Transcrição

Agora que já organizamos o código, podemos continuar com a parte de mapeamentos. Precisaremos fazer uma mudança na entidade `Produto`. Na tabela de "produtos" - no cadastro de produtos - pediram que adicionássemos mais informações.

Então, além do **nome**, do **preço** e da **descrição** do produto, precisamos cadastrar também: a **data**, isto é, quando esse produto foi cadastrado no sistema; e a **categoria**, pois temos algumas categorias de produtos que são vendidos na loja e eles precisam estar registrados.

Como são informações referentes ao produto, na própria entidade `Produto`, vamos adicionar essas novas informações, que são atributos que a JPA vai mapear para colunas no banco de dados. Para a data, podemos utilizar a API de datas do Java 8, então, pode ser um `private LocalDate`, se desejarmos salvar apenas a data, ou o `LocalDateTime` para salvar a data e a hora. No nosso caso, será apenas a data de cadastro, logo, `private LocalDate dataCadastro`.

```
@Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    private LocalDate dataCadastro
```

COPIAR CÓDIGO

Nós podemos instanciar com `LocalDate.now();` (para pegar a data atual). Sempre que um objeto `Produto` for instanciado, automaticamente preencherá o atributo com a data atual, por exemplo. O `LocalDate` é mapeado automaticamente no banco de dados, então, o Hibernate já sabe que ele virará uma coluna do tipo `Date` ou `DateTime` no banco de dados, sem que seja necessário colocar anotação nenhuma.

O outro campo é a categoria, `private Categoria categoria;` . A princípio, nos disseram que, por enquanto, a loja só vende produtos de três categorias: celular, informática e livros. Portanto, é fixo a uma dessas três categorias.

```
@Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    private LocalDate dataCadastro = LocalDate.now();
    private Categoria categoria;
```

COPIAR CÓDIGO

Onde temos `Categoria` , poderia ser uma *String*, mas não é tão interessante, porque podemos passar valores que não são os desejados. Então, podemos criar um *enum* do Java. Vamos apertar "Ctrl + 1", selecionar "Create enum 'Categoria'". Na próxima tela, deixaremos no próprio pacote de modelo, "br.com.alura.loja.modelo" e apertaremos "Finish". Agora, `Categoria` será um *enum*.

Dentro de `Categoria.java` , nós teremos as três constantes - os três valores possíveis - que são: `CELULARES` , `INFORMATICA` , ou `LIVROS` .

```
package br.com.alura.loja.modelo;
```

```
public enum Categoria {

    CELULARES,
        INFORMATICA,
        LIVROS;

}
```

COPIAR CÓDIGO

Na entidade `Produto`, categoria é um *enum*. Porém, temos um detalhe importante. Quando formos mapear um *enum*, temos que tomar cuidado em como o Hibernate e a JPA mapeiam a coluna `Categoria` para o banco de dados. Até então, estávamos usando os tipos primitivos - padrões - do Java, *Long*, *Int*, *String*, *BigDecimal*, *Double*, que são implícitos.

Por exemplo, se for *Long*, ele colocará um número. Se for *String*, ele colocará um *varchar* no banco de dados. A data virará um *Date*. O *BigDecimal* virará um *decimal*. E o *enum*? Como ele fará o relacionamento dessa coluna no banco de dados? Por padrão, se não indicarmos nada, o que ele vai inserir? Vamos fazer um teste e ver como funcionará.

Vamos gerar os métodos "Getters e Setters" da data de cadastro e da categoria. Só para facilitar, vamos criar um construtor. Então, vamos apertar o botão direito, depois selecionar "Source > Generate Constructor using Fields". Na próxima tela, desmarcaremos o "id", a "dataCadastro", e apertaremos "Generate".

Queremos gerar um construtor para facilitar na hora de instanciar um produto. Para ir direto no construtor ao passar o nome, descrição, preço e categoria, ao invés de *setar* tudo isso via método *setter*.

Na nossa classe `CadastroDeProduto` vai dar erro, porque temos agora que passar as informações, não mais via *setter*, mas no construtor: o nome; a descrição; o preço; o id não, porque é gerado automaticamente; a data de cadastro também não, porque já está sendo instanciada no atributo; e a categoria, que, como é um *enum*, nós passamos `categoria.CELULARES`.

```
public static void main(String[] args) {
        Produto celular = new Produto("Xiaomi Redmi", "Muito legal", new BigDecimal("800"), Categor

    }
```

COPIAR CÓDIGO

Mas, o que ele salvará no banco de dados, já que temos um *enum*? Por padrão, se não indicarmos, a JPA não colocará a coluna `Categoria.CELULARES` como um *varchar*, com o texto "CELULARES". Ela vai colocar uma coluna do tipo *Int* (Inteiro), e o valor que ela preenche lá é o valor da posição da constante. Então, `CELULAR` será 1, `INFORMATICA`, 2, e `LIVROS`, 3. Ela faz isso automaticamente.

```
1 CELULARES
2 INFORMATICA
3 LIVROS
```

COPIAR CÓDIGO

Se adicionarmos um produto com a `categoria.CELULARES`, ela mandará para o banco de dados - para a coluna de categoria - o número 1. Se adicionarmos `INFORMATICA`, será o número 2. Se adicionarmos `LIVROS`, será o número 3. Então, se trata da ordem da constante no *enum*. Isso um tanto estranho, porque existe um risco de alguém alterar essa ordem, teremos um desordenamento.

Além disso, se surgir uma nova constante, e alguém, ao invés de adicionar ao final, inserir em cima ou no meio, também embaralhará as ordens, e ele não atualizará sozinho, no banco de dados, as colunas dos registros que já existem. Sendo assim, mapear *enum* pela ordem das constantes é algo arriscado. O ideal é mapear pelo nome da constante.

Significa que não queremos que a coluna seja do tipo inteiro e sim um texto, um *varchar*, e que ele insira a constante `CELULARES`, `INFORMATICA` e `LIVROS`, de maneira independente da ordem declarada na constante. Sendo assim, se alguém

altera a ordem, nada muda no banco de dados.

Para ensinar isso à JPA, que não é mais o padrão, em cima do atributo `Categoria`, na classe `Produto.java`, colocaremos a anotação `@Enumerated()`. E, nessa anotação, temos como opções os parâmetros "ORDINAL" (que é o padrão, a ordem) ou "STRING". Logo, escolheremos `STRING`, para que ele cadastre o nome da constante no banco de dados, não a ordem.

```
@Enumerated(EnumType.STRING)
private Categoria categoria;

    public Produto(String nome, String descricao, BigDecimal preco, Categoria categoria) {
        this.nome = nome;
        this.descricao = descricao;
        this.preco = preco;
        this.categoria = categoria;
    }
```

COPIAR CÓDIGO

Vamos rodar o `CadastroDeProduto`, (Apertando "Run As > 1 Java Aplication"), e olhar o Console. Verificaremos que ele criou a tabela, "Hibernate: create table produtos". Ao analisar as colunas que ele criou, veremos "categoria varchar(255)". Se tivéssemos deixado como "ORDINAL", teríamos um *Int*, e ele mandaria a ordem da constante. Ele também fez o *insert* corretamente. Está pronto mais um mapeamento de um atributo do tipo *enum*.

Nos tipos do próprio Java, *Int, String, Long, Float, Double*, ou nas classes do Java, como a *LocalDate* e a *BigDecimal*, a JPA faz o mapeamento correto automaticamente, ou seja, não precisamos configurar nada. Apenas no caso de *enum* que, se não configurarmos, ele colocará o "ORDINAL" (pela ordem), mas, o ideal é sempre salvar o nome da constante, aí entra o `@Enumerated`.

A aula de hoje foi para discutirmos isso e, na próxima, faremos uma mudança em relação à `Categoria` para transformá-la em uma entidade e deixar o cadastro mais flexível e teremos que discutir um pouco sobre mapeamento de relacionamentos.
Vejo vocês lá!! Abraços!!