02

Consultas com funções de agregação

Transcrição

[00:00] Olá, pessoal! Estamos de volta ao treinamento de JPA. Nesta aula vamos discutir um pouco mais sobre consultas. Estou aqui com aquela que nossa classe "ProdutoDao" aberta e, nessa classe, além dos métodos de persistência para salvar, atualizar e excluir, nesta classe acabamos criando alguns métodos para aprender um pouco sobre consultas com a JPA.

[00:18] Por exemplo, temos o método `buscarPorId`, que simplesmente chama o `em.find()`, quando você quer buscar uma entidade pelo ID, `em.find()`, qual é a entidade que você quer fazer a consulta e o ID, ele vai fazer a consulta pela chave primária. Vimos também como usar o JPQL, essa linguagem que é tipo um SQL orientado a objetos para fazer consultas.

[00:38] Nós vimos como é que faz uma consulta simples, por exemplo, `buscarTodos`, buscar todos os produtos, `SELECT p FROM Produto P`; buscar usando filtros, buscar o produto pelo nome `SELECT p FROM Produto p WHERE p.nome = :nome`, e aprendemos como passar parâmetros para a *query*.

[00:55] Também buscando, filtrando por um atributo de um relacionamento, então simplesmente navegamos pelos relacionamentos `SELECT p FROM Produto p WHERE p.categoria.nome = :nome`. Sai navegando e a JPA gera os *joins* específicos para realizar essa consulta.

[01:10] Por fim, uma consulta em que eu não quero trazer a entidade inteira, não quero carregar todos os atributos, todas as colunas, eu quero trazer uma única coluna, um único atributo, então você não precisa carregar a entidade

inteira e fazer um *get* e o nome do atributo. Você simplesmente pode fazer um `SELECT p.preco FROM Produto p WHERE p.nome`. A JPA sabe que deve trazer a coluna preço, não é para carregar a entidade inteira.

[01:32] Aqui, no caso, seria para buscar o preço de um determinado produto. Então, partindo desse princípio, vamos continuar estudando um pouco sobre consultas. No vídeo de hoje, vamos aprender sobre como fazer consultas de agregação, consultas onde utilizamos aquelas funções para fazer um somatório, um valor mínimo, um valor *max*, uma média, enfim, consultas utilizando funções de agregação.

[01:56] No nosso caso, o que vamos fazer? Vamos na classe "Pedido" e eu quero criar um método para fazer uma consulta relacionada com os pedidos. Imagine que eu vou precisar, em alguma tela, mostrar o valor total que nós já vendemos no site, de todos os pedidos, qual é o valor total. Eu tenho que somar o valor total de todos os pedidos e imprimir em uma tela.

[02:18] Aqui, em "PedidoDao", eu vou fazer um novo método para realizar essa lógica. Vou só quebrar a linha aqui, vamos criar um novo método: `public` - como é o valor total, é um *big decimal*. `public BigDecimal valorTotalVendido()`, vai ser o nome do meu método. Vamos implementar - deixa só eu importar o *big decimal*, "Ctrl + Shift + O", importou.

[02:41] Aqui será parecido com o que já tínhamos feito no "ProdutoDao". Eu preciso de uma `String jpql = "";`. Vou criar aqui, separado, essa *string*. Como é que será essa consulta? Eu preciso fazer um *sum*, fazer um somatório. De novo, lembre, JPQL é tipo um SQL, então, é bem parecido com o que você faria em um SQL puro.

[03:05] `String jpql = "SELECT SUM()";` - não existe a função *sum*, que é a função que faz o somatório? Abre e fecha parênteses. Aqui você também pode chamar essas funções *sum*, *min*, *max*, AVG para média, mais ou menos as funções que existem nos bancos de dados. `SELECT SUM();`, qual é o nome do atributo, da propriedade que você quer somar?

[03:24] No nosso caso, se abrirmos a entidade "Pedido", lembra que na entidade "Pedido" tem um atributo aqui chamado `valorTotal` ? Esse é o atributo que eu quero somar. `SELECT SUM(p.valorTotal);` . Então pega o p, acessa o atributo `valorTotal` , *from* - só colocar aqui maiúsculo, `SELECT SUM(p.valorTotal) FROM;` .

[03:49] Não é obrigatório ter maiúsculo, não lembro se eu comentei isso. Eu, pessoalmente, costumo colocar os comandos SQL em maiúsculo, mas pode colocar em minúsculo que funcionará corretamente, `SELECT SUM(p.valorTotal) FROM Pedido` . A única coisa que você tem que tomar cuidado é que lembra, tem que ser o mesmo nome da entidade.

[04:06] Perceba que aqui eu já não coloquei maiúsculo, eu coloquei maiúscula só a primeira letra. E os nomes dos atributos também, tem que ser exatamente igual ao que está na entidade. `SELECT SUM(p.valorTotal) FROM Pedido p` . Dessa maneira aqui, `SELECT SIM(p.valorTotal) FROM Pedido p` .

[04:20] Daremos um *select* na tabela de pedidos e vai fazer um somatório do valor total. Simples assim. Para fazer a *query*, lembre `return em.createQuery(jpql)` , passa a variável `jpql` , que é a *string*, vírgula, passa o que será devolvido nessa *query*, no caso, não é a entidade "Pedido", é só um *big decimal*, porque é um somatório.

[04:45] Continuando, `(joql, BigDecimal.class).get` - JPQL, o que ele está reclamando aqui? *string* e classe, JPQL - digitei errado aqui em cima, JPQL, estava JQPL. `.getSingleResult();` , no caso é *single result* porque ele vai somar tudo e trazer um único resultado, não é uma lista, então não é o *get result list*.

[05:12] Pronto, simples assim. A princípio, é assim que vai funcionar. Vamos testar isso na nossa classe "CadastroDePedido"? Aqui nós já não tínhamos criado um pedido? Vamos lá, no final, antes de fechar a transação, o `.commit()` , vou jogar para o final. Vou criar aqui uma variável `BigDecimal` .

[05:35] Aliás, depois de *commitar* a transação. *Commitei* a transação, terminei, não vou fazer mais escritas, aqui `BigDecimal totalVendido = pedidoDao.valorTotalVendido();`. Chamei aquele método, vou dar um *system out*, `System,out,println("VALOR TOTAL: " +totalVendido);` e vou concatenar com essa variável `totalVendido`. Vamos ver se ela vai pegar certo. Nesse código, o que nós fizemos?

[06:08] Criamos lá, populamos o banco de dados, criamos um pedido que tem só um item, que é aquele produto celular, com 10 unidades. Eu estou comprando aquele celular, que foi gerado no `popularBancoDeDados`, que o preço é 800. Se eu estou comprando 10, o valor total deveria ser 8 mil. Vamos ver se vai funcionar, "Run As > Java Application". Vamos abrir o console. Ele vai imprimir um milhão de coisas aqui, e retornou nulo.

[06:36] Estranho. Retornou nulo? Não encontrou nada no banco de dados? Será que ele não salvou? Se olharmos aqui em cima, ele fez o *insert* do pedido, fez o *insert* do "itens_pedido" e fez o *select* aqui com o *sum*.

[06:48] Perceba: função de agregação, funções do banco de dados, nós conseguimos fazer *select* normalmente com elas. "select sum(pedido0_.valor_total)", "from pedidos". Está correto. Por que veio um nulo? Veio porque eu cometi um vacilo na última aula, eu esqueci de um detalhe mega importante.

[07:08] Vamos abrir a nossa identidade "Pedido". Lembra que o "Pedido", ele tem um atributo `valorTotal`? Só que eu não preenchi esse atributo em nenhum local, então está nulo, se eu for somar todos os pedidos que estão no banco de dados, vai estar tudo nulo. Isso deveria ser feito aqui, naquele método `adIcionarItem`.

[07:25] Toda vez que eu pego um pedido e adiciono um item, eu *seto* os dois lados do relacionamento bidirecional, mas eu preciso incrementar o valor total. Eu preciso fazer o seguinte, o valor total, eu vou inicializar aqui em cim no `@Column`, como 0, `private BigDecimal valorTotal = BigDecimal.zero;`.

[07:44] Vou inicializar no atributo. Instancie um objeto `Pedido`, o valor total dele começa como 0. Sempre que o método `adicionarItem` for chamado, eu incremento o valor com o valor desse item adicionado. Eu faço o seguinte, `this.valorTotal = this.valorTotal`, pego o valor total atual, que na primeira vez vale 0, `this.valorTotal.add();` e eu adiciono com o valor total deste item `(item.getValor());`.

[08:14] O item não tem um método `.getValor`, vou criar um método `getValor` na classe "ItemPedido". Na classe "ItemPedido" eu preciso fazer esse cálculo, o valor será a quantidade vezes o preço, o preço unitário do pedido. Eu vou criar aqui esse método `public BigDecimal getValor()`. O que eu vou retornar aqui?

[08:34] Seguindo, `return precoUnitario;`. `precoUnitario` é um *mul*, então ele tem um método - perdão, é um *big decimal*, então ele tem um método *multiply*. Multiplicado por quantidade, porém eu não posso passar a quantidade aqui como parâmetro para o método *multiply*. Deixa eu só quebrar a linha aqui.

[08:54] Porque quantidade não é um *big decimal*, é um *int*. Eu vou ter que transformar em um *big decimal*, então `return precoUnitario.multiply(new BigDecimal());`, passa como parâmetro aqui `(new BigDecimal(quantidade));`. Eu só posso fazer cálculo de *big decimal* para *big decimal*.

[09:09] Agora o meu item, a classe "ItemPedido", ela tem um método para devolver o valor, que é o `precoUnitario` vezes a quantidade. Na classe "Pedido", toda vez que eu adicionar um item ao pedido, eu atualizo o `valorTotal` somando o valor total atual com o valor deste item que foi passado como parâmetro.

[09:27] A princípio é isso, vamos rodar a nossa classe "CadastroDePedido" novamente. Agora olha lá: "8000.00". Ele fez o *sum*, então agora ele populou certo, fez o *insert* corretamente. [09:40] Eu esqueci deste detalhe no último vídeo, corrigi agora. Perceba, o nosso *select* aqui, de uma função de agregaçã

funcionou corretamente. Perceba, fazer um *select*, fazer uma consulta com função de agregação, é simples, é igual você faria no SQL, `SELECT SUM` para fazer um somatório, MIN pegar o valor mínimo, MAX pegar o valor máximo, AVG pegar a média.

[10:04] Você passa o nome da função e qual é a coluna, qual é o atributo, na verdade, que ele vai fazer essa função. Bem simples, bem tranquilo, não tem mistério, a JPA suporta funções de agregação. E se eu chamar uma função específica do banco de dados? No curso, eu estou utilizando o H2. Vamos imaginar que existe uma função chamada XPTO, que só existe no H2. O que vai acontecer?

[10:28] Essas funções MIN, MAX, AVG e SUM, são funções que são padrão do SQL, funcionam em qualquer banco de dados, a JPA reconhece e ela já sabe o que fazer. Quando é uma função que a JPA não reconhece, é uma função que é proprietária de um banco de dados, a JPA simplesmente vai pegar essa função e delegar para o banco de dados.

[10:47] Se o banco de dados entender, ele executa. Se o banco de dados não conhecer essa função, ele vai jogar um erro. Vamos rodar aqui, não existe a função XPTO, foi um nome que eu inventei aqui. Vamos rodar novamente aquela nossa classe "CadastroDePedido" e ver o que vai acontecer quando passamos uma função que não existe. Olha lá, o esperado, vem uma *exception* aqui.

[11:09] Ele fala "method XPTO", tal, ele diz que não achou esse método maluco. Ele dá um "QueryException" e ele fala: "method name", o XPTO, ele não achou que diabos é isso, não conseguiu resolver isso no banco de dados. Então, se você usar uma função nativa, cuidado, porque ela é nativa daquele banco, então ela tem que existir naquele banco.

[11:32] E se você trocar de banco de dados, você vai ter um problema, porque você vai ter que alterar essas funções nativas e isso terá que ser feito manualmente aqui. Esse era o objetivo do vídeo de hoje, espero que vocês

tenham gostado. No próximo vídeo continuaremos explorando as partes de consulta com a JPA. Vejo vocês lá, um abraço.