



TUTORIAL

APLICAÇÃO CRUD DE CLIENTES EM JAVA COM
SWING, JAKARTA PERSISTENCE, MAVEN, HIBERNATE E MYSQL

PROFESSOR: MAROMO

Parte 1 - Introdução Completa ao Aplicativo de CRUD de Clientes em Java com Swing, Hibernate e MySQL

Neste laboratório prático, vamos criar um **aplicativo desktop** em Java utilizando as seguintes tecnologias:

- **Swing:** Para construir a interface gráfica (GUI) onde o usuário pode interagir com o sistema.
- **Hibernate:** Para realizar a persistência dos dados no banco de dados usando o mapeamento objeto-relacional (ORM).
- **Jakarta Persistence (Jakarta API):** Para definir as anotações e o padrão ORM, facilitando a persistência de dados entre o modelo de objetos e o banco de dados relacional.
- **MySQL:** Como banco de dados relacional para armazenar as informações dos clientes.
- **Maven:** Para gerenciamento de dependências e organização do projeto.

O objetivo principal deste projeto é criar uma aplicação **CRUD** (Create, Read, Update, Delete) completa, onde você poderá cadastrar, listar, alterar e excluir clientes em uma interface gráfica amigável. Vamos usar o **Hibernate** para gerenciar a conexão com o banco de dados e realizar operações de persistência sem que você precise escrever SQL manualmente. O **MySQL** será usado como o banco de dados para armazenar as informações dos clientes, e o **Swing** nos fornecerá os componentes gráficos (como botões, campos de texto e tabelas) para a interação com o usuário.

Uma novidade neste projeto é o uso do **Jakarta Persistence API** (anteriormente conhecido como **Java Persistence API - JPA**). **Jakarta Persistence** é o padrão mais recente para mapeamento objeto-relacional em Java, substituindo os pacotes antigos de `javax.persistence` por `jakarta.persistence`. Essa mudança faz parte de uma transição da Oracle para a Eclipse Foundation, e o **Jakarta Persistence** é amplamente suportado por frameworks como o Hibernate.

Ao final deste tutorial, você terá uma aplicação desktop robusta e funcional, com uma interface amigável que permite gerenciar clientes diretamente do seu computador.

Estrutura do Projeto

Este projeto será estruturado em pacotes bem organizados para separar as responsabilidades e manter o código modular e fácil de manter. A estrutura de pastas será organizada da seguinte forma:

```
src/
|
|-- model/
|   |-- Cliente.java    # Entidade Cliente mapeada no Hibernate
|
|-- persistence/
|   |-- ClienteDAO.java # Classe DAO para gerenciar as operações de banco de dados do Cliente
|   |-- HibernateUtil.java # Classe utilitária para gerenciar a SessionFactory do Hibernate
```

```
|
|— view/
|   — ClienteView.java # Interface gráfica do Swing para interagir com os dados do Cliente
|
|— GerenciarClientes.java # Classe principal que inicia o aplicativo
```

Tecnologias Abordadas

1. Swing (Interface Gráfica)

- **O que é:** O **Swing** é uma biblioteca de componentes gráficos (GUI) que faz parte da plataforma Java. Ele fornece elementos de interface, como **botões**, **caixas de texto**, **tabelas** e **janelas**, permitindo que você crie interfaces gráficas desktop completas.
- **Como Usaremos:** Utilizaremos o **Swing** para criar uma interface gráfica para gerenciar clientes. A interface permitirá que o usuário insira dados do cliente (como nome e email) e visualize os clientes cadastrados em uma tabela.

2. Hibernate (Persistência de Dados)

- **O que é:** O **Hibernate** é uma ferramenta ORM (Object-Relational Mapping), que facilita a comunicação entre classes Java e bancos de dados relacionais. Com o Hibernate, você pode persistir objetos Java diretamente no banco de dados, sem precisar escrever consultas SQL manualmente.
- **Como Usaremos:** O **Hibernate** será usado para gerenciar as operações de banco de dados no MySQL. Vamos criar uma classe **ClienteDAO** para salvar, listar, alterar e excluir clientes. O Hibernate mapeará a classe **Cliente** para uma tabela no banco de dados, automatizando as operações CRUD.

3. MySQL (Banco de Dados Relacional)

- **O que é:** O **MySQL** é um sistema de gerenciamento de banco de dados relacional popular, utilizado para armazenar dados de forma estruturada.
- **Como Usaremos:** Utilizaremos o **MySQL** para armazenar os dados dos clientes. O Hibernate se conectará ao banco de dados MySQL para realizar operações de inserção, atualização, exclusão e leitura de dados.

4. Maven (Gerenciamento de Dependências)

- **O que é:** O **Maven** é uma ferramenta de automação de build e gerenciamento de dependências para projetos Java. Ele facilita a configuração de bibliotecas e frameworks no projeto e garante que todas as dependências necessárias sejam gerenciadas e baixadas automaticamente.
- **Como Usaremos:** Utilizaremos o **Maven** para gerenciar as dependências do Hibernate, MySQL e outras bibliotecas necessárias. Isso tornará o projeto mais fácil de configurar e manter.

Funcionalidades da Aplicação

O aplicativo será uma aplicação desktop simples para gerenciar clientes com as seguintes funcionalidades:

1. **Adicionar Clientes:**

- O usuário poderá adicionar um novo cliente inserindo o nome e o email nos campos de texto e clicando no botão "Salvar". O cliente será salvo no banco de dados MySQL.
2. **Listar Clientes:**
- Todos os clientes cadastrados serão exibidos em uma tabela na interface gráfica. A tabela será atualizada sempre que um novo cliente for adicionado ou uma operação de alteração ou exclusão for realizada.
3. **Alterar Clientes:**
- O usuário poderá selecionar um cliente existente na tabela. Os dados do cliente serão carregados nos campos de texto, permitindo que o usuário faça alterações. Após modificar as informações, o usuário poderá clicar no botão "Alterar" para atualizar os dados no banco de dados.
4. **Excluir Clientes:**
- O usuário poderá excluir um cliente selecionando-o na tabela e clicando no botão "Excluir". O cliente será removido do banco de dados e a tabela será atualizada.
-

Estrutura dos Pacotes

1. Pacote model

- Este pacote contém a classe **Cliente**, que é a **entidade** do projeto. A classe **Cliente** representa um cliente no sistema e será mapeada para uma tabela no banco de dados MySQL usando o Hibernate.
- Cada instância de **Cliente** terá um **id**, **nome**, e **email**. O **id** será gerado automaticamente pelo banco de dados.

2. Pacote persistence

- Este pacote contém as classes responsáveis pela **persistência de dados** no banco de dados.
- A classe **HibernateUtil** fornecerá a configuração necessária para o Hibernate se conectar ao banco de dados e criar uma sessão para realizar as operações de banco.
- A classe **ClienteDAO** (Data Access Object) gerenciará as operações de inserção, leitura, atualização e exclusão (CRUD) dos clientes no banco de dados.

3. Pacote view

- Este pacote contém a interface gráfica, onde os usuários interagem com a aplicação.
- A classe **ClienteView** será a janela da aplicação, com campos de texto para digitar o nome e o email do cliente, além de botões para as operações CRUD (Salvar, Alterar, Excluir).
- Também terá uma **JTable** para exibir a lista de clientes cadastrados no banco.

4. Classe GerenciarClientes

- A classe **GerenciarClientes** ficará fora dos pacotes e será responsável por iniciar o aplicativo. Ela conterá o método `main`, que executará a janela **ClienteView**.

Fluxo do Aplicativo

1. **Início:** A aplicação é iniciada pela classe **GerenciarClientes**, que abre a janela **ClienteView**.
2. **Interação:** O usuário poderá:
 - Inserir o nome e o email de um cliente e clicar em "Salvar" para adicioná-lo à tabela e ao banco de dados.
 - Selecionar um cliente na tabela para editar seus dados ou removê-lo do sistema.
3. **Persistência:** O **Hibernate** é responsável por conectar o aplicativo ao banco de dados MySQL e realizar todas as operações de CRUD (inserir, listar, atualizar e excluir).
4. **Exibição de Dados:** A **JTable** na interface gráfica é constantemente atualizada para refletir o estado atual do banco de dados.

Parte 2 - Laboratório Prático: Criando um CRUD Completo em Java com Swing, Hibernate e MySQL

Este laboratório passo a passo mostrará como criar uma aplicação completa em **Java** utilizando **Swing** para a interface gráfica e **Hibernate** para persistência no banco de dados **MySQL**. O projeto será estruturado em pacotes com o objetivo de manter o código organizado e modular. Ao final, você terá uma aplicação **CRUD (Create, Read, Update, Delete)** funcional, capaz de gerenciar clientes.

Estrutura do Projeto

1. **model**: Contém a classe **Cliente**, que é a entidade que será persistida.
2. **persistence**: Contém as classes **ClienteDAO** e **HibernateUtil**, responsáveis pela persistência de dados.
3. **view**: Contém a classe **ClienteView**, que representa a interface gráfica da aplicação.
4. **GerenciarClientes**: A classe que inicia a aplicação e a interface gráfica.

Passo 1: Configuração do Projeto com Maven

1.1 Criar um Projeto Maven

No **IntelliJ IDEA**, siga os passos abaixo para criar o projeto:

1. Vá em **File > New > Project > Java**
2. **Em Name digite:** cadastro-cliente
3. Em Build system, selecione **Maven**.
4. Em Advanced Settings: Defina o **GroupId** como `net.maromo.swing` e o **ArtifactId** como `cadastro-cliente`.
5. Clique em **Create** para criar o projeto.

1.2 Adicionar Dependências ao pom.xml

No arquivo **pom.xml**, adicione as dependências do Hibernate, Jakarta Persistence, MySQL e Swing:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>net.maromo.swing</groupId>
  <artifactId>cadastro-cliente</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>20</maven.compiler.source>
    <maven.compiler.target>20</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
```

```

        <!-- Hibernate -->
        <dependency>
            <groupId>org.hibernate.orm</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.5.2.Final</version>
        </dependency>
        <!-- MySQL -->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <version>8.4.0</version>
        </dependency>
        <dependency>
            <groupId>jakarta.persistence</groupId>
            <artifactId>jakarta.persistence-api</artifactId>
            <version>3.1.0</version>
        </dependency>
    </dependencies>
</project>

```

Passo 2: Configuração do Hibernate

2.1 Criar o Arquivo de Configuração hibernate.cfg.xml

No diretório `src/main/resources`, crie o arquivo `hibernate.cfg.xml`. Esse arquivo contém as configurações de conexão com o banco de dados **MySQL**:

```

<hibernate-configuration>
<session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/cadastro_clientes</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">1234</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.show_sql">true</property>
    <!-- Mapeamento de entidade -->
    <mapping class="model.Cliente" />
</session-factory>
</hibernate-configuration>

```

Nota: Substitua `root` e senha com seu usuário e senha do MySQL.

2.2 Criar a Classe HibernateUtil para Gerenciar a Conexão

No pacote **persistence**, crie a classe `HibernateUtil` para fornecer a configuração da `SessionFactory`:

```

package persistence;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {

```



```

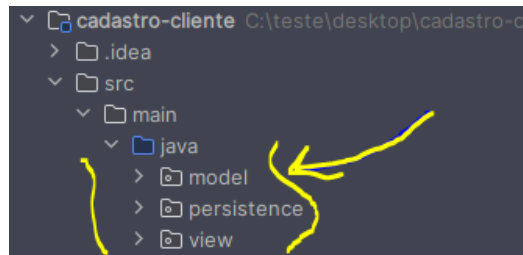
        return sessionFactory;
    }

    public static void shutdown() {
        getSessionFactory().close();
    }
}

```

Passo 3: Criar a Entidade cliente

Crie os pacotes model, persistence e view. Veja a figura:



Agora vamos criar a entidade **Cliente** no pacote **model**. Ela representará os dados do cliente que serão persistidos no banco de dados.

3.1 Criar a Classe Cliente

No pacote **model**, crie a classe **Cliente** com os seguintes campos:

```

package model;
import jakarta.persistence.*;

@Entity
@Table(name = "cliente")
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;
    private String email;

    public Cliente() {}

    public Cliente(String nome, String email) {
        this.nome = nome;
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

    public String getEmail() {

```

```

        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

Passo 4: Criar o DAO (Data Access Object)

Vamos criar a classe **ClienteDAO** no pacote **persistence**, que gerenciará as operações de persistência, como salvar, atualizar, excluir e listar clientes.

4.1 Criar a Classe ClienteDAO

```

package persistence;

import model.Cliente;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.List;

public class ClienteDAO {

    public void salvar(Cliente cliente) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.save(cliente);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public List<Cliente> listarTodos() {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("from Cliente", Cliente.class).list();
        }
    }

    public void atualizar(Cliente cliente) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.update(cliente);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public void excluir(Cliente cliente) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.delete(cliente);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
        }
    }
}

```

```

        e.printStackTrace();
    }
}

public Cliente buscarPorId(Long id) {
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        return session.get(Cliente.class, id);
    }
}
}
}

```

Passo 5: Criar a Interface Gráfica com Swing

Agora, vamos criar a interface gráfica no pacote **view**, onde o usuário pode interagir com a aplicação e realizar as operações CRUD.

5.1 Criar a Classe ClienteView

No pacote **view**, crie a classe **ClienteView**:

```

package view;

import model.Cliente;
import persistence.ClienteDAO;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

public class ClienteView extends JFrame {
    private JTextField nomeField;
    private JTextField emailField;
    private DefaultTableModel tableModel;
    private JTable clienteTable;
    private Cliente clienteSelecionado = null;

    public ClienteView() {
        setTitle("Cadastro de Clientes");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel(new GridLayout(4, 2));
        inputPanel.add(new JLabel("Nome:"));
        nomeField = new JTextField();
        inputPanel.add(nomeField);

        inputPanel.add(new JLabel("Email:"));
        emailField = new JTextField();
        inputPanel.add(emailField);

        JButton salvarButton = new JButton("Salvar");
        JButton alterarButton = new JButton("Alterar");
        JButton excluirButton = new JButton("Excluir");

        inputPanel.add(salvarButton);
        inputPanel.add(alterarButton);
        inputPanel.add(excluirButton);

        add(inputPanel, BorderLayout.NORTH);

        tableModel = new DefaultTableModel(new Object[]{"ID", "Nome", "Email"}, 0);
        clienteTable = new JTable(tableModel);
        add(new JScrollPane(clienteTable), BorderLayout.CENTER);
    }
}

```

```

salvarButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String nome = nomeField.getText();
        String email = emailField.getText();
        Cliente cliente = new Cliente(nome, email);
        ClienteDAO clienteDAO = new ClienteDAO();
        clienteDAO.salvar(cliente);
        atualizarTabela();
        limparCampos();
    }
});

alterarButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (clienteSelecionado != null) {
            clienteSelecionado.setNome(nomeField.getText());
            clienteSelecionado.setEmail(emailField.getText());
            ClienteDAO clienteDAO = new ClienteDAO();
            clienteDAO.atualizar(clienteSelecionado);
            atualizarTabela();
            limparCampos();
            clienteSelecionado = null;
        } else {
            JOptionPane.showMessageDialog(null, "Selecione um cliente para alterar.");
        }
    }
});

excluirButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (clienteSelecionado != null) {
            ClienteDAO clienteDAO = new ClienteDAO();
            clienteDAO.excluir(clienteSelecionado);
            atualizarTabela();
            limparCampos();
            clienteSelecionado = null;
        } else {
            JOptionPane.showMessageDialog(null, "Selecione um cliente para excluir.");
        }
    }
});

clienteTable.getSelectionModel().addListSelectionListener(event -> {
    if (!event.getValueIsAdjusting() && clienteTable.getSelectedRow() != -1) {
        Long id = (Long) clienteTable.getValueAt(clienteTable.getSelectedRow(), 0);
        ClienteDAO clienteDAO = new ClienteDAO();
        clienteSelecionado = clienteDAO.buscarPorId(id);
        if (clienteSelecionado != null) {
            nomeField.setText(clienteSelecionado.getNome());
            emailField.setText(clienteSelecionado.getEmail());
        }
    }
});

atualizarTabela();
}

private void atualizarTabela() {
    tableModel.setRowCount(0);
    ClienteDAO clienteDAO = new ClienteDAO();
    List<Cliente> clientes = clienteDAO.listarTodos();
    for (Cliente cliente : clientes) {
        tableModel.addRow(new Object[]{cliente.getId(), cliente.getNome(), cliente.getEmail()});
    }
}

private void limparCampos() {
    nomeField.setText("");
    emailField.setText("");
}
}

```

A classe `ClienteView` é a parte responsável pela **interface gráfica** da nossa aplicação CRUD. Utilizando componentes do **Swing**, ela fornece a interface onde o usuário pode **inserir, visualizar, alterar** e **excluir** dados de clientes. A seguir, vamos detalhar cada parte da classe para que você compreenda como ela funciona e como os componentes interagem entre si.

Aqui está um resumo das principais funcionalidades:

- **Campos de entrada** para o nome e o email do cliente.
- **Botões** para realizar operações CRUD: Salvar, Alterar e Excluir.
- **Tabela (JTable)** para listar os clientes cadastrados.
- Ações associadas aos botões para realizar as operações de CRUD (criar, listar, alterar e excluir).

Nela temos um painel chamado **inputPanel** que será responsável por abrigar os campos de entrada para o nome e email, além de botões para realizar as ações:

- **JPanel com GridLayout(3, 2):** O painel usa um **GridLayout** com 3 linhas e 2 colunas. Cada linha contém um rótulo (`JLabel`) seguido de um campo de entrada (`TextField`).
- **JLabel("Nome:")** e **JLabel("Email:")**: Esses rótulos são adicionados para identificar os campos de entrada.
- **nomeField** e **emailField** são os componentes onde o usuário insere os dados.
- **salvarButton**: Ao clicar neste botão, o usuário poderá salvar um novo cliente.
- **alterarButton**: Permite que o usuário altere os dados de um cliente selecionado.
- **excluirButton**: Exclui o cliente selecionado da tabela e do banco de dados.

Além disso temos a tabela de clientes, local onde os dados serão exibidos, com três colunas: **ID, Nome, e Email**.

- **DefaultTableModel**: Define o modelo de dados da tabela, especificando que a tabela terá três colunas (ID, Nome, e Email).
- **JTable(clienteTable)**: Cria a tabela usando o modelo `TableModel`.
- **JScrollPane**: A tabela é colocada dentro de um `JScrollPane` para permitir o **scroll** vertical e horizontal, caso haja muitos dados.

Passo 6: Criar a Classe `GerenciarClientes`

Crie a classe **GerenciarClientes** que iniciará a interface gráfica:

6.1 Criar a Classe `GerenciarClientes`

```
import view.ClienteView;

public class GerenciarClientes {
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(() -> {
            ClienteView clienteView = new ClienteView();
            clienteView.setVisible(true);
        });
    }
}
```

```
}  
}
```

Passo 7: Criando o Banco de Dados MySQL

Antes de executar a aplicação e testar o CRUD completo, precisamos configurar o banco de dados **MySQL** que será utilizado para armazenar os dados dos clientes. Vamos criar o banco de dados e uma tabela para os clientes, embora o Hibernate cuide automaticamente da criação da tabela (graças à configuração `hibernate.hbm2ddl.auto`), vamos garantir que o banco de dados está pronto para receber os dados.

Requisitos:

- O **MySQL** deve estar instalado na sua máquina.
- Você deve ter um usuário com permissão para criar bancos de dados e tabelas.
- Use um cliente MySQL (como o **MySQL Workbench**, **phpMyAdmin** ou o próprio **MySQL CLI**) para executar os comandos SQL.

7.1: Conectando ao MySQL

No **MySQL Workbench** ou outro cliente gráfico, conecte-se ao servidor MySQL como faria normalmente.

7.2: Criando o Banco de Dados

Vamos criar um banco de dados chamado **cadastro_clientes**, que será utilizado para armazenar os dados dos clientes na nossa aplicação.

Execute o seguinte comando SQL no MySQL CLI ou Workbench:

```
CREATE DATABASE cadastro_clientes;
```

Este comando cria um banco de dados chamado `cadastro_clientes`.

7.3: Usando o Banco de Dados Criado

Após criar o banco de dados, precisamos garantir que todas as operações subsequentes ocorram dentro dele. Para selecionar o banco de dados recém-criado, execute o comando:

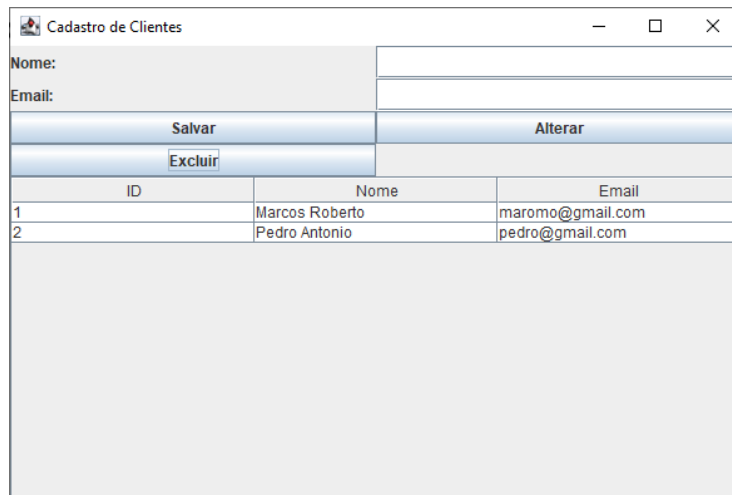
```
USE cadastro_clientes;
```

Agora todas as operações serão executadas dentro do banco de dados **cadastro_clientes**.

Passo 8: Testando o CRUD Completo

Agora que você tem todas as partes configuradas, execute a classe GerenciarClientes. Isso abrirá a interface gráfica onde você poderá:

1. **Inserir:** Adicionar um novo cliente.
2. **Listar:** Visualizar todos os clientes cadastrados.
3. **Alterar:** Atualizar os dados de um cliente existente.
4. **Excluir:** Remover um cliente.



A interface gráfica, intitulada 'Cadastro de Clientes', apresenta campos de entrada para 'Nome' e 'Email'. Abaixo desses campos, há três botões: 'Salvar', 'Alterar' e 'Excluir'. Na parte inferior, uma tabela exibe a lista de clientes cadastrados.

ID	Nome	Email
1	Marcos Roberto	maromo@gmail.com
2	Pedro Antonio	pedro@gmail.com

Conclusão

Este projeto oferece uma visão prática de como combinar **Swing** para construir uma interface gráfica amigável, **Hibernate** e **Jakarta Persistence** para gerenciar a persistência dos dados, e **MySQL** como banco de dados. A estrutura organizada em pacotes torna o código modular, facilitando a manutenção e o entendimento das diferentes partes do sistema.

Cordialmente,

Prof. Me. Marcos R Moraes (Maromo)