04

Relacionamentos many-to-many

Transcrição

[00:00] Olá, pessoal! Continuando, agora nós precisamos relacionar o pedido com o produto. Aqui nós temos o mapeamento já da classe do `Cliente`, da entidade `Cliente`. A de `Pedido`.

[00:11] E `Pedido` está relacionado com `Cliente`, um *many to one*, todo pedido pertence a um cliente, um cliente pode estar vinculado a mais de um pedido. E agora nós precisamos relacionar o pedido com o produto, em um pedido eu terei vários produtos. Se formos analisar, pensando no banco de dados, precisaremos ter essa tabela aqui.

[00:30] Uma nova tabela, que é justamente a tabela que faz essa junção entre um pedido e um produto. É aquela famosa tabela de *join*, de relacionamento de muitos para muitos, porque um pedido pode ter vários produtos, mas um mesmo produto pode estar presente em vários pedidos, então é um relacionamento muitos para muitos.

[00:49] Se fosse um relacionamento muito para muitos simples, conforme está demonstrado nesse diagrama, nós teríamos essa tabela "itens_pedido", essa tabela seria apenas a tabela de *join*, então teria apenas o ID do pedido e o ID do produto. Ela seria populada, por exemplo, o pedido 1 está vinculado com o produto 1, o pedido 1 está vinculado com o produto 2, o pedido 1 está vinculado ao produto 3, e por aí vai.

[01:12] Porém, se fosse fazer esse mapeamento na JPA, seria algo simples, bastaria vir na entidade Pedido, por exemplo, e colocaríamos aqui um `priva`

`List<>` , já que são vários, `private List <Produto>` , que é a nossa entidade Produto. Chamaria aqui de `private List <Produto> produtos;` .

[01:30] Em cima desse atributo, como Produto é uma outra entidade, lembre, precisamos colocar a anotação da cardinalidade, nós colocaríamos `@ManyToMany` . Eu estaria falando: JPA, de Pedido para Produto, é muitos para muitos. A JPA já ia assumir que teria uma tabela de *join*.

[01:52] A JPA tem um padrão de nomenclatura, mas se quiséssemos trocar o padrão dessa tabela, não é o `@Table` , o `@Table` é só em cima de entidade. Nós teríamos que colocar aqui uma outra anotação, que é o `@JoinTable()` . Aqui nós conseguimos personalizar qual é o nome da tabela, quais os nomes das colunas de *join*, o "produto_id", "pedido_id", enfim.

[02:16] Então esse é o mapeamento *many to many* simples, onde a tabela tem apenas as duas colunas, o ID das duas tabelas que ela referencia. Porém, no nosso caso, o relacionamento, ele é *many to many*, ele é muitos para muitos, só que a nossa tabela, ela vai precisar de mais colunas.

[02:33] Por exemplo, eu preciso saber a quantidade do produto. Então, nesse pedido eu estou comprando o produto XPTO, mas em qual quantidade? É uma, são 2, são 5, são 30? Eu preciso saber isso. Outra coisa que eu preciso saber também é qual é o preço do produto.

[02:49] Você pode perguntar, “Mas Rodrigo, já não tem aqui o ID do produto? E na tabela do produto já não tem o preço?” Verdade, só que esse preço pode sofrer um reajuste, futuramente ele pode aumentar, mas na data daquele pedido, ele tinha um determinado valor, em um determinado pedido ele custava 100 reais, hoje foi reajustado, custa 180, mas quando eu fiz o pedido, cada unidade custava 100 reais.

[03:10] Eu preciso ter meio que um histórico do preço do produto naquela venda, naquele pedido. Percebe? Como eu terei mais atributos, mais colunas

mais informações nesse relacionamento, a nossa tabela, ela fica dessa maneira.

[03:25] Ela muda um pouco de figura. Ela terá um ID próprio, e terá que ter um relacionamento com o pedido e com o produto. Eu preciso saber: esse item pedido, de qual produto ele se referencia, e a qual pedido ele se referencia. E tem as outras colunas, no caso, o preço unitário e a quantidade. Por conta disso, o mapeamento não vai ficar dessa maneira.

[03:48] Não será um `@ManyToMany`, vamos mudar um pouco aqui. Precisamos ter uma nova entidade. Sempre que você tiver um relacionamento muitos para muitos que precise de mais colunas, o ideal é você criar uma nova entidade para representar essa tabela de uma maneira mais apropriada. Aqui eu vou apagar. Teremos uma outra entidade, chamada `<ItemPedido>`, que é justamente aqui a ideia, "itens_pedidos".

[04:14] Eu preciso criar essa nova entidade. "Ctrl + 1", "Create Class", no pacote "modelo" mesmo. Vou copiar o `@Entity`, o `@Table`, o mesmo esquema, mapear a entidade agora, você já sabe como funciona. O nome da tabela é `@Table(name = "itens_pedido")`, conforme nos foi passado. Aqui dentro, vamos trazer, vai ter um `@Id`, vou copiar de "Pedido".

[04:43] O ID desta tabela também será gerado automaticamente. O item pedido, ele tem o preço unitário, então `private BigDecimal precoUnitario;`, que é o preço na data da venda do produto. Ele vai ter também o que mais? Uma quantidade, `private int quantidade;`. É um inteiro a quantidade dos produtos.

[05:07] E o relacionamento para produto e para pedido, que nada mais é do que um relacionamento *many to one*. Então, essa tabela "itens_pedido", embora ela seja uma tabela de muitos para muitos, como ela tem essa característica, se formos parar para pensar, ela é muitos para um para produto e muitos para um para pedido.

[05:24] Então nós vamos mapear exatamente dessa forma aqui, na entidade. Deixa eu copiar aqui, de "Pedido", aquele relacionamento. Aliás, em "ItemPedido" são novos relacionamentos: `private Pedido pedido`, vou chamar de `pedido`, e `private Produto produto`. Então a nossa `ItemPedido` está relacionada com Pedido e com um Produto. Apaguei sem querer, "Ctrl + Z".

[05:50] E *many to one*, `@ManyToOne`. Então esse relacionamento, em si, será *many to one*, porque virou uma nova entidade. Se não precisasse, seria um *many to many*. Por padrão, o nome da coluna será "pedido_id", "produto_id", conforme eu desejo aqui. Vamos fazer aquele mesmo esquema, criar o construtor, criar os *getters* e *setters*, conforme já fizemos aqui várias vezes.

[06:14] Botão direito, "Source > Generate Constructor". Nesse caso, quando eu criar um "ItemPedido", eu vou passar o ID, a quantidade, o pedido e o produto. Eu não vou passar o preço unitário, porque o preço unitário eu pego do produto, já que, na hora que eu estou instanciando o produto, eu pego o preço dele, naquele momento.

[06:34] Vou gerar aqui. Lembre que a JPA necessita do construtor *default*. Agora botão direito, "Source > Generate Getters and Setters". Vou gerar os *getters* e *setters*. Na teoria, do jeito que eu estou fazendo aqui no projeto, não precisaria dos *setters*, porque eu já estou recebendo no construtor, mas vou gerar aqui, que é o comum do mercado.

[06:55] Mapeei aqui a minha entidade e agora na entidade "Pedido", ele vai ter um *list* de itens, então vou chamar aqui de `private List<ItemPedido> itens;`, e será um `@OneToMany`. Olha que legal, uma nova anotação, um novo mapeamento. Então o "ItemPedido", ele conhece quem é o pedido, quem é o produto, e o "Pedido" conhece essa lista.

[07:20] Entretanto, daqui para lá é um para muitos, então aqui, no "ItemPedidos", nós usamos essa anotação `@OneToMany`. Porém, aqui tem um detalhe: com isso aqui, eu estou mapeando agora os dois lados do

relacionamento. De "ItemPedido" eu estou mapeando o "Pedido", e de "Pedido" eu estou mapeando o "ItemPedido".

[07:43] Então isso se tornou um relacionamento bidirecional, onde eu estou mapeando os dois lados. Porém, no próximo vídeo, vamos discutir um pouco como funciona essa questão do relacionamento bidirecional e algumas boas práticas para lidarmos com ele. Vejo vocês lá, um abraço.