Banco de Dados

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

LABORATÓRIO: Relacionamentos Muitos-para-Muitos - Parte 03

Objetivo da Parte 03

Esta etapa visa implementar e diferenciar os dois tipos de relacionamentos Muitos-para-Muitos (N:M) no JPA:

1. **N:M com Atributo Próprio:** Implementado através de uma **Entidade de Junção Explícita** (`Matricula`), usando `@EmbeddedId`. (Aluno ↔ Disciplina).
2. **N:M Simples:** Implementado através do **Mapeamento Automático** do Hibernate com *@ManyToMany*. (Disciplina ↔ Professor).

N:M com Atributo Próprio (Matricula)

O relacionamento Aluno ↔ Disciplina possui o atributo `nota`, exigindo a criação de uma entidade de junção (`Matricula`) e uma Chave Primária Composta.

Chave Composta: `MatriculaId.java`

Define a estrutura da chave primária, baseada nas chaves de `Aluno` e `Disciplina`.

```
package ifsc.eng.Lab_Ana_BCD;

import java.io.Serializable;
import jakarta.persistence.Embeddable;
import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter
@EqualsAndHashCode
@AllArgsConstructor @NoArgsConstructor @ToString
@Embeddable
/**
* Chave Primária Composta para a entidade Matricula,
* baseada nas chaves de Aluno e Disciplina.
*/
public class MatriculaId implements Serializable {

  // A chave primária do Aluno (FK para Aluno)
  private Long alunoId;

  // A chave primária da Disciplina (FK para Disciplina)
  private Long disciplinaId;
}
```

Entidade de Junção: `Matricula.java`

A Limitação do Lombok no Construtor

O Lombok, por si só, não consegue criar a lógica para inicializar o objeto `MatriculaId` a partir das entidades `Aluno` e `Disciplina`. Por isso, um construtor manual é essencial.

Ação Crucial no Construtor Manual

O construtor adicional (manual) em `Matricula` é o que garante a unicidade da matrícula no banco de dados, inicializando a chave composta (`@EmbeddedId`):
`this.id = new MatriculaId(aluno.getMatricula(), disciplina.getId());`

```java
package ifsc.eng.Lab_Ana_BCD;

import jakarta.persistence.*;
import lombok.*;
import java.util.Date;

@Entity
@Getter @Setter
@NoArgsConstructor
@RequiredArgsConstructor
// Excluímos as entidades relacionadas para evitar recursão
@EqualsAndHashCode(exclude = {"aluno", "disciplina"})
@ToString(exclude = {"aluno", "disciplina"})
public class Matricula {

  @EmbeddedId
  private MatriculaId id;

  @NonNull private Double nota; // ATRIBUTO PRÓPRIO

  @NonNull @Temporal(TemporalType.DATE)
  private Date dataMatricula;

  // Mapeamento N:1 com Aluno - Usa @MapsId para mapear o campo da chave composta
  @ManyToOne(fetch = FetchType.LAZY)
  @MapsId("alunoId") // Mapeia o campo 'alunoId' da chave composta para a PK de Aluno
  @JoinColumn(name = "aluno_id")
  private Aluno aluno;

  // Mapeamento N:1 com Disciplina - Usa @MapsId para mapear o campo da chave composta
  @ManyToOne(fetch = FetchType.LAZY)
  @MapsId("disciplinaId")
  @JoinColumn(name = "disciplina_id")
  private Disciplina disciplina;

  // Construtor manual para inicializar a chave composta (ID)
  public Matricula(@NonNull Double nota, @NonNull Date dataMatricula, Aluno aluno, Disciplina disciplina) {
    this.nota = nota;
    this.dataMatricula = dataMatricula;
    this.aluno = aluno;
    this.disciplina = disciplina;

    // Ação crucial que o Lombok NÃO faz:
    // Inicializa o ID composto usando as chaves primárias das entidades relacionadas.
    this.id = new MatriculaId(aluno.getMatricula(), disciplina.getId());
  }
}
```

N:M Simples (`Professor` ↔ `Disciplina`)

O relacionamento não tem atributos próprios, permitindo o uso do mapeamento automático `@ManyToMany`. O Hibernate cria e gerencia a tabela de junção automaticamente.

Entidade `Professor` (Lado Inverso)

```java
package ifsc.eng.Lab_Ana_BCD;


import jakarta.persistence.*;
```

```java
import lombok.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Getter @Setter
@NoArgsConstructor
@RequiredArgsConstructor
@EqualsAndHashCode(exclude = "disciplinas")
@ToString(exclude = "disciplinas")
public class Professor {

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

  @NonNull
  private String nome;

  @NonNull
  private String area;

  // Relação N:M Simples: Mapeado pelo campo 'professores' em Disciplina
  @ManyToMany(mappedBy = "professores")
  private Set<Disciplina> disciplinas = new HashSet<>();
}
```

Entidade `Disciplina` (Lado Dono)

```java
package ifsc.eng.Lab_Ana_BCD;

import jakarta.persistence.*;
import lombok.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Getter @Setter
@NoArgsConstructor
@RequiredArgsConstructor
@EqualsAndHashCode(exclude = {"matriculas", "professores"})
@ToString(exclude = {"matriculas", "professores"})
public class Disciplina {

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

  @NonNull private String nome;
  @NonNull private Integer creditos;

  // --- RELACIONAMENTO N:M (com nota) via Matricula ---
  @OneToMany(mappedBy = "disciplina", cascade = CascadeType.ALL, orphanRemoval = true)
  private Set<Matricula> matriculas = new HashSet<>();

  // --- RELACIONAMENTO N:M (simples) com Professor ---
  @ManyToMany
  @JoinTable(
      name = "disciplina_professor",
      joinColumns = @JoinColumn(name = "disciplina_id"),
      inverseJoinColumns = @JoinColumn(name = "professor_id")
  )
  private Set<Professor> professores = new HashSet<>();

  // Métodos de ajuda
  public void adicionarMatricula(Matricula matricula) {
    this.matriculas.add(matricula);
    matricula.setDisciplina(this);
  }

```

```
  public void adicionarProfessor(Professor professor) {
    this.professores.add(professor);
  }
}
```

Configuração e Lógica de Teste

Repositórios

```
// AlunoRepository
@Repository
public interface AlunoRepository extends CrudRepository<Aluno, Long> {}

// DisciplinaRepository
public interface DisciplinaRepository extends CrudRepository<Disciplina, Long> {}

// ProfessorRepository
public interface ProfessorRepository extends CrudRepository<Professor, Long> {}

// MatriculaRepository
public interface MatriculaRepository extends CrudRepository<Matricula, Long> {}
```

Componente de Teste (`TesteComponent.java`)

A transação (`@Transactional`) é vital para a criação e consulta das relações N:M, especialmente para garantir que os IDs sejam gerados antes de construir o `MatriculaId`.

```
package ifsc.eng.Lab_Ana_BCD;

import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import lombok.RequiredArgsConstructor;
import java.time.LocalDate;
import java.util.Date;
import java.util.List;

@Component
@RequiredArgsConstructor
public class TesteComponent {

  private final AlunoRepository alunoRepository;
  private final DisciplinaRepository disciplinaRepository;
  private final ProfessorRepository professorRepository;
  private final MatriculaRepository matriculaRepository;

  @Transactional // ESSENCIAL: Garante que as consultas LAZY funcionem
  public void rodarTestes() {

    // I. CRIAÇÃO DE ENTIDADES BÁSICAS (SALVANDO PARA GERAR IDS)
    Aluno alunoA = alunoRepository.save(new Aluno("Mariana Luz", true, LocalDate.of(2024, 2, 1)));
    Aluno alunoB = alunoRepository.save(new Aluno("Pedro Rocha", true, LocalDate.of(2023, 8, 15)));
    Professor profPedro = professorRepository.save(new Professor("Dr. Pedro Alvares", "Banco de Dados"));
    Professor profAna = professorRepository.save(new Professor("Dra. Ana Costa", "Programação"));
    Disciplina discBD = disciplinaRepository.save(new Disciplina("Banco de Dados II", 4));
    Disciplina discPOO = disciplinaRepository.save(new Disciplina("Programação Orientada a Objetos", 6));

    // ------------------------------------------------
    // II. TESTE N:M SIMPLES (DISCIPLINA <=> PROFESSOR)
    // ------------------------------------------------
    discBD.adicionarProfessor(profPedro);
    discPOO.adicionarProfessor(profAna);
    discPOO.adicionarProfessor(profPedro);
    disciplinaRepository.save(discBD);
    disciplinaRepository.save(discPOO);

    // CONSULTA: Acessar a lista de Professores
    System.out.println("--- 1. TESTE N:M SIMPLES ---");
```

```
41      Disciplina dPOO = disciplinaRepository.findById(discPOO.getId()).get();
42      System.out.println("Disciplina: " + dPOO.getNome());
43      System.out.print("Lecionada por: ");
44      dPOO.getProfessores().forEach(p -> System.out.print(p.getNome() + " | "));
45      System.out.println();
46
47      // ------------------------------------------------
48      // III. TESTE N:M COM ATRIBUTO (Matricula)
49      // ------------------------------------------------
50      System.out.println("\n--- 2. TESTE N:M COM ATRIBUTO ---");
51
52      // 1. Criar as Matrículas (o construtor cria o MatriculaId)
53      Matricula m1 = new Matricula(8.5, new Date(), alunoA, discBD);
54      Matricula m2 = new Matricula(9.8, new Date(), alunoA, discPOO);
55      Matricula m3 = new Matricula(6.0, new Date(), alunoB, discBD);
56
57      // 2. Manter a bidirecionalidade na memória
58      alunoA.adicionarMatricula(m1);
59      alunoA.adicionarMatricula(m2);
60      alunoB.adicionarMatricula(m3);
61
62      // 3. Salvar as Matrículas
63      matriculaRepository.saveAll(List.of(m1, m2, m3));
64
65      // 4. CONSULTA: Recarregar Aluno para acessar a coleção de Matrículas
66      Aluno aA = alunoRepository.findById(alunoA.getMatricula()).get();
67      System.out.println("Aluno: " + aA.getNome());
68      System.out.println("Histórico de Notas (Matrículas):");
69
70      aA.getMatriculas().forEach(m ->
71          System.out.println("- Disciplina: " + m.getDisciplina().getNome() + ", Nota: " + m.getNota())
72      );
73    }
74  }
```

Resumo

- Foi estabelecida a diferença entre modelar N:M com entidade de junção explícita (`@EmbeddedId`) e N:M simples (`@ManyToMany`).
- A importância da criação de um construtor manual na entidade de junção foi destacada, superando a limitação de inicialização do Lombok para chaves compostas.