



Certified Tech Developer

The Ultimate Degree

Mapeamento de relacionamento

Modelo DER

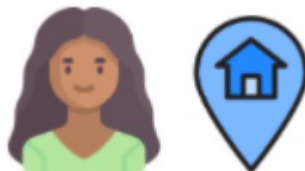
Certamente você se lembra de quando estudou Banco de Dados, que um sistema de gerenciamento de banco de dados relacional administra todas as informações que desejamos guardar em diferentes tabelas, como se fossem planilhas do Excel.

No entanto, uma das diferenças entre um banco de dados relacional e uma planilha do Excel, é que **o banco de dados gerencia os relacionamentos entre as tabelas por meio de chaves**.

Um registro em uma tabela possui uma chave primária que o identifica exclusivamente. Quando uma tabela armazena em uma coluna a chave primária de outra tabela, este campo é conhecido como **chave estrangeira**. Esse fato permite que relações sejam realizadas entre tabelas. Ou seja, o fato de persistir uma chave primária de uma tabela em uma coluna de outra tabela cria um relacionamento entre elas.

Vamos recordar com um exemplo

Qual é a relação entre um usuário e um endereço?



Cada um deles possui uma chave primária que podemos chamar de id. Mas queremos que nossa tabela de usuários (users) armazene a chave primária da tabela de endereços (address). Com essa relação, sabemos qual é o registro da tabela de endereços que está associado a um usuário.

Podemos ver essa relação na imagem:



A imagem representa um relacionamento de 1 para 1 entre duas tabelas.

Mas... como podemos implementar esse e outros relacionamentos no Spring Data JPA com o Hibernate?

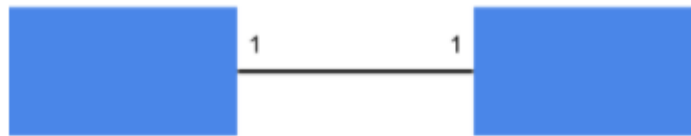
Anotações

Para cada um dos tipos de relacionamentos, temos uma anotação equivalente para JPA, sendo:

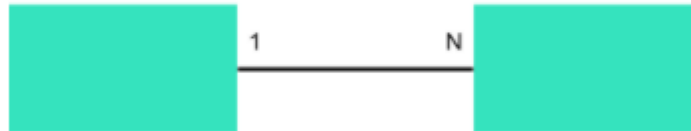
Relação	Anotação
Um para um	@OneToOne
Um para muitos	@OneToMany
Muitos para um	@ManyToOne
Muitos para muitos	@ManyToMany

Chegou o momento de aprender como mapear os seguintes tipos de relações do modelo relacional de banco de dados para o modelo orientado a objetos.

Relação UM para UM



Relação UM para MUITOS



Relação MUITOS para MUITOS



Relação um para um

Nas classes em Java, você aprenderá a mapear, por meio de anotações do Hibernate, um relacionamento 1 para 1 de acordo com a navegabilidade de nosso modelo orientado a objetos.

Relação um para muitos

Nas classes em Java você aprenderá a mapear, por meio de anotações do Hibernate, um relacionamento 1 para N de acordo com a navegabilidade de nosso modelo orientado a objetos.

Relação muitos para muitos

Por fim, você aprenderá a mapear nas classes em Java, por meio de anotações do Hibernate, um relacionamento N para N de acordo com a **navegação** que temos em nosso modelo orientado a objetos.

JoinColumn, Cascade e Fetch type

Em continuidade veremos três tipos de anotações importantes: JoinColumn, Cascade e Fetch type.

@JoinColumn

Em uma relação **@OneToOne** pode ser usada para indicar que uma coluna na entidade pai se refere a uma chave primária na entidade filho. Por exemplo, Office se relacione com Address por meio de uma chave estrangeira, que é o addressId;

```
@Entity
public class Office {
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "addressId")
    private Address address;

}
```

Em uma relação **@OneToMany** usar o atributo “mappedBy” para indicar que a coluna se refere a outra entidade. Por exemplo, Email que é a entidade pai, se une a Employee pelo id.

```
@Entity
public class Employee {
    @Id
    private Long id;


    @OneToMany(fetch = FetchType.LAZY, mappedBy = "employee")
    private List emails;

    @Entity
    public class Email {

        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "employee_id")
        private Employee employee;

    }
}
```

@JoinColumns

Atenção! Foi adicionada a letra S no final. 

Essa anotação pode ser usada nos casos em que temos a intenção de criar joins com várias colunas. Por exemplo, na entidade Office, criaremos duas chaves estrangeiras que apontam para as colunas id e zip na entidade Address.

```
@Entity
public class Office{
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumns({name = "student_id"
        @JoinColumn(name = "ADDR_ID", referencedColumnName = "ID"),
        @JoinColumn(name = "ADDR_ZIP", referencedColumnName = "ZIP")
    })
}
```

```
} )
private Address address;
}
```

Cascading

Na maioria das vezes os relacionamentos entre entidades dependem da existência de outra entidade.

Sem um, o outro não poderia existir e, se modificarmos um, devemos modificar ambos.

Portanto, se executarmos uma ação em uma entidade, a mesma ação deve ser aplicada à entidade associada.

Os operações em cascata mais utilizadas são:

- CascadeType.ALL
- CascadeType.PERSIST
- CascadeType.REMOVE
- CascadeType.MERGE

```
@Entity
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String name;
    @OneToMany(mappedBy = "person", cascade = CascadeType.ALL)
    private List
```

```
addresses;
}
```

```
@Entity
public class Address {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String street;
    private int houseNumber;
    private String city;
    private int zipCode;
    @ManyToOne(fetch = FetchType.LAZY)
    private Person person;
}
```

Objeto Literal	Json
CascadeType.ALL	Realiza a propagação de todas as operações.
CascadeType.PERSIST	Propaga as operações de persistência de uma entidade pai para entidades filhos.
CascadeType.MERGE	Propaga as operações de mesclagem de uma entidade pai para entidades filho.
CascadeType.REMOVE	Propaga as operações de exclusão de uma entidade pai para entidades filho.