da tabela Clientes são lidas de rs e armazenadas em ResultSets. A função next() permite passar por cada registro em rs. A função getString("id") retorna o valor contido na coluna definida entre parênteses. Por fim, a função add(row) insere o registro em ResultSets.

```
public ClientesModelo(ResultSet rs) {
     ResultSets = new ArrayList<String[]>();
     try{
       while(rs.next()){
             String[] row = {
               rs.getString("id"),rs.getString("nome"),
               rs.getString("email"),rs.getString("usuario"),
               rs.getString("senha")
           };
           ResultSets.add(row);
        }
     }
     catch(Exception e){
         System.out.println("Exception em ClientesModelo");
     }
}
```

Os demais métodos nesta classe são utilizados para conseguir informações específicas sobre dados na tabela, são eles: o método getValueAt(int rowindex, int columnindex) retorna o conteúdo de acordo com o índice da linha em rowindex e o índice da coluna em columnindex; o método getRowCount() retorna o número de registros na tabela; o método getColumnCount() retorna o número de colunas; e o método getColumnName(int param) retorna o nome de uma coluna de acordo com o índice definido em param.

```
public Object getValueAt(int rowindex, int columnindex) {
   String[] row = ResultSets.get(rowindex);
   return row[columnindex];
}

public int getRowCount() {
    return ResultSets.size();
}

public int getColumnCount() {
    return colnum;
}

public String getColumnName(int param) {
   return colNames[param];
}
```

7 A CLASSE CADASTROCLIENTES

A classe `CadastroClientes` será usada para mostrar os dados da tabela para o usuário e permitir que sejam inseridos e apagados registros nesta tabela. Inicialmente, o construtor `CadastroClientes()` faz toda inicialização necessária para fazer a conexão com o banco de dados, da seguinte forma:

- `mdbc = new Conexao()` cria uma nova instância de conexão através da classe `Conexao`;
- `mdbc.init()` executa o método `init()` que faz a conexão com o banco de dados;
- `getMyConnection()` retorna a conexão criada;
- `createStatement()` cria o `statement` que permitirá executar os comandos SQL; e
- `initComponents()` é usado para inicializar os componentes no formulário e atualizar a exibição dos dados.

```
public CadastroClientes() throws Exception{
    mdbc = new Conexao();
    mdbc.init();
    Connection conn = mdbc.getMyConnection();
    stmt = conn.createStatement();
    initComponents();
}
```

Um ponto importante é definido pelo método `getResultFromClientes()`. Este método é responsável por montar e executar o comando SQL que retornará os dados da tabela através do array `rs`. Neste caso, a função `executeQuery("Select * from Clientes")` executa o comando SQL definido entre parênteses.

```
public ResultSet getResultFromClientes() {
    ResultSet rs = null;
    try{
            rs = stmt.executeQuery("select * from Clientes");
    }
    catch(SQLException e){}
    return rs;
}
```

O método `jBCadastrarActionPerformed` é responsável por pegar os valores que serão inseridos na tabela, montar e executar o comando SQL. Para isto, a função `getText()` é usada para pegar o valor inserido em um campo no formulário. Por exemplo, `nome = jTNome.getText()` pega o valor inserido no campo `jTNome` e armazena na variável `nome`. O comando `SQL` é montado e armazenado na variável `insertStr`. Assim, a instrução `done = stmt.executeUpdate(insertStr)` executa o comando `SQL` e, se o registro for inserido na tabela, a variável `done` recebe o valor "1", caso contrário recebe "0". Note que a função `executeUpdate` é usada quando o comando `SQL` não irá retornar nenhum valor (diferentemente da função `executeQuery` mostrada anteriormente). As instruções `getContentPane().removeAll()` e `initComponents()` inicializa os componentes no formulário e mostra o conteúdo atualizado da tabela.

```
private void jBCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
        String nome = jTNome.getText();
        String email = jTEmail.getText();
        String usuario = jTUsuario.getText();
        String senha = jTSenha.getText();
        String insertStr = "";
        try{
            insertStr = "insert into Clientes (nome, email, usuario, senha) values('"
                    +nome+"','"
                    +email+"','"
                    +usuario+"','"
                    +senha+"')";
            System.out.println(insertStr);
            int done = stmt.executeUpdate(insertStr);
            getContentPane().removeAll();
            initComponents();
            jLComentarios.setText("Um registro inserido!");
        }
        catch(Exception e){
            jLComentarios.setText("Erro ao inserir registro!");
        }
}
```

8 Apêndice

8.1 Classe Conexao

```
package AulaBD01;
import java.sql.*;

public class Conexao {
    private Connection myConnection;
    public Conexao() {
    }

    public void init(){
       try{
          Class.forName("com.mysql.jdbc.Driver");
          myConnection = DriverManager.getConnection(
            "jdbc:mysql://localhost/aulaBD","aulaUS", "123");
       }
        catch(Exception e){
            System.out.println("Falhou ao Fazer a conexão!");
            e.printStackTrace();
        }
    }

    public Connection getMyConnection(){
        return myConnection;
    }

    public void close(ResultSet rs){
        if(rs !=null){
            try{
               rs.close();
            }
            catch(Exception e){}
        }
    }

    public void close(java.sql.Statement stmt){
        if(stmt != null){
            try{
               stmt.close();
            }
            catch(Exception e){}
        }
    }

  public void destroy(){
    if(myConnection != null){
         try{
               myConnection.close();
         }
         catch(Exception e){}
    }
  }
}
```

8.2 Classe ClientesModelo

```
package AulaBD01;

import javax.swing.table.AbstractTableModel;
import java.sql.*;
import java.util.ArrayList;

public class ClientesModelo extends AbstractTableModel {

    private int colnum = 5;
    private int rownum;
    private String[] colNames = {
        "id","nome","email","usuario", "senha"
    };
    private  ArrayList<String[]> ResultSets;

    public ClientesModelo(ResultSet rs) {

      ResultSets=new ArrayList<String[]>();

      try{
        while(rs.next()){
            String[] row = {
                rs.getString("id"),rs.getString("nome"),
                rs.getString("email"),rs.getString("usuario"),
                rs.getString("senha")
            };
            ResultSets.add(row);
        }
      }
      catch(Exception e){
          System.out.println("Exception em ClientesModelo");
      }
    }

    public Object getValueAt(int rowindex, int columnindex) {
       String[] row = ResultSets.get(rowindex);
       return row[columnindex];

    }

    public int getRowCount() {
        return ResultSets.size();
    }

    public int getColumnCount() {
        return colnum;
    }

    public String getColumnName(int param) {
       return colNames[param];
    }
}
```