Ao trabalharmos com banco de dados, existem relacionamentos com diferentes tipos de direção: os relacionamentos unidirecionais e os bidirecionais. Os unidirecionais deixam a relação visível apenas em um lado, enquanto relacionamentos bidirecionais permitem que os objetos de ambos os lados acessem e/ou alterem o objeto do outro lado. Isso é muito útil quando você quer ter um controle maior sobre seus objetos e as operações que você pode executar neles, como no caso visto em vídeo.

Vamos dar uma olhada em como podemos definir, configurar e controlar esses relacionamentos.

**Definindo um Relacionamento Bidirecional**

Para definir um relacionamento bidirecional, precisamos ter duas entidades que estão de alguma forma conectadas. Por exemplo, vamos considerar um simples sistema de blog onde temos posts e comentários. Cada post pode ter vários comentários.

@Entity

**public** **class** **Post** {

@Id

@GeneratedValue

**private** Long id;

**private** String title;

**private** String content;

@OneToMany(mappedBy = "post")

**private** List<Comment> comments = **new** **ArrayList**<>();

}

@Entity

**public** **class** **Comment** {

@Id

@GeneratedValue

**private** Long id;

**private** String content;

@ManyToOne

@JoinColumn(name = "post\_id")

**private** Post post;

}

COPIAR CÓDIGO

No exemplo acima, definimos um relacionamento "OneToMany" de Post para Comentários e um relacionamento "ManyToOne" inverso de Comentários para Post. mappedBy = "post" no Post se refere ao campo post na classe Comentário.

Erros comuns ao configurarmos mapeamentos bidirecionais

**Erro 1: Problemas de mapeamento bidirecional**

Quando se tem uma relação bidirecional entre duas entidades, como, por exemplo, uma relação entre os objetos "Aluno" e "Disciplina", em que um aluno pode se matricular em diversas disciplinas e uma disciplina pode ter vários alunos, é necessário atentar para o mapeamento de ambos os lados da relação.

Exemplo prático:

@Entity

**public** **class** **Aluno** {

@OneToMany(mappedBy = "aluno")

**private** List<Disciplina> disciplinas;

}

@Entity

**public** **class** **Disciplina** {

@ManyToOne

**private** Aluno aluno;

}

COPIAR CÓDIGO

Neste exemplo, a entidade "Disciplina" está mapeando para a entidade "Aluno". No entanto, a entidade "Aluno" também precisa mapear de volta para a "Disciplina". A falta desse mapeamento bidirecional é uma causa comum de erros.

Para resolver, inclua o mapeamento no lado "Aluno" da relação:

@Entity

**public** **class** **Aluno** {

@OneToMany(mappedBy = "aluno")

**private** List<Disciplina> disciplinas;

// inclua este método

**public** **void** **addDisciplina**(Disciplina disciplina) {

this.disciplinas.add(disciplina);

disciplina.setAluno(this);

}

}

COPIAR CÓDIGO

**Erro 2: Falha ao escolher o lado de posse corretamente**

Em uma associação bidirecional, um lado é o **proprietário**, e o outro é o **lado invertido**. Na JPA, o lado do proprietário é sempre usado quando se atualiza a relação no banco de dados.

Por exemplo:

@Entity

**public** **class** **Carro** {

@ManyToOne

**private** Dono dono;

}

@Entity

**public** **class** **Dono** {

@OneToMany(mappedBy = "dono")

**private** List<Carro> carros;

}

COPIAR CÓDIGO

"Neste caso, Carro é a entidade proprietária. Se esquecermos de fato de atualizar o lado do proprietário, a JPA não poderá sincronizar corretamente a associação com o banco de dados.

Para corrigir, você deve atualizar sempre o lado proprietário do relacionamento:

**Dono** dono = **new** **Dono**();

**Carro** carro = **new** **Carro**();

carro.setDono(dono); // Carro é a entidade proprietária, então atualizamos este lado

dono.getCarros().add(carro);

entityManager.persist(dono);

COPIAR CÓDIGO

Estes são apenas dois exemplos de erros comuns que podem ocorrer ao configurar os mapeamentos na JPA. Outros erros também podem ocorrer, mas a chave para resolvê-los é entender bem como a JPA funciona e sempre analisar e testar cuidadosamente o seu código