

CAMPUS: Estácio Via Corpvs

CURSO: Desenvolvimento FullStack

DISCIPLINA: Bankend sem banco, não tem!

TURMA: 2025.1

SEMESTRE LETIVO: Primeiro Semestre (2025)

ALUNO: Francisco Ernaldo Vieira

MATRÍCULA: 202401191817

**Título da prática:**

Criação de aplicativo Java, com acesso ao banco de dados SQL Server através de middleware JDBC.

**Objetivos da prática:**

* Implementar persistência com base no middleware JDBC.
* Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
* Implementar o mapeamento objeto-relacional em sistemas Java.
* Criar sistemas cadastrais com persistência em banco relacional.

**Procedimento 1:**

**Códigos usados neste roteiro:**

**CadastroBDTeste.java**

package cadastrobd;

import cadastrobd.model.PessoaFisica;

import cadastrobd.model.PessoaJuridica;

import cadastrobd.model.PessoaFisicaDAO;

import cadastrobd.model.PessoaJuridicaDAO;

public class CadastroBDTeste {

public static void main(String[] args) {

// Instanciar DAOs

PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();

PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

// OPERAÇÃO 1: Criar e persistir uma pessoa física

System.out.println("=== CRIANDO PESSOA FÍSICA ===");

PessoaFisica pessoaFisica = new PessoaFisica(0, "Ernaldo", "Rua A, 123", "Fortaleza", "CE", "1119292222", "ernal@email.com", "666.666.777-77");

if (!pessoaFisicaDAO.cpfExiste(pessoaFisica.getCpf())) {

pessoaFisicaDAO.incluir(pessoaFisica);

System.out.println("Pessoa física criada e persistida!");

} else {

System.out.println("Erro: CPF já existe no banco de dados!");

}

// OPERAÇÃO 2: Alterar os dados da pessoa física

System.out.println("\n=== ALTERANDO PESSOA FÍSICA ===");

pessoaFisica.setNome("frffff ddddddda");

pessoaFisica.setCpf("555.663.666-88");

pessoaFisicaDAO.alterar(pessoaFisica);

System.out.println("Pessoa física alterada!");

// OPERAÇÃO 3: Consultar todas as pessoas físicas

System.out.println("\n=== LISTANDO TODAS AS PESSOAS FÍSICAS ===");

for (PessoaFisica pf : pessoaFisicaDAO.getPessoas()) {

pf.exibir();

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

}

// OPERAÇÃO 4: Excluir a pessoa física

System.out.println("\n=== EXCLUINDO PESSOA FÍSICA ===");

pessoaFisicaDAO.excluir(pessoaFisica.getId());

System.out.println("Pessoa física excluída!");

// OPERAÇÃO 5: Criar e persistir uma pessoa jurídica

System.out.println("\n=== CRIANDO PESSOA JURÍDICA ===");

PessoaJuridica pessoaJuridica = new PessoaJuridica(0, "xxxx Empresa Ltda.", "Rua nnnl, 555", "Fortaleza", "CE", "3333377778", "nnnnnnnnnnn@email.com", "89.999.432/0001-01");

if (!pessoaJuridicaDAO.cnpjExiste(pessoaJuridica.getCnpj())) {

pessoaJuridicaDAO.incluir(pessoaJuridica);

System.out.println("Pessoa jurídica criada e persistida!");

} else {

System.out.println("Erro: CNPJ já existe no banco de dados!");

}

// OPERAÇÃO 6: Alterar os dados da pessoa jurídica

System.out.println("\n=== ALTERANDO PESSOA JURÍDICA ===");

pessoaJuridica.setNome("Elite Ltda.");

pessoaJuridica.setCnpj("87.678.999/0001-02");

pessoaJuridicaDAO.alterar(pessoaJuridica);

System.out.println("Pessoa jurídica alterada!");

// OPERAÇÃO 7: Consultar todas as pessoas jurídicas

System.out.println("\n=== LISTANDO TODAS AS PESSOAS JURÍDICAS ===");

for (PessoaJuridica pj : pessoaJuridicaDAO.getPessoas()) {

pj.exibir();

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

}

// OPERAÇÃO 8: Excluir a pessoa jurídica

System.out.println("\n=== EXCLUINDO PESSOA JURÍDICA ===");

pessoaJuridicaDAO.excluir(pessoaJuridica.getId());

System.out.println("Pessoa jurídica excluída!");

}

**Main.java**

package cadastrobd;

import cadastrobd.model.PessoaFisica;

import cadastrobd.model.PessoaJuridica;

public class Main {

public static void main(String[] args) {

// Criando uma Pessoa Física

PessoaFisica pessoaFisica = new PessoaFisica(1, "João Silva", "Rua A", "São Paulo", "SP", "1199999999", "joao@email.com", "123.456.789-00");

System.out.println("Dados da Pessoa Física:");

pessoaFisica.exibir();

System.out.println();

// Criando uma Pessoa Jurídica

PessoaJuridica pessoaJuridica = new PessoaJuridica(2, "Empresa ABC", "Rua B", "Rio de Janeiro", "RJ", "2188888888", "empresa@email.com", "12.345.678/0001-90");

System.out.println("Dados da Pessoa Jurídica:");

pessoaJuridica.exibir();

}

}

**Pessoa.java**

package cadastrobd.model;

public class Pessoa {

// Atributos

private int id;

private String nome;

private String logradouro;

private String cidade;

private String estado;

private String telefone;

private String email;

// Construtor padrão

public Pessoa() {

}

// Construtor completo

public Pessoa(int 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;

}

// Getters e Setters (opcionalmente implementados)

public int getId() {

return id;

}

public void setId(int 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;

}

// Método exibir

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.java**

package cadastrobd.model;

public class PessoaFisica extends Pessoa {

// Atributo especÃ­fico

private String cpf;

// Construtor padrÃ£o

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;

}

// Getter e Setter para CPF

public String getCpf() {

return cpf;

}

public void setCpf(String cpf) {

this.cpf = cpf;

}

// Sobrescrita do mÃ©todo exibir

@Override

public void exibir() {

super.exibir(); // Chama o mÃ©todo da classe pai

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

}

}

**PessoaFisicaDao.java**

package cadastrobd.model;

import cadastrobd.model.util.ConectorBD;

import cadastrobd.model.util.SequenceManager;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class PessoaFisicaDAO {

// Método para obter uma pessoa física pelo ID

public PessoaFisica getPessoa(int id) {

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

PessoaFisica pessoaFisica = null;

try {

connection = ConectorBD.getConnection();

// Consulta SQL ajustada

String sql = "SELECT p.id\_pessoa, p.nome, p.endereco, p.telefone, p.email, pf.cpf " +

"FROM pessoa p INNER JOIN pessoa\_fisica pf ON p.id\_pessoa = pf.id\_pessoa WHERE p.id\_pessoa = ?";

preparedStatement = ConectorBD.getPrepared(connection, sql);

preparedStatement.setInt(1, id);

resultSet = ConectorBD.getSelect(preparedStatement);

if (resultSet != null && resultSet.next()) {

pessoaFisica = new PessoaFisica(

resultSet.getInt("id\_pessoa"),

resultSet.getString("nome"),

resultSet.getString("endereco"), // Logradouro mapeado para 'endereco'

"", // Cidade não existe no banco

"", // Estado não existe no banco

resultSet.getString("telefone"),

resultSet.getString("email"),

resultSet.getString("cpf")

);

}

} catch (SQLException e) {

System.out.println("Erro ao buscar pessoa física!");

e.printStackTrace();

} finally {

ConectorBD.close(resultSet);

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

return pessoaFisica;

}

// Método para obter todas as pessoas físicas

public List<PessoaFisica> getPessoas() {

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

List<PessoaFisica> pessoasFisicas = new ArrayList<>();

try {

connection = ConectorBD.getConnection();

// Consulta SQL ajustada

String sql = "SELECT p.id\_pessoa, p.nome, p.endereco, p.telefone, p.email, pf.cpf " +

"FROM pessoa p INNER JOIN pessoa\_fisica pf ON p.id\_pessoa = pf.id\_pessoa";

preparedStatement = ConectorBD.getPrepared(connection, sql);

resultSet = ConectorBD.getSelect(preparedStatement);

while (resultSet != null && resultSet.next()) {

PessoaFisica pessoaFisica = new PessoaFisica(

resultSet.getInt("id\_pessoa"),

resultSet.getString("nome"),

resultSet.getString("endereco"), // Logradouro mapeado para 'endereco'

"", // Cidade não existe no banco

"", // Estado não existe no banco

resultSet.getString("telefone"),

resultSet.getString("email"),

resultSet.getString("cpf")

);

pessoasFisicas.add(pessoaFisica);

}

} catch (SQLException e) {

System.out.println("Erro ao buscar pessoas físicas!");

e.printStackTrace();

} finally {

ConectorBD.close(resultSet);

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

return pessoasFisicas;

}

// Método para incluir uma pessoa física

public void incluir(PessoaFisica pessoaFisica) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

connection = ConectorBD.getConnection();

// Obter o próximo ID da sequência

int id = SequenceManager.getValue("seq\_pessoa\_id");

// Inserir na tabela Pessoa

String sqlPessoa = "INSERT INTO pessoa (id\_pessoa, nome, endereco, telefone, email) VALUES (?, ?, ?, ?, ?)";

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoa);

preparedStatement.setInt(1, id);

preparedStatement.setString(2, pessoaFisica.getNome());

preparedStatement.setString(3, pessoaFisica.getLogradouro()); // Logradouro mapeado para 'endereco'

preparedStatement.setString(4, pessoaFisica.getTelefone());

preparedStatement.setString(5, pessoaFisica.getEmail());

preparedStatement.executeUpdate();

ConectorBD.close(preparedStatement);

// Inserir na tabela PessoaFisica

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoaFisica);

preparedStatement.setInt(1, id);

preparedStatement.setString(2, pessoaFisica.getCpf());

preparedStatement.executeUpdate();

} catch (SQLException e) {

System.out.println("Erro ao incluir pessoa física!");

e.printStackTrace();

} finally {

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

}

// Método para alterar uma pessoa física

public void alterar(PessoaFisica pessoaFisica) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

connection = ConectorBD.getConnection();

// Atualizar na tabela Pessoa

String sqlPessoa = "UPDATE pessoa SET nome = ?, endereco = ?, telefone = ?, email = ? WHERE id\_pessoa = ?";

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoa);

preparedStatement.setString(1, pessoaFisica.getNome());

preparedStatement.setString(2, pessoaFisica.getLogradouro()); // Logradouro mapeado para 'endereco'

preparedStatement.setString(3, pessoaFisica.getTelefone());

preparedStatement.setString(4, pessoaFisica.getEmail());

preparedStatement.setInt(5, pessoaFisica.getId());

preparedStatement.executeUpdate();

ConectorBD.close(preparedStatement);

// Atualizar na tabela PessoaFisica

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoaFisica);

preparedStatement.setString(1, pessoaFisica.getCpf());

preparedStatement.setInt(2, pessoaFisica.getId());

preparedStatement.executeUpdate();

} catch (SQLException e) {

System.out.println("Erro ao alterar pessoa física!");

e.printStackTrace();

} finally {

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

}

// Método para excluir uma pessoa física

public void excluir(int id) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

connection = ConectorBD.getConnection();

// Excluir da tabela PessoaFisica

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoaFisica);

preparedStatement.setInt(1, id);

preparedStatement.executeUpdate();

ConectorBD.close(preparedStatement);

// Excluir da tabela Pessoa

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoa);

preparedStatement.setInt(1, id);

preparedStatement.executeUpdate();

} catch (SQLException e) {

System.out.println("Erro ao excluir pessoa física!");

e.printStackTrace();

} finally {

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

}

public boolean cpfExiste(String cpf) {

// TODO Auto-generated method stub

return false;

}

}

**PessoaJuridica.java**

package cadastrobd.model;

public class PessoaJuridica extends Pessoa {

// Atributo específico

private String cnpj;

// Construtor padrão

public PessoaJuridica() {

}

// Construtor completo

public PessoaJuridica(int 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;

}

// Getter e Setter para CNPJ

public String getCnpj() {

return cnpj;

}

public void setCnpj(String cnpj) {

this.cnpj = cnpj;

}

// Sobrescrita do método exibir

@Override

public void exibir() {

super.exibir(); // Chama o método da classe pai

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

}

}

**PessoaJuridicaDAO.java**

package cadastrobd.model;

import cadastrobd.model.util.ConectorBD;

import cadastrobd.model.util.SequenceManager;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class PessoaJuridicaDAO {

// Método para obter uma pessoa jurídica pelo ID

public PessoaJuridica getPessoa(int id) {

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

PessoaJuridica pessoaJuridica = null;

try {

connection = ConectorBD.getConnection();

// Consulta SQL ajustada

String sql = "SELECT p.id\_pessoa, p.nome, p.endereco, p.telefone, p.email, pj.cnpj " +

"FROM pessoa p INNER JOIN pessoa\_juridica pj ON p.id\_pessoa = pj.id\_pessoa WHERE p.id\_pessoa = ?";

preparedStatement = ConectorBD.getPrepared(connection, sql);

preparedStatement.setInt(1, id);

resultSet = ConectorBD.getSelect(preparedStatement);

if (resultSet != null && resultSet.next()) {

pessoaJuridica = new PessoaJuridica(

resultSet.getInt("id\_pessoa"),

resultSet.getString("nome"),

resultSet.getString("endereco"), // Logradouro mapeado para 'endereco'

"", // Cidade não existe no banco

"", // Estado não existe no banco

resultSet.getString("telefone"),

resultSet.getString("email"),

resultSet.getString("cnpj")

);

}

} catch (SQLException e) {

System.out.println("Erro ao buscar pessoa jurídica!");

e.printStackTrace();

} finally {

ConectorBD.close(resultSet);

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

return pessoaJuridica;

}

// Método para obter todas as pessoas jurídicas

public List<PessoaJuridica> getPessoas() {

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

try {

connection = ConectorBD.getConnection();

// Consulta SQL ajustada

String sql = "SELECT p.id\_pessoa, p.nome, p.endereco, p.telefone, p.email, pj.cnpj " +

"FROM pessoa p INNER JOIN pessoa\_juridica pj ON p.id\_pessoa = pj.id\_pessoa";

preparedStatement = ConectorBD.getPrepared(connection, sql);

resultSet = ConectorBD.getSelect(preparedStatement);

while (resultSet != null && resultSet.next()) {

PessoaJuridica pessoaJuridica = new PessoaJuridica(

resultSet.getInt("id\_pessoa"),

resultSet.getString("nome"),

resultSet.getString("endereco"), // Logradouro mapeado para 'endereco'

"", // Cidade não existe no banco

"", // Estado não existe no banco

resultSet.getString("telefone"),

resultSet.getString("email"),

resultSet.getString("cnpj")

);

pessoasJuridicas.add(pessoaJuridica);

}

} catch (SQLException e) {

System.out.println("Erro ao buscar pessoas jurídicas!");

e.printStackTrace();

} finally {

ConectorBD.close(resultSet);

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

return pessoasJuridicas;

}

// Método para incluir uma pessoa jurídica

public void incluir(PessoaJuridica pessoaJuridica) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

connection = ConectorBD.getConnection();

// Obter o próximo ID da sequência

int id = SequenceManager.getValue("seq\_pessoa\_id");

// Inserir na tabela Pessoa

String sqlPessoa = "INSERT INTO pessoa (id\_pessoa, nome, endereco, telefone, email) VALUES (?, ?, ?, ?, ?)";

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoa);

preparedStatement.setInt(1, id);

preparedStatement.setString(2, pessoaJuridica.getNome());

preparedStatement.setString(3, pessoaJuridica.getLogradouro()); // Logradouro mapeado para 'endereco'

preparedStatement.setString(4, pessoaJuridica.getTelefone());

preparedStatement.setString(5, pessoaJuridica.getEmail());

preparedStatement.executeUpdate();

ConectorBD.close(preparedStatement);

// Inserir na tabela PessoaJuridica

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoaJuridica);

preparedStatement.setInt(1, id);

preparedStatement.setString(2, pessoaJuridica.getCnpj());

preparedStatement.executeUpdate();

} catch (SQLException e) {

System.out.println("Erro ao incluir pessoa jurídica!");

e.printStackTrace();

} finally {

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

}

// Método para alterar uma pessoa jurídica

public void alterar(PessoaJuridica pessoaJuridica) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

connection = ConectorBD.getConnection();

// Atualizar na tabela Pessoa

String sqlPessoa = "UPDATE pessoa SET nome = ?, endereco = ?, telefone = ?, email = ? WHERE id\_pessoa = ?";

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoa);

preparedStatement.setString(1, pessoaJuridica.getNome());

preparedStatement.setString(2, pessoaJuridica.getLogradouro()); // Logradouro mapeado para 'endereco'

preparedStatement.setString(3, pessoaJuridica.getTelefone());

preparedStatement.setString(4, pessoaJuridica.getEmail());

preparedStatement.setInt(5, pessoaJuridica.getId());

preparedStatement.executeUpdate();

ConectorBD.close(preparedStatement);

// Atualizar na tabela PessoaJuridica

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoaJuridica);

preparedStatement.setString(1, pessoaJuridica.getCnpj());

preparedStatement.setInt(2, pessoaJuridica.getId());

preparedStatement.executeUpdate();

} catch (SQLException e) {

System.out.println("Erro ao alterar pessoa jurídica!");

e.printStackTrace();

} finally {

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

}

// Método para excluir uma pessoa jurídica

public void excluir(int id) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

connection = ConectorBD.getConnection();

// Excluir da tabela PessoaJuridica

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoaJuridica);

preparedStatement.setInt(1, id);

preparedStatement.executeUpdate();

ConectorBD.close(preparedStatement);

// Excluir da tabela Pessoa

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

preparedStatement = ConectorBD.getPrepared(connection, sqlPessoa);

preparedStatement.setInt(1, id);

preparedStatement.executeUpdate();

} catch (SQLException e) {

System.out.println("Erro ao excluir pessoa jurídica!");

e.printStackTrace();

} finally {

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

}

public boolean cnpjExiste(String cnpj) {

// TODO Auto-generated method stub

return false;

}

}

**ConectorBD.java**

package cadastrobd.model.util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class ConectorBD {

// Método para obter uma conexão com o banco de dados

public static Connection getConnection() {

String url = "jdbc:postgresql://localhost:5432/loja";

String user = "loja";

String password = "loja";

try {

return DriverManager.getConnection(url, user, password);

} catch (SQLException e) {

System.out.println("Erro ao conectar ao banco de dados!");

e.printStackTrace();

}

return null;

}

// Método para obter um PreparedStatement a partir de uma consulta SQL

public static PreparedStatement getPrepared(Connection connection, String sql) {

try {

return connection.prepareStatement(sql);

} catch (SQLException e) {

System.out.println("Erro ao preparar a instrução SQL!");

e.printStackTrace();

}

return null;

}

// Método para executar uma consulta SELECT e retornar o ResultSet

public static ResultSet getSelect(PreparedStatement preparedStatement) {

try {

return preparedStatement.executeQuery();

} catch (SQLException e) {

System.out.println("Erro ao executar a consulta!");

e.printStackTrace();

}

return null;

}

// Métodos sobrecarregados para fechar recursos

// Fechar ResultSet

public static void close(ResultSet resultSet) {

if (resultSet != null) {

try {

resultSet.close();

} catch (SQLException e) {

System.out.println("Erro ao fechar ResultSet!");

e.printStackTrace();

}

}

}

// Fechar Statement ou PreparedStatement

public static void close(Statement statement) {

if (statement != null) {

try {

statement.close();

} catch (SQLException e) {

System.out.println("Erro ao fechar Statement!");

e.printStackTrace();

}

}

}

// Fechar Connection

public static void close(Connection connection) {

if (connection != null) {

try {

connection.close();

} catch (SQLException e) {

System.out.println("Erro ao fechar Connection!");

e.printStackTrace();

}

}

}

}

**SequenceManager.java**

package cadastrobd.model.util;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class SequenceManager {

public static int getValue(String sequenceName) {

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

int nextValue = -1;

try {

connection = ConectorBD.getConnection();

String sql = "SELECT nextval('" + sequenceName + "')";

preparedStatement = connection.prepareStatement(sql);

resultSet = preparedStatement.executeQuery();

if (resultSet != null && resultSet.next()) {

nextValue = resultSet.getInt(1);

}

} catch (SQLException e) {

System.out.println("Erro ao obter o próximo valor da sequência!");

e.printStackTrace();

} finally {

ConectorBD.close(resultSet);

ConectorBD.close(preparedStatement);

ConectorBD.close(connection);

}

return nextValue;

}

}

**Análise e Conclusão:**

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

O JDBC (Java Database Connectivity) é um componente middleware essencial que atua como intermediário entre aplicações Java e sistemas gerenciadores de banco de dados. Sua principal função é permitir a comunicação eficiente e independente entre a aplicação e diferentes bancos de dados, promovendo portabilidade, controle de conexões, execução de comandos SQL e gerenciamento de transações. Ao abstrair as especificidades dos bancos, o JDBC facilita o desenvolvimento de sistemas robustos e escaláveis, sendo amplamente utilizado em ambientes corporativos e acadêmicos.

**Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?**

No contexto do JDBC, Statement e PreparedStatement são utilizados para a execução de comandos SQL, mas diferem quanto à segurança, desempenho e reutilização. O Statement é mais simples e indicado para comandos ocasionais, porém menos seguro, pois está sujeito a ataques de SQL Injection ao concatenar dados diretamente na string SQL. Já o PreparedStatement oferece maior segurança e eficiência por meio da utilização de parâmetros (?) e da pré-compilação das instruções SQL, sendo ideal para consultas frequentes e com variáveis. Além disso, melhora a legibilidade do código e reduz o risco de erros de formatação de dados.

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

O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao **separar a lógica de acesso a dados da lógica de negócio**. Isso permite que alterações na forma como os dados são armazenados (por exemplo, mudança de banco de dados ou da tecnologia de persistência) sejam feitas sem impactar outras camadas da aplicação. Além disso, o DAO centraliza e organiza as operações de persistência (como inserção, consulta, atualização e exclusão), promovendo **baixo acoplamento, maior reutilização de código e facilidade nos testes unitários**. Com isso, o software se torna mais modular, fácil de entender, evoluir e manter ao longo do tempo.

**Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?**

Em modelos estritamente relacionais, a herança da programação orientada a objetos não é representada diretamente, sendo necessário aplicar estratégias de mapeamento para preservar a estrutura hierárquica entre classes. As três estratégias mais comuns são:

1. **Tabela única por hierarquia (single table)**: toda a hierarquia de classes é representada em uma única tabela com colunas para todos os atributos e um campo discriminador.
2. **Tabela por classe (joined strategy)**: cada classe da hierarquia possui sua própria tabela, ligadas por chave estrangeira; promove normalização e integridade referencial.
3. **Tabela por classe concreta (table per class)**: cada classe concreta possui sua própria tabela com todos os campos herdados, sem relacionamento com as demais.

Essas abordagens buscam equilibrar **desempenho, integridade e flexibilidade**, e sua escolha depende do domínio da aplicação e da complexidade do modelo. O mapeamento da herança é frequentemente tratado por frameworks ORM (como Hibernate ou JPA), que traduzem a lógica orientada a objetos para o paradigma relacional.

**LINK DO GITHUB:**

https://github.com/ernaldo777/semestre3---missao-pratica-3