Intensivão Java Spring

- Crie um projeto para seu currículo

- Descubra o caminho para se tornar um desenvolvedor back end profissional

Conteúdo preparatório:

Relacionamento N-N e ORM

https://devsuperior.com.br

DEVSUPERIOR

Dr. Nelio Alves

Faaaaala dev! Prof. Nelio Alves aqui.

O Intensivão Java Spring está quase começando, e como a área de back end envolve muito a manipulação de banco de dados, neste conteúdo quero repassar um tópico bem bacana com vocês:

- **Relacionamentos N-N**
- **Mapeamento objeto-relacionalo (ORM)**

Organizei aqui neste material de uma forma bem didática e direto ao ponto. Bora!

Relacionamento N-N

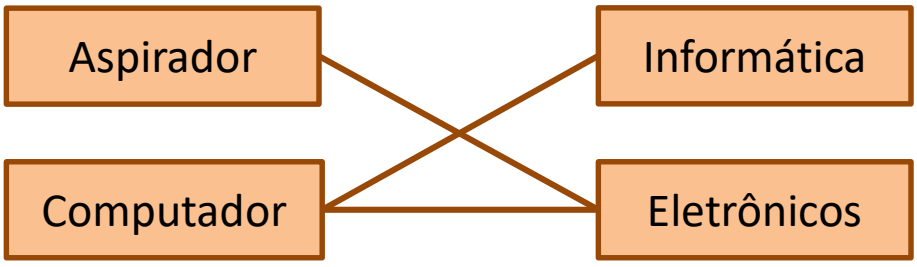
Muitas vezes precisamos armazenar no banco de dados relacional (na forma de tabelas), duas entidades que mantém um relacionamento "muitos para muitos", ou N-N, entre elas. **Por exemplo**:

Relacionamento Produto-Categoria

Suponhamos que no negócio em questão:

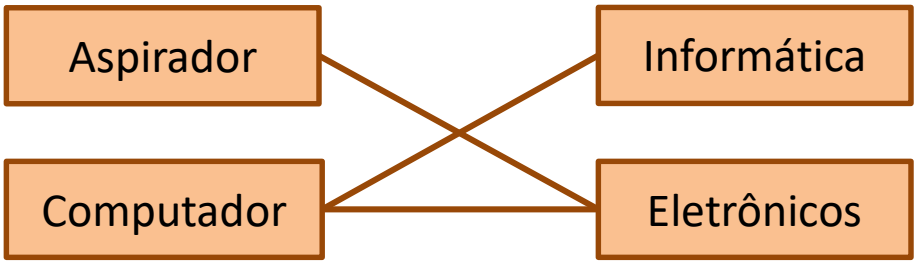
- 1 Produto pode ter **várias** categorias
- 1 Categoria pode ter **vários** produtos

Esta regra de negócio caracteriza uma relação N-N entre produto e categoria. Por exemplo:



Repare que, no exemplo, a categoria "Eletrônicos" tem vários produtos, e o produto "Computador" tem várias categorias.

Relacionamento N-N



Para representar isso em tabelas no banco de dados relacional, nós aprendemos que é preciso criar uma **tabela de associação**, para dizer qual produto está relacionado com qual categoria:

tb_product

id	name	price
1	Aspirador	1500
2	Computador	4500

tb_category

id	description
1	Informática
2	Eletrônicos

tb_product_category

product_id	category_id
1	2
2	1
2	2

Mapeamento Objeto-Relacional (ORM)

Beleza, então agora nosso desafio como programadores é: ler essas informações do banco de dados relacional, que estão na forma de tabelas, e trazer para a memória do computador na forma de objetos:

tb_product

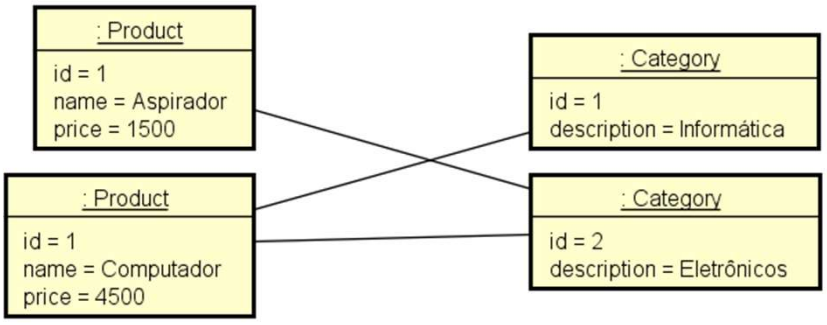
id	name	price
1	Aspirador	1500
2	Computador	4500

tb_category

id	description
1	Informática
2	Eletrônicos

tb_product_category

product_id	category_id
1	2
2	1
2	2



Mapeamento Objeto-Relacional (ORM)

O processo de traduzir do modelo relacional para o modelo de objetos, e vice-versa, se chama **Mapeamento Objeto-Relacional**, ou **ORM**.

Fazer esse processo programaticamente é bem trabalhoso, mas felizmente as linguagens modernas contam com ferramentas para facilitar nossa vida.

No caso da linguagem Java, a ferramenta mais utilizada para automatizar o processo de ORM se chama **JPA** (**Java Persistence API**).

Mapeamento Objeto-Relacional (ORM)

Conforme vamos aprender na prática durante o Intensivão Java Spring, a JPA permite definir as classes de entidades com ***annotations*** especiais para fazer o mapeamento objeto-relacional e, desta forma, instanciar objetos em memória de forma automatizada.

Abaixo temos um trecho de código da classe Java que representa uma categoria, com *annotations* especificando a **tabela**, a **chave primária**, e o **relacionamento** com os produtos:

```java
@Entity
@Table(name = "tb_category")
public class Category {

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

    @ManyToMany(mappedBy = "categories")
    private Set<Product> products = new HashSet<>();
```

Mapeamento Objeto-Relacional (ORM)

Agora temos abaixo um trecho de código da classe Java que representa um produto, com *annotations* especificando a **tabela**, a **chave primária**, o **relacionamento** com as categorias, e o mais legal: a especificação da **tabela de associação**:

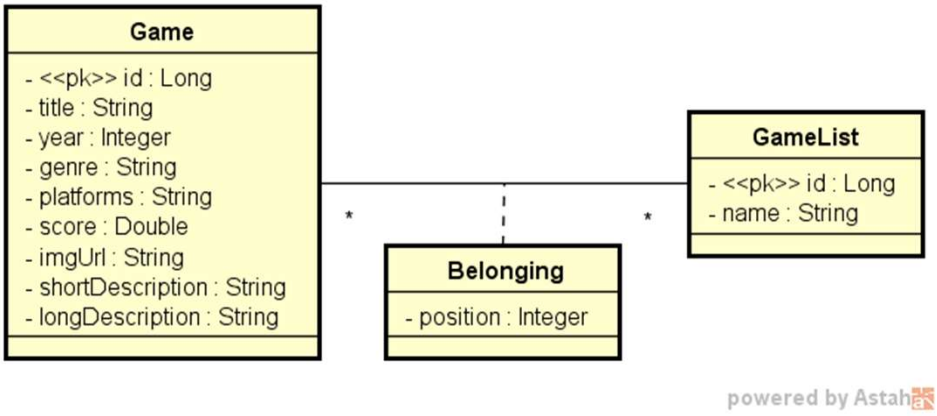
```
@Entity
@Table(name = "tb_product")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Double price;
    private String imgUrl;

    @ManyToMany
    @JoinTable(name = "tb_product_category",
        joinColumns =
            @JoinColumn(name = "product_id"),
        inverseJoinColumns =
            @JoinColumn(name = "category_id"))
    Set<Category> categories = new HashSet<>();
```

Mapeamento Objeto-Relacional (ORM)

Durante o **Intensivão Java Spring**, nós vamos fazer tudo isso passo a passo com um modelo de classes do nosso sistema de listas de jogos:



E repara que vai ter uma outra novidade que ainda não falei aqui: uma **classe de associação** (classe **Belonging**) entre as duas entidades **Game** e **GameList** do relacionamento N-N.

Essa classe de associação é necessária quando há atributos adicionais no relacionamento, como é o caso do atributo "position".

Mas não se preocupe que vamos explicar tudo isso passo a passo no treinamento, ok? Vai ser muito top ☺