UNIVERSIDADE CATÓLICA DO SALVADOR - UCSAL

ALANA DE JESUS COSTA SANTOS E KAYKE PEREIRA CARVALHO QUEIROZ

GERENCIADOR DE LIVROS
BANCO DE DADOS

SALVADOR
2024

1. INTRODUÇÃO

O sistema foi projetado para permitir o controle eficiente de informações relacionadas a autores e livros, oferecendo funcionalidades como inserção, consulta, edição e exclusão dos dados. Além disso, o sistema também disponibiliza recursos adicionais, como a listagem de informações por critérios específicos e a exibição de estatísticas sobre o acervo de livros.

O sistema foi desenvolvido utilizando a linguagem de programação Java, com foco na integração com banco de dados para garantir a persistência e a organização dos dados. A interface do sistema é baseada em um menu interativo, que oferece uma experiência de uso simples e intuitiva para os usuários.

2. REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

2.1 FUNCIONAIS

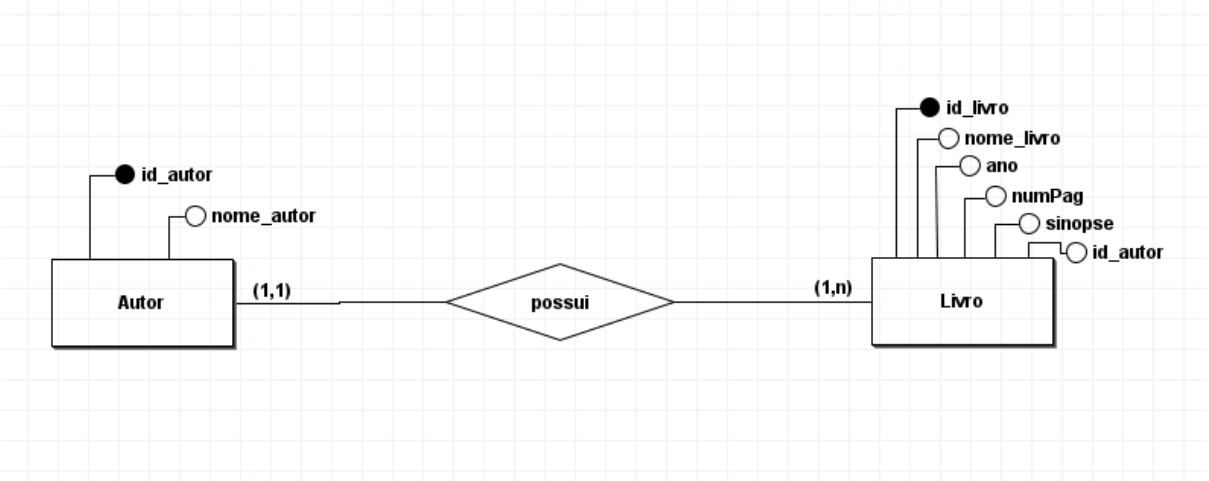
RF001 - Permitir a inserção de informações detalhadas de livros
RF002 - Permitir o cadastro de informações de autores
RF003 - Consulta de dados pelo identificador do livro e autor
RF004 - Permitir a atualização de informações de livros e autores já cadastrados no sistema.
RF005 - Disponibilizar a remoção de livros e autores do sistema com confirmação antes de finalizar a operação.
RF006 - Garantir que todas as informações cadastradas sejam armazenadas em um banco de dados e possam ser recuperadas ao reiniciar o sistema.
RF007 - Disponibilizar uma interface baseada em menu que permita navegação intuitiva entre as funcionalidades.

2.2 NÃO FUNCIONAIS

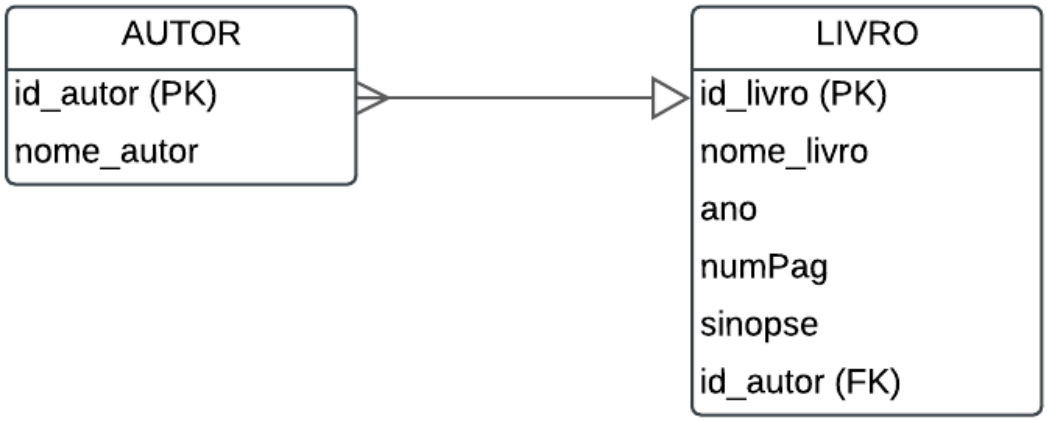
RNF001 - O sistema será desenvolvido na linguagem Java, aproveitando seus recursos para integração com banco de dados e desenvolvimento robusto.
RNF002 - A interface gráfica do sistema será projetada para ser simples e intuitiva
RNF003 - O sistema estará em conformidade com a Lei Geral de Proteção de Dados (LGPD), garantindo a privacidade e segurança das informações pessoais armazenadas
RNF004 - Consentimento para coleta de dados sensíveis.
RNF005 - Exclusão de informações mediante solicitação.
RNF006 - Os dados serão armazenados em um banco de dados relacional para garantir organização e confiabilidade na recuperação de informações.
RNF007 - A documentação do código e do sistema deve ser clara e completa, com explicações sobre as principais funcionalidades e dependências.

3. MODELO DE DADOS

Modelo Conceitual:



Modelo Lógico:



4. SCRIPT DO BANCO DE DADOS - DDL (DEFINIÇÃO DE ESTRUTURA)

```
-- Criando tabela AUTOR
CREATE TABLE autor (
    id_autor SERIAL PRIMARY KEY,
    nome_autor VARCHAR(200)
);

-- Criando tabela LIVRO
CREATE TABLE livro (
    id_livro SERIAL PRIMARY KEY,
    nome_livro VARCHAR(200),
    ano_publicacao INTEGER,
    num_pag INTEGER,
    sinopse TEXT,
    id_autor INTEGER,
    FOREIGN KEY (id_autor) REFERENCES autor(id_autor)
);

-- Criando VIEW para simplificar consulta de livros e autores
CREATE VIEW view_livros_autores AS
SELECT livro.nome_livro, autor.nome_autor
```

```
FROM livro
JOIN autor ON livro.id_autor = autor.id_autor;

-- Função para contar livros cadastrados
CREATE FUNCTION total_livro() RETURNS INTEGER AS $$
DECLARE
    livro_total INTEGER;
BEGIN
    SELECT COUNT(*) INTO livro_total FROM livro;
    RETURN livro_total;
END;
$$ LANGUAGE plpgsql;

-- Função para ordenar os autores pelo ID
CREATE OR REPLACE FUNCTION autores_ordem()
    RETURNS TABLE(id_autor INT, nome_autor VARCHAR) AS $$
BEGIN
    RETURN QUERY
    SELECT autor.id_autor, autor.nome_autor
    FROM autor
    ORDER BY autor.id_autor;
END;
$$ LANGUAGE plpgsql;

-- Função para ordenar os livros pelo ID
CREATE OR REPLACE FUNCTION livros_ordem()
    RETURNS TABLE(id_livro INT, nome_livro VARCHAR) AS $$
BEGIN
    RETURN QUERY
    SELECT livro.id_livro, livro.nome_livro
    FROM livro
    ORDER BY livro.id_livro;
END;
$$ LANGUAGE plpgsql;
```

5. SCRIPT DO BANCO DE DADOS - DML (MANIPULAÇÃO DE DADOS)

```
-- Inserindo registros na tabela AUTOR
INSERT INTO autor (nome_autor) VALUES ('Autor 1');
INSERT INTO autor (nome_autor) VALUES ('Autor 2');
INSERT INTO autor (nome_autor) VALUES ('Autor 3');

-- Inserindo registros na tabela LIVRO
INSERT INTO livro (nome_livro, ano_publicacao, num_pag, sinopse,
id_autor)
VALUES ('Livro A', 2021, 300, 'Sinopse do Livro A', 1);

INSERT INTO livro (nome_livro, ano_publicacao, num_pag, sinopse,
id_autor)
VALUES ('Livro B', 2022, 150, 'Sinopse do Livro B', 2);

INSERT INTO livro (nome_livro, ano_publicacao, num_pag, sinopse,
id_autor)
VALUES ('Livro C', 2023, 200, 'Sinopse do Livro C', 3);

-- Consultando todos os autores
SELECT * FROM autor;

-- Consultando todos os livros
SELECT * FROM livro;
```

```sql
-- Usando a VIEW para consultar livros e autores
SELECT * FROM view_livros_autores;

-- Usando a função para contar livros
SELECT total_livro();

-- Usando a função para listar autores ordenados pelo ID
SELECT * FROM autores_ordem();

-- Usando a função para listar livros ordenados pelo ID
SELECT * FROM livros_ordem();
```

6. PROGRAMAÇÃO COM JAVA

1. CLASSE: AUTORDB

```java
// metodo para inserir autor
public void inserirAutor(Autor autor) throws ClassNotFoundException, SQLException {

    // SQL para inserir um novo registro de autor com o retorno do ID gerado
    String sql = "INSERT INTO autor (nome_autor) VALUES (?) RETURNING id_autor";

    // garantir que a conexão e o PreparedStatement sejam fechados automaticamente
    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {

        //preenchendo o parâmetro do sql com os dados do autor
        statement.setString(parameterIndex:1, autor.getNome());

        // executando a query e obtendo o ID gerado
        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {
            int generatedId = resultSet.getInt(columnLabel:"id_autor");
            autor.setId(generatedId); // atribuir o ID gerado ao objeto Autor
        }
    }
}
```

1.1 Declaração do Método:

`public void inserirAutor(Autor autor)`:

O método é público, ou seja, pode ser acessado por outras classes.

Não retorna nenhum valor (void).

Recebe como parâmetro um objeto da classe `Autor`, que contém os dados do autor a ser inserido.

1.2 Construção da Query SQL:

`String sql = "INSERT INTO autor (nome_autor) VALUES (?) RETURNING id_autor";`:

A string `sql` armazena a instrução SQL que será executada no banco de dados.

`INSERT INTO autor`: Insere um novo registro na tabela `autor`.

`(nome_autor) VALUES (?)`: Especifica que o campo `nome_autor` será preenchido com o valor do parâmetro `?`.

`RETURNING id_autor`: Retorna o valor gerado para o campo `id_autor` (geralmente uma chave primária autoincrementada), que será o identificador único do novo registro.

1.3 Criação da Conexão e PreparedStatement:

`try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) { ... }`:

Utiliza um bloco `try-with-resources` para garantir que a conexão e o `PreparedStatement` sejam fechados automaticamente ao final do bloco, mesmo que ocorram exceções.

`Conexao.criarConexao()`: Chama um método (não mostrado no código) para estabelecer a conexão com o banco de dados.

`prepareStatement(sql)`: Cria um `PreparedStatement` a partir da string SQL, permitindo a parametrização da query.

1.4 Preenchimento do Parâmetro:

`statement.setString(1, autor.getNome());`:

Define o valor do primeiro parâmetro (`?`) da query SQL com o nome do autor obtido do objeto `Autor`.

1.5 Execução da Query e Obtenção do ID Gerado:

`ResultSet resultSet = statement.executeQuery();`:

Executa a query e armazena o resultado em um objeto `ResultSet`.

`if (resultSet.next()) { ... }`:

Verifica se há algum resultado (o ID gerado) no `ResultSet`.

`int generatedId = resultSet.getInt("id_autor");`:

Obtém o valor do campo `id_autor` do primeiro registro do `ResultSet` e armazena em `generatedId`.

`autor.setId(generatedId);`:

Atribui o ID gerado ao objeto `Autor` passado como parâmetro, atualizando o objeto com o novo identificador.

2.

```java
// deletar um autor pelo ID
public void deletarAutor(int idAutor) throws ClassNotFoundException, SQLException {
    String sql = "DELETE FROM autor WHERE id_autor = ?";

    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {
        statement.setInt(parameterIndex:1, idAutor); //definindo o ID do livro a ser deletado
        statement.executeUpdate(); //executa o delete
    }
}
```

2.1 Método para deletar um autor pelo ID:

> **`String sql = "DELETE FROM autor WHERE id_autor = ?";`:**
> A string `sql` define a instrução para deletar um registro da tabela `autor` onde o campo `id_autor` seja igual ao valor fornecido pelo parâmetro `idAutor`.
> O símbolo `?` Representa o parâmetro que será preenchido posteriormente.

2.2 Criação da conexão e do PreparedStatement:

`try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) { ... }`:

> Utiliza um bloco `try-with-resources` para garantir o fechamento automático da conexão e do `PreparedStatement` ao final do bloco.
>
> `Conexao.criarConexao()`: Chama um método (não mostrado) para estabelecer a conexão com o banco de dados.
>
> `prepareStatement(sql)`: Cria um `PreparedStatement` a partir da string SQL, permitindo a parametrização da query.

2.3 Definindo o valor do parâmetro e executando a operação:

`statement.setInt(1, idAutor);`

> O método `setInt(1, idAutor)` define o valor do primeiro parâmetro (o `?`) no `PreparedStatement`, substituindo-o pelo valor de `idAutor` passado para o método `deletarAutor`.

`statement.executeUpdate();`

> O método `executeUpdate()` executa a query SQL, que neste caso é um `DELETE`, removendo o autor com o `id_autor` correspondente ao valor fornecido.

3. Consultar autor por ID

```java
// consultar autor por ID
public Autor consultarAutor(int idAutor) throws ClassNotFoundException, SQLException {
    String sql = "SELECT * FROM autor WHERE id_autor = ?";

    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {
        statement.setInt(parameterIndex:1, idAutor);

        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {
            return new Autor(resultSet.getInt(columnLabel:"id_autor"), resultSet.getString(columnLabel:"nome_autor")); //retorna o autor encontrado
        } else {
            return null; // retorna null se o autor não for encontrado
        }
    }
}
```

3.1 Método para consultar um autor por ID:

```
String sql = "SELECT * FROM autor WHERE id_autor = ?";
```

A string `sql` define a instrução SQL para selecionar todos os campos (`*`) da tabela `autor` onde o campo `id_autor` seja igual ao valor fornecido pelo parâmetro `idAutor`.

O símbolo `?` é um parâmetro que será substituído pelo valor de `idAutor` quando a query for executada.

3.2 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao();
PreparedStatement statement = conexao.prepareStatement(sql)) {
... }
```

O código utiliza um bloco `try-with-resources` para garantir o fechamento automático da conexão e do `PreparedStatement` quando o bloco for finalizado.

`Conexao.criarConexao()`: Chama o método responsável por criar e retornar a conexão com o banco de dados.

`conexao.prepareStatement(sql)`: Cria um `PreparedStatement` utilizando a string SQL definida, permitindo a execução de consultas parametrizadas.

3.3 Definindo o valor do parâmetro e executando a consulta:

```
statement.setInt(1, idAutor);
```

O método `setInt(1, idAutor)` define o valor do primeiro parâmetro (o `?`) no `PreparedStatement`, substituindo-o pelo valor de `idAutor` passado para o método `consultarAutor`.

```
ResultSet resultSet = statement.executeQuery();
```

O método `executeQuery()` executa a consulta SQL e retorna um `ResultSet`, que contém o conjunto de resultados da consulta.

3.4 Processando o resultado da consulta:

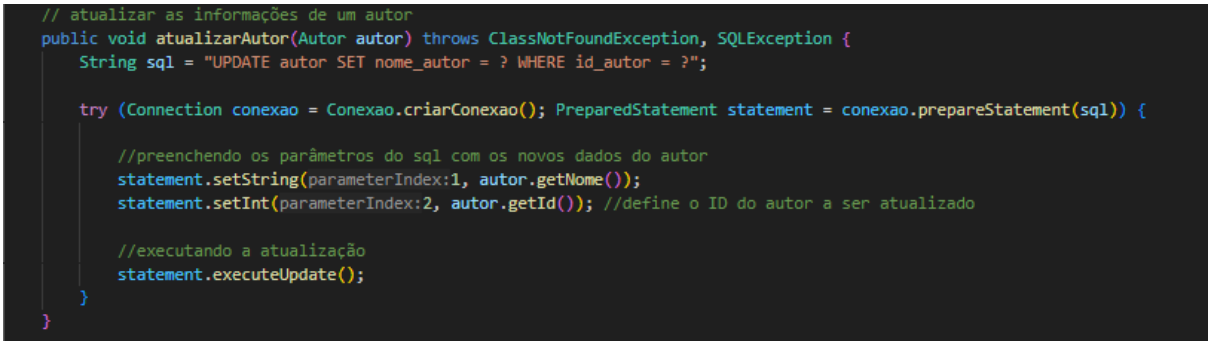
```
if (resultSet.next()) {
                return new Autor(resultSet.getInt("id_autor"),
resultSet.getString("nome_autor"));
} else {
    return null;
}
```

`resultSet.next()`: Verifica se há algum registro no `ResultSet`. Se houver, move o cursor para o próximo registro.

Se o autor for encontrado, um objeto `Autor` é criado utilizando os dados retornados no `ResultSet` (`id_autor` e `nome_autor`), e este objeto é retornado.

Caso contrário, o método retorna `null`, indicando que o autor não foi encontrado no banco de dados.

4. Atualizar as informações de um autor



```java
// atualizar as informações de um autor
public void atualizarAutor(Autor autor) throws ClassNotFoundException, SQLException {
    String sql = "UPDATE autor SET nome_autor = ? WHERE id_autor = ?";

    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {

        //preenchendo os parâmetros do sql com os novos dados do autor
        statement.setString(parameterIndex:1, autor.getNome());
        statement.setInt(parameterIndex:2, autor.getId()); //define o ID do autor a ser atualizado

        //executando a atualização
        statement.executeUpdate();
    }
}
```

4.1 Método para atualizar as informações de um autor:

```
String sql = "UPDATE autor SET nome_autor = ? WHERE id_autor = ?";
```

- A string `sql` define a instrução SQL para atualizar o campo `nome_autor` da tabela `autor`, onde o campo `id_autor` seja igual ao valor fornecido.
- O símbolo `?` é um parâmetro que será substituído pelos valores fornecidos pelo objeto `autor` durante a execução do código.

4.2 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao(); PreparedStatement
statement = conexao.prepareStatement(sql)) { ... }
```

- O código utiliza um bloco `try-with-resources` para garantir que a conexão e o `PreparedStatement` sejam fechados automaticamente quando o bloco for finalizado.
- `Conexao.criarConexao()`: Chama o método responsável por criar e retornar a conexão com o banco de dados.
- `conexao.prepareStatement(sql)`: Cria um `PreparedStatement` a partir da string SQL definida, permitindo a execução de comandos SQL parametrizados.

4.3 Preenchendo os parâmetros do SQL com os novos dados do autor:

```
statement.setString(1, autor.getNome());
statement.setInt(2, autor.getId());
```

- `statement.setString(1, autor.getNome())`: Define o valor do primeiro parâmetro (representado pelo `?`) como o nome do autor, obtido através do método `getNome()` do objeto `autor`.
- `statement.setInt(2, autor.getId())`: Define o valor do segundo parâmetro (representado pelo `?`) como o ID do autor, obtido através do método `getId()` do objeto `autor`.

4.4 Executando a atualização:

```
statement.executeUpdate();
```

- O método `executeUpdate()` executa a query SQL, que neste caso é um `UPDATE`, atualizando o nome do autor no banco de dados para o valor fornecido, onde o `id_autor` corresponde ao valor passado.

5. Listar autores em ordem pelo ID usando o function:

```java
//listar autores em ordem pelo ID usando o function
public void listarAutores() throws ClassNotFoundException, SQLException {
    String sql = "SELECT * FROM autores_ordem()";

    try (Connection conexao = Conexao.criarConexao();
            PreparedStatement statement = conexao.prepareStatement(sql);
            ResultSet resultSet = statement.executeQuery()) {

        System.out.println(x:"ID \t NOME DO AUTOR");
        System.out.println(x:"----------------------");

        while (resultSet.next()) {
            int idAutor = resultSet.getInt(columnLabel:"id_autor");
            String nomeAutor = resultSet.getString(columnLabel:"nome_autor");
            System.out.println(idAutor + "\t" + nomeAutor);
        }
    }
}
```

5.1 Método para listar autores em ordem pelo ID utilizando a função:

```
    String sql = "SELECT * FROM autores_ordem()";
```

- A string `sql` define a instrução SQL para chamar a função `autores_ordem()`, que retorna os autores em ordem crescente de `id_autor`.
- A função `autores_ordem()` é presumivelmente uma função definida no banco de dados que realiza a ordenação dos autores.

5.2 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao();
     PreparedStatement statement = conexao.prepareStatement(sql);
```

```
     ResultSet resultSet = statement.executeQuery()) {
    ...
}
```

- O código utiliza um bloco `try-with-resources` para garantir que a conexão com o banco de dados (`conexao`), o `PreparedStatement` (`statement`) e o `ResultSet` (`resultSet`) sejam fechados automaticamente após o uso.
- `Conexao.criarConexao()`: Chama o método que cria e retorna a conexão com o banco de dados.
- `conexao.prepareStatement(sql)`: Cria um `PreparedStatement` com a instrução SQL definida.
- `statement.executeQuery()`: Executa a query e retorna um `ResultSet` contendo os resultados da consulta.

5.3 Exibindo o cabeçalho da tabela:

```
System.out.println("ID \t NOME DO AUTOR");
System.out.println("-----------------------");
```

- Exibe o cabeçalho da tabela com os títulos das colunas, "ID" e "NOME DO AUTOR", e uma linha de separação.

5.4 Iterando sobre o ResultSet e exibindo os dados:

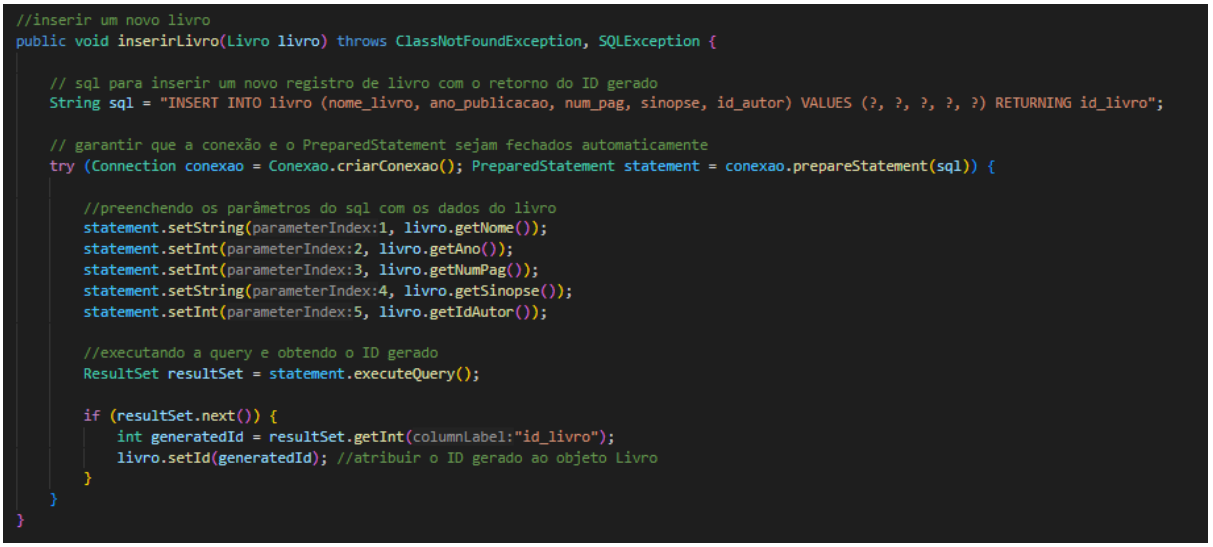
```
while (resultSet.next()) {
    int idAutor = resultSet.getInt("id_autor");
    String nomeAutor = resultSet.getString("nome_autor");
    System.out.println(idAutor + "\t" + nomeAutor);
}
```

- O método `resultSet.next()` verifica se há mais registros no `ResultSet`. Se houver, ele move o cursor para o próximo registro.
- `resultSet.getInt("id_autor")`: Obtém o valor da coluna `id_autor` para o registro atual.
- `resultSet.getString("nome_autor")`: Obtém o valor da coluna `nome_autor` para o registro atual.
- Os dados do autor (ID e nome) são então exibidos no formato "ID \t NOME DO AUTOR".

5.5 Fechamento automático:

Ao final do bloco `try-with-resources`, a conexão com o banco de dados (`conexao`), o `PreparedStatement` (`statement`) e o `ResultSet` (`resultSet`) são fechados automaticamente, garantindo a liberação de recursos.

7. CLASSE: LIVRODB



```java
//inserir um novo livro
public void inserirLivro(Livro livro) throws ClassNotFoundException, SQLException {

    // sql para inserir um novo registro de livro com o retorno do ID gerado
    String sql = "INSERT INTO livro (nome_livro, ano_publicacao, num_pag, sinopse, id_autor) VALUES (?, ?, ?, ?, ?) RETURNING id_livro";

    // garantir que a conexão e o PreparedStatement sejam fechados automaticamente
    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {

        //preenchendo os parâmetros do sql com os dados do livro
        statement.setString(parameterIndex:1, livro.getNome());
        statement.setInt(parameterIndex:2, livro.getAno());
        statement.setInt(parameterIndex:3, livro.getNumPag());
        statement.setString(parameterIndex:4, livro.getSinopse());
        statement.setInt(parameterIndex:5, livro.getIdAutor());

        //executando a query e obtendo o ID gerado
        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {
            int generatedId = resultSet.getInt(columnLabel:"id_livro");
            livro.setId(generatedId); //atribuir o ID gerado ao objeto Livro
        }
    }
}
```

1.1 Método para inserir um novo livro com retorno do ID gerado:

```
String sql = "INSERT INTO livro (nome_livro, ano_publicacao, num_pag, sinopse, id_autor) VALUES (?, ?, ?, ?, ?) RETURNING id_livro";
```

- A string `sql` define a instrução SQL para inserir um novo registro na tabela `livro`.
- Os valores para as colunas `nome_livro`, `ano_publicacao`, `num_pag`, `sinopse` e `id_autor` são fornecidos como parâmetros representados por `?`.
- A cláusula `RETURNING id_livro` especifica que o ID gerado automaticamente (chave primária) deve ser retornado após a inserção.

1.2 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) { ... }
```

- O código utiliza um bloco `try-with-resources` para garantir que a conexão com o banco de dados (`conexao`) e o `PreparedStatement` (`statement`) sejam fechados automaticamente após o uso.
- `Conexao.criarConexao()`: Cria e retorna uma conexão com o banco de dados.
- `conexao.prepareStatement(sql)`: Cria um `PreparedStatement` a partir da string SQL definida.

1.3 Preenchendo os parâmetros do SQL com os dados do livro:

```
statement.setString(1, livro.getNome());
statement.setInt(2, livro.getAno());
statement.setInt(3, livro.getNumPag());
statement.setString(4, livro.getSinopse());
statement.setInt(5, livro.getIdAutor());
```

- Os métodos `setX` substituem os parâmetros `?` no SQL pelos valores correspondentes obtidos do objeto `Livro`:

- `setString`: Define o nome do livro.
- `setInt`: Define o ano de publicação, o número de páginas e o ID do autor.
- `setString`: Define a sinopse do livro.

1.4 Executando a query e obtendo o ID gerado:

```
ResultSet resultSet = statement.executeQuery();
```

- O método `executeQuery()` executa a instrução SQL e retorna um `ResultSet` contendo o ID gerado pelo banco de dados.

1.5 Processando o resultado para obter o ID:

```
if (resultSet.next()) {
    int generatedId = resultSet.getInt("id_livro");
    livro.setId(generatedId);
}
```

- `resultSet.next()`: Move o cursor para o próximo registro no `ResultSet`, caso exista.
- `resultSet.getInt("id_livro")`: Obtém o valor do ID gerado a partir da coluna `id_livro`.
- `livro.setId(generatedId)`: Define o ID gerado no objeto `Livro`, permitindo que ele fique acessível para uso posterior.

2. Deletar um livro pelo ID

```
//deletar um livro pelo ID
public void deletarLivro(int idLivro) throws ClassNotFoundException, SQLException {
    String sql = "DELETE FROM livro WHERE id_livro = ?";

    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {
        statement.setInt(parameterIndex:1, idLivro); //definindo o ID do livro a ser deletado
        statement.executeUpdate(); //executando o comando de exclusão
    }
}
```

2.1 Método para deletar um livro pelo ID:

```
String sql = "DELETE FROM livro WHERE id_livro = ?";
```

- A string `sql` define a instrução SQL para excluir um registro da tabela `livro` onde o campo `id_livro` é igual ao valor fornecido.
- O símbolo `?` representa um parâmetro que será preenchido com o valor do `idLivro`.

2.2 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao(); PreparedStatement
statement = conexao.prepareStatement(sql)) { ... }
```

- O bloco `try-with-resources` garante o fechamento automático da conexão e do `PreparedStatement` ao final da execução do bloco.
- `Conexao.criarConexao()`: Cria e retorna uma conexão com o banco de dados.
- `conexao.prepareStatement(sql)`: Cria um `PreparedStatement` utilizando a string SQL definida.

2.3 Definindo o parâmetro e executando o comando de exclusão:

```
statement.setInt(1, idLivro); // definindo o ID do livro a ser deletado
```

- `setInt(1, idLivro)`: Define o valor do primeiro parâmetro (`?`) da query como o ID do livro a ser excluído.
- **`statement.executeUpdate(); // executando o comando de exclusão`**
-
- O método `executeUpdate()` executa o comando SQL, que neste caso é um `DELETE`, removendo o registro do banco de dados.

3. Método para consultar um livro pelo ID:

```java
//consultar livro pelo ID
public Livro consultarLivro(int idLivro) throws ClassNotFoundException, SQLException {
    String sql = "SELECT * FROM livro WHERE id_livro = ?";

    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {
        statement.setInt(parameterIndex:1, idLivro);

        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {
            Livro livro = new Livro(resultSet.getInt(columnLabel:"id_livro"), resultSet.getString(columnLabel:"nome_livro"),
                    resultSet.getInt(columnLabel:"ano_publicacao"), resultSet.getInt(columnLabel:"num_pag"),
                    resultSet.getString(columnLabel:"sinopse"),resultSet.getInt(columnLabel:"id_autor"));

            return livro; //retorna o livro encontrado
        } else {
            return null; //retorna null se o livro não for encontrado
        }
    }
}
```

```
String sql = "SELECT * FROM livro WHERE id_livro = ?";
```

- A instrução SQL busca todos os campos de um registro na tabela `livro` onde o `id_livro` corresponde ao valor fornecido.
- O `?` é um placeholder para o parâmetro que será definido no `PreparedStatement`.

3.1 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) { ... }
```

- Utiliza um bloco `try-with-resources` para gerenciar automaticamente o fechamento da conexão com o banco de dados (`conexao`) e do `PreparedStatement` (`statement`).
- `Conexao.criarConexao()`: Método responsável por estabelecer a conexão com o banco de dados.
- `conexao.prepareStatement(sql)`: Cria um `PreparedStatement` parametrizado com a consulta SQL.

3.2 Definindo o parâmetro do SQL:

```
statement.setInt(1, idLivro);
```

- Define o valor do parâmetro `?` no SQL, substituindo-o pelo valor do ID do livro fornecido.

3.3 Executando a consulta e processando o resultado:

```
ResultSet resultSet = statement.executeQuery();
```

- O método `executeQuery()` executa a consulta SQL e retorna um `ResultSet` contendo os resultados.

3.4 Verificando o resultado e criando o objeto Livro:

```
if (resultSet.next()) {
          Livro livro = new Livro(resultSet.getInt("id_livro"),
resultSet.getString("nome_livro"),
                                  resultSet.getInt("ano_publicacao"),
resultSet.getInt("num_pag"),
                                      resultSet.getString("sinopse"),
resultSet.getInt("id_autor"));
    return livro;
} else {
    return null;
}
```

- `resultSet.next()`: Move o cursor para o próximo registro no `ResultSet`. Retorna `true` se um registro estiver disponível, ou `false` caso contrário.
- Se um registro for encontrado:
  - Cria um novo objeto `Livro` preenchendo os campos com os valores obtidos do `ResultSet`.
  - Retorna o objeto `Livro`.
- Caso nenhum registro seja encontrado, retorna `null`.

4. Método para atualizar um livro:

```java
//atualizar as informações de um livro
public void atualizarLivro(Livro livro) throws ClassNotFoundException, SQLException {
    String sql = "UPDATE livro SET nome_livro = ?, ano_publicacao = ?, num_pag = ?, sinopse = ?, id_autor = ? WHERE id_livro = ?";

    try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) {

        //preenchendo os parâmetros do sql com os novos dados do livro
        statement.setString(parameterIndex:1, livro.getNome());
        statement.setInt(parameterIndex:2, livro.getAno());
        statement.setInt(parameterIndex:3, livro.getNumPag());
        statement.setString(parameterIndex:4, livro.getSinopse());
        statement.setInt(parameterIndex:5, livro.getIdAutor());
        statement.setInt(parameterIndex:6, livro.getId()); //define o ID do livro a ser atualizado

        //executando a atualização
        statement.executeUpdate();
    }
}
```

```
String sql = "UPDATE livro SET nome_livro = ?, ano_publicacao = ?, num_pag = ?, sinopse = ?, id_autor = ? WHERE id_livro = ?";
```

- A instrução SQL atualiza os campos `nome_livro`, `ano_publicacao`, `num_pag`, `sinopse` e `id_autor` na tabela `livro`.
- A cláusula `WHERE id_livro = ?` garante que apenas o registro correspondente ao ID fornecido seja atualizado.
- Os parâmetros são representados por `?`, que serão substituídos no `PreparedStatement`.

4.1 Criação da conexão e do PreparedStatement:

```
try (Connection conexao = Conexao.criarConexao(); PreparedStatement statement = conexao.prepareStatement(sql)) { ... }
```

- Utiliza um bloco `try-with-resources` para garantir o fechamento automático da conexão com o banco de dados (`conexao`) e do `PreparedStatement` (`statement`).
- `Conexao.criarConexao()`: Método responsável por estabelecer a conexão com o banco.
- `conexao.prepareStatement(sql)`: Cria um `PreparedStatement` para a execução da instrução SQL.

4.2 Definindo os parâmetros do SQL:

```
statement.setString(1, livro.getNome());
statement.setInt(2, livro.getAno());
statement.setInt(3, livro.getNumPag());
statement.setString(4, livro.getSinopse());
statement.setInt(5, livro.getIdAutor());
statement.setInt(6, livro.getId());
```

- Define os valores para os parâmetros do SQL:
    - Nome do livro.
    - Ano de publicação.
    - Número de páginas.
    - Sinopse.
    - ID do autor.
    - ID do livro a ser atualizado.

4.3 Executando a atualização:

```
statement.executeUpdate();
```

- O método `executeUpdate()` executa a instrução SQL de atualização no banco de dados.
- Atualiza os dados do livro correspondente ao ID fornecido.

5. Executando a consulta para listar livros e autores:

```java
//listar livros e seus respectivos autores usando o view
public List<String> consultarLivrosComAutores() throws ClassNotFoundException, SQLException {
    String sql = "SELECT * FROM view_livros_autores";
    List<String> livrosComAutores = new ArrayList<>();

    try (Connection conexao = Conexao.criarConexao();
            PreparedStatement statement = conexao.prepareStatement(sql);
            ResultSet resultSet = statement.executeQuery()) {

        System.out.println(x:"TITULO\t NOME DO AUTOR");
        System.out.println(x:"----------------------");

        while (resultSet.next()) {
            String livro = resultSet.getString(columnLabel:"nome_livro");
            String autor = resultSet.getString(columnLabel:"nome_autor");
            livrosComAutores.add(livro + " - " + autor);
        }
    }

    return livrosComAutores;
}
```

```
ResultSet resultSet = statement.executeQuery();
```

- O método `executeQuery()` executa a instrução SQL e retorna um `ResultSet` contendo os dados da view `view_livros_autores`.

5.1 Iterando sobre o resultado e processando os dados:

```
while (resultSet.next()) {
    String livro = resultSet.getString("nome_livro");
    String autor = resultSet.getString("nome_autor");
    livrosComAutores.add(livro + " - " + autor);
```

```
}
```

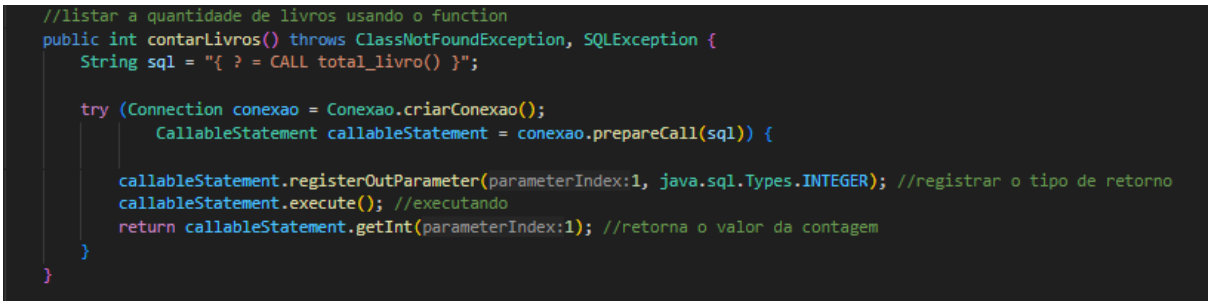
- `resultSet.next()`: Move o cursor para o próximo registro no `ResultSet`.
- Para cada registro:
  - Obtém o título do livro da coluna `nome_livro`.
  - Obtém o nome do autor da coluna `nome_autor`.
  - Adiciona a combinação "livro - autor" à lista `livrosComAutores`.

5.2 Retornando a lista de livros e autores:

```
return livrosComAutores;
```

- Após processar todos os registros, retorna a lista com as combinações de livros e seus respectivos autores.

6. Contar livros usando uma função no banco de dados:



```
//listar a quantidade de livros usando o function
public int contarLivros() throws ClassNotFoundException, SQLException {
    String sql = "{ ? = CALL total_livro() }";

    try (Connection conexao = Conexao.criarConexao();
            CallableStatement callableStatement = conexao.prepareCall(sql)) {

        callableStatement.registerOutParameter(parameterIndex:1, java.sql.Types.INTEGER); //registrar o tipo de retorno
        callableStatement.execute(); //executando
        return callableStatement.getInt(parameterIndex:1); //retorna o valor da contagem
    }
}
```

```
String sql = "{ ? = CALL total_livro() }";
```

- A instrução SQL chama uma função no banco de dados chamada `total_livro`, que retorna a quantidade total de livros.
- O símbolo `?` representa o parâmetro de saída que conterá o resultado da contagem.

6.1 Preparando e registrando o parâmetro de saída:

```
CallableStatement callableStatement = conexao.prepareCall(sql);
callableStatement.registerOutParameter(1, java.sql.Types.INTEGER);
```

`prepareCall(sql)`: Cria um `CallableStatement` para executar a função armazenada.

`registerOutParameter(1, java.sql.Types.INTEGER)`: Define que o primeiro parâmetro retornará um valor do tipo inteiro.

6.2 Executando a função:

```
callableStatement.execute();
```

- Executa a função armazenada no banco de dados e calcula a quantidade de livros.

6.3 Obtendo o resultado da contagem:

```
return callableStatement.getInt(1);
```

- Recupera o valor retornado pela função através do parâmetro de saída registrado.

```
//listar livros em ordem pelo ID usando o function
public void listarLivros() throws ClassNotFoundException, SQLException {
    String sql = "SELECT * FROM livros_ordem()";

    try (Connection conexao = Conexao.criarConexao();
            PreparedStatement statement = conexao.prepareStatement(sql);
            ResultSet resultSet = statement.executeQuery()) {

        System.out.println(x:"ID \t NOME DO LIVRO");
        System.out.println(x:"----------------------");

        while (resultSet.next()) {
            int idLivro = resultSet.getInt(columnLabel:"id_livro");
            String nomeLivro = resultSet.getString(columnLabel:"nome_livro");
            System.out.println(idLivro + "\t" + nomeLivro);
        }
    }
}
```

7.13 Listar livros em ordem pelo ID usando uma função:

```
String sql = "SELECT * FROM livros_ordem()";
```

- A instrução SQL utiliza uma função no banco de dados chamada `livros_ordem`, que retorna os livros ordenados pelo ID.

7.14 Executando a consulta:

```
ResultSet resultSet = statement.executeQuery();
```

- O método `executeQuery()` executa a função e retorna um `ResultSet` com os dados.

7.15 Iterando sobre o resultado e exibindo os livros:

```
while (resultSet.next()) {
    int idLivro = resultSet.getInt("id_livro");
    String nomeLivro = resultSet.getString("nome_livro");
    System.out.println(idLivro + "\t" + nomeLivro);
}
```

- `resultSet.next()`: Move o cursor para o próximo registro.
- Para cada registro:
    - Obtém o ID do livro da coluna `id_livro`.
    - Obtém o nome do livro da coluna `nome_livro`.
    - Exibe os dados formatados.

8. CONCLUSÃO:

Com a conclusão do trabalho orientado pela docente Ângela Peixoto, foi possível compreender a relevância do banco de dados na arquitetura do sistema proposto. A estrutura adotada, baseada em um banco de dados relacional, foi cuidadosamente planejada para atender às necessidades do programa de gerenciador de livros, facilitando o gerenciamento de informações vitais, como os dados dos autores, e as informações dos livros, as interações dos usuários com o sistema e os processos de pesquisa e cadastramento.