Banco de Dados

Profª. Ana Luiza Scharf
ana.scharf@ifsc.edu.br

LABORATÓRIO – Introdução a JPA + Spring Boot - Parte 02

Objetivo da Parte 02

Esta etapa foca em implementar um relacionamento **Um-Para-Muitos (Orientador ↔ Aluno)** bidirecional, garantindo a integridade dos dados e otimizando as entidades com técnicas avançadas do Lombok e do JPA.

Tradução de BCD para Java (Revisão Rápida)

Abaixo está um resumo das principais anotações JPA para traduzir o Modelo Conceitual do Banco de Dados (BCD) para as Entidades Java:

Tabela 1: Mapeamento BCD → Java/JPA

Conceito BCD	Conceito JPA	Anotação Java
Tabela	Entidade	`@Entity` e `@Table`
Chave Primária (PK)	Identificador	`@Id` e `@GeneratedValue`
Chave Estrangeira (FK)	Lado Dono (Many)	`@ManyToOne` e `@JoinColumn`
Relacionamento 1:N	Lado Inverso (One)	`@OneToMany(mappedBy)`
Campo Obrigatório	Restrição Lombok	`@NonNull` (com `@RequiredArgsConstructor`)

Gerenciamento de Relações Bidirecionais

Evitando Loops Infinitos com Lombok

Em um relacionamento bidirecional, se as classes se referenciam mutuamente, chamar `toString()` ou `hashCode()` pode levar a um `StackOverflowError` (loop infinito).

- O Orientador chama o `toString()` do Aluno, que chama o `toString()` do Orientador, e assim por diante.

Para evitar isso, usamos as anotações do Lombok para **excluir** o campo da relação mútua:

Regra de Exclusão de Campos Bidirecionais

Classe	Anotação Lombok	O que Excluir
Orientador (Lado One/Coleção)	`@ToString` e `@EqualsAndHashCode`	O campo da coleção
Aluno (Lado Many/Referência)	`@ToString` e `@EqualsAndHashCode`	O campo da referência

Otimizando Construtores com `@NonNull` e `@RequiredArgsConstructor`

O uso do `@RequiredArgsConstructor` em conjunto com `@NonNull` permite criar um construtor com **apenas** os atributos intrínsecos e obrigatórios da entidade, excluindo o campo de relacionamento.

Caso da Classe `Aluno`

Por que o `orientador` é excluído do construtor?

- **Segurança contra `null` no construtor:** No JPA, o objeto de relacionamento (`Orientador orientador`) geralmente é estabelecido **após** a criação do objeto `Aluno`. Excluir o `orientador` do construtor gerado pelo `@RequiredArgsConstructor` (ao não marcá-lo como `@NonNull`) permite que o `Aluno` seja criado usando apenas seus dados intrínsecos, evitando que o construtor exija um objeto `Orientador` potencialmente nulo.
- **Fluxo de Persistência Ideal:** O fluxo recomendado é: (1) Crie `Aluno` (sem orientador), (2) Busque/Crie `Orientador`, (3) Use `aluno.setOrientador(orientador)` para vincular, e (4) Salve.

Integridade Bidirecional (Método Helper)

Na relação Orientador $\leftrightarrow$ Aluno, o lado `Aluno` é o **dono** da relação (possui a Chave Estrangeira, mapeada com `@ManyToOne`). Para manter a integridade bidirecional, o lado `Orientador` deve ter um método de *Helper*.

Método `public void adicionarAluno(Aluno aluno);` Este método, definido na classe `Orientador`, garante que a atualização ocorra nos dois lados da relação:

1. Atualiza o Lado Inverso (`Orientador`): Adiciona o aluno à lista `this.alunos.add(aluno)`.
2. Atualiza o Lado Dono (`Aluno`): Define a referência de volta, garantindo que a FK seja persistida: `aluno.setOrientador(this)`.

Implementação das Entidades (Java)

Entidade `Orientador` (Lado One/Coleção)

```java
package ifsc.eng.Lab_Ana_BCD;

import jakarta.persistence.*;
import lombok.*;
import java.util.List;
import java.util.ArrayList;

@Entity
@Getter @Setter
@NoArgsConstructor
@RequiredArgsConstructor
@EqualsAndHashCode(exclude = "alunos") // Exclui a lista
@ToString(exclude = "alunos") // Exclui a lista
public class Orientador {

@Id
private String siape;

@NonNull // Incluído no construtor @RequiredArgsConstructor
private String nome;

@NonNull // Incluído no construtor @RequiredArgsConstructor
private String area;

@OneToMany(mappedBy = "orientador",
cascade = CascadeType.ALL, // Deleção em cascata
fetch = FetchType.LAZY)
private List<Aluno> alunos = new ArrayList<>();


// MÉTODO HELPER para manter a integridade da relação Bidirecional
public void adicionarAluno(Aluno aluno) {
this.alunos.add(aluno);
aluno.setOrientador(this); // Atualiza o lado DONO da FK
}
}
```

Entidade `Aluno` (Lado Many/Referência)

```java
package ifsc.eng.Lab_Ana_BCD;


import jakarta.persistence.*;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import java.time.LocalDate;

@Entity
@Getter @Setter
@NoArgsConstructor
@RequiredArgsConstructor // Gera construtor com campos @NonNull
@EqualsAndHashCode(exclude = "orientador") // Exclui a referência
@ToString(exclude = "orientador") // Exclui a referência
public class Aluno {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long matricula;

@NonNull // Incluído no construtor
private String nome;

@NonNull // Incluído no construtor
private Boolean matriculaAtiva;

@NonNull // Incluído no construtor
private LocalDate dataInicio;

// Lado DONO do relacionamento - Mapeia a FK
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "orientador_siape", nullable = true)
// NOTA: 'orientador' não é @NonNull, logo, é excluído do construtor
private Orientador orientador;
}
```

Repositórios e Lógica de Teste

Repositório do Orientador

```java
package ifsc.eng.Lab_Ana_BCD;

import  ifsc.eng.Lab_Ana_BCD.Orientador;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface OrientadorRepository extends CrudRepository<Orientador, String> {
// Tipo do ID é String, conforme definido na entidade Orientador (siape).
}
```

Componente de Teste (`TesteComponent.java`)

A lógica de teste é separada em um componente transacional para garantir que a sessão do Hibernate esteja aberta ao acessar os dados `LAZY`.

```java
package ifsc.eng.Lab_Ana_BCD;

```

```java
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.time.LocalDate;
import java.util.Optional;

@Component
public class TesteComponent {

private final AlunoRepository alunoRepository;
private final OrientadorRepository orientadorRepository;

public TesteComponent(AlunoRepository alunoRepository, OrientadorRepository orientadorRepository) {
this.alunoRepository = alunoRepository;
this.orientadorRepository = orientadorRepository;
}

// @Transactional é CRUCIAL para garantir acesso LAZY
@Transactional
public void rodarTestes() {

// --- 1. CREATE ORIENTADOR ---
System.out.println("--- 1. INSERT: Salvando Orientador ---");
// Construtor usa apenas campos @NonNull (nome, area)
Orientador drPedro = new Orientador("Dr. Pedro Álvares", "Banco de Dados");
drPedro.setSiape("123456");
orientadorRepository.save(drPedro);

// --- 2. CREATE ALUNOS E RELACIONAR ---
System.out.println("\n--- 2. INSERT: Salvando Alunos e relacionando ---");
// Construtor usa apenas campos @NonNull (nome, matriculaAtiva, dataInicio)
Aluno alunoA = new Aluno("Mariana Luz", true, LocalDate.of(2024, 2, 1));
Aluno alunoB = new Aluno("Pedro Rocha", true, LocalDate.of(2023, 8, 15));

// Método HELPER garantindo a relação bidirecional
drPedro.adicionarAluno(alunoA);
drPedro.adicionarAluno(alunoB);

alunoRepository.save(alunoA);
alunoRepository.save(alunoB);

// --- 3. READ: Buscando um aluno e acessando o Orientador ---
Aluno alunoEncontrado = alunoRepository.findById(alunoA.getMatricula()).get();
System.out.println("\nAluno: " + alunoEncontrado.getNome());
System.out.println("Orientador do Aluno: " + alunoEncontrado.getOrientador().getNome()); // Acesso LAZY

// --- 4. READ: Buscando um orientador e acessando a lista de alunos ---
Orientador orientadorEncontrado = orientadorRepository.findById("123456").get();

System.out.println("\nOrientador: " + orientadorEncontrado.getNome());
System.out.println("Alunos sob orientação: ");
orientadorEncontrado.getAlunos().forEach(a ->
System.out.println("- " + a.getNome())); // Acesso LAZY
}
}
```

Classe Principal (`LabAnaBcdApplication.java`)

```java
package ifsc.eng.Lab_Ana_BCD;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class LabAnaBcdApplication {

public static void main(String[] args) {
SpringApplication.run(LabAnaBcdApplication.class, args);
}
```

```
14
15 @Bean
16 // O CommandLineRunner chama o componente de teste.
17 // O @Transactional é aplicado no TesteComponent.
18 public CommandLineRunner brincarComDados(TesteComponent testeComponent) {
19 return args -> {
20 testeComponent.rodarTestes(); // Chama o método transacional
21 };
22 }
23 }
```

Resumo

Este laboratório avançou na introdução dos conceitos de:

- Gerenciamento de relações Bidirecionais no JPA.
- Uso estratégico das exclusões do Lombok (`@ToString` e `@EqualsAndHashCode`).
- Combinação `@NonNull` e `@RequiredArgsConstructor` para criação de construtores otimizados.
- Implementação de métodos *helper* para garantir a integridade da Chave Estrangeira.