|  |  |
| --- | --- |
|  | *Universidade Estácio Campus Polo Casa Amarela*  *Curso de Desenvolvimento Full Stack Relatório da Missão Prática 2 - Mundo 3* |
| *Disciplina:* | RPG0016 – Backend sem banco de dados não tem |
| *Nome:* | João Gilberto dos Santos |
| *Turma:* | 2022.4 |

***1º Título da Prática***: **Criando o Banco de Dados**

### 2º Objetivo da Prática

O objetivo deste relatório é demonstrar os resultados obtidos durante a criação de um banco de dados e toda sua funcionalidade.

### 

### 

### 3º Códigos Solicitados:

### USE Loja;

### GO

### CREATE TABLE [pessoa] (

### id\_pessoa integer NOT NULL IDENTITY(1,1),

### nome varchar(255) NOT NULL,

### endereco varchar(255) NOT NULL,

### cidade varchar(255) NOT NULL,

### estado char(2) NOT NULL,

### telefone varchar(11) NOT NULL,

### email varchar(255) NOT NULL,

### CONSTRAINT [PK\_PESSOA] PRIMARY KEY CLUSTERED ([id\_pessoa] ASC)

### )

### GO

### CREATE TABLE [pessoa\_fisica] (

### id\_pessoa integer NOT NULL,

### cpf varchar(11) NOT NULL,

### CONSTRAINT [PK\_PESSOA\_FISICA] PRIMARY KEY CLUSTERED ([id\_pessoa] ASC),

### CONSTRAINT [FK\_PESSOA\_FISICA\_PESSOA] FOREIGN KEY ([id\_pessoa]) REFERENCES [pessoa] ([id\_pessoa])

### ON UPDATE CASCADE

### ON DELETE CASCADE

### )

### GO

### CREATE TABLE [pessoa\_juridica] (

### id\_pessoa integer NOT NULL,

### cnpj varchar(20) NOT NULL,

### CONSTRAINT [PK\_PESSOA\_JURIDICA] PRIMARY KEY CLUSTERED ([id\_pessoa] ASC),

### CONSTRAINT [FK\_PESSOA\_JURIDICA\_PESSOA] FOREIGN KEY ([id\_pessoa]) REFERENCES [pessoa] ([id\_pessoa])

### ON UPDATE CASCADE

### ON DELETE CASCADE

### )

### GO

### CREATE TABLE [produto] (

### id\_produto integer NOT NULL IDENTITY(1,1),

### nome varchar(255) NOT NULL,

### quantidade integer NOT NULL,

### precoVenda numeric(10,2) NOT NULL,

### CONSTRAINT [PK\_PRODUTO] PRIMARY KEY CLUSTERED ([id\_produto] ASC)

### )

### GO

### CREATE TABLE [usuario] (

### id\_usuario integer NOT NULL IDENTITY(1,1),

### login varchar(25) NOT NULL,

### senha varchar(25) NOT NULL,

### CONSTRAINT [PK\_USUARIO] PRIMARY KEY CLUSTERED ([id\_usuario] ASC)

### )

### GO

### CREATE TABLE [movimento] (

### id\_movimento integer NOT NULL IDENTITY(1,1),

### id\_pessoa integer NOT NULL,

### id\_produto integer NOT NULL,

### id\_usuario integer NOT NULL,

### quantidade integer NOT NULL,

### tipo char(1) NOT NULL,

### valor\_unitario numeric(10,2) NOT NULL,

### CONSTRAINT [PK\_MOVIMENTO] PRIMARY KEY CLUSTERED ([id\_movimento] ASC)

### )

### GO

### ALTER TABLE [movimento] WITH CHECK ADD CONSTRAINT [movimento\_fk0] FOREIGN KEY ([id\_pessoa])

### REFERENCES [pessoa] ([id\_pessoa])

### ON UPDATE CASCADE

### ON DELETE CASCADE

### GO

### ALTER TABLE [movimento] CHECK CONSTRAINT [movimento\_fk0]

### GO

### ALTER TABLE [movimento] WITH CHECK ADD CONSTRAINT [movimento\_fk1] FOREIGN KEY ([id\_produto])

### REFERENCES [produto] ([id\_produto])

### ON UPDATE CASCADE

### GO

### ALTER TABLE [movimento] CHECK CONSTRAINT [movimento\_fk1]

### GO

### ALTER TABLE [movimento] WITH CHECK ADD CONSTRAINT [movimento\_fk2] FOREIGN KEY ([id\_usuario])

### REFERENCES [usuario] ([id\_usuario])

### ON UPDATE CASCADE

### GO

### ALTER TABLE [movimento] CHECK CONSTRAINT [movimento\_fk2]

### GO

### 4º Os resultados da execução dos códigos:

### 4.1 Tabela dbo.pessoa

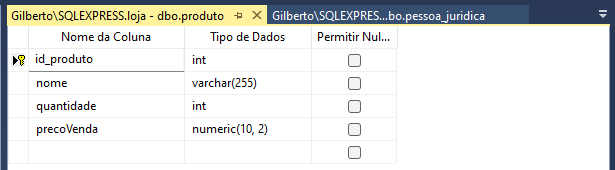
### 

### 4.2 Tabela dbo.pessoa\_fisica

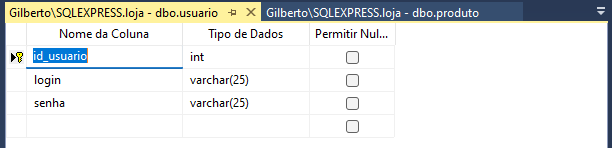
### 

### 4.3 Tabela dbo.pessoa\_juridica

### 4.4 Tabela dbo.produto

******

### 4.5 Tabela dbo.usuario

******

***5°*** *Análise e Conclusão:*

**A° Qual a importância dos componentes de middleware, como o JDBC?**

**R:** Em termos simples, o middleware é um software que atua como uma ponte entre diferentes aplicativos e sistemas. Ele facilita a comunicação e a integração entre diversas tecnologias, como bancos de dados, sistemas operacionais e linguagens de programação.

**Abstração:** O middleware esconde a complexidade da infraestrutura subjacente, permitindo que os desenvolvedores se concentrem na lógica de negócios da aplicação.

**Reutilização:** Componentes de middleware podem ser reutilizados em diversos projetos, acelerando o desenvolvimento e reduzindo custos.

**Interoperabilidade:** O middleware facilita a integração de sistemas heterogêneos, permitindo que diferentes aplicações se comuniquem de forma eficiente.

**Escalabilidade:** Muitos componentes de middleware são projetados para suportar cargas de trabalho crescentes, garantindo a performance e a disponibilidade dos sistemas.

O JDBC (Java Database Connectivity) é um API (Interface de Programação de Aplicativos) para conectar aplicações Java a bancos de dados. Ele é um dos componentes de middleware mais utilizados no mundo.

**Abstração de Bancos de Dados:** O JDBC fornece uma interface padrão para acessar diferentes bancos de dados (MySQL, Oracle, PostgreSQL, etc.), permitindo que os desenvolvedores escrevam código que seja independente do banco de dados utilizado.

**Simplificação do Acesso a Dados:** O JDBC simplifica tarefas comuns como a execução de consultas SQL, a manipulação de resultados e o tratamento de exceções.

**Integração com Outras Tecnologias:** O JDBC se integra facilmente com outras tecnologias Java, como frameworks de persistência (Hibernate, JPA) e servidores de aplicação (Tomcat, JBoss).

***B° Qual a diferença no uso de Statement ou PreparedStatement para a***

***manipulação de dados?***

***R:*** A escolha entre Statement e PreparedStatement para manipulação de dados em Java, especialmente quando se trabalha com bancos de dados relacionais via JDBC, é crucial para otimizar o desempenho e a segurança das suas aplicações.

***Statement:***

***Criação:*** A cada execução de uma consulta SQL, um novo objeto Statement é criado.

***Preenchimento:*** A consulta SQL é concatenada diretamente com os valores dos parâmetros, o que pode levar a problemas de injeção de SQL.

***PreparedStatement:***

***Criação***: Um único objeto PreparedStatement é criado para uma determinada consulta SQL, com placeholders para os parâmetros.

***Preenchimento:*** Os valores dos parâmetros são definidos separadamente, utilizando métodos como setString(), setInt(), etc., evitando problemas de injeção de SQL.

***C° Como o padrão DAO melhora a manutenibilidade do software?***

***R:*** O padrão DAO (Data Access Object) é uma ferramenta poderosa na construção de aplicações, especialmente quando se trata de manter a organização e a flexibilidade do código. Ao separar a lógica de acesso a dados da lógica de negócio, o DAO contribui significativamente para a manutenibilidade do software.

**Principais Benefícios para a Manutenibilidade:**

**Abstração da Fonte de Dados:Isolamento:** O DAO cria uma camada de abstração entre o código da aplicação e a fonte de dados (banco de dados, arquivo, etc.). Isso significa que você pode mudar a fonte de dados sem afetar a lógica de negócios.

**Flexibilidade:** Ao trocar a fonte de dados, é necessário apenas modificar a implementação do DAO, mantendo o restante do código intacto.

***D° Como a herança é refletida no banco de dados, quando lidamos com um***

***modelo estritamente relacional?***

***R:*** A herança, um conceito fundamental na programação orientada a objetos, não tem um mapeamento direto para o modelo relacional de dados. Isso ocorre porque os bancos relacionais são baseados em tabelas, enquanto a herança implica em hierarquias de classes.

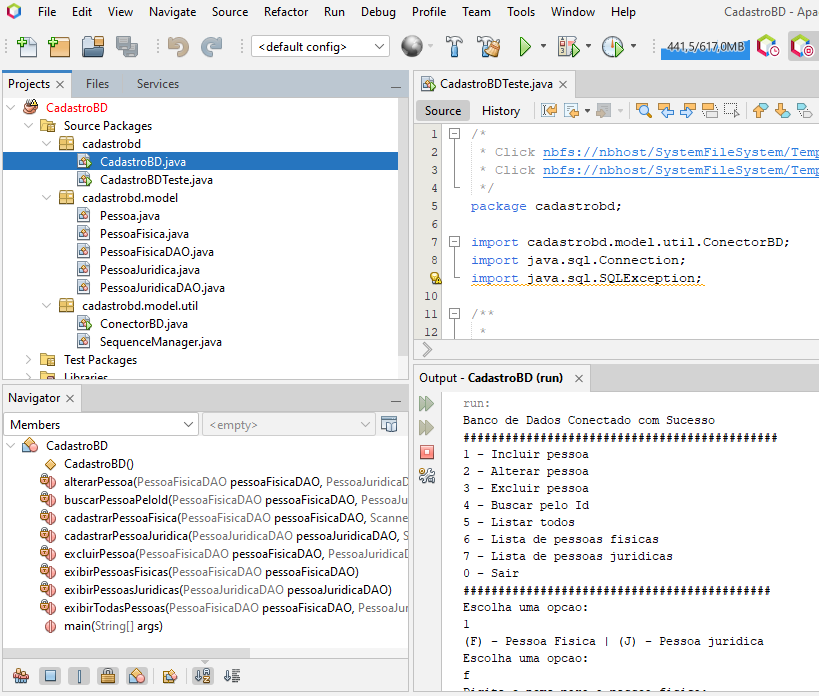
No entanto, existem algumas estratégias comuns para modelar a herança em bancos relacionais:

Em Resumo, modelar a herança em bancos relacionais exige um cuidadoso planejamento e pode envolver trade-offs entre diferentes estratégias. A escolha da melhor abordagem dependerá das necessidades específicas da aplicação e do banco de dados.

***1º Título da Prática***: **Conectando ao Banco de Dados**

### 2º Objetivo da Prática

O objetivo deste relatório é demonstrar os resultados obtidos após a implementação de um banco de dados e seu funcionamento conforme foi solicitado pelo docente desta materia.



### 

### 3º Códigos Solicitados:

### 3.1

### package cadastrobd;

### import cadastrobd.model.PessoaFisicaDAO;

### import cadastrobd.model.PessoaJuridicaDAO;

### import cadastrobd.model.PessoaFisica;

### import cadastrobd.model.PessoaJuridica;

### import cadastrobd.model.util.ConectorBD;

### import java.sql.Connection;

### import java.sql.SQLException;

### import java.util.List;

### import java.util.Scanner;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class CadastroBD {

### 

### public static void main(String[] args) {

### 

### 

### Scanner scanner = new Scanner(System.in);

### Connection conn = ConectorBD.getConnection();

### PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO(conn);

### PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO(conn);

### 

### int escolha;

### do {

### System.out.println("#############################################");

### System.out.println("1 - Incluir pessoa");

### System.out.println("2 - Alterar pessoa");

### System.out.println("3 - Excluir pessoa");

### System.out.println("4 - Buscar pelo Id");

### System.out.println("5 - Listar todos");

### System.out.println("6 - Lista de pessoas fisicas");

### System.out.println("7 - Lista de pessoas juridicas");

### System.out.println("0 - Sair");

### System.out.println("############################################");

### System.out.println("Escolha uma opcao: ");

### escolha = scanner.nextInt();

### scanner.nextLine();

### try {

### switch (escolha) {

### case 1:

### System.out.println("(F) - Pessoa Fisica | (J) - Pessoa juridica");

### System.out.println("Escolha uma opcao: ");

### char tipoInclusao = scanner.next().charAt(0);

### scanner.nextLine();

### if (tipoInclusao == 'F' || tipoInclusao == 'f') {

### cadastrarPessoaFisica(pessoaFisicaDAO, scanner);

### } else if (tipoInclusao == 'J' || tipoInclusao == 'j') {

### cadastrarPessoaJuridica(pessoaJuridicaDAO, scanner);

### } else {

### System.out.println("## > A opcao escolhida invalida.");

### }

### break;

### case 2:

### alterarPessoa(pessoaFisicaDAO, pessoaJuridicaDAO, scanner);

### break;

### case 3:

### excluirPessoa(pessoaFisicaDAO, pessoaJuridicaDAO, scanner);

### break;

### case 4:

### buscarPessoaPeloId(pessoaFisicaDAO, pessoaJuridicaDAO, scanner);

### break;

### case 5:

### exibirTodasPessoas(pessoaFisicaDAO, pessoaJuridicaDAO);

### break;

### case 6:

### exibirPessoasFisicas(pessoaFisicaDAO);

### break;

### case 7:

### exibirPessoasJuridicas(pessoaJuridicaDAO);

### break;

### case 0:

### System.out.println("## > Saindo....");

### break;

### default:

### System.out.println("## > Opcao invalida. Verifique a opcao escolhida e tente novamente");

### }

### } catch (SQLException e) {

### System.out.println("## > Erro de banco de dados: " + e.getMessage());

### 

### }

### } while (escolha != 0);

### }

### // Métods para cadastrar Pessoa Fisica

### private static void cadastrarPessoaFisica(PessoaFisicaDAO pessoaFisicaDAO, Scanner scanner) throws SQLException {

### System.out.println("Digite o nome para a pessoa fisica:");

### String nome = scanner.nextLine();

### System.out.println("Digite o endereco:");

### String logradouro = scanner.nextLine();

### System.out.println("Digite a cidade:");

### String cidade = scanner.nextLine();

### System.out.println("Digite o estado:");

### String estado = scanner.nextLine();

### System.out.println("Digite o telefone:");

### String telefone = scanner.nextLine();

### System.out.println("Digite o e-mail:");

### String email = scanner.nextLine();

### System.out.println("Digite o CPF:");

### String cpf = scanner.nextLine();

### PessoaFisica novaPessoaFisica = new PessoaFisica(0, nome, logradouro, cidade, estado, telefone, email, cpf);

### pessoaFisicaDAO.inserirPessoaFisica(novaPessoaFisica);

### System.out.println("## > Pessoa Fisica cadastrada com sucesso!");

### }

### // Métods para cadastrar Pessoa Juridica

### private static void cadastrarPessoaJuridica(PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner) throws SQLException {

### System.out.println("Digite o nome para a pessoa juridica:");

### String nome = scanner.nextLine();

### System.out.println("Digite o endereco:");

### String logradouro = scanner.nextLine();

### System.out.println("Digite a cidade:");

### String cidade = scanner.nextLine();

### System.out.println("Digite o estado:");

### String estado = scanner.nextLine();

### System.out.println("Digite o telefone:");

### String telefone = scanner.nextLine();

### System.out.println("Digite o e-mail:");

### String email = scanner.nextLine();

### System.out.println("Digite o CNPJ:");

### String cnpj = scanner.nextLine();

### PessoaJuridica novaPessoaJuridica = new PessoaJuridica(0, nome, logradouro, cidade, estado, telefone, email, cnpj);

### pessoaJuridicaDAO.incluir(novaPessoaJuridica);

### System.out.println("## > Pessoa Jurídica cadastrada com sucesso.");

### }

### // Métods para alterar Pessoas

### private static void alterarPessoa(PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner) throws SQLException {

### System.out.println("F - Alterar pessoa fisica | J - Alterar pessoa juridica");

### System.out.println("Escolha uma opcao: ");

### char tipoPessoa = scanner.next().charAt(0);

### scanner.nextLine();

### int id;

### if (tipoPessoa == 'F' || tipoPessoa == 'f') {

### System.out.println("Digite o ID da pessoa fisica que deseja alterar:");

### id = scanner.nextInt();

### scanner.nextLine();

### PessoaFisica pessoaExistente = pessoaFisicaDAO.getPessoa(id);

### if (pessoaExistente != null) {

### System.out.println("Digite o novo nome da pessoa fisica:");

### String novoNome = scanner.nextLine();

### System.out.println("Digite o novo endereco:");

### String novoLogradouro = scanner.nextLine();

### System.out.println("Digite a nova cidade:");

### String novaCidade = scanner.nextLine();

### System.out.println("Digite o novo estado:");

### String novoEstado = scanner.nextLine();

### System.out.println("Digite o novo telefone:");

### String novoTelefone = scanner.nextLine();

### System.out.println("Digite o novo e-mail:");

### String novoEmail = scanner.nextLine();

### System.out.println("Digite o novo CPF:");

### String novoCpf = scanner.nextLine();

### PessoaFisica novaPessoa = new PessoaFisica(id, novoNome, novoLogradouro, novaCidade, novoEstado, novoTelefone, novoEmail, novoCpf);

### pessoaFisicaDAO.alterar(novaPessoa);

### System.out.println("## > Pessoa fisica atualizada com sucesso.");

### } else {

### System.out.println("## > Pessoa fisica não encontrada.");

### }

### } else if (tipoPessoa == 'J' || tipoPessoa == 'j') {

### System.out.println("Digite o ID da pessoa juridica que deseja alterar:");

### 

### if (scanner.hasNextInt()) {

### id = scanner.nextInt();

### System.out.println("Voce adicionou o ID: " + id);

### } else {

### String input = scanner.next();

### System.out.println("Nao e permitido Strings: " + input);

### return;

### }

### 

### scanner.nextLine();

### PessoaJuridica pessoaExistente = pessoaJuridicaDAO.getPessoa(id);

### if (pessoaExistente != null) {

### System.out.println("Digite o novo nome da pessoa juridica:");

### String novoNome = scanner.nextLine();

### System.out.println("Digite o novo endereco:");

### String novoLogradouro = scanner.nextLine();

### System.out.println("Digite a nova cidade:");

### String novaCidade = scanner.nextLine();

### System.out.println("Digite o novo estado:");

### String novoEstado = scanner.nextLine();

### System.out.println("Digite o novo telefone:");

### String novoTelefone = scanner.nextLine();

### System.out.println("Digite o novo email:");

### String novoEmail = scanner.nextLine();

### System.out.println("Digite o novo CNPJ:");

### String novoCnpj = scanner.nextLine();

### PessoaJuridica novaPessoa = new PessoaJuridica(id, novoNome, novoLogradouro, novaCidade, novoEstado, novoTelefone, novoEmail, novoCnpj);

### pessoaJuridicaDAO.alterar(novaPessoa);

### System.out.println("## > ## >Pessoa juridica atualizada com sucesso.");

### } else {

### System.out.println("## > Pessoa juridica não encontrada.");

### }

### } else {

### System.out.println("## > Opcao invalida.");

### }

### }

### // Métods para excluir Pessoas

### private static void excluirPessoa(PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner) throws SQLException {

### System.out.println("(F) - Excluir pessoa fisica | (J) - Excluir pessoa juridica");

### System.out.println("Escolha uma opcao: ");

### char tipoPessoa = scanner.next().charAt(0);

### scanner.nextLine();

### if (tipoPessoa == 'F' || tipoPessoa == 'f') {

### System.out.println("Digite o ID da pessoa fisica para ser removida:");

### int id = scanner.nextInt();

### pessoaFisicaDAO.excluir(id);

### System.out.println("## > Pessoa fisica removida com sucesso.");

### } else if (tipoPessoa == 'J' || tipoPessoa == 'j') {

### System.out.println("Digite o ID da pessoa juridica para ser removida:");

### int id = scanner.nextInt();

### pessoaJuridicaDAO.excluir(id);

### System.out.println("## > Pessoa juridica excluida com sucesso.");

### } else {

### System.out.println("## > Opcao invalida");

### }

### }

### // Métods para buscar Pessoas

### private static void buscarPessoaPeloId(PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner) throws SQLException {

### System.out.println("(F) - Buscar pessoa fisica | (J) - Buscar pessoa juridica");

### System.out.println("Escolha uma opcao: ");

### char tipoPessoa = scanner.next().charAt(0);

### scanner.nextLine();

### if (tipoPessoa == 'F' || tipoPessoa == 'f') {

### System.out.println("Digite o ID da pessoa fisica:");

### int id = scanner.nextInt();

### scanner.nextLine();

### PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);

### if (pessoaFisica != null) {

### System.out.println("Informacoes da pessoa fisica:");

### System.out.println("ID: " + pessoaFisica.getId());

### System.out.println("Nome: " + pessoaFisica.getNome());

### System.out.println("Endereco: " + pessoaFisica.getEndereco());

### System.out.println("Cidade: " + pessoaFisica.getCidade());

### System.out.println("Estado: " + pessoaFisica.getEstado());

### System.out.println("Telefone: " + pessoaFisica.getTelefone());

### System.out.println("E-mail: " + pessoaFisica.getEmail());

### System.out.println("CPF: " + pessoaFisica.getCpf());

### } else {

### System.out.println("## > Nao foi possivel encontrar a pessoa fisica");

### }

### } else if (tipoPessoa == 'J' || tipoPessoa == 'j') {

### System.out.println("Digite o ID da pessoa juridica:");

### int id = scanner.nextInt();

### scanner.nextLine();

### PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(id);

### if (pessoaJuridica != null) {

### System.out.println("Informacaoes da pessoa juridica:");

### System.out.println("ID: " + pessoaJuridica.getId());

### System.out.println("Nome: " + pessoaJuridica.getNome());

### System.out.println("Endereco: " + pessoaJuridica.getEndereco());

### System.out.println("Cidade: " + pessoaJuridica.getCidade());

### System.out.println("Estado: " + pessoaJuridica.getEstado());

### System.out.println("Telefone: " + pessoaJuridica.getTelefone());

### System.out.println("E-mail: " + pessoaJuridica.getEmail());

### System.out.println("CNPJ: " + pessoaJuridica.getCnpj());

### } else {

### System.out.println("## > Nao foi possivel encontrar a pessoa juridica.");

### }

### } else {

### System.out.println("## > Opcao invalida.");

### }

### }

### // Métods para exibir todas as Pessoas

### private static void exibirTodasPessoas(PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {

### System.out.println("Relacao de pessoas fisicas cadastradas:");

### List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();

### if (pessoasFisicas.isEmpty()) {

### System.out.println("## > Nenhuma cadastro de pessoa fisica.");

### } else {

### for (PessoaFisica pf : pessoasFisicas) {

### System.out.println("ID: " + pf.getId());

### System.out.println("Nome: " + pf.getNome());

### System.out.println("Endereco: " + pf.getEndereco());

### System.out.println("Cidade: " + pf.getCidade());

### System.out.println("Estado: " + pf.getEstado());

### System.out.println("Telefone: " + pf.getTelefone());

### System.out.println("E-mail: " + pf.getEmail());

### System.out.println("CPF: " + pf.getCpf());

### System.out.println();

### }

### }

### System.out.println("Relacao de pessoas juridicas cadastradas:");

### List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoasJuridicas();

### if (pessoasJuridicas.isEmpty()) {

### System.out.println("## > Nenhuma pessoa juridica cadastrada.");

### } else {

### for (PessoaJuridica pj : pessoasJuridicas) {

### System.out.println("ID: " + pj.getId());

### System.out.println("Nome: " + pj.getNome());

### System.out.println("Endereco: " + pj.getEndereco());

### System.out.println("Cidade: " + pj.getCidade());

### System.out.println("Estado: " + pj.getEstado());

### System.out.println("Telefone: " + pj.getTelefone());

### System.out.println("E-mail: " + pj.getEmail());

### System.out.println("CNPJ: " + pj.getCnpj());

### System.out.println();

### }

### }

### }

### // Métods para exibir Pessoas Fisicas

### private static void exibirPessoasFisicas(PessoaFisicaDAO pessoaFisicaDAO) throws SQLException {

### List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();

### if (pessoasFisicas.isEmpty()) {

### System.out.println("## > Nenhuma pessoa fisica cadastrada.");

### } else {

### for (PessoaFisica pf : pessoasFisicas) {

### System.out.println("ID: " + pf.getId());

### System.out.println("Nome: " + pf.getNome());

### System.out.println("Endereco: " + pf.getEndereco());

### System.out.println("Cidade: " + pf.getCidade());

### System.out.println("Estado: " + pf.getEstado());

### System.out.println("Telefone: " + pf.getTelefone());

### System.out.println("E-mail: " + pf.getEmail());

### System.out.println("CPF: " + pf.getCpf());

### System.out.println();

### }

### }

### }

### // Métods para exibir Pessoas Fisicas

### private static void exibirPessoasJuridicas(PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {

### List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoasJuridicas();

### if (pessoasJuridicas.isEmpty()) {

### System.out.println("## > Nenhuma pessoa juridica cadastrada.");

### } else {

### for (PessoaJuridica pj : pessoasJuridicas) {

### System.out.println("ID: " + pj.getId());

### System.out.println("Nome: " + pj.getNome());

### System.out.println("Endereco: " + pj.getEndereco());

### System.out.println("Cidade: " + pj.getCidade());

### System.out.println("Estado: " + pj.getEstado());

### System.out.println("Telefone: " + pj.getTelefone());

### System.out.println("E-mail: " + pj.getEmail());

### System.out.println("CNPJ: " + pj.getCnpj());

### System.out.println();

### }

### }

### }

### 

### }

### 3.2

### package cadastrobd.model;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class Pessoa {

### 

### private int id;

### private String nome;

### private String endereco;

### private String cidade;

### private String estado;

### private String telefone;

### private String email;

### public Pessoa() {}

### public Pessoa(int id, String nome, String endereco, String cidade, String estado, String telefone, String email) {

### this.id = id;

### this.nome = nome;

### this.endereco = endereco;

### this.cidade = cidade;

### this.estado = estado;

### this.telefone = telefone;

### this.email = email;

### }

### public void exibir() {

### System.out.println("ID: " + id);

### System.out.println("Nome: " + nome);

### System.out.println("Endereco: " + endereco);

### System.out.println("Cidade: " + cidade);

### System.out.println("Estado: " + estado);

### System.out.println("Telefone: " + telefone);

### System.out.println("Email: " + email);

### }

### public Integer getId() {

### return id;

### }

### public void setId(Integer id) {

### this.id = id;

### }

### public String getNome() {

### return nome;

### }

### public void setNome(String nome) {

### this.nome = nome;

### }

### public String getEndereco() {

### return endereco;

### }

### public void setEndereco(String endereco) {

### this.endereco = endereco;

### }

### public String getCidade() {

### return cidade;

### }

### public void setCidade(String cidade) {

### this.cidade = cidade;

### }

### public String getEstado() {

### return estado;

### }

### public void setEstado(String estado) {

### this.estado = estado;

### }

### public String getTelefone() {

### return telefone;

### }

### public void setTelefone(String telefone) {

### this.telefone = telefone;

### }

### public String getEmail() {

### return email;

### }

### public void setEmail(String email) {

### this.email = email;

### }

### }

### 

### 3.3

### package cadastrobd.model;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class PessoaFisica extends Pessoa {

### 

### private String cpf;

### // Construtor padrão

### public PessoaFisica() {}

### // Construtor completo

### public PessoaFisica(int id, String nome, String logradouro, String cidade, String estado, String telefone, String email, String cpf) {

### super(id, nome, logradouro, cidade, estado, telefone, email);

### this.cpf = cpf;

### }

### public String getCpf() {

### return cpf;

### }

### public void setCpf(String cpf) {

### this.cpf = cpf;

### }

### 

### @Override

### public void exibir() {

### super.exibir();

### System.out.println("CPF: " + cpf);

### }

### }

### 

### 3.4

### package cadastrobd.model;

### import java.sql.\*;

### import java.util.ArrayList;

### import java.util.List;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class PessoaFisicaDAO {

### 

### private final Connection conn;

### public PessoaFisicaDAO(Connection conexao) {

### this.conn = conexao;

### }

### public void inserirPessoaFisica(PessoaFisica pf) throws SQLException {

### String sqlPessoa = "INSERT INTO Pessoa (nome, endereco, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?)";

### String sqlPessoaFisica = "INSERT INTO pessoa\_fisica (id\_pessoa, cpf) VALUES (?, ?)";

### try {

### conn.setAutoCommit(false);

### int pessoaId = 0;

### // Inserir na tabela Pessoa

### try (PreparedStatement stPessoa = conn.prepareStatement(sqlPessoa, Statement.RETURN\_GENERATED\_KEYS)) {

### stPessoa.setString(1, pf.getNome());

### stPessoa.setString(2, pf.getEndereco());

### stPessoa.setString(3, pf.getCidade());

### stPessoa.setString(4, pf.getEstado());

### stPessoa.setString(5, pf.getTelefone());

### stPessoa.setString(6, pf.getEmail());

### stPessoa.executeUpdate();

### // Obter o ID gerado

### try (ResultSet rs = stPessoa.getGeneratedKeys()) {

### if (rs.next()) {

### pessoaId = rs.getInt(1);

### }

### }

### }

### // Inserir na tabela PessoaFisica

### try (PreparedStatement stPessoaFisica = conn.prepareStatement(sqlPessoaFisica)) {

### stPessoaFisica.setInt(1, pessoaId);

### stPessoaFisica.setString(2, pf.getCpf());

### stPessoaFisica.executeUpdate();

### }

### conn.commit();

### } catch (SQLException e) {

### conn.rollback();

### throw e;

### } finally {

### conn.setAutoCommit(true);

### }

### }

### public void alterar(PessoaFisica pf) throws SQLException {

### String sqlPessoa = "UPDATE Pessoa SET nome = ?, endereco = ?, cidade = ?, estado = ?, telefone = ?, email = ? WHERE id\_pessoa = ?";

### String sqlPessoaFisica = "UPDATE pessoa\_fisica SET cpf = ? WHERE id\_pessoa = ?";

### try {

### conn.setAutoCommit(false);

### // Atualizar na tabela Pessoa

### try (PreparedStatement stPessoa = conn.prepareStatement(sqlPessoa)) {

### stPessoa.setString(1, pf.getNome());

### stPessoa.setString(2, pf.getEndereco());

### stPessoa.setString(3, pf.getCidade());

### stPessoa.setString(4, pf.getEstado());

### stPessoa.setString(5, pf.getTelefone());

### stPessoa.setString(6, pf.getEmail());

### stPessoa.setInt(7, pf.getId());

### stPessoa.executeUpdate();

### }

### // Atualizar na tabela PessoaFisica

### try (PreparedStatement stPessoaFisica = conn.prepareStatement(sqlPessoaFisica)) {

### stPessoaFisica.setString(1, pf.getCpf());

### stPessoaFisica.setInt(2, pf.getId());

### stPessoaFisica.executeUpdate();

### }

### conn.commit();

### } catch (SQLException e) {

### conn.rollback();

### throw e;

### } finally {

### conn.setAutoCommit(true);

### }

### }

### public void excluir(Integer id) throws SQLException {

### String sqlPessoaFisica = "DELETE FROM pessoa\_fisica WHERE id\_pessoa = ?";

### String sqlPessoa = "DELETE FROM Pessoa WHERE id\_pessoa = ?";

### try {

### conn.setAutoCommit(false);

### // Excluir da tabela PessoaFisica

### try (PreparedStatement stPessoaFisica = conn.prepareStatement(sqlPessoaFisica)) {

### stPessoaFisica.setInt(1, id);

### stPessoaFisica.executeUpdate();

### }

### // Excluir da tabela Pessoa

### try (PreparedStatement stPessoa = conn.prepareStatement(sqlPessoa)) {

### stPessoa.setInt(1, id);

### stPessoa.executeUpdate();

### }

### conn.commit();

### } catch (SQLException e) {

### conn.rollback();

### throw e;

### } finally {

### conn.setAutoCommit(true);

### }

### }

### public PessoaFisica getPessoa(Integer id) throws SQLException {

### String sql = "SELECT Pessoa.id\_pessoa, Pessoa.nome, Pessoa.endereco, Pessoa.cidade, Pessoa.estado, Pessoa.telefone, Pessoa.email, PF.cpf "

### + "FROM Pessoa JOIN pessoa\_fisica PF ON Pessoa.id\_pessoa = PF.id\_pessoa WHERE Pessoa.id\_pessoa = ?";

### try (PreparedStatement st = conn.prepareStatement(sql)) {

### st.setInt(1, id);

### try (ResultSet rs = st.executeQuery()) {

### if (rs.next()) {

### return new PessoaFisica(

### rs.getInt("id\_pessoa"),

### rs.getString("nome"),

### rs.getString("endereco"),

### rs.getString("cidade"),

### rs.getString("estado"),

### rs.getString("telefone"),

### rs.getString("email"),

### rs.getString("cpf")

### );

### }

### }

### }

### return null;

### }

### public List<PessoaFisica> getPessoas() throws SQLException {

### List<PessoaFisica> list = new ArrayList<>();

### String sql = "SELECT Pessoa.id\_pessoa, Pessoa.nome, Pessoa.endereco, Pessoa.cidade, Pessoa.estado, Pessoa.telefone, Pessoa.email, PF.cpf "

### + "FROM Pessoa JOIN pessoa\_fisica PF ON Pessoa.id\_pessoa = PF.id\_pessoa";

### try (PreparedStatement st = conn.prepareStatement(sql); ResultSet rs = st.executeQuery()) {

### while (rs.next()) {

### list.add(new PessoaFisica(

### rs.getInt("id\_pessoa"),

### rs.getString("nome"),

### rs.getString("endereco"),

### rs.getString("cidade"),

### rs.getString("estado"),

### rs.getString("telefone"),

### rs.getString("email"),

### rs.getString("cpf")

### ));

### }

### }

### return list;

### }

### }

### 3.5

### package cadastrobd.model;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class PessoaJuridica extends Pessoa {

### 

### private String cnpj;

### 

### public PessoaJuridica() {}

### 

### public PessoaJuridica(int id, String nome, String endereco, String cidade, String estado, String telefone, String email, String cnpj) {

### super(id, nome, endereco, cidade, estado, telefone, email);

### this.cnpj = cnpj;

### }

### public String getCnpj() {

### return cnpj;

### }

### public void setCnpj(String cnpj) {

### this.cnpj = cnpj;

### }

### 

### @Override

### public void exibir() {

### super.exibir();

### System.out.println("CNPJ: " + cnpj);

### }

### }

### 3.6

### package cadastrobd.model;

### import cadastrobd.model.util.ConectorBD;

### import cadastrobd.model.util.SequenceManager;

### import java.sql.Connection;

### import java.sql.Statement;

### import java.sql.PreparedStatement;

### import java.sql.ResultSet;

### import java.sql.SQLException;

### import java.util.ArrayList;

### import java.util.List;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class PessoaJuridicaDAO {

### 

### private Connection conn;

### public PessoaJuridicaDAO(Connection conn) {

### this.conn = conn;

### }

### private PessoaJuridica extrairPessoaJuridica(ResultSet rs) throws SQLException {

### return new PessoaJuridica(

### rs.getInt("id\_pessoa"),

### rs.getString("nome"),

### rs.getString("endereco"),

### rs.getString("cidade"),

### rs.getString("estado"),

### rs.getString("telefone"),

### rs.getString("email"),

### rs.getString("cnpj")

### );

### }

### public PessoaJuridica getPessoa(int id) throws SQLException {

### final String sql = "SELECT Pessoa.id\_pessoa, Pessoa.nome, Pessoa.endereco, Pessoa.cidade, Pessoa.estado, Pessoa.telefone, Pessoa.email, PJ.cnpj\n"

### + "FROM Pessoa AS Pessoa JOIN pessoa\_juridica AS PJ ON Pessoa.id\_pessoa = PJ.id\_pessoa WHERE Pessoa.id\_pessoa = ?;";

### try (PreparedStatement stmt = conn.prepareStatement(sql)) {

### stmt.setInt(1, id);

### try (ResultSet rs = stmt.executeQuery()) {

### if (rs.next()) {

### return extrairPessoaJuridica(rs);

### }

### }

### }

### return null;

### }

### public List<PessoaJuridica> getPessoasJuridicas() throws SQLException {

### List<PessoaJuridica> list = new ArrayList<>();

### final String sql = "SELECT Pessoa.id\_pessoa, Pessoa.nome, Pessoa.endereco, Pessoa.cidade, Pessoa.estado, Pessoa.telefone, Pessoa.email, PJ.cnpj\n"

### + "FROM Pessoa AS Pessoa JOIN pessoa\_juridica AS PJ ON Pessoa.id\_pessoa = PJ.id\_pessoa;";

### try (PreparedStatement stmt = conn.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {

### while (rs.next()) {

### list.add(extrairPessoaJuridica(rs));

### }

### }

### return list;

### }

### public void incluir(PessoaJuridica pessoa) throws SQLException {

### final String sqlPessoa = "INSERT INTO Pessoa (nome, endereco, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?)";

### final String sqlPessoaJuridica = "INSERT INTO pessoa\_juridica (id\_pessoa, cnpj) VALUES (?, ?)";

### try {

### conn.setAutoCommit(false);

### int pessoaId = 0;

### try (PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa, Statement.RETURN\_GENERATED\_KEYS)) {

### stmtPessoa.setString(1, pessoa.getNome());

### stmtPessoa.setString(2, pessoa.getEndereco());

### stmtPessoa.setString(3, pessoa.getCidade());

### stmtPessoa.setString(4, pessoa.getEstado());

### stmtPessoa.setString(5, pessoa.getTelefone());

### stmtPessoa.setString(6, pessoa.getEmail());

### stmtPessoa.executeUpdate();

### try (ResultSet generatedKeys = stmtPessoa.getGeneratedKeys()) {

### if (generatedKeys.next()) {

### pessoaId = generatedKeys.getInt(1);

### }

### }

### }

### if (pessoaId == 0) {

### throw new SQLException("Falha ao inserir pessoa, nenhum ID foi gerado.");

### }

### try (PreparedStatement stmtPessoaJuridica = conn.prepareStatement(sqlPessoaJuridica)) {

### stmtPessoaJuridica.setInt(1, pessoaId);

### stmtPessoaJuridica.setString(2, pessoa.getCnpj());

### stmtPessoaJuridica.executeUpdate();

### }

### conn.commit();

### } catch (SQLException e) {

### conn.rollback();

### throw e;

### } finally {

### conn.setAutoCommit(true);

### }

### }

### public void alterar(PessoaJuridica pessoa) throws SQLException {

### final String sqlPessoa = "UPDATE Pessoa SET nome = ?, endereco = ?, cidade = ?, estado = ?, telefone = ?, email = ? WHERE id\_pessoa = ?";

### final String sqlPessoaJuridica = "UPDATE pessoa\_juridica SET cnpj = ? WHERE id\_pessoa = ?";

### try {

### conn.setAutoCommit(false);

### try (PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa)) {

### stmtPessoa.setString(1, pessoa.getNome());

### stmtPessoa.setString(2, pessoa.getEndereco());

### stmtPessoa.setString(3, pessoa.getCidade());

### stmtPessoa.setString(4, pessoa.getEstado());

### stmtPessoa.setString(5, pessoa.getTelefone());

### stmtPessoa.setString(6, pessoa.getEmail());

### stmtPessoa.setInt(7, pessoa.getId());

### stmtPessoa.executeUpdate();

### }

### try (PreparedStatement stmtPessoaJuridica = conn.prepareStatement(sqlPessoaJuridica)) {

### stmtPessoaJuridica.setString(1, pessoa.getCnpj());

### stmtPessoaJuridica.setInt(2, pessoa.getId());

### stmtPessoaJuridica.executeUpdate();

### }

### conn.commit();

### } catch (SQLException e) {

### conn.rollback();

### throw e;

### } finally {

### conn.setAutoCommit(true);

### }

### }

### public void excluir(int id) throws SQLException {

### String sqlPessoaJuridica = "DELETE FROM pessoa\_juridica WHERE id\_pessoa = ?";

### String sqlPessoa = "DELETE FROM Pessoa WHERE id\_pessoa = ?";

### try {

### conn.setAutoCommit(false);

### try (PreparedStatement stmtPessoaJuridica = conn.prepareStatement(sqlPessoaJuridica)) {

### stmtPessoaJuridica.setInt(1, id);

### stmtPessoaJuridica.executeUpdate();

### }

### try (PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa)) {

### stmtPessoa.setInt(1, id);

### stmtPessoa.executeUpdate();

### }

### conn.commit();

### } catch (SQLException e) {

### conn.rollback();

### throw e;

### } finally {

### conn.setAutoCommit(true);

### }

### }

### 

### }

### 3.7

### package cadastrobd.model.util;

### import java.sql.Connection;

### import java.sql.DriverManager;

### import java.sql.ResultSet;

### import java.sql.SQLException;

### import java.sql.Statement;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class ConectorBD {

### 

### public static void main(String[] args) {

### getConnection();

### 

### }

### private static Connection conn = null;

### public static Connection getConnection() {

### if (conn == null) {

### try {

### // URL de conexão atualizada para incluir o banco de dados

### String url = "jdbc:sqlserver://localhost\\DESKTOP-3F3K0EB:1433;databaseName=loja;encrypt=true;trustServerCertificate=true;";

### String user = "loja";

### String password = "loja";

### conn = DriverManager.getConnection(url, user, password);

### System.out.println("Banco de Dados Conectado com Sucesso");

### } catch (SQLException e) {

### System.out.println("Erro no Banco de Dados");

### throw new RuntimeException("Erro ao obter conexao com o banco de dados: " + e.getMessage(), e);

### }

### }

### return conn;

### }

### public static void closeStatement(Statement statement) {

### try {

### if (statement != null) {

### statement.close();

### }

### } catch (SQLException e) {

### throw new RuntimeException("Erro ao fechar o statement: " + e.getMessage(), e);

### }

### }

### public static void closeResultSet(ResultSet resultSet) {

### try {

### if (resultSet != null) {

### resultSet.close();

### }

### } catch (SQLException e) {

### throw new RuntimeException("Erro ao fechar o resultSet: " + e.getMessage(), e);

### }

### }

### public static void closeConnection() {

### try {

### if (conn != null) {

### conn.close();

### }

### } catch (SQLException e) {

### throw new RuntimeException("Erro ao fechar a conexão com o banco de dados: " + e.getMessage(), e);

### }

### }

### void close(ResultSet resultSet) {

### throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody

### }

### }

### 

### 3.8

### package cadastrobd.model.util;

### import java.sql.Connection;

### import java.sql.PreparedStatement;

### import java.sql.ResultSet;

### import java.sql.SQLException;

### /\*\*

### \*

### \* @author mrjoa

### \*/

### public class SequenceManager {

### 

### private final ConectorBD conectorBD;

### public SequenceManager() {

### this.conectorBD = new ConectorBD();

### }

### public int getNextValue(String sequenceName) throws SQLException {

### int proximoValor = 0;

### Connection connection = null;

### PreparedStatement preparedStatement = null;

### ResultSet resultSet = null;

### try {

### connection = conectorBD.getConnection();

### String sql = "SELECT NEXT VALUE FOR " + sequenceName + " AS NextVal";

### preparedStatement = connection.prepareStatement(sql);

### resultSet = preparedStatement.executeQuery();

### if (resultSet.next()) {

### proximoValor = resultSet.getInt("NextVal");

### }

### } finally {

### conectorBD.closeResultSet(resultSet);

### conectorBD.closeStatement(preparedStatement);

### conectorBD.closeConnection();

### }

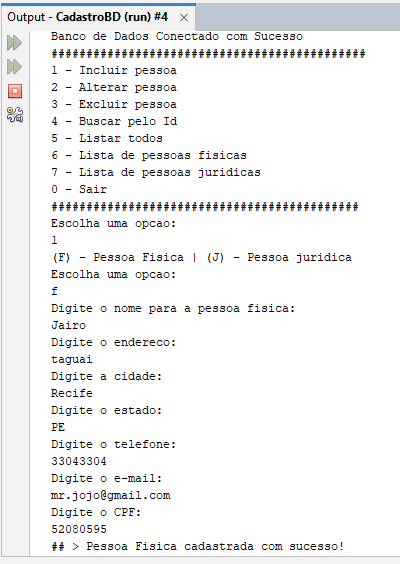
### return proximoValor;

### }

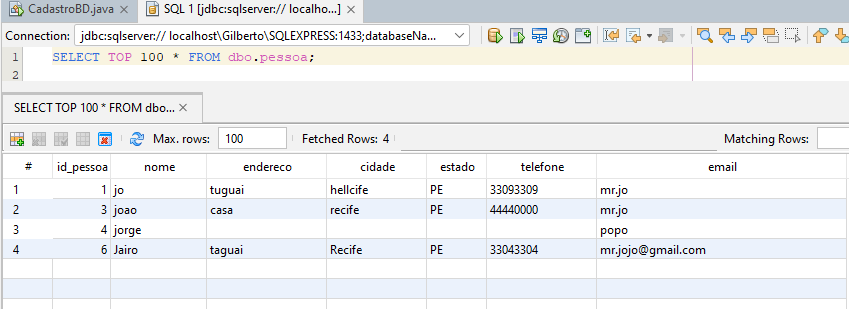
### 

### 4º Os resultados da execução dos códigos:

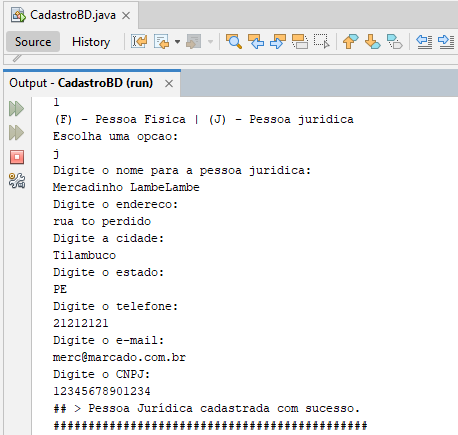
### 4.1 Incluir Pessoa Fisica:



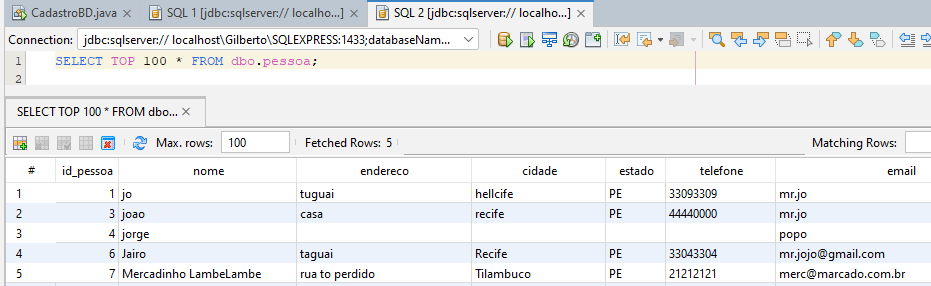
### Resultado:



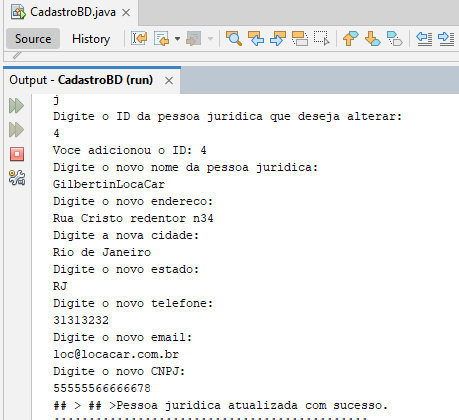
### 4.1 Incluir Pessoa Juridica:



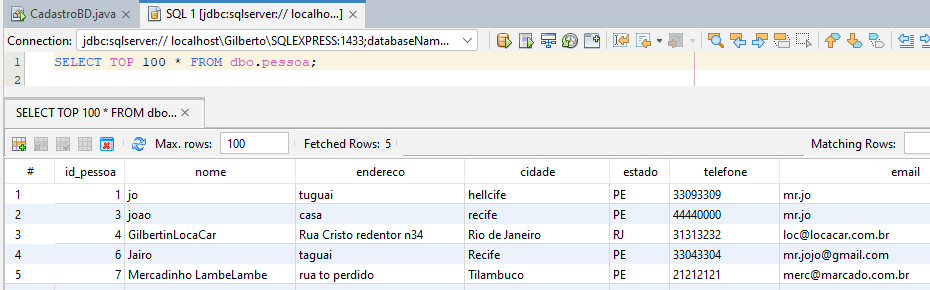
### Resultado:



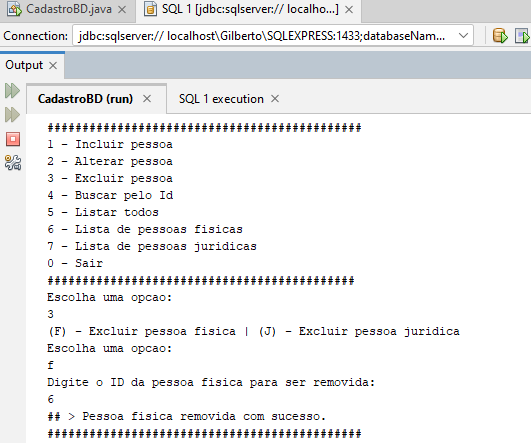
***4.2*** Alterar Pessoa



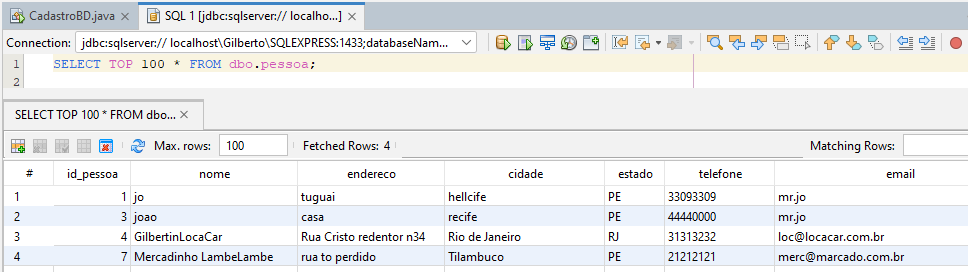
**Resultado:** O ID: 4 foi alterado com o novo nome GilbertinLocaCar e os demais dados.



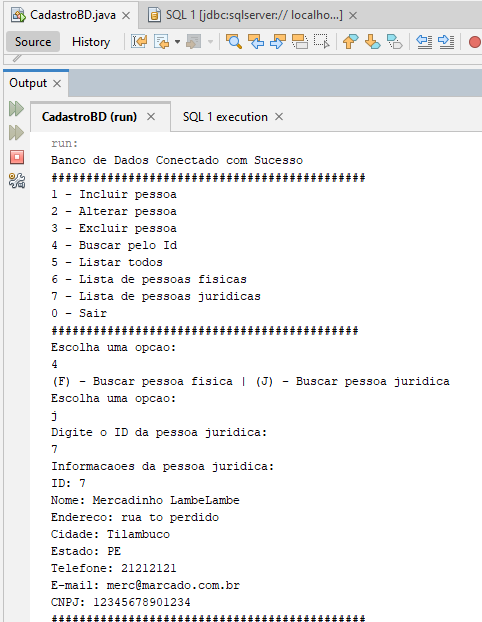
***4.3*** Excluir Pessoa



**Resultado:** Foi excluido o ID 6 .

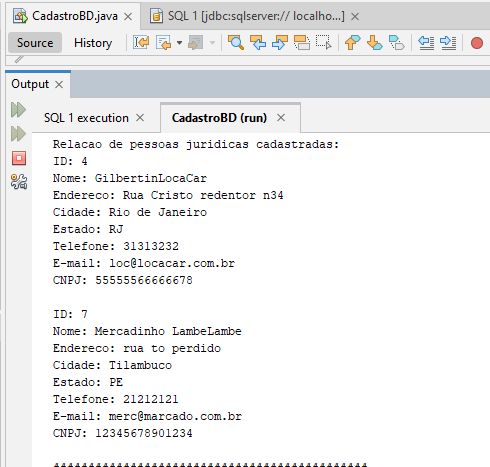
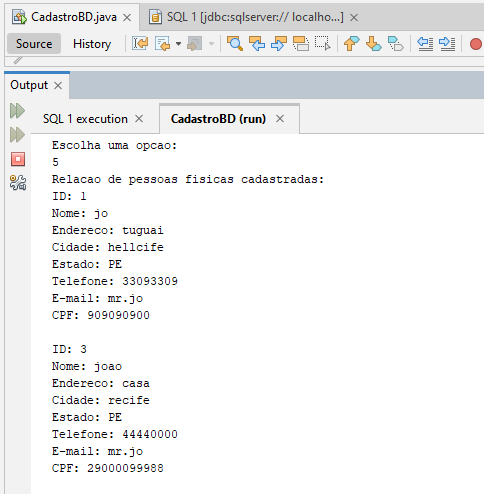


***4.4*** Buscar ID

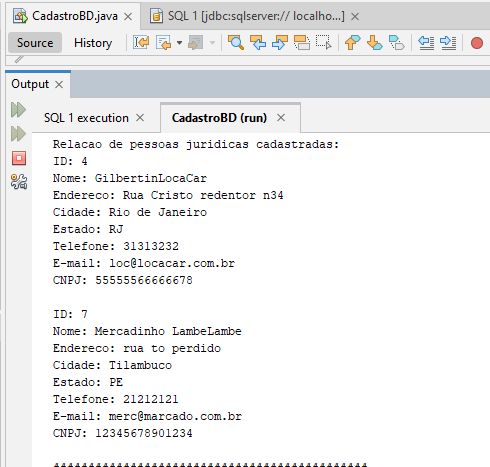


**Resultado:** Neste caso, foi mostrado em tela a busca pelo ID 7 e o resultado sao os dados que foi solicitado.

***4.5*** Listar Todos

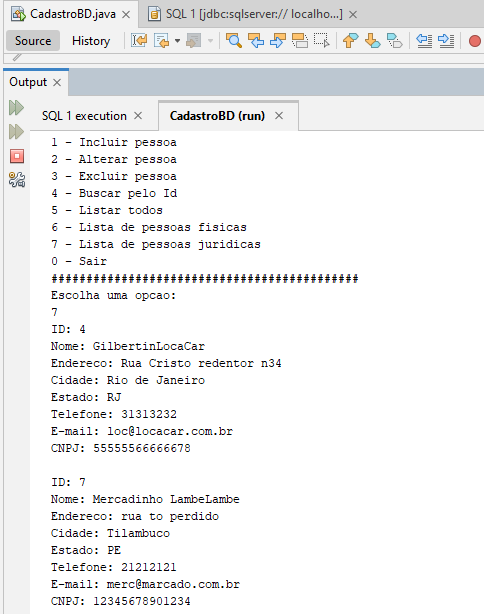


***4.6*** Listar Pessoas Fisicas



**Resultado:** Neste caso foi listados todas Pessoas Fisicas cadastradas

***4.7*** Listar Pessoas Juridicas



**Resultado:** Neste caso foi listados todas Pessoas Juridicas cadastrada

***5°*** *Análise e Conclusão:*

1. Quais as diferenças entre a persistência em arquivo e a persistência em

**A°** Quais as diferenças entre a persistência em arquivo e a persistência em bancos de dados?

**R**: A decisão de como persistir os dados de uma aplicação, seja em arquivos ou em um banco de dados, é crucial e pode influenciar significativamente o desempenho, a escalabilidade e a manutenibilidade do sistema.

persistência em arquivo:

**Conceito:** Nessa abordagem, os dados são armazenados em arquivos no sistema de arquivos, podendo ser em formatos simples como texto ou em formatos mais complexos como JSON ou XML.

persistência em banco de dados:

**Conceito:** Os dados são armazenados em um sistema gerenciador de banco de dados (SGBD), que oferece ferramentas para organizar, armazenar e recuperar dados de forma eficiente.

**B°** Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

**R**: Os operadores lambda, introduzidos no Java 8, trouxeram uma nova forma de escrever código mais conciso e expressivo, especialmente quando se trata de lidar com coleções e expressões funcionais. No contexto da impressão de valores contidos em entidades, os lambdas oferecem uma sintaxe mais elegante e simplificada em comparação com as abordagens tradicionais.

**C°** Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static?

**R:**  **Pertence à classe, não ao objeto:** Um método estático está associado à classe em si, e não a uma instância específica daquela classe. Isso significa que ele pode ser chamado diretamente pelo nome da classe, sem a necessidade de criar um objeto.

 **Acesso a membros estáticos:** Métodos estáticos só podem acessar outros membros estáticos da classe, como variáveis estáticas e outros métodos estáticos. Eles não podem acessar diretamente membros não estáticos (variáveis de instância e métodos de instância) porque estes estão associados a objetos específicos.

**Ponto de entrada:** O método main é o ponto de partida da execução de um programa Java. É chamado diretamente pela máquina virtual Java (JVM) quando um programa é executado.

**Método estático:** O main é sempre declarado como static por dois motivos principais:

1. **Acessibilidade:** A JVM precisa chamar o main sem criar uma instância da classe, já que ainda não existe nenhum objeto no início da execução.
2. **Convenção:** É uma convenção estabelecida em Java que o método main seja sempre estático.