03

Mapeando novas entidades

Transcrição

[00:00] Olá, pessoal! Vamos começar o nosso segundo treinamento? Eu estou aqui com o Eclipse aberto, você pode usar a IDE de sua preferência, e eu estou com aquele projeto que nós finalizamos no primeiro curso, da JPA.

[00:10] É o mesmo projeto da última aula, o finalizado. Só recapitulando, é uma aplicação Maven, tem o arquivo "pom.xml". Aqui, no "pom.xml", basicamente só tem a dependência do `org.hibernate` e do `com.h2database`, que é o nosso banco de dados em memória.

[00:25] Você pode usar o banco de dados de sua preferência, isso também não vai fazer muita diferença. E aqui, no Eclipse, no "src/main/resources", na pasta "META-INF", o nosso "persistence.xml".

[00:32] Bem simples, só com as configurações básicas da JPA. Tem um único `persistence-unit`, estamos usando apenas um banco de dados.

[00:41] E aqui as configurações da JPA, do JDBC, do banco de dados, URL, *driver* do banco de dados, *login* e senha, e algumas configurações do *hibernate*, o dialeto do H2 para exibir e formatar o SQL toda vez que ele for acessar o banco de dados no console e para gerar as tabelas automaticamente. Bem simples também.

[01:00] De código, temos aqui alguns pacotes. No pacote "modelo", acabamos ficando com duas classes de modelo, duas entidades da JPA. Tem aquela que representa um produto, que está mapeada aqui, para uma tabela chamada de produtos.

[01:12] O produto tem um nome, uma descrição, um preço, uma data de cadastro e uma categoria, um relacionamento `@ManyToOne` com uma categoria. Categoria é outra entidade, que está mapeada pela tabela de categorias.

[01:22] Tem um ID e um nome. De resto, nada de mais aqui, só construtor, *getter* e *setter*. Bem simples. Criamos também as classes "dao", seguindo o padrão *Data Access Object* para isolar o acesso ao banco de dados.

[01:37] Aqui a classe Dao utilizando um EntityManager, tem só um construtor, que recebe o EntityManager para nos preocupar com a infra do EntityManager, e os métodos de persistência: cadastrar, atualizar, remover, utilizando o EntityManager com os métodos específicos e alguns métodos de consulta. Bem tranquilo.

[01:54] Além disso, temos a classe "JPAUtil.Java", a classe utilitária só para isolar a criação do EntityManager e do EntityManagerFactory, bem simples também.

[02:02] Como essa aplicação não é uma aplicação web, não estamos utilizando nenhum *framework*, é JPA puro, estamos fazendo os testes em uma classe utilitária, em uma classe com o método *name*.

[02:13] Então bem simples, só para ficar bem focado na JPA e não ter interferência de *frameworks*, só alguns métodos para criar o EntityManager, instanciar a nossa classe Dao, criar uns objetos, as nossas entidades, preencher os atributos, persistir e fazer algum *system outs* para ver se tudo está funcionando corretamente.

[02:31] Esse foi o projeto que nós finalizamos no primeiro treinamento. Vamos dar continuidade em cima desse projeto. Nesse primeiro vídeo, o que vamos fazer? Vamos aprender, continuaremos fazendo novos mapeamentos. Temos essas duas novas tabelas. Eu coloquei esse diagrama aqui, que é a tabela de "clientes" e de "pedidos". Até então não tínhamos essa necessidade e agora surgiu essa necessidade.

[02:53] Teremos que mapear duas novas tabelas. Nós já sabemos: toda tabela, no banco de dados, é mapeada para entidades no mundo Java, no lado da orientação a objetos. Teremos uma classe "cliente", com ID, nome e CPF, e uma classe "pedido", com ID, data, o ID do cliente, o cliente em si, e o valor total.

[03:12] Bem tranquilo, a princípio nada de novo. E o relacionamento de "pedidos" para "clientes", no caso, *many to one*. Então um pedido está vinculado com um único cliente, mas um cliente pode ter múltiplos pedidos. De "pedidos" para "clientes", *many to one*. Vamos fazer o mapeamento dessas duas entidades.

[03:30] De volta ao Eclipse, no pacote de "modelo", vou criar uma nova classe, "Ctrl + N", "Java Class", "Cliente" será a minha primeira classe. É bem simples, vou até abrir aqui a classe de "Produto.java", vou copiar essas duas anotações, `@Entity` e `@Table`, só para não ter que digitar na mão.

[03:45] Vou colar em "Clientes". Ao invés de `produtos`, o nome da tabela é `@Tabel(name = "clientes")`. Só relembrando que, por padrão, o nome da tabela é o mesmo nome da entidade. No nosso caso, como a tabela é no plural, `clientes`, então tem que colocar o `@Table`.

[03:59] Vou copiar o `@Id` e o `@GeneratedValue` do "Produto", copiar o `private Long id;` e o `private String nome;`, que cliente tem nome. Deixa eu maximizar a tela. Então `Cliente` tem um `private Long id`, um `private String nome` e um CPF. Vou dar um "Ctrl + C", vou dar um "Ctrl + V", `private String cpf;`.

[04:12] Vou fazer aquele mesmo esquema que tínhamos feito nas outras classes, vou gerar aqui um construtor. Vou mandar ele gerar um construtor com todos os atributos menos o ID, o ID é gerado pelo banco. Só que lembre que a JPA exige um construtor *default*, padrão, sem nenhum argumento, então eu vou gerar aqui também.

[04:31] E vou gerar os métodos *getters* e *setters*, "Source > Generate Getters and Setters". No caso, vou gerar com todas as opções, dar um "Ctrl + Shift + F" e está pronto. Aqui, bem simples, um novo mapeamento de uma entidade.

[04:50] Agora vou mapear - já mapeei a tabela de "clientes", agora eu tenho que mapear "pedidos". A "pedidos" têm data, valor total, e um relacionamento com "clientes", então é bem tranquilo também. Vou copiar a classe "Produto". Vou dar um "Ctrl + C" e "Ctrl + V" mesmo, e é a classe de "Pedido".

[05:10] Colei o código. Agora tenho que adaptar. Em `@Table`, é `@Table(name = "pedidos")` - deixa eu só maximizar a tela. `Pedido` eu tenho um `id`, `Pedido` tem uma data, então não é `dataCadastro`. A `data` é a data de criação, quando foi instanciado o pedido, eu já instancio a data atual.

[05:28] Além disso, no `Pedido` eu tenho um `valorTotal` e tenho o que mais? Data, valor total e o relacionamento com o "Cliente". Vou apagar esse `nome`, vou apagar essa `descricao`. E aqui, em `@ManyToOne`, será um relacionamento com `Cliente` e vou chamar de `private Cliente cliente;`, e é `ManyToOne`, então aqui eu não preciso mexer.

[05:46] Está mapeado, agora eu só preciso corrigir aqui, apagar esse código que não vamos usar. Vou gerar aqui o construtor com os atributos, para ficar parecido com o que fizemos nos outros projetos, facilitar na hora de criar um objeto. Mas esse construtor só recebe o valor total - aliás, esse construtor só recebe o cliente.

[06:05] Porque a data será a data atual, o valor total, nós vamos mexer nele posteriormente, quando formos trabalhar com os itens do pedido, então eu só preciso passar quem é o cliente desse pedido. *Getters* e *setters*, posso apagar esses daqui, apagar esses daqui que eu já tinha.

[06:22] Apaguei tudo. Agora vou gerar os *getters* e *setters* dos outros atributos: cliente, data e valor total. "Ctrl + Shift + F", está formatado. Está mapeada a nossa entidade `Pedido`. Tem o ID, valor total, data e cliente.

[06:40] ID, valor total, data e cliente. A coluna "valor_total", ela é separada por underline. No Java, nós usamos o *camel case*. Por padrão, o *hibernate* sabe que quando tem *camel case* é para separar com underline, então não precisamos configurar, esse já é o comportamento padrão.

[06:58] Então duas novas tabelas, duas novas entidades mapeadas, bem simples, bem tranquilo. Mas vem essa questão: como eu vou relacionar o pedido com um produto? Porque no pedido eu tenho o cliente, eu tenho a data, eu tenho o valor total. Esse valor é baseado nos produtos. Quais são os produtos desse pedido?

[07:19] Então precisamos também fazer esse mapeamento do pedido com os produtos. Só que, na realidade, não é um mapeamento direto com a entidade "Produto", será um pouco diferente, porém, isso nós vamos discutir no próximo vídeo. Nós vamos ver como podemos fazer esse mapeamento e aprender outras coisas da JPA. Vejo vocês lá.