RPG0016 - BackEnd sem banco não tem

Paulo Wuéliton Horacio Fernandes (202302025919)

507 Polo Tucuruvi – São Paulo/SP
Nível 3: Back-end Sem Banco Não Tem – 2024.1 FLEX - 3º Semestre Letivo

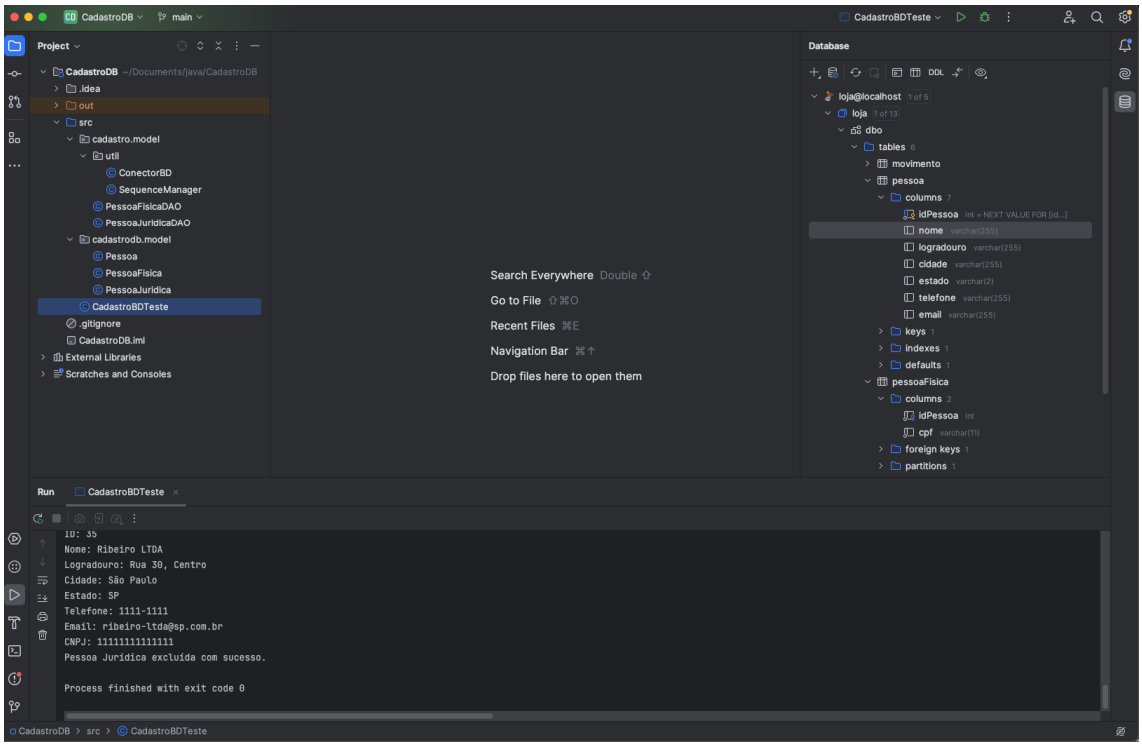
Objetivo da Prática

1. Implementar persistência com base no middleware JDBC.
2. Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
3. Implementar o mapeamento objeto-relacional em sistemas Java.
4. Criar sistemas cadastrais com persistência em banco relacional.
5. No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

GIT: https://github.com/wueliton/rpg0016-cadastrodb

Conexão com o banco de dados JDBC SQL Server:



cadastrodb.model

Pessoa

```
package cadastrodb.model;

public class Pessoa {
    private Integer id;
    private String nome;
    private String logradouro;
    private String cidade;
    private String estado;
    private String telefone;
    private String email;

    public Pessoa() {}

    public Pessoa(Integer id, String nome, String logradouro, String
cidade, String estado, String telefone, String email) {
        this.id = id;
        this.nome = nome;
        this.logradouro = logradouro;
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }

    public Integer getId() {
        return id;
    }
```

```java
    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

    public String getLogradouro() {
        return logradouro;
    }

    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

    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;
    }

    public void exibir() {
        System.out.println("ID: " + id);
        System.out.println("Nome: " + nome);
        System.out.println("Logradouro: " + logradouro);
        System.out.println("Cidade: " + cidade);
        System.out.println("Estado: " + estado);
        System.out.println("Telefone: " + telefone);
        System.out.println("Email: " + email);
    }
}
```

PessoaFisica

```
package cadastrodb.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    public PessoaFisica() {}

    public PessoaFisica(Integer 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);
    }
}
```

PessoaJuridica

```
package cadastrodb.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {}

    public PessoaJuridica(Integer id, String nome, String
logradouro, String cidade, String estado, String telefone, String
email, String cnpj) {
        super(id, nome, logradouro, 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);
    }
}
```

cadastro.model.util

ConectorBD

```java
package cadastro.model.util;

import java.sql.*;

public class ConectorBD {
    private Connection connection = null;

    public Connection getConnection() {
        if(connection != null) return connection;

        try {
            String url =
"jdbc:sqlserver://localhost;databaseName=loja;encrypt=true;trustServ
erCertificate=true";
            String usuario = "loja";
            String senha = "loja";

            connection = DriverManager.getConnection(url, usuario,
senha);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return connection;
    }

    public PreparedStatement getPrepared(String sql) throws
SQLException {
        Connection conn = getConnection();
        return conn.prepareStatement(sql);
    }

    public ResultSet getSelect(String sql) throws SQLException {
        PreparedStatement statement = getPrepared(sql);
        return statement.executeQuery();
    }

    public void close(Statement statement) {
        try {
            statement.close();
        }  catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void close(ResultSet resultSet) {
        try {
            resultSet.close();
        }  catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void close(Connection connection) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

SequenceManager

```java
package cadastro.model.util;

import java.sql.ResultSet;
import java.sql.SQLException;

public class SequenceManager {
    ConectorBD conectorBD;

    public SequenceManager(ConectorBD conectorBD) {
        this.conectorBD = conectorBD;
    }

    public int getValue(String nomeSequencia) throws SQLException {
        String sql = "SELECT NEXT VALUE FOR " + nomeSequencia;
        try(ResultSet result = conectorBD.getSelect(sql)) {
            if(result.next()) {
                return result.getInt(1);
            }
        }
        return -1;
    }
}
```

cadastro.model

PessoaFisicaDAO

```
package cadastro.model;

import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import cadastrodb.model.PessoaFisica;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
    private ConectorBD conectorBD;
    private SequenceManager sequenceManager;

    public PessoaFisicaDAO(ConectorBD conectorBD, SequenceManager
sequenceManager) {
        this.conectorBD = conectorBD;
        this.sequenceManager = sequenceManager;
    }

    public PessoaFisica getPessoa(int id) {
        PessoaFisica pessoaFisica = null;
        String sql = "SELECT * FROM pessoa AS p INNER JOIN
pessoaFisica AS pf ON p.idPessoa = pf.idPessoa WHERE p.idPessoa =
?";
        try {
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setInt(1, id);
            ResultSet resultSet = preparedStatement.executeQuery();
            if(resultSet.next()) {
                pessoaFisica = converterPessoa(resultSet);
            }
            conectorBD.close(preparedStatement);
            conectorBD.close(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return pessoaFisica;
    }

public List<PessoaFisica> getPessoas() {
    List<PessoaFisica> pessoas = new ArrayList<PessoaFisica>();
    String sql = "SELECT * FROM pessoa AS p INNER JOIN pessoaFisica
AS pf ON p.idPessoa = pf.idPessoa";
    try(ResultSet resultSet = conectorBD.getSelect(sql)) {
        while(resultSet.next()) {
            pessoas.add(converterPessoa(resultSet));
        }
        conectorBD.close(resultSet);
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
        return pessoas;
    }

    public PessoaFisica inserir(PessoaFisica pessoa) {
        try {
            int idPessoa = sequenceManager.getValue("idPessoa");
            pessoa.setId(idPessoa);
            String sql = "INSERT INTO pessoa (idPessoa, nome,
logradouro, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?,
?, ?)";
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setInt(1, idPessoa);
            preparedStatement.setString(2, pessoa.getNome());
            preparedStatement.setString(3, pessoa.getLogradouro());
            preparedStatement.setString(4, pessoa.getCidade());
            preparedStatement.setString(5, pessoa.getEstado());
            preparedStatement.setString(6, pessoa.getTelefone());
            preparedStatement.setString(7, pessoa.getEmail());
            preparedStatement.executeUpdate();
            conectorBD.close(preparedStatement);

            String sqlPF = "INSERT INTO pessoaFisica (idPessoa, cpf)
VALUES (?, ?)";
            PreparedStatement preparedStatementPF =
conectorBD.getPrepared(sqlPF);
            preparedStatementPF.setInt(1, idPessoa);
            preparedStatementPF.setString(2, pessoa.getCpf());
            preparedStatementPF.executeUpdate();
            conectorBD.close(preparedStatementPF);

            System.out.println("Pessoa Física salva com sucesso.");
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return pessoa;
    }

    public void alterar(PessoaFisica pessoa) {
        try {
            String sql = "UPDATE pessoa SET nome = ?, logradouro =
?, cidade = ?, estado = ?, telefone = ?, email = ? WHERE idPessoa =
?";
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setString(1, pessoa.getNome());
            preparedStatement.setString(2, pessoa.getLogradouro());
            preparedStatement.setString(3, pessoa.getCidade());
            preparedStatement.setString(4, pessoa.getEstado());
            preparedStatement.setString(5, pessoa.getTelefone());
            preparedStatement.setString(6, pessoa.getEmail());
            preparedStatement.setInt(7, pessoa.getId());
            preparedStatement.executeUpdate();
            conectorBD.close(preparedStatement);

            String sqlPF = "UPDATE pessoaFisica SET cpf = ? WHERE
idPessoa = ?";
```

```
            PreparedStatement preparedStatementPF =
conectorBD.getPrepared(sqlPF);
            preparedStatementPF.setString(1, pessoa.getCpf());
            preparedStatementPF.setInt(2, pessoa.getId());
            preparedStatementPF.executeUpdate();
            conectorBD.close(preparedStatementPF);

            System.out.println("Pessoa Física atualizada com
sucesso.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void excluir(int id) {
        try {
            String sqlPF = "DELETE FROM pessoaFisica WHERE idPessoa
= ?";
            PreparedStatement preparedStatementPF =
conectorBD.getPrepared(sqlPF);
            preparedStatementPF.setInt(1, id);
            preparedStatementPF.executeUpdate();

            String sql = "DELETE FROM pessoa WHERE idPessoa = ?";
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setInt(1, id);
            preparedStatement.executeUpdate();
            conectorBD.close(preparedStatement);

            System.out.println("Pessoa Física excluída com
sucesso.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private PessoaFisica converterPessoa(ResultSet resultSet) throws
SQLException {
        PessoaFisica pessoa = new PessoaFisica();
        pessoa.setId(resultSet.getInt("idPessoa"));
        pessoa.setNome(resultSet.getString("nome"));
        pessoa.setLogradouro(resultSet.getString("logradouro"));
        pessoa.setCidade(resultSet.getString("cidade"));
        pessoa.setEstado(resultSet.getString("estado"));
        pessoa.setTelefone(resultSet.getString("telefone"));
        pessoa.setEmail(resultSet.getString("email"));
        pessoa.setCpf(resultSet.getString("cpf"));
        return pessoa;
    }
}
```

PessoaJuridicaDAO

```
package cadastro.model;

import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import cadastrodb.model.PessoaJuridica;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaDAO {
    private ConectorBD conectorBD;
    private SequenceManager sequenceManager;

    public PessoaJuridicaDAO(ConectorBD conectorBD, SequenceManager
sequenceManager) {
        this.conectorBD = conectorBD;
        this.sequenceManager = sequenceManager;
    }

    public PessoaJuridica getPessoa(int id) {
        PessoaJuridica pessoaJuridica = null;
        String sql = "SELECT * FROM pessoa AS p INNER JOIN
pessoaJuridica AS pj ON p.idPessoa = pj.idPessoa WHERE p.idPessoa =
?";
        try {
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setInt(1, id);
            ResultSet resultSet = preparedStatement.executeQuery();
            if(resultSet.next()) {
                pessoaJuridica = converterPessoa(resultSet);
            }
            conectorBD.close(preparedStatement);
            conectorBD.close(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return pessoaJuridica;
    }

    public List<PessoaJuridica> getPessoas() {
        List<PessoaJuridica> pessoas = new
ArrayList<PessoaJuridica>();
        String sql = "SELECT * FROM pessoa AS p INNER JOIN
pessoaJuridica AS pj ON p.idPessoa = pj.idPessoa";
        try(ResultSet resultSet = conectorBD.getSelect(sql)) {
            while(resultSet.next()) {
                pessoas.add(converterPessoa(resultSet));
            }
            conectorBD.close(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return pessoas;
    }

    public PessoaJuridica inserir(PessoaJuridica pessoa) {
        try {
            int idPessoa = sequenceManager.getValue("idPessoa");
            pessoa.setId(idPessoa);
            String sql = "INSERT INTO pessoa (idPessoa, nome,
logradouro, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?,
?, ?)";
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setInt(1, idPessoa);
            preparedStatement.setString(2, pessoa.getNome());
            preparedStatement.setString(3, pessoa.getLogradouro());
            preparedStatement.setString(4, pessoa.getCidade());
            preparedStatement.setString(5, pessoa.getEstado());
            preparedStatement.setString(6, pessoa.getTelefone());
            preparedStatement.setString(7, pessoa.getEmail());
            preparedStatement.executeUpdate();
            conectorBD.close(preparedStatement);

            String sqlPJ = "INSERT INTO pessoaJuridica (idPessoa,
cnpj) VALUES (?, ?)";
            PreparedStatement preparedStatementPJ =
conectorBD.getPrepared(sqlPJ);
            preparedStatementPJ.setInt(1, idPessoa);
            preparedStatementPJ.setString(2, pessoa.getCnpj());
            preparedStatementPJ.executeUpdate();
            conectorBD.close(preparedStatementPJ);

            System.out.println("Pessoa Jurídica salva com sucesso");
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return pessoa;
    }

    public void alterar(PessoaJuridica pessoa) {
        try {
            String sql = "UPDATE pessoa SET nome = ?, logradouro =
?, cidade = ?, estado = ?, telefone = ?, email = ? WHERE idPessoa =
?";
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setString(1, pessoa.getNome());
            preparedStatement.setString(2, pessoa.getLogradouro());
            preparedStatement.setString(3, pessoa.getCidade());
            preparedStatement.setString(4, pessoa.getEstado());
            preparedStatement.setString(5, pessoa.getTelefone());
            preparedStatement.setString(6, pessoa.getEmail());
            preparedStatement.setInt(7, pessoa.getId());
            preparedStatement.executeUpdate();
            conectorBD.close(preparedStatement);
```

```
            String sqlPJ = "UPDATE pessoaJuridica SET cnpj = ? WHERE
idPessoa = ?";
            PreparedStatement preparedStatementPJ =
conectorBD.getPrepared(sqlPJ);
            preparedStatementPJ.setString(1, pessoa.getCnpj());
            preparedStatementPJ.setInt(2, pessoa.getId());
            preparedStatementPJ.executeUpdate();
            conectorBD.close(preparedStatementPJ);

            System.out.println("Pessoa Jurídica atualizada com
sucesso.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void excluir(int id) {
        try {
            String sqlPf = "DELETE FROM pessoaJuridica WHERE
idPessoa = ?";
            PreparedStatement preparedStatementPJ =
conectorBD.getPrepared(sqlPf);
            preparedStatementPJ.setInt(1, id);
            preparedStatementPJ.executeUpdate();

            String sql = "DELETE FROM pessoa WHERE idPessoa = ?";
            PreparedStatement preparedStatement =
conectorBD.getPrepared(sql);
            preparedStatement.setInt(1, id);
            preparedStatement.executeUpdate();
            conectorBD.close(preparedStatement);

            System.out.println("Pessoa Jurídica excluída com
sucesso.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private PessoaJuridica converterPessoa(ResultSet resultSet)
throws SQLException {
        PessoaJuridica pessoa = new PessoaJuridica();
        pessoa.setId(resultSet.getInt("idPessoa"));
        pessoa.setNome(resultSet.getString("nome"));
        pessoa.setLogradouro(resultSet.getString("logradouro"));
        pessoa.setCidade(resultSet.getString("cidade"));
        pessoa.setEstado(resultSet.getString("estado"));
        pessoa.setTelefone(resultSet.getString("telefone"));
        pessoa.setEmail(resultSet.getString("email"));
        pessoa.setCnpj(resultSet.getString("cnpj"));
        return pessoa;
    }
}
```

CadastroDBTeste

```
import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import cadastrodb.model.PessoaFisica;
import cadastrodb.model.PessoaJuridica;

import java.sql.SQLException;
import java.util.List;

public class CadastroDBTeste {
    public static void main(String[] args) {
        ConectorBD conectorBD = new ConectorBD();
        SequenceManager sequenceManager = new
SequenceManager(conectorBD);
        PessoaFisicaDAO pessoaFisicaDAO = new
PessoaFisicaDAO(conectorBD, sequenceManager);
        PessoaJuridicaDAO pessoaJuridicaDAO = new
PessoaJuridicaDAO(conectorBD, sequenceManager);

        // Instanciar uma pessoa física e persistir no banco de
dados.
        PessoaFisica pessoaFisica = new PessoaFisica(0, "Ribeiro",
"Rua 30, Centro", "São Paulo", "SP", "1111-1111",
"ribeiro@sp.com.br", "11111111111");
        pessoaFisica = pessoaFisicaDAO.inserir(pessoaFisica);

        // Alterar os dados da pessoa física no banco.
        pessoaFisica.setNome("Ribeiro Alberto");
        pessoaFisica.setCidade("Guarulhos");
        pessoaFisicaDAO.alterar(pessoaFisica);

        // Consultar todas as pessoas físicas do banco de dados e
listar no console.
        List<PessoaFisica> listaPF = pessoaFisicaDAO.getPessoas();
        listaPF.forEach(PessoaFisica::exibir);

        // Excluir a pessoa física criada anteriormente no banco.
        pessoaFisicaDAO.excluir(pessoaFisica.getId());

        // Instanciar uma pessoa jurídica e persistir no banco de
dados.
        PessoaJuridica pessoaJuridica = new PessoaJuridica(0,
"Ribeiro LTDA", "Rua 30, Centro", "São Paulo", "SP", "1111-1111",
"ribeiro-ltda@sp.com.br", "11111111111111");
        pessoaJuridica = pessoaJuridicaDAO.inserir(pessoaJuridica);

        // Alterar os dados da pessoa jurídica no banco.
        pessoaJuridica.setNome("Ribeiro Soares LTDA");
        pessoaJuridica.setLogradouro("Rua 31, Centro");
        pessoaFisicaDAO.alterar(pessoaFisica);

        // Consultar todas as pessoas jurídicas do banco e listar no
console.
        List<PessoaJuridica> listaPJ =
pessoaJuridicaDAO.getPessoas();
        listaPJ.forEach(PessoaJuridica::exibir);
```

```
        // Excluir a pessoa jurídica criada anteriormente no banco.
        pessoaJuridicaDAO.excluir(pessoaJuridica.getId());

        conectorBD.close(conectorBD.getConnection());
    }
}
```

Análise e Conclusão

a) **Qual a importância dos componentes de middleware, como o JDBC?**
Os componentes de middleware são responsáveis por permitir a integração simplificada, de forma transparente. Os middlewares como o JDBC são responsáveis pela comunicação com uma fonte de dados, fornecendo uma interface padronizada, que muitas vezes requer pouca ou nenhuma alteração na implementação para alterar o fornecedor, além de abstrair a forma como as consultas são realizadas, criando um padrão que pode ser utilizado independente do banco de dados utilizado, desde que esteja disponível um driver para o banco utilizado.

b) **Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?**
O Statement executa uma instrução SQL estática, onde todos os dados que compõem a consulta devem ser informados, ele é indicado apenas para consultas sem parâmetros, pois não oferece um tratamento nos dados informados, tornando seu uso uma vulnerabilidade para SQL Injection. O PreparedStatement recebe seus parâmetros após a definição da instrução SQL definida, que previne que dados de tipos não esperados sejam informados ou instruções que exploram o SQL Injection, além de facilitar a escrita do comando SQL, pois não é necessário o uso de apóstrofe ou qualquer outro delimitador.

c) **Como o padrão DAO melhora a manutenibilidade do software?**
Ele tem como objetivo organizar e centralizar todos os comandos SQL que são utilizados pela aplicação, é indicado possuir uma classe DAO para cada classe de entidade relevante para o sistema. Utilizar o padrão DAO tem como principais benefícios a reutilização de comandos sem duplicação de código e a facilidade na manutenção do código, pois todas as consultas estão centralizadas em classes DAO.

d) **Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?**
Na modelagem de herança em um banco de dados relacional, a superclasse e a classe são representadas por duas tabelas distintas, relacionadas por meio de uma chave primária e uma chave estrangeira, estabelecendo um relacionamento 1 para 1. Durante a inserção de dados, a tabela que representa a classe depende de uma chave primária existente na tabela que representa a superclasse. Esse tipo de relacionamento simplifica a recuperação de dados e se assemelha muito à estrutura de herança encontrada em Java.

2º Procedimento | Alimentando a Base

GIT: https://github.com/wueliton/rpg0016-cadastrodb/tree/2-procedimento-alimentando-a-base

CadastroDBTeste2.java

```
import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import cadastrodb.model.PessoaFisica;
import cadastrodb.model.PessoaJuridica;

import java.util.List;
import java.util.Scanner;

public class CadastroDBTeste {
    public static void main(String[] args) {
        ConectorBD conectorBD = new ConectorBD();
        SequenceManager sequenceManager = new
SequenceManager(conectorBD);
        PessoaFisicaDAO pessoaFisicaDAO = new
PessoaFisicaDAO(conectorBD, sequenceManager);
        PessoaJuridicaDAO pessoaJuridicaDAO = new
PessoaJuridicaDAO(conectorBD, sequenceManager);

        Scanner scanner = new Scanner(System.in);

        while(true) {
            try {
                int opcaoMenu = mostrarMenu(scanner);

                scanner.nextLine();

                if (opcaoMenu == 0) {
                    break;
                }

                String tipoPessoa = tipoPessoa(scanner);

                switch (opcaoMenu) {
                    case 1:
                        inserir(pessoaFisicaDAO, pessoaJuridicaDAO,
scanner, tipoPessoa);
                        break;
                    case 2:
                        alterar(pessoaFisicaDAO, pessoaJuridicaDAO,
scanner, tipoPessoa);
                        break;
                    case 3:
                        excluirPorId(pessoaFisicaDAO,
pessoaJuridicaDAO, scanner, tipoPessoa);
                        break;
                    case 4:
                        buscarPorId(pessoaFisicaDAO,
```

```
pessoaJuridicaDAO, scanner, tipoPessoa);
                        break;
                    case 5:
                        exibirTodos(pessoaFisicaDAO,
pessoaJuridicaDAO, tipoPessoa);
                        break;
                }
            } catch (Exception err) {
                System.out.println("Opção inválida, tente
novamente.");
            }
        }
    }

    public static int mostrarMenu(Scanner scanner) {
        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 - Exibir Todos");
        System.out.println("0 - Finalizar Programa");
        System.out.println("================================");

        return scanner.nextInt();
    }

    public static String tipoPessoa(Scanner scanner) throws
Exception {
        System.out.println("F - Pessoa Física | J - Pessoa
Jurídica");
        String tipoPessoa = scanner.nextLine();
        if(tipoPessoa.equals("F") || tipoPessoa.equals("J")) return
tipoPessoa;
        throw new Exception("Tipo de Pessoa inválida");
    }

    public static void inserir(PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner, String
tipoPessoa) {
        exibirTitulo("Insira os dados da Pessoa " +
tipoPessoaString(tipoPessoa));

        System.out.println("Digite o nome:");
        String nome = scanner.nextLine();

        System.out.println("Digite o logradouro:");
        String logradouro = scanner.nextLine();

        System.out.println("Digite a cidade:");
        String cidade = scanner.nextLine();

        System.out.println("Digite o estado (2 caracteres):");
        String estado = scanner.nextLine();

        System.out.println("Digite o telefone:");
        String telefone = scanner.nextLine();

        System.out.println("Digite o email:");
        String email = scanner.nextLine();
```

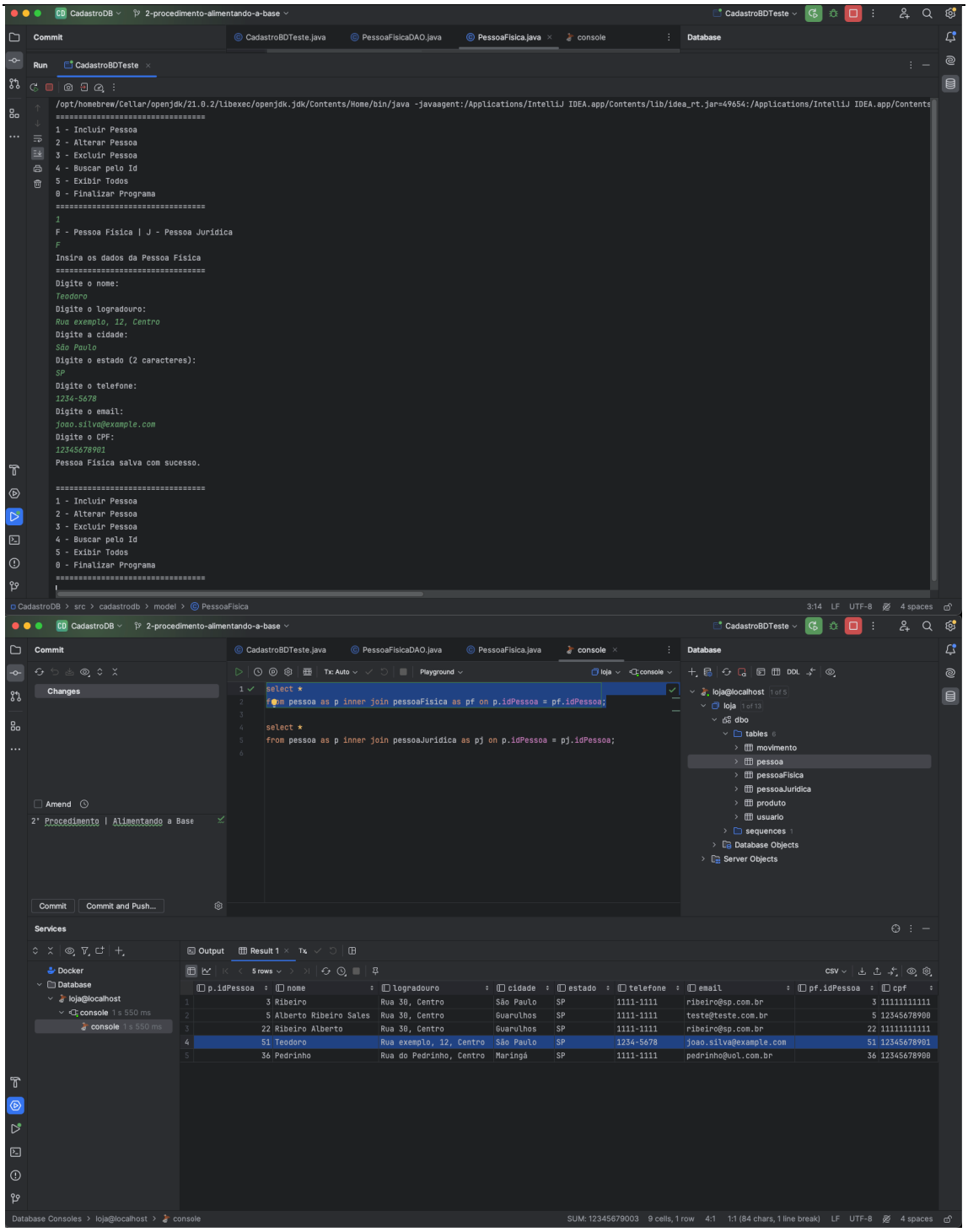
```
        if(tipoPessoa.equals("F")) {
            PessoaFisica pessoaFisica = criarPF(scanner, 0, nome,
logradouro, cidade, estado, telefone, email);
            pessoaFisicaDAO.inserir(pessoaFisica);
        } else {
            PessoaJuridica pessoaJuridica = criarPJ(scanner, 0,
nome, logradouro, cidade, estado, telefone, email);
            pessoaJuridicaDAO.inserir(pessoaJuridica);
        }
        System.out.println();
    }

    private static void alterar(PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner, String
tipoPessoa) {
        exibirTitulo("Insira os dados da " +
tipoPessoaString(tipoPessoa));

        System.out.println("Digite o ID:");
        int id = scanner.nextInt();

        scanner.nextLine();

        System.out.println("Digite o nome:");
        String nome = scanner.nextLine();

        System.out.println("Digite o logradouro:");
        String logradouro = scanner.nextLine();

        System.out.println("Digite a cidade:");
        String cidade = scanner.nextLine();

        System.out.println("Digite o estado (2 caracteres):");
        String estado = scanner.nextLine();

        System.out.println("Digite o telefone:");
        String telefone = scanner.nextLine();

        System.out.println("Digite o email:");
        String email = scanner.nextLine();

        if(tipoPessoa.equals("F")) {
            PessoaFisica pessoaFisica = criarPF(scanner, id, nome,
logradouro, cidade, estado, telefone, email);
            pessoaFisicaDAO.alterar(pessoaFisica);
        } else {
            PessoaJuridica pessoaJuridica = criarPJ(scanner, id,
nome, logradouro, cidade, estado, telefone, email);
            pessoaJuridicaDAO.alterar(pessoaJuridica);
        }
        System.out.println();
    }

    private static void excluirPorId(PessoaFisicaDAO
pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO, Scanner
scanner, String tipoPessoa) {
        System.out.println("Digite o ID:");
        int id = scanner.nextInt();
        scanner.nextLine();
```

```
        if(tipoPessoa.equals("F")) {
            pessoaFisicaDAO.excluir(id);
        } else {
            pessoaJuridicaDAO.excluir(id);
        }
        System.out.println();
    }

    private static void buscarPorId(PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO, Scanner scanner, String
tipoPessoa) {
        System.out.println("Digite o ID:");
        int id = scanner.nextInt();
        scanner.nextLine();

        exibirTitulo("Exibindo Pessoa " +
tipoPessoaString(tipoPessoa));

        if(tipoPessoa.equals("F")) {
            PessoaFisica pessoaFisica =
pessoaFisicaDAO.getPessoa(id);
            if(pessoaFisica == null) {
                System.out.println("Nenhuma Pessoa Física encontrada
com o id " + id);
            } else {
                pessoaFisica.exibir();
            }
        }
        else {
            PessoaJuridica pessoaJuridica =
pessoaJuridicaDAO.getPessoa(id);
            if(pessoaJuridica == null) {
                System.out.println("Nenhuma Pessoa Jurídica
encontrada com o id " + id);
            } else {
                pessoaJuridica.exibir();
            }
        }
        System.out.println();
    }

    private static void exibirTodos(PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO, String tipoPessoa) {
        exibirTitulo("Exibindo dados de Pessoas " +
tipoPessoaString(tipoPessoa) + "s");
        if(tipoPessoa.equals("F")) {
            List<PessoaFisica> pessoaFisicaList =
pessoaFisicaDAO.getPessoas();
            pessoaFisicaList.forEach(pessoaFisica -> {
                pessoaFisica.exibir();
                System.out.println();
            });
        } else {
            List<PessoaJuridica> pessoaJuridicaList =
pessoaJuridicaDAO.getPessoas();
            pessoaJuridicaList.forEach(pessoaJuridica -> {
                pessoaJuridica.exibir();
                System.out.println();
            });
```

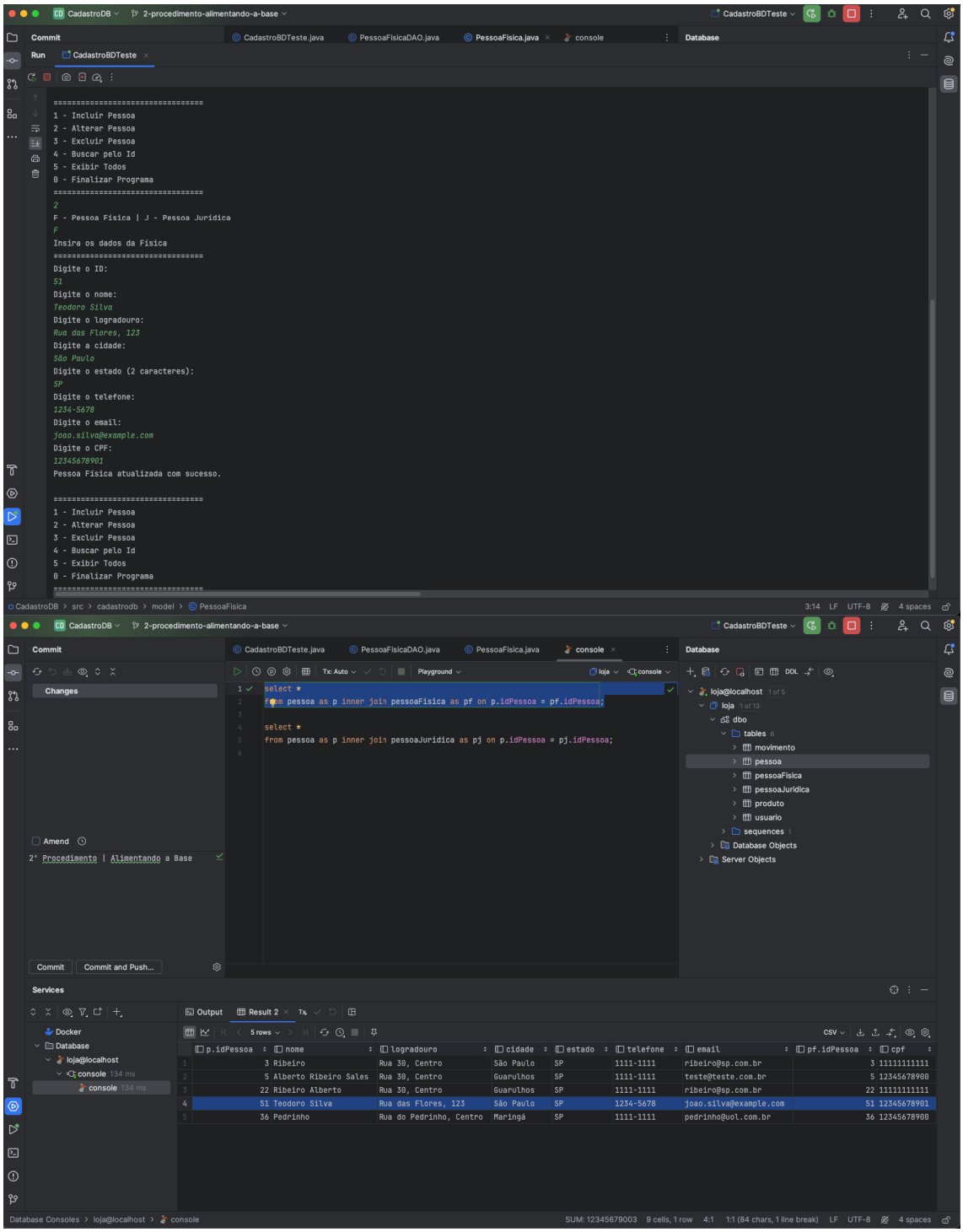
```
        }
    }

    private static PessoaFisica criarPF(Scanner scanner, int id,
String nome, String logradouro, String cidade, String estado, String
telefone, String email) {
        System.out.println("Digite o CPF:");
        String cpf = scanner.nextLine();

        return new PessoaFisica(id, nome, logradouro, cidade,
estado, telefone, email, cpf);
    }

    private static PessoaJuridica criarPJ(Scanner scanner, int id,
String nome, String logradouro, String cidade, String estado, String
telefone, String email) {
        System.out.println("Digite o CNPJ:");
        String cnpj = scanner.nextLine();

        return new PessoaJuridica(id, nome, logradouro, cidade,
estado, telefone,email, cnpj);
    }

    private static void exibirTitulo(String msg) {
        System.out.println(msg);
        System.out.println("==================================");
    }

    private static String tipoPessoaString(String tipoPessoa) {
        return tipoPessoa.equals("F") ? "Física" : "Jurídica";
    }
}
```

Teste das funcionalidades do sistema

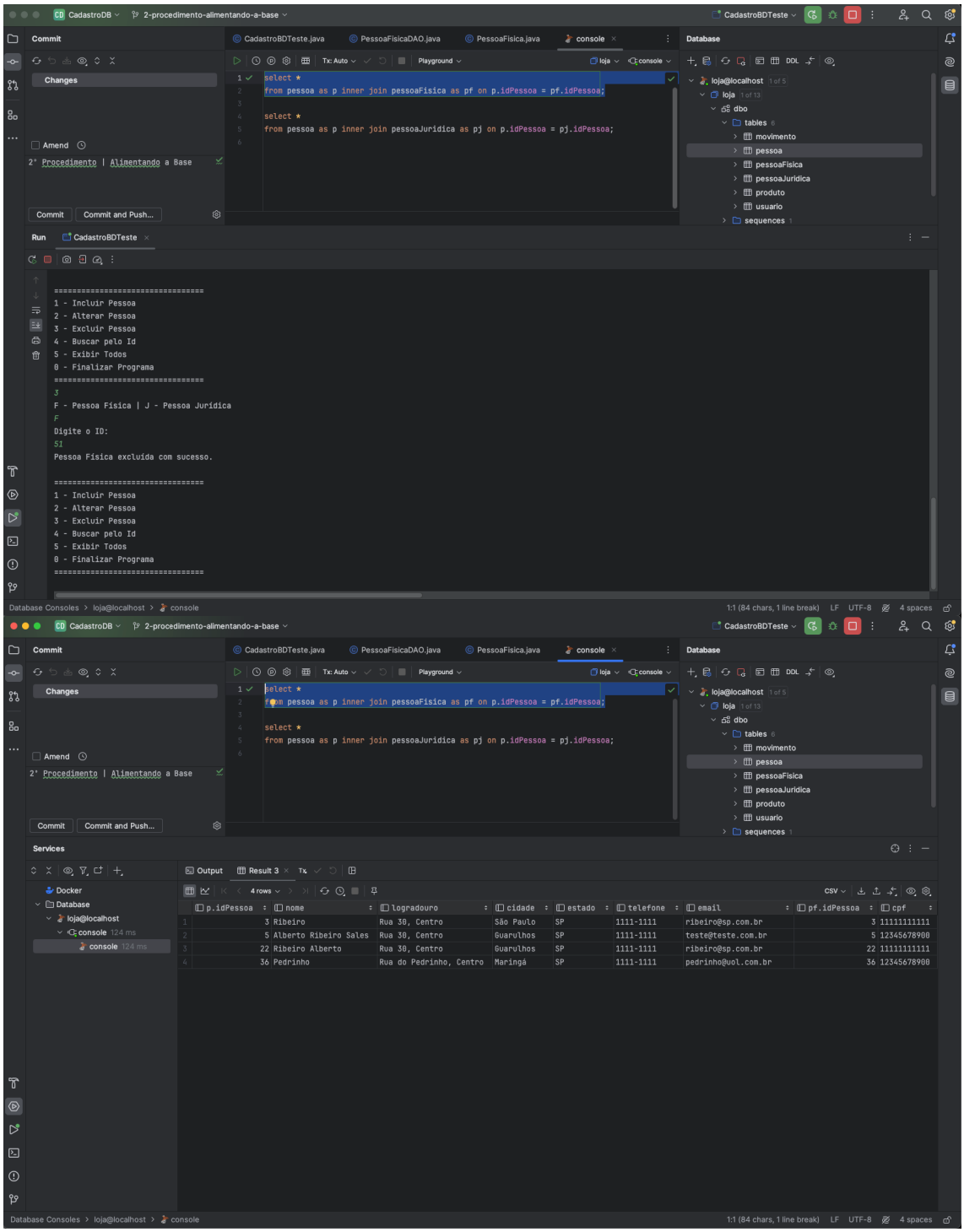
Inserir Pessoa Física



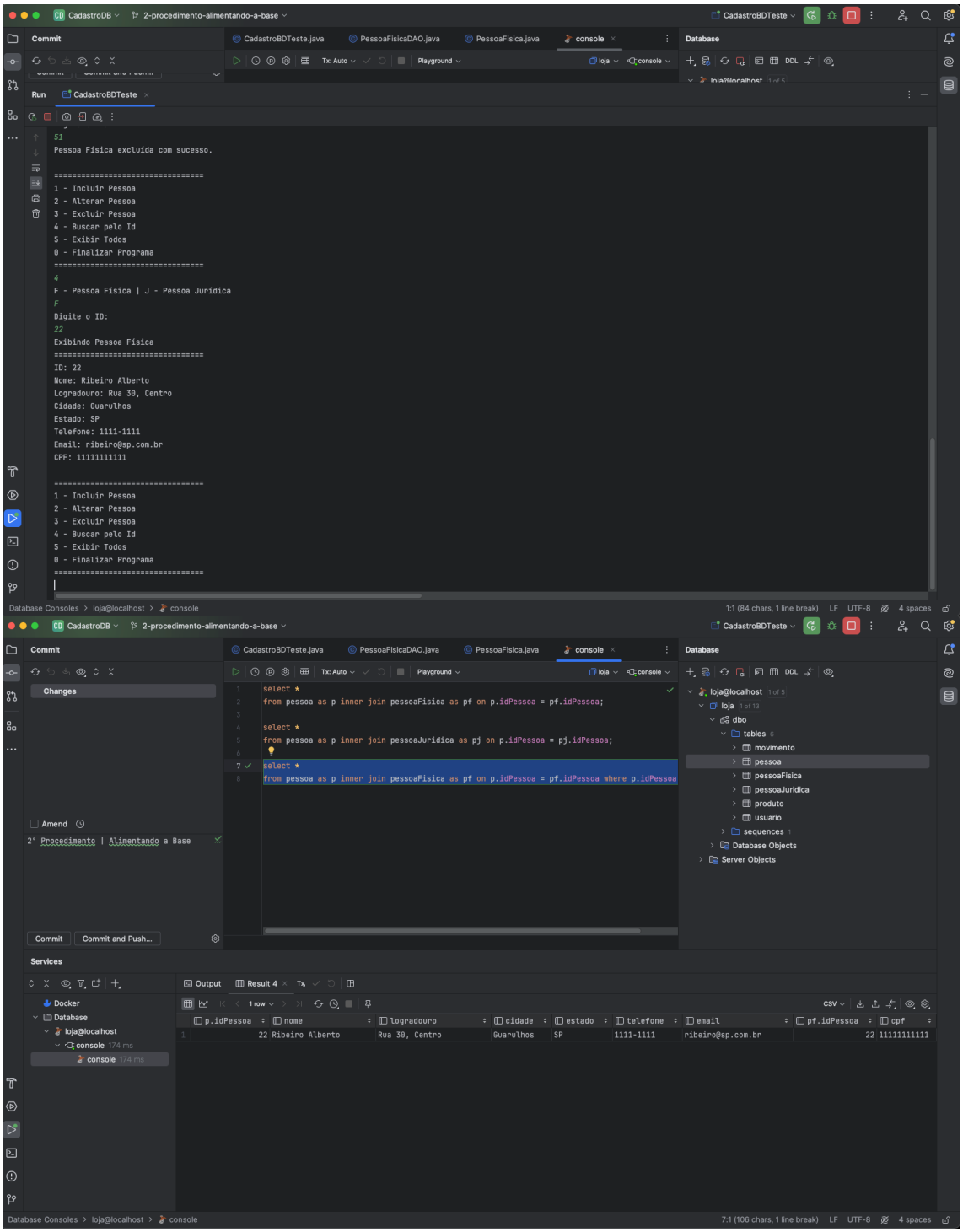
Alterar Pessoa Física



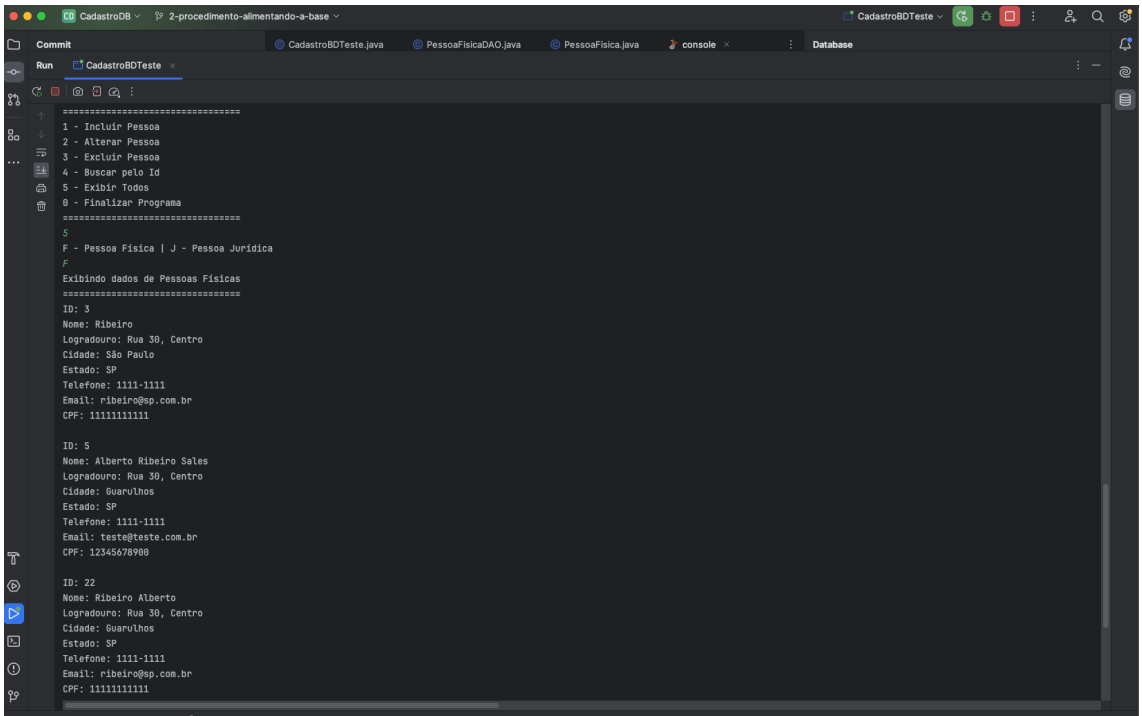
Excluir Pessoa Física

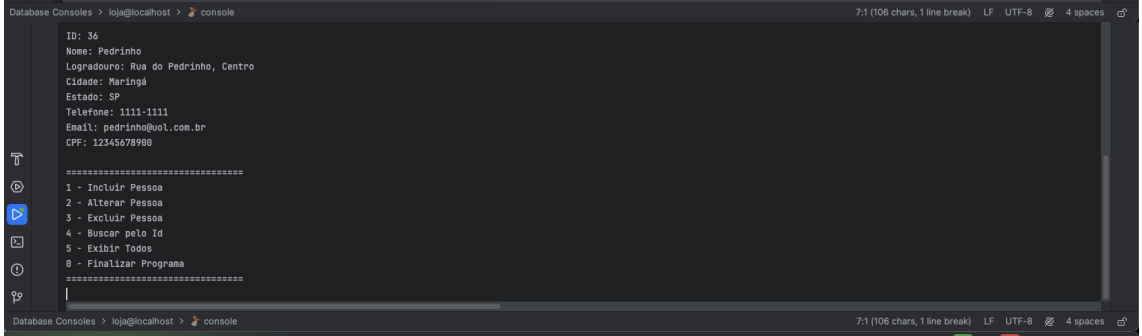


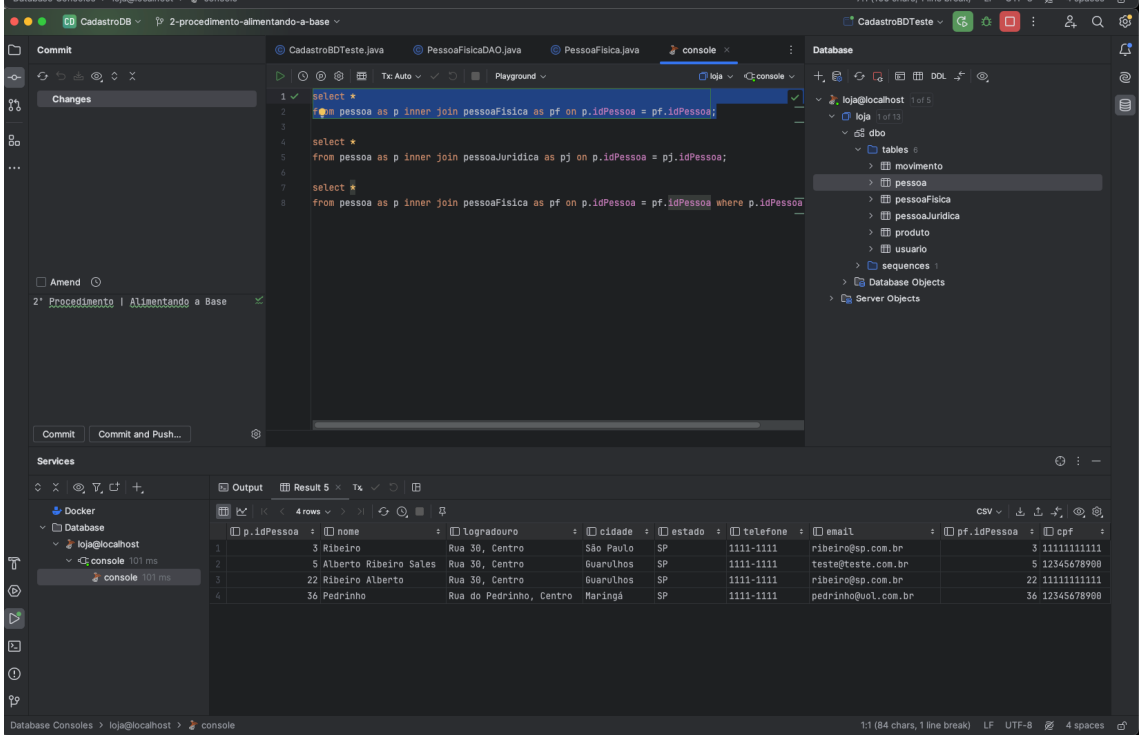
Buscar Pessoa Física por ID



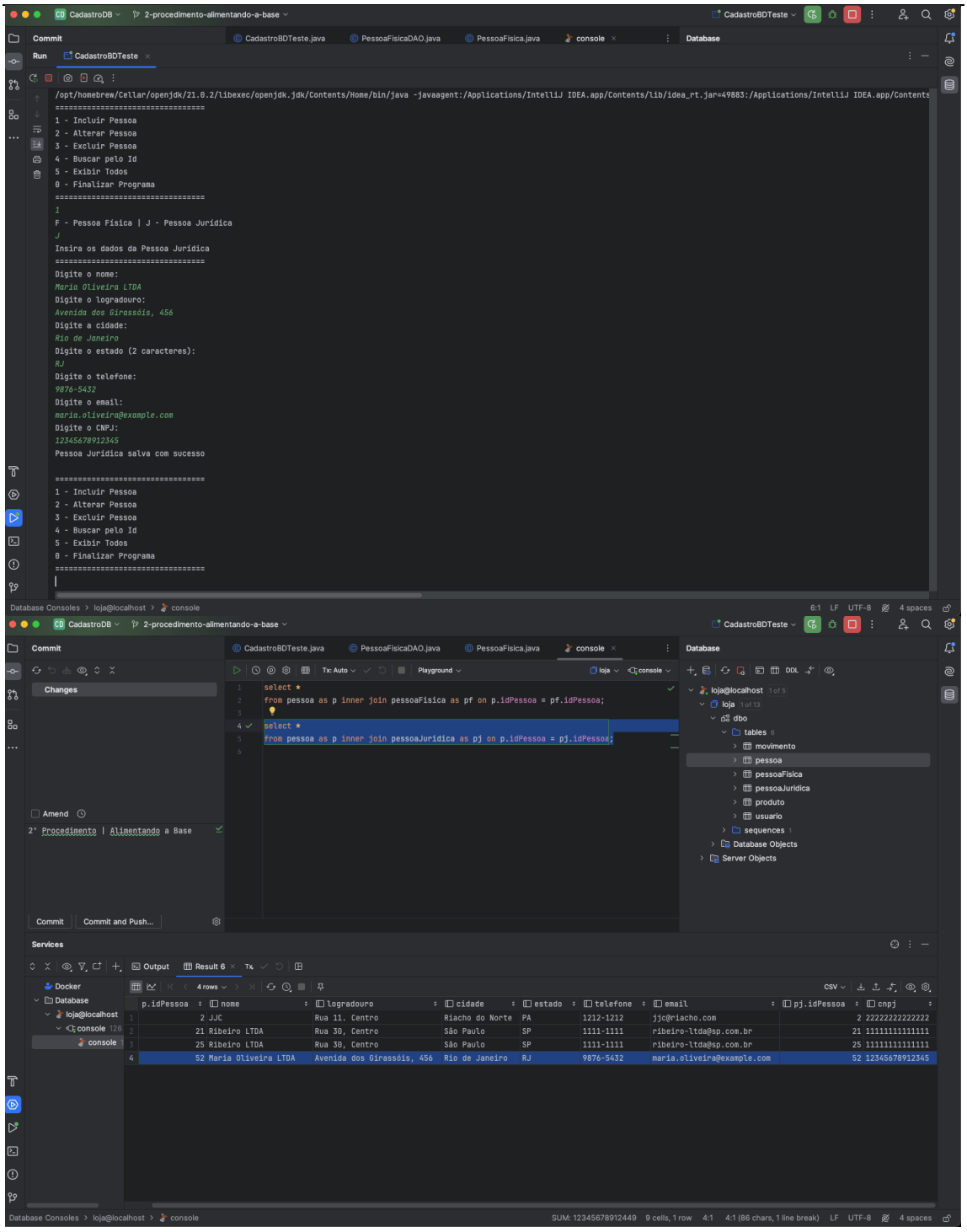
Exibir todas as Pessoas Físicas


CadastroDB
2-procedimento-alimentando-a-base
CadastroBDTeste
Commit
CadastroBDTeste.java
PessoaFisicaDAO.java
PessoaFisica.java
console
Database
Run
CadastroBDTeste
==========
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==========
5
F - Pessoa Física | J - Pessoa Jurídica
F
Exibindo dados de Pessoas Físicas
==========
ID: 3
Nome: Ribeiro
Logradouro: Rua 30, Centro
Cidade: São Paulo
Estado: SP
Telefone: 1111-1111
Email: ribeiro@sp.com.br
CPF: 11111111111
ID: 5
Nome: Alberto Ribeiro Sales
Logradouro: Rua 30, Centro
Cidade: Guarulhos
Estado: SP
Telefone: 1111-1111
Email: teste@teste.com.br
CPF: 12345678900
ID: 22
Nome: Ribeiro Alberto
Logradouro: Rua 30, Centro
Cidade: Guarulhos
Estado: SP
Telefone: 1111-1111
Email: ribeiro@sp.com.br
CPF: 11111111111
Database Consoles
loja@localhost
console
7:1 (106 chars, 1 line break)
LF
UTF-8
4 spaces

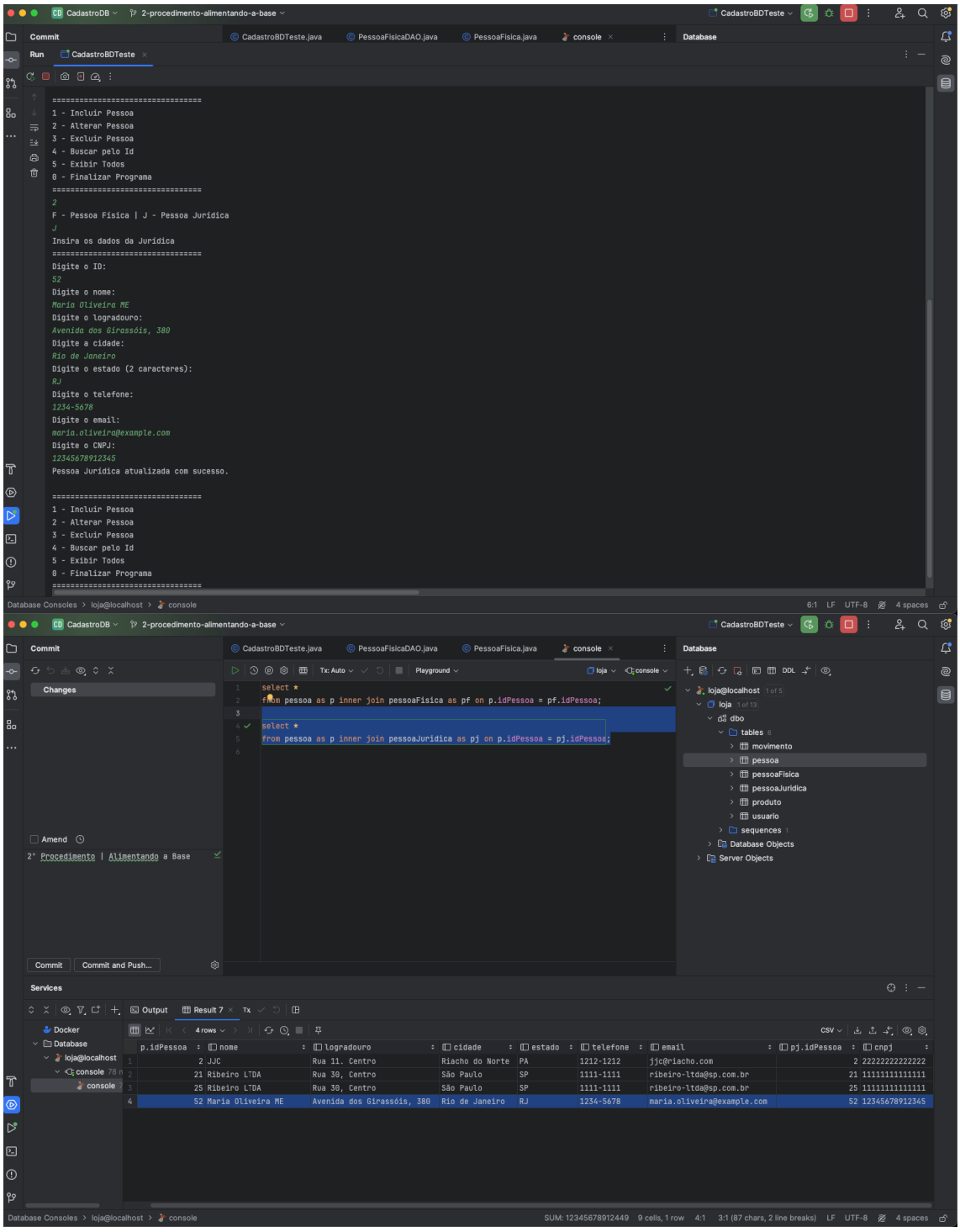

ID: 36
Nome: Pedrinho
Logradouro: Rua do Pedrinho, Centro
Cidade: Maringá
Estado: SP
Telefone: 1111-1111
Email: pedrinho@uol.com.br
CPF: 12345678900
==========
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==========
Database Consoles
loja@localhost
console
7:1 (106 chars, 1 line break)
LF
UTF-8
4 spaces


CadastroDB
2-procedimento-alimentando-a-base
CadastroBDTeste
Commit
Changes
Amend
2° Procedimento | Alimentando a Base
Commit
Commit and Push...
CadastroBDTeste.java
PessoaFisicaDAO.java
PessoaFisica.java
console
Tx: Auto
Playground
loja
console
1 select *
2 from pessoa as p inner join pessoaFisica as pf on p.idPessoa = pf.idPessoa;
3
4 select *
5 from pessoa as p inner join pessoaJuridica as pj on p.idPessoa = pj.idPessoa;
6
7 select *
8 from pessoa as p inner join pessoaFisica as pf on p.idPessoa = pf.idPessoa where p.idPessoa
Database
loja@localhost 1 of 5
loja 1 of 13
dbo
tables 6
movimento
pessoa
pessoaFisica
pessoaJuridica
produto
usuario
sequences 1
Database Objects
Server Objects
Services
Docker
Database
loja@localhost
console 101 ms
console 101 ms
Output
Result 5
4 rows
CSV
p.idPessoa | nome | logradouro | cidade | estado | telefone | email | pf.idPessoa | cpf
1 3 Ribeiro Rua 30, Centro São Paulo SP 1111-1111 ribeiro@sp.com.br 3 11111111111
2 5 Alberto Ribeiro Sales Rua 30, Centro Guarulhos SP 1111-1111 teste@teste.com.br 5 12345678900
3 22 Ribeiro Alberto Rua 30, Centro Guarulhos SP 1111-1111 ribeiro@sp.com.br 22 11111111111
4 36 Pedrinho Rua do Pedrinho, Centro Maringá SP 1111-1111 pedrinho@uol.com.br 36 12345678900
Database Consoles
loja@localhost
console
1:1 (84 chars, 1 line break)
LF
UTF-8
4 spaces

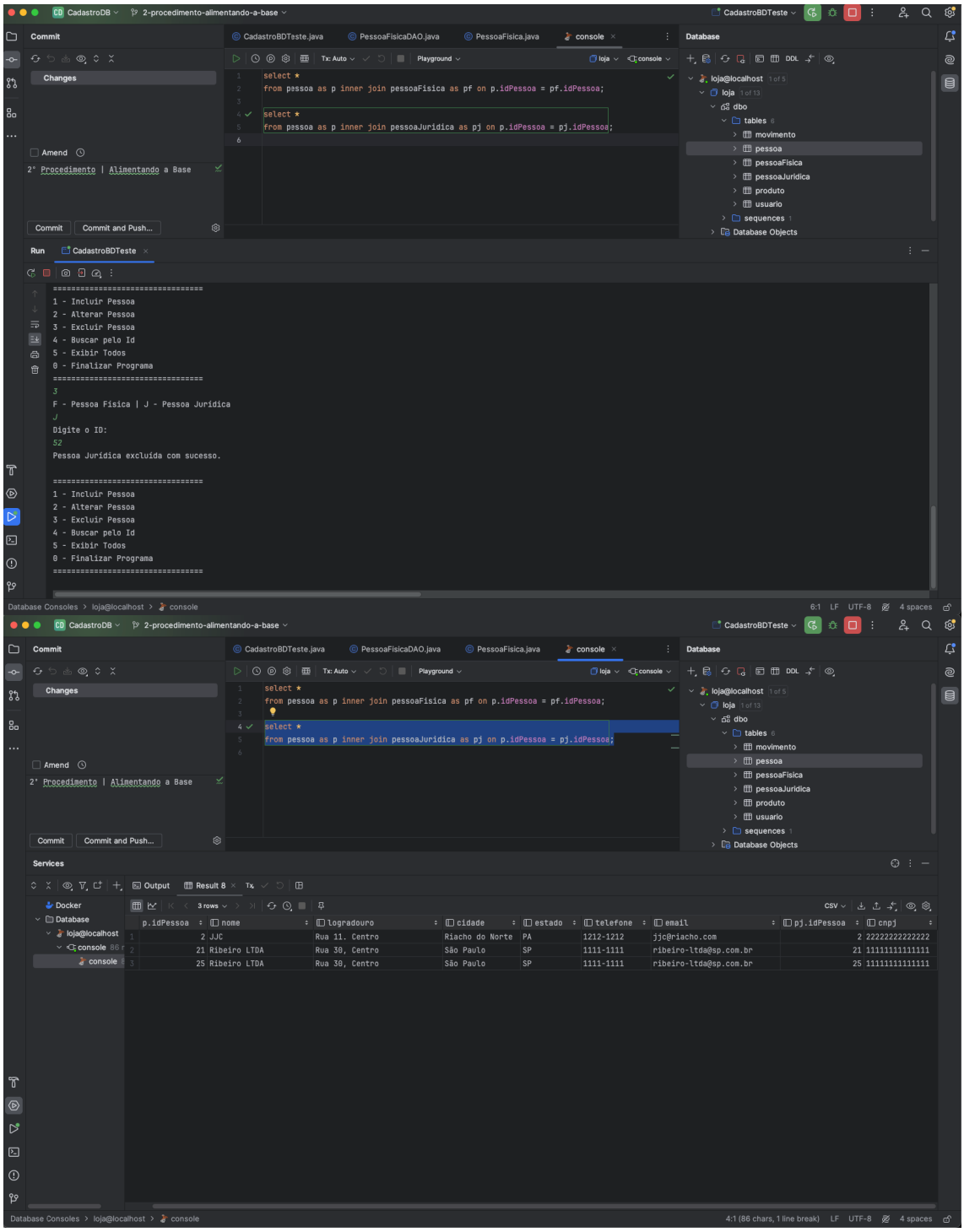
Inserir Pessoa Jurídica



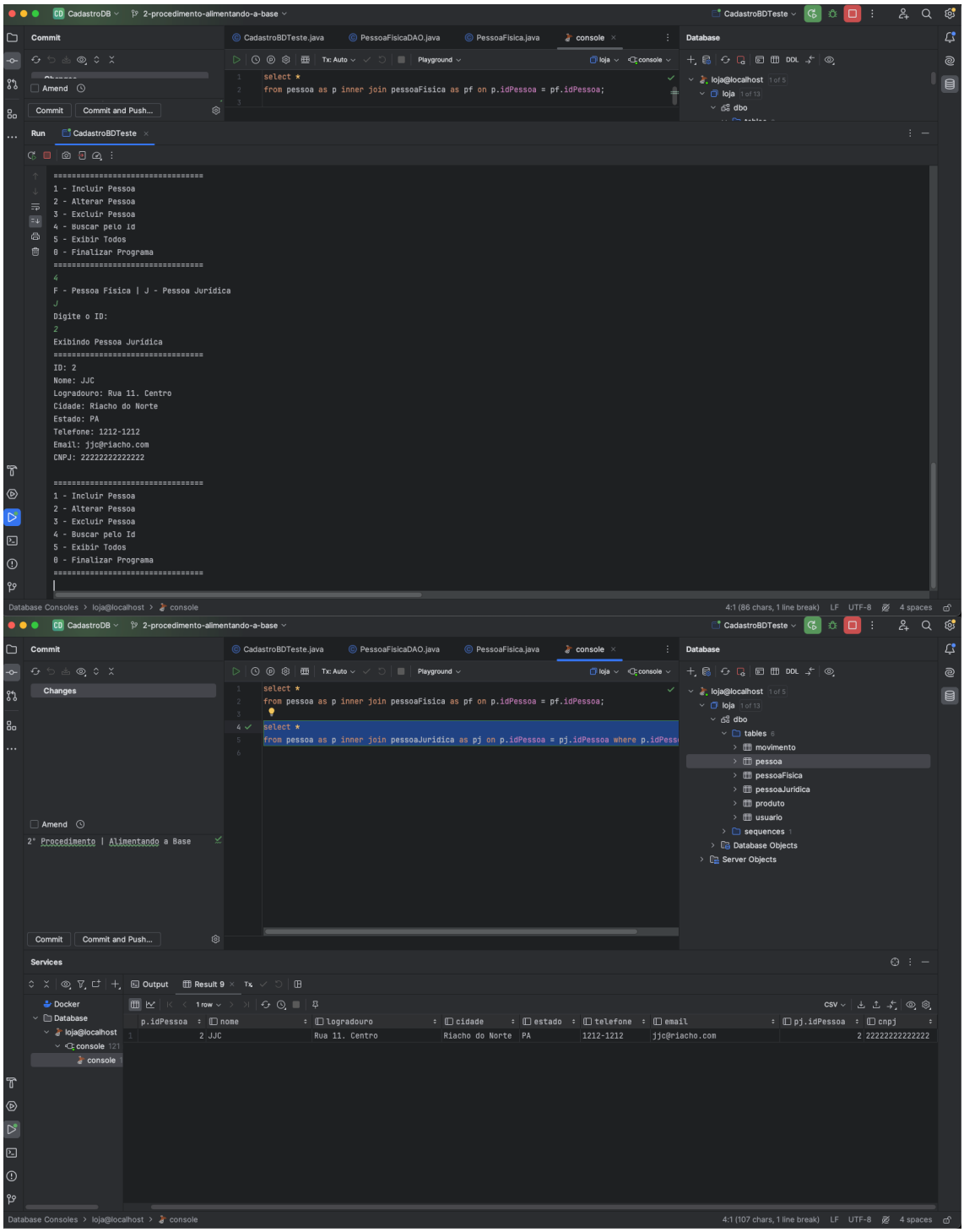
Alterar Pessoa Jurídica



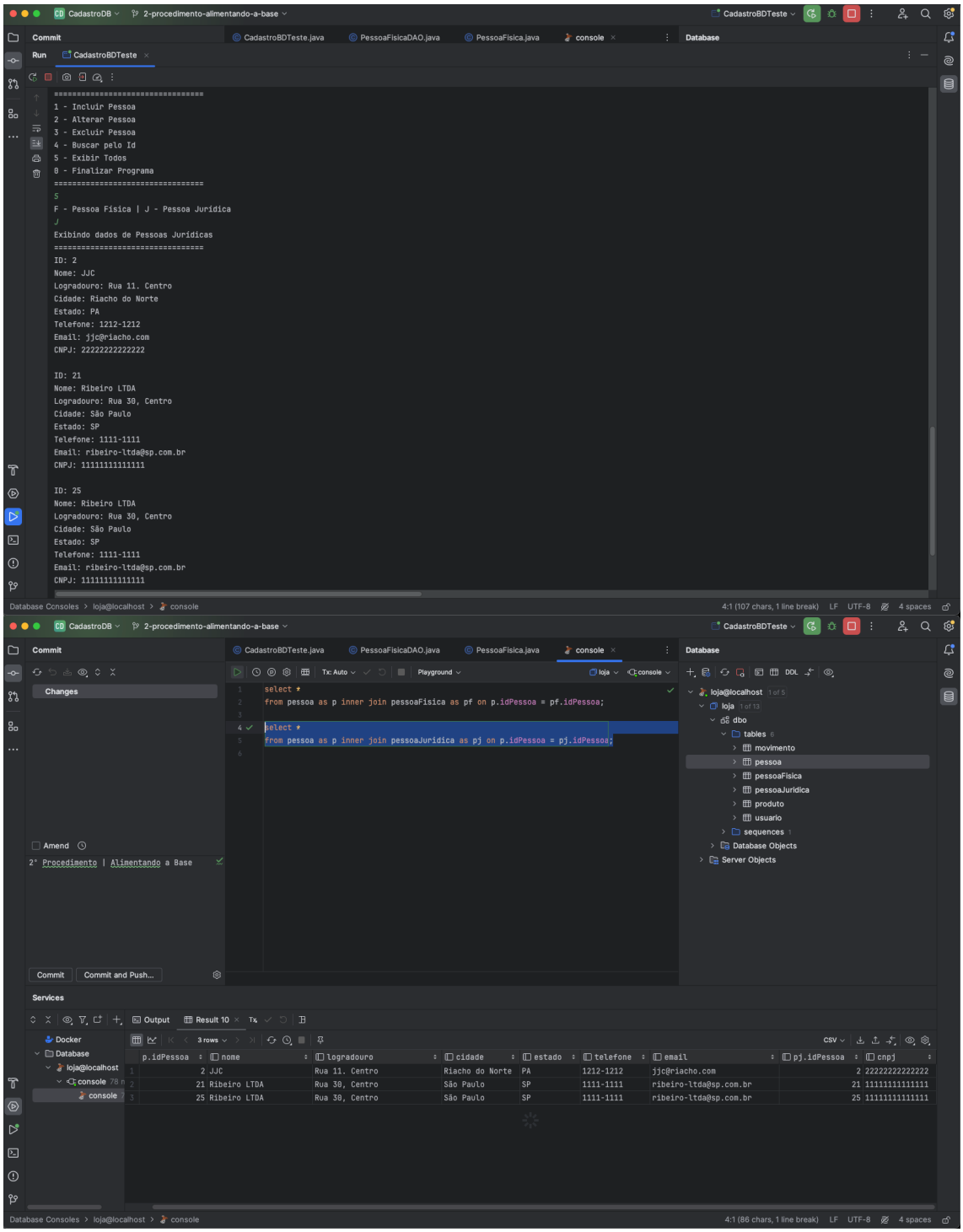
Excluir Pessoa Jurídica



Buscar Pessoa Jurídica por ID



Exibir Todas as Pessoas Jurídicas


CadastroDB
2-procedimento-alimentando-a-base
CadastroBDTeste
Commit
CadastroBDTeste.java
PessoaFisicaDAO.java
PessoaFisica.java
console
Database
Run
CadastroBDTeste
==========
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==========
5
F - Pessoa Física | J - Pessoa Jurídica
J
Exibindo dados de Pessoas Jurídicas
==========
ID: 2
Nome: JJC
Logradouro: Rua 11. Centro
Cidade: Riacho do Norte
Estado: PA
Telefone: 1212-1212
Email: jjc@riacho.com
CNPJ: 22222222222222
ID: 21
Nome: Ribeiro LTDA
Logradouro: Rua 30, Centro
Cidade: São Paulo
Estado: SP
Telefone: 1111-1111
Email: ribeiro-ltda@sp.com.br
CNPJ: 11111111111111
ID: 25
Nome: Ribeiro LTDA
Logradouro: Rua 30, Centro
Cidade: São Paulo
Estado: SP
Telefone: 1111-1111
Email: ribeiro-ltda@sp.com.br
CNPJ: 11111111111111
Database Consoles
loja@localhost
console
4:1 (107 chars, 1 line break)
LF
UTF-8
4 spaces

CadastroDB
2-procedimento-alimentando-a-base
CadastroBDTeste
Commit
Changes
Amend
2° Procedimento | Alimentando a Base
Commit
Commit and Push...
CadastroBDTeste.java
PessoaFisicaDAO.java
PessoaFisica.java
console
Tx: Auto
Playground
loja
console
select *
from pessoa as p inner join pessoaFisica as pf on p.idPessoa = pf.idPessoa;
select *
from pessoa as p inner join pessoaJuridica as pj on p.idPessoa = pj.idPessoa;
Database
loja@localhost
loja
dbo
tables
movimento
pessoa
pessoaFisica
pessoaJuridica
produto
usuario
sequences
Database Objects
Server Objects
Services
Output
Result 10
Docker
Database
loja@localhost
console
console
3 rows
CSV
p.idPessoa
nome
logradouro
cidade
estado
telefone
email
pj.idPessoa
cnpj
1 2 JJC Rua 11. Centro Riacho do Norte PA 1212-1212 jjc@riacho.com 2 22222222222222
2 21 Ribeiro LTDA Rua 30, Centro São Paulo SP 1111-1111 ribeiro-ltda@sp.com.br 21 11111111111111
3 25 Ribeiro LTDA Rua 30, Centro São Paulo SP 1111-1111 ribeiro-ltda@sp.com.br 25 11111111111111
Database Consoles
loja@localhost
console
4:1 (86 chars, 1 line break)
LF
UTF-8
4 spaces

Análise e Conclusão

a) **Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?**

A persistência em arquivos é uma prática que dificulta o processo de leitura, alteração e gravação dos dados, visto que depende da integridade de um arquivo gravado em disco, que pode ser corrompido ou até mesmo lido por qualquer outra fonte que tenha acesso ao disco rígido, além de tornar o processo de backup mais moroso, pois envolve processos manuais de versionamento dos backups, ele deve ser utilizado em cenários onde o volume de dados e volume de acessos é bem baixo, já o armazenamento em banco de dados permite uma flexibilidade maior, pois qualquer aplicação pode acessar o banco utilizando as credenciais de acesso, o que deixa o acesso e o controle do acesso aos dados mais protegido e possibilita a integração com diversas aplicações alimentadas pelas mesmas informações, além de permitir ler, gravar e manipular os dados com maior facilidade, ou executar filtragem avançada dos dados por possuir métodos nativos, o banco de dados também possui na maioria das vezes um próprio sistema de backup que pode ser configurado e seu versionamento é auto gerenciado.

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

Nas versões mais recentes do Java é possível imprimir valores utilizando menos código, que permite informar para um loop forEach por exemplo, como cada dado deve ser impresso na tela, ou até mesmo somente informar qual método de uma classe deve ser invocado para cada item do loop, isso facilita a compressão e leitura do código e reduz de forma siginificável o boilerplate que era causado por loops tradicionais como for ou foreach.

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

O java inicia a aplicação invocando diretamente o método main da classe principal sem instanciar a classe, se o método main não for do tipo static a JVM não irá conseguir iniciar a aplicação. O static indica que o método faz parte da classe e pode ser invocado diretamente sem estar vinculado a um objeto em específico, assim como todos os métodos que são invocados diretamente, sem instâncias de objetos, precisam ser do tipo static, para que possam ser executados sem objetos vinculados.