08

Teste do relacionamento bidirecional

Transcrição

[00:00] Olá, pessoal, vamos continuar? No último vídeo nós entendemos essa questão do relacionamento bidirecional e terminamos de fazer os mapeamentos, já estamos com o nosso mapeamento de pedido para item, de item para pedido, já temos o método utilitário para adicionar um item, está tudo certo, tudo mapeado. Agora podemos testar, simular, ver se tudo vai funcionar conforme o esperado.

[00:20] Nós tínhamos aquela classe "CadastroDeProduto", como vamos cadastrar pedido agora, só para não mexer nesse código que já existia, vou criar uma classe de teste aqui. "Ctrl + N", "class", nesse pacote "testes" mesmo, vou chamá-la de "CadastroDePedido". Vou marcar aqui para ele gerar um método *main*. Criou a classe.

[00:43] Mas aqui tem um problema: para cadastrar um pedido, eu preciso ter um produto, para cadastrar um produto, eu preciso ter uma categoria, tem essa necessidade. Como no meu projeto eu estou com aquela configuração do *hibernate* sempre criar as tabelas do zero, então sempre que rodo a classe *main*, o meu banco está zerado, ele cria todas as tabelas e não tem nenhum registro.

[01:01] Eu vou reaproveitar, vou copiar esse código "cadastrarProduto", que tínhamos feito na classe "CadastroDeProduto", que era um método que criava um produto, criava a categoria, salvava tudo no banco de dados, tudo certo, só para popular os dados. Vou copiar esse código - podíamos até pensar em uma maneira de reaproveitar, mas vai ficar como um desafio tentar organizar esses códigos de teste.

[01:25] Só para não perdermos muito tempo, vou fazer direto na "CadastroDePedido". É aquele mesmo código, eu tenho um método utilitário que cria uma `Categoria`, chamada `CELULARES`, cria um produto `celular` nessa categoria, de preço 800 reais. Cria o `EntityManager`, cria os Daos, manda salvar tudo no banco.

[01:46] Está tudo certo. Aqui, no meu método *main*, eu vou chamar esse método `cadastrarProduto();`. Quando eu rodar o meu *main*, ele já vai criar um produto e uma categoria no banco de dados. Agora eu já posso cadastrar o pedido, simular como vai funcionar esse cadastro de pedido.

[02:01] Vou maximizar o código aqui. Vamos lá. Eu preciso criar um novo pedido. `Pedido pedido = new Pedido()` e vamos relembrar que temos um construtor aqui. Para criar um pedido eu preciso de um cliente. Então vou precisar de um cliente primeiro, `Pedido(cliente);`. Vou passar essa variável `cliente`, só que ela não existe.

[02:24] Vou criar uma variável local aqui `Cliente cliente = new Cliente();`. Na hora de instanciar o Cliente, eu preciso passar o nome dele. Vou passar aqui `Cliente("Rodrigo");`, preciso passar um CPF, `Cliente("Rodrigo", "123456");`. Como não tem validação nem nada no CPF, só para simplificar vou passar "123456".

[02:44] Já tenho o cliente, já tenho um pedido. Porém, o Pedido só tem o cliente, eu preciso adicionar itens. Lembra que na classe "Pedido", não criamos esse método `adicionarItem`? É assim que vamos adicionar itens nesse pedido. Aqui, na linha de baixo, o que eu vou fazer? `pedido.adicionarItem();` e eu tenho que passar um novo item, `(new ItemPedido());`.

[03:08] E tem aquele construtor, que eu recebo qual é a quantidade, qual é o pedido e quem é o produto que esse pedido está relacionado. O produto é esse que eu criei aqui em `cadastrarProduto`. Entretanto, eu vou precisar desse produto, então vou precisar do `EntityManager`, vou precisar de todo aquele código.

[03:24] Eu vou copiar essa linha do EntityManager para o Cadastro de Pedido, vou precisar criar um EntityManager, vou fazer isso aqui.

[03:33] Peguei o EntityManager e agora eu preciso carregar um produto que esteja cadastrado no banco de dados. Como eu criei um produto aqui e o meu banco está zerado, ele será o produto de ID 1. Eu vou recuperar aqui: EntityManager, `em.` - aliás, temos a classe Dao, então posso criar a classe Dao utilizando esse EntityManager e chamar o método, tem o método de buscar por ID, se não me engano.

[03:55] `Produto produto = produtoDao.buscarPorId(1l);` e eu passo o ID 1. Já carreguei aqui o meu produto, acredito que agora eu consigo passar um pedido. Vou jogar essa linha `pedido.adicionarItem` para baixo. Aliás, vou passar essas linhas `EntityManager`, `ProdutoDao` e `Produto produto` para cima.

[04:16] Nelas eu criei e cadastrei o produto, salvei a categoria no banco de dados, recuperei o produto recém-salvo do banco de dados. Agora eu estou criando o cliente, estou criando o pedido, estou adicionando o item.

[04:30] Eu preciso passar a quantidade: eu comprei 10 celulares daquele. Quem é o pedido, `pedido.adicionarItem(new ItemPedido(10, pedido, ));` e quem é o produto, `(10, pedido, produto)`. Está aí o meu pedido e está aqui o meu produto. Adicionei, vamos considerar que esse pedido tem apenas um único item. Adicionei aqui o item e agora eu quero salvar esse pedido no banco de dados.

[04:52] Da mesma maneira que eu tenho a classe `ProdutoDao`, eu vou precisar de um `PedidoDao`. Então vou copiar aquela minha classe `ProdutoDao`, dar um "Ctrl + C" e "Ctrl + V", vou renomear para "PedidoDao". E eu só preciso trocar no código onde está `Produto` vai ser `Pedido`.

[05:21] A princípio eu só vou ter o método para cadastrar um pedido, então vou apagar todos esses outros aqui, que vieram quando eu copiei a classe. Eu só tenho

um método que cadastra um novo pedido. Agora vamos voltar na nossa classe *main* de teste e agora eu só preciso criar um `PedidoDao`.

[05:38] `PedidoDao pedidoDao = new PedidoDao(em);`, passo o mesmo EntityManager `(em);` como parâmetro, que já está aberto. Agora eu tenho que fazer `pedidoDao.cadastrar(pedido);`, passando esse pedido aqui. Será que vai funcionar? Vamos dar uma analisada aqui no código.

[06:01] Chamei o meu método utilitário, que cadastrou um produto e cadastrou uma categoria no banco de dados. Carreguei aqui, instanciei, peguei o EntityManager, criei o ProdutoDao, criei o Produto cadastrado no banco de dados nessa linha aqui, o produto 1. Criei um Cliente, criei um Pedido.

[06:19] Mandei adicionarItem, criei o PedidoDao, mandei cadastrar nesse pedido, que está vinculado com esse item, que é desse pedido, desse produto, desse cliente. Vamos rodar e ver o que vai acontecer. Eu acho que não vai funcionar, vamos ver. Ele fez aqui uns *selects*, será que ele rodou a classe correta?

[06:39] Deixa eu ver, cadastro de pedido, e, a princípio, vamos ver o que ele fez. Ele rodou aqui os *client tables*, ele fez o *insert* na categoria e no produto e depois fez só o *select* do produto, buscou o produto, porque esqueci de iniciar a transação, então lembre que precisa de uma transação.

[06:58] Eu preciso desse código aqui em cima. Tudo isso vai ficar dentro de uma transação. Dei um `em.getTransaction().begin();` e no final vou dar um *commit*. Vou colar a mesma linha, só que é `em.getTransaction().commit();`.

[07:08] Vou rodar de novo, "Run As > Java Application". Vamos ver se agora vai funcionar, eu ainda acho que não vai funcionar. Realmente não funcionou. Deu aquela *exception*.

[07:18] Ele carregou o produto, na hora que ele ia fazer o *insert* na tabela de pedido, ele teve um problema, ele teve aquele mesmo problema do *transient property*

value, porque eu tenho uma entidade que não está salva, não está *managed,* gerenciada, sendo vinculada a uma entidade que estou criando agora, que no caso é o cliente, "Pedido.cliente".

[07:43] Verdade, eu não salvei o cliente no banco de dados, o cliente está transiente, então ele não pode. Precisarei criar também um ClienteDao. Vou copiar e colar o "PedidoDao", vou criar a nossa classe "ClienteDao". Vou abrir o "ClienteDao", só preciso renomear "pedido" por "cliente". "Ctrl + Shift + O", `this.em.persist(cliente);`.

[08:11] Já tenho o "ClienteDao", volta para a nossa classe "CadastroDePedido". Percebe que é um pouco chato, porque eu tenho que criar toda essa infraestrutura, então em um projeto, que estamos fazendo no manual, é chato fazer essa parte de testar. Criei `cadastrarProduto();`. O que eu posso fazer aqui? Tem esse `cadastrarProduto();`, vou renomear esse método, invés de `cadastrarProduto`, vou renomear para `PopularBancoDeDados();`.

[08:45] Porque, na verdade é isso que ele está fazendo, ele está populando o banco de dados, ele está criando um produto, uma categoria. Eu vou criar agora o código que cria um cliente e joga no banco de dados, esse código aqui.

[08:54] Vou extrair esse código, vou jogar para "CadastroDePedido". Acabei de criar um cliente. Vou criar também um `ClienteDao`. `ClienteDao clienteDao = new ClienteDao(em);`, passei o EntityManager. Esse `ClienteDao`, dei o `begin();` na transação, salvei a categoria, salvei o produto, vou salvar o cliente: `ClienteDao.cadastrar(cliente);` passando o meu `cliente`.

[09:27] Agora esse método popula o banco de dados, cria uma categoria, um produto e um cliente. Aqui eu preciso também recuperar esse cliente do banco de dados, `Cliente cliente =` - vou precisar também, além de um `ProdutoDao`, vou precisar de um `ClienteDao`. Vou copiar essa linha do `ClienteDao clienteDao = new ClientDao(em);`, vou dar um "Ctrl + C", vou jogar para cima.

[09:52] Aqui eu não tinha carregado um produto pelo ID? Preciso fazer a mesma coisa: `Cliente cliente = clienteDao.buscarPorId(1l);`. Só que não tem esse método `buscarPorId()` na classe "ClienteDao", tem ela no "ProdutoDao", então vou copiar esse código `buscarPorId`, vou jogar para o meu "ClienteDao", só que não é `(Produto.class, id);`, é `return em.find(Cliente.class, id);`. Vou dar um "Ctrl + Shift + O".

[10:19] Aqui não é `Produto`, é `public Cliente buscarPorId(Long id)`. Volto para o meu "CadastroDePedido". Então eu já tenho o meu cliente certo, já carreguei ele do banco de dados, então estou vinculando um cliente com um pedido que já está salvo no banco de dados. Vou rodar o código de novo, vamos ver se agora vai funcionar.

[10:37] Eu ainda acho que não vai funcionar. Não deu a *exception*, então funcionou, não é? Não sei, vamos ver. Ele fez um *insert* da categoria, fez um *insert* do produto, fez o *insert* do cliente, selecionou o produto do banco de dados, o produto de ID 1, selecionou o cliente de ID 1 e agora é o que me interessa.

[10:59] Fez o *insert* na tabela de pedido. Só que ele fez um *insert* só na tabela de pedido. Só que lembra, faltou ele fazer um *insert* - deixa eu ver se está aberto aqui.

[11:09] Ele fez um *insert* na tabela de pedido, só que ele não fez o *insert* aqui na tabela "itens_pedido". Então ele não salvou o relacionamento, ele não salvou os itens, salvou um pedido solto, sem itens. Está errado, está incompleto. Por que ele não salvou? Porque, de novo, o que eu estou salvando neste teste?

[11:27] Eu estou salvando o pedido, o `pedidoDao.cadastrar(pedido)`, eu não salvei o item pedido. Poxa, Rodrigo, vou ter que criar um `ItemPedidoDao`, vou ter 300 classes Dao no projeto? É, a princípio sim. Só que, na verdade, nesse caso, não precisa, porque o item pedido, ele é vinculado a um pedido, sem um pedido não faz sentido existir o item pedido.

[11:50] Então o item pedido, ele é dependente diretamente do pedido. Nós podemos usar um recurso da JPA. Na classe "Pedido", na nossa entidade, essa `private List<ItemPedido>`, invés de eu ter que salvar o "pedido" no banco de dados e depois salvar separadamente, no banco de dados, o `ItemPedido`, eu posso pedir para a JPA já salvar.

[12:11] JPA, estou salvando um pedido no banco de dados, já salva junto, já faz um *insert* junto na tabela de *join*, na tabela "ItemPedido". Aqui, na anotação `@OneToMany`, além do atributo `(mappedBy = "pedido")`, tem um outro atributo muito importante chamado *cascade*, que é para falarmos para fazer o efeito cascata: tudo o que acontecer com o `"pedido"`, faça também no `ItemPedido`.

[12:33] Mas eu tenho que dizer qual é o tipo de *cascade* que eu quero, se é quanto tiver um *persist*, só quando tiver um *persist* no `pedido` faz no `ItemPedido`? *Remove*? *Merge*? Qual é o tipo de *cascade*? Nesse meu caso, eu vou colocar `(mappedBy = "pedido", cascade = CascadeType.ALL)`.

[12:48] Tudo o que eu fizer no `pedido`, faz também no `ItemPedido`. Isso vale também para a exclusão, se eu excluir um pedido, não faz sentido eu ter um item pedido voando, se eu matei, se eu deletei o pedido, apaga todos os itens pedidos, não faz mais sentido ter um item pedido sem o pedido, um não existe sem o outro. Então eu vou colocar `cascade = CascadeType.ALL`.

[13:09] Um novo recurso que acabamos de aprender. Vamos voltar para a nossa classe de teste, vamos rodar. Agora eu acredito que deva funcionar. Olha só, vamos voltar tudo para o começo. Vamos lá: fez um *insert* na categoria, fez um *insert* no produto, fez um *insert* no cliente, carregou o produto 1, carregou o cliente 1, fez o *insert* no pedido e fez o *insert* no item pedido.

[13:35] Como eu só tenho um único item no pedido, ele só fez um único *insert*. Se eu tivesse, naquela lista, adicionado mais itens, para cada um ele faria um *insert* aqui. Funcionou corretamente. Só tem um problema, que eu acabei de perceber: o

nome da coluna está `precoUnitario`. Está *camel case*, igual o do Java, não está com underline.

[13:56] Na realidade, a JPA, quando vai mapear as colunas do banco de dados, quando é uma coluna de relacionamentos, por exemplo, o "cliente", por padrão, ela já coloca o `_id`. Mas para os atributos que não são de relacionamento, para os atributos que têm o nome composto, igual "valorTotal", ele não coloca o underline para separar, ele coloca o mesmo nome do atributo, então ele coloca *camel case*. Mas eu não quero assim.

[14:26] No meu banco de dados, eu quero separar com underline. Aqui, nós não estamos seguindo a convenção, tem que usar aquela anotação `@Column`. Vou importar do *javax.persistence* e o *name* dessa coluna, `(name = "valor_total")`.

[14:42] Então, se não formos seguir a convenção, temos que configurar. E no "itemPedido", o preço unitário: `@Column(name = "preco_unitario")`, não quero usar *camel case*, eu quero usar *snake case*, que é o underline para separar. Vamos rodar de novo, só para ver se ele fez essa mudança? Vou rodar aqui. Cadê o console? Olha lá: "preco_unitario". E o pedido "valor_total".

[15:12] Só um detalhe, só para ficar certo, igual na nossa tabela, igual está no *slide* de modelagem do banco de dados. Foi um pouco mais chato esse vídeo, vocês viram, como, de novo, o foco do treinamento é JPA, não temos *spring boot*, não temos nenhum *framework* MVC, nenhuma biblioteca. É JPA puro.

[15:32] Então, para testarmos as coisas, fazemos via método *main* e é a parte chata, tem que instanciar essas classes na mão, não dá para fazer ingestão de dependências aqui, não tem controle de transação automático, então para testar é um pouco chato. Mas nós vimos que funcionou, o que nos interessa é a JPA, é a parte dos relacionamentos, mapeamentos, e funcionou tudo corretamente.

[15:53] Com isso nós fechamos aqui os testes do relacionamento *many to many*. Ele inseriu corretamente na tabela de pedido e na tabela de *join*, na tabela de