**Anexo 5**

**Código Relacionado ao Exemplo do Capitulo 5**

O código foi desenvolvido na ferramenta IntelliJ IDEA 2023.1 (Community Edition).

**Autor.java**

package model.entities;

import java.io.Serializable;

import java.util.Date;

public class Autor implements Serializable {

private static final long serialVersionUID = 1L;

private Integer id;

private String nome;

private Date dtNasc;

public Autor() {

}

public Autor(Integer id, String nome, Date dtNasc) {

this.id = id;

this.nome = nome;

this.dtNasc = dtNasc;

}

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 Date getDtNasc() {

return dtNasc;

}

public void setDtNasc(Date dtNasc) {

this.dtNasc = dtNasc;

}

@Override

public String toString() {

return "Autor [id=" + id + ", Autor=" + nome + ", Nascimento=" + dtNasc + " HashCode=" + hashCode() + "]";

}

}

**Editora.java**

package model.entities;

import java.io.Serializable;

public class Editora implements Serializable {

private static final long serialVersionUID = 1L;

private Integer id;

private String nome;

public Editora() {

}

public Editora(Integer id, String nome) {

this.id = id;

this.nome = nome;

}

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;

}

@Override

public String toString() {

return "Editora [id=" + id + ", editora=" + nome + " HashCode=" + hashCode() + "]";

}

}

**Livro.java**

package model.entities;

import java.io.Serializable;

public class Livro implements Serializable {

private static final long serialVersionUID = 1L;

private Integer id;

private String titulo;

private Integer anoPub;

private Double preco;

private Editora editora;

public Livro() {

}

public Livro(Integer id, String titulo, Integer anoPub, Double preco, Editora editora) {

this.id = id;

this.titulo = titulo;

this.anoPub = anoPub;

this.preco = preco;

this.editora = editora;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getTitulo() {

return titulo;

}

public void setTitulo(String titulo) {

this.titulo = titulo;

}

public Integer getAnoPub() {

return anoPub;

}

public void setAnoPub(Integer email) {

this.anoPub = anoPub;

}

public Double getPreco() {

return preco;

}

public void setPreco(Double preco) {

this.preco = preco;

}

public Editora getEditora() {

return editora;

}

public void setEditora(Editora editora) {

this.editora = editora;

}

@Override

public String toString() {

return "Seller [id=" + id + ", titulo=" + titulo + ", anoPub=" + anoPub + ", preco="

+ preco + " HashCode=" + hashCode() + ", editora=" + editora + "]";

}

}

**LivroAutor.java**

package model.entities;

import java.io.Serializable;

import java.util.Date;

public class LivroAutor implements Serializable {

private static final long serialVersionUID = 1L;

private Integer id;

private String papel;

private Livro livro;

private Autor autor;

public LivroAutor() {

}

public LivroAutor(Integer id, String papel, Livro livro, Autor autor) {

this.id = id;

this.papel = papel;

this.livro = livro;

this.autor = autor;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getPapel() {

return papel;

}

public void setPapel(String name) {

this.papel = papel;

}

public Livro getLivro() {

return livro;

}

public void setLivro(Livro livro) {

this.livro = livro;

}

public Autor getAutor() {

return autor;

}

public void setAutor(Autor autor) {

this.autor = autor;

}

@Override

public String toString() {

return "Livro Autor [id=" + id + ", papel=" + papel

+ " HashCode=" + hashCode() + ", livro=" + livro + ", autor" + autor + "]";

}

}

package model.dao.interfaces;

import model.dao.EditoraDao;

import model.dao.AutorDao;

import model.dao.LivroDao;

import model.dao.LivroAutorDao;

**DaoFactory.java**

public class DaoFactory {

public static EditoraDaoI createEditoraDao() {

return new EditoraDao();

}

public static AutorDaoI createAutorDao() {

return new AutorDao();

}

public static LivroDaoI createLivroDao() {

return new LivroDao();

}

public static LivroAutorDaoI createLivroAutorDao() {

return new LivroAutorDao();

}

}

package model.dao.interfaces;

import model.entities.Autor;

import java.util.List;

**AutorDaoI.java**

public interface AutorDaoI {

void insert(Autor autor);

void update(Autor autor);

void deleteById(Integer id);

Autor findById(Integer id);

List<Autor> findAll();

List<Autor> select(Autor autor);

}

**EditoraDaoI.java**

package model.dao.interfaces;

import model.entities.Editora;

import java.util.List;

public interface EditoraDaoI {

void insert(Editora editora);

void update(Editora editora);

void deleteById(Integer id);

Editora findById(Integer id);

List<Editora> findAll();

List<Editora> select(Editora editora);

}

**LivroAutorDaoI.java**

package model.dao.interfaces;

import model.entities.LivroAutor;

import java.util.List;

public interface LivroAutorDaoI {

void insert(LivroAutor livroautor);

void update(LivroAutor livroautor);

void deleteById (Integer id);

LivroAutor findById(Integer id);

List<LivroAutor> findAll();

List<LivroAutor> listAll(LivroAutor livroautor);

}

**LivroDaoI.java**

package model.dao.interfaces;

import model.entities.Livro;

import java.util.List;

public interface LivroDaoI {

void insert(Livro livro);

void update(Livro livro);

void deleteById(Integer id);

Livro findById(Integer id);

List<Livro> findAll();

List<Livro> listAll(Livro livro);

}

**DbConnection.java**

package model.dao;

import java.io.FileInputStream;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Properties;

abstract class DbConnection {

public static void DbException(String message) {

throw new RuntimeException("Exception -> "+ message);

}

public static Connection getConnection() {

Connection conn = null;

if (conn == null) {

try {

Properties props = loadProperties();

String url = props.getProperty("dburl");

conn = DriverManager.getConnection(url, props);

}

catch (SQLException e) {

DbException(e.getMessage());

}

}

return conn;

}

public static ResultSet executeQuery(PreparedStatement st) {

ResultSet rs = null;

if (st != null) {

try {

rs = st.executeQuery();

} catch (SQLException e) {

DbException(e.getMessage());

}

}

return rs;

}

public static int executeUpdate(PreparedStatement st) {

int rowsAffected = 0;

if (st != null) {

try {

rowsAffected = st.executeUpdate();

} catch (SQLException e) {

DbException(e.getMessage());

}

}

return rowsAffected;

}

private static Properties loadProperties() {

try (FileInputStream fs = new FileInputStream("db.properties")) {

Properties props = new Properties();

props.load(fs);

return props;

}

catch (IOException e) {

DbException(e.getMessage());

}

return null;

}

}

**AutorDao.java**

package model.dao;

import model.dao.DbConnection;

import model.dao.interfaces.AutorDaoI;

import model.entities.Autor;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

public class AutorDao extends DbConnection implements AutorDaoI {

private static Connection conn;

public AutorDao() {

conn = getConnection();

}

@Override

public Autor findById(Integer id) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT \* FROM Autor WHERE id = ?");

st.setInt(1, id);

rs = DbConnection.executeQuery(st);

if (rs.next()) {

Autor obj = new Autor();

obj.setId(rs.getInt("id"));

obj.setNome(rs.getString("nome"));

obj.setDtNasc(rs.getDate("dtnasc"));

return obj;

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public List<Autor> findAll() {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT \* FROM Autor ORDER BY nome");

rs = DbConnection.executeQuery(st);

List<Autor> list = new ArrayList<>();

while (rs.next()) {

Autor obj = new Autor();

obj.setId(rs.getInt("id"));

obj.setNome(rs.getString("nome"));

obj.setDtNasc(rs.getDate("dtnasc"));

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public List<Autor> select(Autor autor) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT \* FROM Autor " +

"WHERE (id = ? OR " + Objects.isNull(autor.getId()) + ")" +

"AND (nome = ? OR " + Objects.isNull(autor.getNome()) + ")" +

"AND (dtnasc = ? OR " + Objects.isNull(autor.getDtNasc()) + ")" );

st.setInt(1, autor.getId());

st.setString(2, autor.getNome());

st.setDate(3, new java.sql.Date(autor.getDtNasc().getTime()));

rs = DbConnection.executeQuery(st);

List<Autor> list = new ArrayList<>();

while (rs.next()) {

Autor obj = new Autor();

obj.setId(rs.getInt("id"));

obj.setNome(rs.getString("nome"));

obj.setDtNasc(rs.getDate("dtnasc"));

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public void insert(Autor autor) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"INSERT INTO Autor " +

"(nome) " +

"(dtnasc) " +

"VALUES " +

"(?), (?)", +

Statement.RETURN\_GENERATED\_KEYS);

st.setString(1, autor.getNome());

st.setDate(2, new java.sql.Date(autor.getDtNasc().getTime()));

int rowsAffected = DbConnection.executeUpdate(st);

if (rowsAffected > 0) {

ResultSet rs = st.getGeneratedKeys();

if (rs.next()) {

int id = rs.getInt(1);

autor.setId(id);

}

}

else {

DbException("Unexpected error! No rows affected!");

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void update(Autor autor) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"UPDATE Autor " +

"SET nome = ?, dtnasc = ? " +

"WHERE id = ?");

st.setString(1, autor.getNome());

st.setDate(2, new java.sql.Date(autor.getDtNasc().getTime()));

st.setInt(3, autor.getId());

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void deleteById(Integer id) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"DELETE FROM Autor WHERE id = ?");

st.setInt(1, id);

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

}

**EditoraDao.java**

package model.dao;

import model.dao.DbConnection;

import model.dao.interfaces.EditoraDaoI;

import model.entities.Editora;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

public class EditoraDao extends DbConnection implements EditoraDaoI {

private static Connection conn;

public EditoraDao() {

conn = getConnection();

}

@Override

public Editora findById(Integer id) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT \* FROM Editora WHERE id = ?");

st.setInt(1, id);

rs = DbConnection.executeQuery(st);

if (rs.next()) {

Editora obj = new Editora();

obj.setId(rs.getInt("id"));

obj.setNome(rs.getString("nome"));

return obj;

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public List<Editora> findAll() {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT \* FROM editora ORDER BY nome");

rs = DbConnection.executeQuery(st);

List<Editora> list = new ArrayList<>();

while (rs.next()) {

Editora obj = new Editora();

obj.setId(rs.getInt("id"));

obj.setNome(rs.getString("nome"));

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public List<Editora> select(Editora editora) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT \* FROM editora " +

"WHERE (id = ? OR " + Objects.isNull(editora.getId()) + ")" +

"AND (nome = ? OR " + Objects.isNull(editora.getNome()) + ")" );

st.setInt(1, editora.getId());

st.setString(2, editora.getNome());

rs = DbConnection.executeQuery(st);

List<Editora> list = new ArrayList<>();

while (rs.next()) {

Editora obj = new Editora();

obj.setId(rs.getInt("id"));

obj.setNome(rs.getString("nome"));

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public void insert(Editora editora) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"INSERT INTO editora " +

"(nome) " +

"VALUES " +

"(?)",

Statement.RETURN\_GENERATED\_KEYS);

st.setString(1, editora.getNome());

int rowsAffected = DbConnection.executeUpdate(st);

if (rowsAffected > 0) {

ResultSet rs = st.getGeneratedKeys();

if (rs.next()) {

int id = rs.getInt(1);

editora.setId(id);

}

}

else {

DbException("Unexpected error! No rows affected!");

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void update(Editora editora) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"UPDATE Editora " +

"SET nome = ? " +

"WHERE id = ?");

st.setString(1, editora.getNome());

st.setInt(2, editora.getId());

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void deleteById(Integer id) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"DELETE FROM Editora WHERE id = ?");

st.setInt(1, id);

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

}

**LivroDao.java**

package model.dao;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.\*;

import java.sql.Statement;

import model.dao.interfaces.LivroDaoI;

import model.entities.Editora;

import model.entities.Livro;

public class LivroDao extends DbConnection implements LivroDaoI {

private Connection conn;

public LivroDao() {

conn = getConnection();

}

@Override

public void insert(Livro livro) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"INSERT INTO livro "

+ "(titulo, anoPub, preco, idEditora) "

+ "VALUES "

+ "(?, ?, ?, ?)",

Statement.RETURN\_GENERATED\_KEYS);

st.setString(1, livro.getTitulo());

st.setInt(2, livro.getAnoPub());

st.setDouble(3, livro.getPreco());

st.setInt(4, livro.getEditora().getId());

int rowsAffected = DbConnection.executeUpdate(st);

if (rowsAffected > 0) {

ResultSet rs = st.getGeneratedKeys();

if (rs.next()) {

int id = rs.getInt(1);

livro.setId(id);

}

st.close();

}

else {

DbException("Unexpected error! No rows affected!");

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void update(Livro livro) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"UPDATE livro "

+ "SET titulo = ?, anoPub = ?, preco = ?, idEditora = ? "

+ "WHERE id = ?");

st.setString(1, livro.getTitulo());

st.setInt(2, livro.getAnoPub());

st.setDouble(3, livro.getPreco());

st.setInt(4, livro.getEditora().getId());

st.setInt(5, livro.getId());

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void deleteById(Integer id) {

PreparedStatement st = null;

try {

st = conn.prepareStatement("DELETE FROM livro WHERE id = ?");

st.setInt(1, id);

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public Livro findById(Integer id) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT Livro.\*,Editora.id as editoraId, Editora.nome as editoraNome "

+ "FROM Livro INNER JOIN Editora "

+ "ON Livro.idEditora = Editora.Id "

+ "WHERE Livro.id = ?");

st.setInt(1, id);

rs = DbConnection.executeQuery(st);

if (rs.next()) {

Editora editora = loadEditora(rs);

Livro obj = loadLivro(rs, editora);

return obj;

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

private Livro loadLivro(ResultSet rs, Editora editora) throws SQLException {

Livro livro = new Livro();

livro.setId(rs.getInt("id"));

livro.setTitulo(rs.getString("titulo"));

livro.setAnoPub(rs.getInt("anoPub"));

livro.setPreco(rs.getDouble("preco"));

livro.setEditora(editora);

return livro;

}

private Editora loadEditora(ResultSet rs) throws SQLException {

Editora editora = new Editora();

editora.setId(rs.getInt("id"));

editora.setNome(rs.getString("nome"));

return editora;

}

@Override

public List<Livro> findAll() {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT Livro.\*,Editora.id as editoraId,Editora.nome as editoraNome "

+ "FROM Livro INNER JOIN Editora "

+ "ON Livro.id = Editora.id "

+ "ORDER BY Livro.nome");

rs = DbConnection.executeQuery(st);

List<Livro> list = new ArrayList<>();

Map<Integer, Editora> map = new HashMap<>();

while (rs.next()) {

Editora editora = map.get(rs.getInt("editoraId"));

if (editora == null) {

editora = loadEditora(rs);

map.put(rs.getInt("editoraId"), editora);

}

Livro obj = loadLivro(rs, editora);

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public List<Livro> listAll(Livro livro) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT Livro.\*,Editora.id as editoraId,Editora.nome as editoraNome "

+ "FROM Livro, Editora "

+ "WHERE Livro.idEditora = Editora.id "

+ "AND (Livro.id = ? OR " + Objects.isNull(livro.getId()) + ") "

+ "AND (Livro.titulo LIKE ? OR " + Objects.isNull(livro.getTitulo()) + ") "

+ "AND (Livro.preco = ? OR " + Objects.isNull(livro.getPreco()) + ") "

+ "AND (Editora.id = ? OR " + (Objects.isNull(livro.getEditora()) || Objects.isNull(livro.getEditora().getId())) + ") "

+ "AND (Editora.nome LIKE ? OR " + (Objects.isNull(livro.getEditora()) || Objects.isNull(livro.getEditora().getNome())) + ") "

+ "ORDER BY Livro.id");

st.setInt(1, Objects.isNull(livro.getId()) ? 0 : livro.getId());

st.setString(2, Objects.isNull(livro.getTitulo()) ?"":"%"+livro.getTitulo()+"%");

st.setString(3, String.valueOf(Objects.isNull(livro.getPreco()) ? 0 : livro.getPreco()));

st.setInt(4, (Objects.isNull(livro.getEditora()) || Objects.isNull(livro.getEditora().getId())) ? 0 : livro.getEditora().getId());

st.setString(5, (Objects.isNull(livro.getEditora()) || Objects.isNull(livro.getEditora().getNome())) ? "" : "%"+livro.getEditora().getNome()+"%");

rs = DbConnection.executeQuery(st);

List<Livro> list = new ArrayList<>();

Map<Integer, Editora> map = new HashMap<>();

while (rs.next()) {

Editora editora = map.get(rs.getInt("idEditora"));

if (editora == null) {

editora = loadEditora(rs);

map.put(rs.getInt("idEditora"), editora);

}

Livro obj = loadLivro(rs, editora);

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

}

**LivroAutorDao.java**

package model.dao;

import model.dao.interfaces.LivroAutorDaoI;

import model.entities.LivroAutor;

import model.entities.Livro;

import model.entities.Editora;

import model.entities.Autor;

import java.sql.\*;

import java.util.\*;

public class LivroAutorDao extends DbConnection implements LivroAutorDaoI {

private Connection conn;

public LivroAutorDao() {

conn = getConnection();

}

@Override

public void insert(LivroAutor livroautor) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"INSERT INTO LivroAutor "

+ "(idLivro, idAutor, papel) "

+ "VALUES "

+ "(?, ?, ?)",

Statement.RETURN\_GENERATED\_KEYS);

st.setInt(1, livroautor.getLivro().getId());

st.setInt(2, livroautor.getAutor().getId());

st.setString(3, livroautor.getPapel());

int rowsAffected = DbConnection.executeUpdate(st);

if (rowsAffected > 0) {

ResultSet rs = st.getGeneratedKeys();

if (rs.next()) {

int id = rs.getInt(1);

livroautor.setId(id);

}

st.close();

}

else {

DbException("Unexpected error! No rows affected!");

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void update(LivroAutor livroautor) {

PreparedStatement st = null;

try {

st = conn.prepareStatement(

"UPDATE LivroAutor "

+ "SET idLivro = ?, idAutor = ?, papel = ? "

+ "WHERE id = ?");

st.setInt(1, livroautor.getLivro().getId());

st.setInt(2, livroautor.getAutor().getId());

st.setString(3, livroautor.getPapel());

st.setInt(5, livroautor.getId());

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public void deleteById(Integer id) {

PreparedStatement st = null;

try {

st = conn.prepareStatement("DELETE FROM LivroAutor WHERE id = ?");

st.setInt(1, id);

DbConnection.executeUpdate(st);

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

}

}

@Override

public LivroAutor findById(Integer id) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT LivroAutor.id, LivroAutor.papel, Livro.id as livroId, Livro.titulo as livroTitulo, "

+ "Livro.anoPub as livroAnoPub, Livro.preco as livroPreco, Editora.id as editoraId, "

+ "Editora.nome as editoraNome, Autor.id as autorId, Autor.nome as autorNome, "

+ "Autor.dtNasc as autorDtNasc "

+ "FROM LivroAutor, Livro, Autor, Editora "

+ "WHERE LivroAutor.idEditora = Editora.Id "

+ "AND Livro.idEditora = Editora.id "

+ "AND LivroAutor.idAutor = Autor.id "

+ "WHERE LivroAutor.id = ?");

st.setInt(1, id);

rs = DbConnection.executeQuery(st);

if (rs.next()) {

Editora editora = loadEditora(rs);

Livro livro = loadLivro(rs, editora);

Autor autor = loadAutor(rs);

LivroAutor obj = loadLivroAutor(rs, livro, autor);

return obj;

}

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

private Livro loadLivro(ResultSet rs, Editora editora) throws SQLException {

Livro livro = new Livro();

livro.setId(rs.getInt("livroId"));

livro.setTitulo(rs.getString("livroTitulo"));

livro.setAnoPub(rs.getInt("livroAnoPub"));

livro.setPreco(rs.getDouble("livroPreco"));

livro.setEditora(editora);

return livro;

}

private Editora loadEditora(ResultSet rs) throws SQLException {

Editora editora = new Editora();

editora.setId(rs.getInt("editoraId"));

editora.setNome(rs.getString("editoraNome"));

return editora;

}

private Autor loadAutor(ResultSet rs) throws SQLException {

Autor autor = new Autor();

autor.setId(rs.getInt("autorId"));

autor.setNome(rs.getString("autorNome"));

autor.setDtNasc(rs.getDate("autorDtNasc"));

return autor;

}

private LivroAutor loadLivroAutor(ResultSet rs, Livro livro, Autor autor) throws SQLException {

LivroAutor livroautor = new LivroAutor();

livroautor.setId(rs.getInt("id"));

livroautor.setPapel(rs.getString("papel"));

livroautor.setLivro(livro);

livroautor.setAutor(autor);

return livroautor;

}

@Override

public List<LivroAutor> findAll() {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement("SELECT LivroAutor.id, LivroAutor.papel, Livro.id as livroId, Livro.titulo as livroTitulo, "

+ "Livro.anoPub as livroAnoPub, Livro.preco as livroPreco, Editora.id as editoraId, "

+ "Editora.nome as editoraNome, Autor.id as autorId, Autor.nome as autorNome, "

+ "Autor.dtNasc as autorDtNasc "

+ "FROM LivroAutor, Livro, Autor, Editora "

+ "WHERE LivroAutor.idEditora = Editora.Id "

+ "AND Livro.idEditora = Editora.id "

+ "AND LivroAutor.idAutor = Autor.id "

+ "ORDER BY LivroAutor.id");

rs = DbConnection.executeQuery(st);

List<LivroAutor> list = new ArrayList<>();

Map<Integer, Editora> mapEditora = new HashMap<>();

Map<Integer, Autor> mapAutor = new HashMap<>();

Map<Integer, Livro> mapLivro = new HashMap<>();

while (rs.next()) {

Editora editora = mapEditora.get(rs.getInt("editoraId"));

if (editora == null) {

editora = loadEditora(rs);

mapEditora.put(rs.getInt("editoraId"), editora);

}

Livro livro = mapLivro.get(rs.getInt("editoraId"));

if (livro == null) {

livro = loadLivro(rs, editora);

mapLivro.put(rs.getInt("editoraId"), livro);

}

Autor autor = mapAutor.get(rs.getInt("autorId"));

if (autor == null) {

autor = loadAutor(rs);

mapAutor.put(rs.getInt("autorId"), autor);

}

LivroAutor obj = loadLivroAutor(rs, livro, autor);

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

@Override

public List<LivroAutor> listAll(LivroAutor livroautor) {

PreparedStatement st = null;

ResultSet rs = null;

try {

st = conn.prepareStatement(

"SELECT LivroAutor.id, LivroAutor.papel, Livro.id as livroId, Livro.titulo as livroTitulo, "

+ "Livro.anoPub as livroAnoPub, Livro.preco as livroPreco, Editora.id as editoraId, "

+ "Editora.nome as editoraNome, Autor.id as autorId, Autor.nome as autorNome, "

+ "Autor.dtNasc as autorDtNasc "

+ "FROM LivroAutor, Livro, Autor, Editora "

+ "WHERE LivroAutor.idEditora = Editora.Id "

+ "AND Livro.idEditora = Editora.id "

+ "AND LivroAutor.idAutor = Autor.id "

+ "AND (LivroAutor.id = ? OR " + Objects.isNull(livroautor.getId()) + ") "

+ "AND (LivroAutor.papel LIKE ? OR " + Objects.isNull(livroautor.getPapel()) + ") "

+ "ORDER BY Livro.nome");

st.setInt(1, Objects.isNull(livroautor.getId()) ? 0 : livroautor.getId());

st.setString(2, Objects.isNull(livroautor.getPapel()) ?"":"%"+livroautor.getPapel()+"%");

rs = DbConnection.executeQuery(st);

List<LivroAutor> list = new ArrayList<>();

Map<Integer, Editora> mapEditora = new HashMap<>();

Map<Integer, Autor> mapAutor = new HashMap<>();

Map<Integer, Livro> mapLivro = new HashMap<>();

while (rs.next()) {

Editora editora = mapEditora.get(rs.getInt("editoraId"));

if (editora == null) {

editora = loadEditora(rs);

mapEditora.put(rs.getInt("editoraId"), editora);

}

Livro livro = mapLivro.get(rs.getInt("editoraId"));

if (livro == null) {

livro = loadLivro(rs, editora);

mapLivro.put(rs.getInt("editoraId"), livro);

}

Autor autor = mapAutor.get(rs.getInt("autorId"));

if (autor == null) {

autor = loadAutor(rs);

mapAutor.put(rs.getInt("autorId"), autor);

}

LivroAutor obj = loadLivroAutor(rs, livro, autor);

list.add(obj);

}

return list;

}

catch (SQLException e) {

DbException(e.getMessage());

}

finally {

st=null;

rs=null;

}

return null;

}

}