03

Consultas para relatórios

Transcrição

[00:00] Olá, pessoal! Agora que já aprendemos como fazer uma consulta com função de agregação, no vídeo de hoje, vamos continuar tratando sobre consultas, mas agora vamos fazer uma consulta um pouco mais complicada, um pouco mais elaborada. Qual é a ideia?

[00:14] Imagine que nos passaram o trabalho de fazer uma nova funcionalidade no sistema, que é um relatório. É esse relatório que eu coloquei aqui como exemplo. Nós precisamos fazer tipo um relatório de vendas.

[00:26] Nesse relatório, o que ele tem que exibir? Ele tem que exibir todos os produtos, para cada produto qual foi a quantidade vendida, então eu tenho que somar de cada produto quanto que já foi vendido, e também a data da última venda de cada um dos produtos. Aqui teria esse exemplo, eu tenho um celular, foram vendidas 240 unidades dele, a última venda foi no dia 1º de fevereiro de 2021.

[00:48] Tenho outro produto, Playstation 5, foram vendidos 98, a última venda foi dia 9 de fevereiro. O Macbook Pro, foram vendidas 71 unidades e a última venda foi em 10 de setembro de 2020. Seria um relatório de vendas. Aqui, perceba, é uma consulta, um relatório nada mais é do que uma consulta.

[01:07] Só que, nesse relatório, vamos misturar, eu tenho que trazer algumas informações, eu tenho que usar as funções de agregação e ele vai ser um pouco mais chato de lidarmos. Vamos para o código, para vermos como vai funcionar. Perceba que vendas está relacionada com os pedidos, então terei que fazer essa consulta baseada nos pedidos que foram realizados no site.

[01:28] Na classe "PedidoDao" vou criar um outro método. Vamos lá, deixa eu só quebrar a linha aqui para subir. `public ?` algo que eu terei que retornar, já eu digo sobre isso. Vou chamar o método de `public ? relatorioDeVendas()`. Ele não recebe parâmetros. Aqui é que está a primeira coisa é o que eu vou retornar neste método? O que essa consulta me devolve?

[01:54] Se pararmos para pensar, se olharmos o relatório, o esboço do relatório, são só três informações, então eu não vou retornar uma entidade inteira, não vou retornar aqui, por exemplo, a entidade `Pedido`. Eu não quero carregar as informações do pedido, eu quero só três informações. Nós já fizemos isso, não é?

[02:12] Se eu abrir aquela classe "ProdutoDao", nós já vimos como faz para fazer uma consulta que só traz parte das informações de uma entidade. Tem aqui, por exemplo - cadê? O último método.

[02:23] Tem esse método `buscarPrecoDoProdutoComNome`. Ele faz um `SELECT p.preco`. Então eu consigo retornar só um atributo de uma entidade, eu não preciso carregar a entidade inteira. Porém, aqui, no relatório de vendas, eu quero três atributos. E o problema: tem atributos de várias entidades.

[02:40] Por exemplo, o nome do produto é da entidade "Produto", só que a quantidade vendida é um somatório, essa quantidade está no pedido, no "ItemPedido". E a data da última venda está no "Pedido". Perceba, eu tenho três informações, três colunas, só que cada coluna vem de uma tabela distinta. Então eu não posso devolver Pedido.

[03:00] Eu não posso colocar como retorno a entidade Pedido, porque eu preciso de informações de outras entidades, de outras tabelas. Então essa consulta diferirá, já nós veremos o que eu devolvo aqui, então vou deixar essa interrogação como mistério no código. Vamos escrever o restante do código.

[03:19] É uma consulta, preciso montar com a *query*, `String jpql = "";`. Vamos lá, como é que será essa consulta? `= "SELECT"`. Poxa, e agora? Eu preciso retornar o

nome do produto, então `"SELECT produto.nome;"`. Só que eu tenho que retornar várias informações., então vírgula, será que a vírgula funciona? Eu consigo trazer várias informações separadas pela vírgula?

[03:43] Vamos testar. Eu preciso trazer o `"SELECT produto.nome, ";`, eu preciso trazer a quantidade vendida. Isso é um somatório, eu preciso somar a quantidade vendida do produto. Nós já vimos como é que faz um SUM, `"SELECT produto.nome, SUM()";` e qual é a coluna que você quer somar. A quantidade vendida, ela está no item da venda. Aqui vai ser `SUM(item.quantidade),`. O que mais?

[04:05] Deixa eu quebrar a linha aqui para facilitar a visualização do código. O que mais? Mais, vírgula - faltou a vírgula aqui. Então traz para mim o `produto.nome`, o somatório do `item.quantidade` e aqui é a última venda, então eu preciso pegar a data da última venda, da venda mais recente.

[04:26] Eu vou trazer aqui, vou usar outra função de agregação, que é o `MAX()`, para trazer o valor máximo, a maior data, a data mais recente. A data, ela vem do "Pedido", então é `MAX(pedido.data)";`. Lembra que no pedido tem o atributo data, então `(pedido.data)`. Estão aqui as minhas três colunas, que eu quero trazer nessa consulta. E o *from*? De onde eu vou fazer esse *select*?

[04:58] De novo, como é um relatório de vendas, isso vem dos pedidos, então eu posso fazer a busca a partir da tabela de pedidos. Então `FROM Pedido pedido";`. Tem que colocar um *alias* aqui, `pedido`. E agora, como é que eu vou fazer para - porque aqui eu estou fazendo só um *from* pedido, só que no SUM, eu tenho um `item.quantidade`, no *select*, nessa primeira linha, tem um `produto.nome`.

[05:20] E agora? Como eu faço os *joins* aqui? Você pode simplesmente colocar um *join* aqui, não tem problema. Eu vou quebrar a linha aqui, `JOIN` e eu faço o *join* como se fosse um SQL mesmo, um `JOIN pedido.`. `JOIN pedido.`, eu tenho que fazer um *join* com quem? Vamos olhar a nossa entidade "Pedido".

[05:45] A entidade "Pedido" tem - cadê? Ela é relacionada com cliente, cliente não tem nada a ver. Ela está relacionada com `ItemPedido`, que é o atributo `itens`, então eu tenho que fazer um *join* com itens. `JOIN pedido.itens item`. Pronto, já fiz o meu primeiro *join*, esse *join* com `item`, que eu estou utilizando ele aqui, `SUM(item.quantidade)`.

[06:09] Agora eu tenho que fazer um *join* com o produto. O produto, o pedido não conhece o produto. Mas o `ItemPedido` sim, ele está relacionado com o `Produto`. Vamos no nosso "PedidoDao", eu faço mais um *join* aqui. Vou quebrar a linha, `JOIN item.produto produto` e eu dou apelido aqui, `produto`.

[06:32] Pronto, já fiz os dois *joins* aqui, já tenho as três entidades relacionadas com a minha consulta e os três atributos delas sendo carregados. Agora, só tem um problema, como eu estou fazendo uma consulta que traz três informações, só que duas delas são funções de agregação e uma delas, que é o nome do produto, não é função de agregação, eu vou ter que fazer um *group by*.

[06:58] Porque nessa consulta, na verdade, eu estou agrupando, é uma consulta de agrupamento. Eu estou agrupando a quantidade vendida e a última venda pelo produto, então esse produto, agrupado, teve essa quantidade de vendas e a data da última venda foi essa. Eu tenho que fazer um *group by* nome do produto.

[07:14] Vou dar um espaço, vou quebrar a linha, `GROUP BY produto.nome`. Aqui precisa conhecer um pouco de SQL para conseguir se virar aqui nos trinta. O que mais? Esse relatório, nós podemos ordenar. Ele está ordenado, a princípio, pela quantidade vendida. Eu quero primeiro os itens que tiveram mais vendas.

[07:37] Vou fazer aqui um *order by*. Acredito que não tínhamos feito o *order by*, é bem tranquilo, `ORDER BY` e qual é o atributo que eu quero ordenar. No caso, é pedido o `item.quantidade`. `ORDER BY item.quantidade`. Mas aqui será do menor para o maior, no nosso relatório é ao contrário, é do maior para o menor, `ORDER BY item.quantidade DESC`, decrescente, do maior para o menor.

[08:03] A princípio está pronta aqui a nossa *query*. Porém, está faltando aqui o que essa *query* vai retornar. O que essa *query* vai retornar? Vamos continuar o nosso código, `return em.createQuery(jpql, ?)`. Lembra que eu tinha que passar o retorno desse método? Que é aquela interrogação do começo?

[08:32] Eu tenho que retornar alguma coisa, `.getResultList();`, porque essa consulta, cada linha da tabela é um registro que ela vai trazer, então ela vai trazer múltiplos registros, `.getResultList();`. Porém, ficou essa interrogação. Como podemos fazer essa consulta?

[08:50] Tem duas maneiras aqui, que eu vou mostrar. Tem a maneira feia, a não ideal, e tem a bonita, a correta, a mais ideal. Como essa consulta, ela devolve três colunas, ela devolve três colunas, mas cada coluna é de um tipo, o nome é uma *string*, o *sum* é um somatório da quantidade, é um *int* e o *max* é a data, é um * local date*. São três informações também distintas.

[09:18] Esse retorno, nós já sabemos que ele tem que ser um `public List<> relatorioDeVendas()`. É um *list*, porque eu estou chamando o `.getResultList`, ele vai me devolver, cada item da lista é uma linha desta tabela, só que para cada linha eu tenho três informações distintas, cada uma pode ser de um tipo.

[09:35] Vou só importar aqui o *list*, "java.util". Como cada coluna pode ser de um tipo, vou usar um `List<Object>`. Porém, não é um único objeto, são vários, então um *array* de *objects*. `List<Object[]>`. Esse é um retorno que eu posso colocar no `return em.createQuery(jpql, Object[]). Essa consulta, ela me devolve um *array* de objeto.

[09:54] (jpql, Object[].class)`, faltou o` .class`. É um *array* de objeto, é um *array*, é uma lista, cada elemento é uma linha dessa lista, é um registro dessa lista, e cada um deles são três *objects*. O primeiro é uma *string*, o segundo é um *int* e o terceiro é um *local date*, por isso é um *array* de *object*.

[10:19] Você pode pensar que é meio bizarro um *array* de *object*. Por isso que eu falei, esse é o jeito não apropriado, mas funciona. Vamos testar? Vamos na nossa classe "CadastroDePedido". Aqui, depois daquele código que nós fizemos para testar o somatório, eu vou chamar aquele nosso novo método.

[10:37] `pedidoDao.relatorioDeVendas();`. Vou dar um "Ctrl + 1", vou pedir para ele jogar em uma variável local. Vou chamar essa variável de `relatorio`. Ele faz a consulta, ok, e vamos imprimir essa lista. Vou fazer um *for each*, `for(Object[] obj: relatorio)`, vou fazer um *for each* tradicional.

[10:58] Para cada *object* no relatório - vou chamar de `obj` - eu quero dar um `System.out.println(obj[0]);` na posição 0. Como são três colunas que eu estou trazendo, vão ter três posições nesse *array* de objetos, vou colar essa linha e colocar na posição `(obj[1]);` e na posição `(obj[2]);`.

[11:17] Vamos rodar o código e ver o que vai acontecer. Rodei, vou abrir o console, e olha, ele trouxe, ele fez. Deixa eu maximizar o console. Ele fez o *select*.

[11:31] Olha lá, *select*, "produto2_.nome", somatório de itens, quantidade, *max* "pedido0_.data", *from* pedido e fez os *inner joins* aqui.

[11:40] Faz o *join* com "itens_pedido", faz um *join* com produto, faz o *group by* pelo nome e faz o *order by*, de maneira decrescente. E ele trouxe.

[11:49] No caso, naquele código de exemplo, eu só tinha cadastrado um único produto, um único pedido com um único produto. Mas ele trouxe certo, "Xiaomi Redmi", foram vendidas 10 unidades, a última venda foi no dia 9 de fevereiro de 2021. Trouxe corretamente. Porém, ficou esse código meio esquisito aqui, que fica um *array* de *object*.

[12:10] Essa é uma maneira de você fazer esse tipo de consulta, uma consulta de um relatório. Você pode trazer cada informação separadamente, só que, como cada

informação, cada coluna é de um tipo, o retorno é um *array* de objeto. Cada posição do *array* é uma coluna, mas como são várias linhas então é um `List<Object[]>`.

[12:30] Essa é uma maneira de você fazer um relatório na JPQL, na JPA, fazendo uma consulta dessa maneira. Só é feio isso, esse `List<Object[]>`, isso é meio esquisito. No próximo vídeo, vamos aprender como dar uma simplificada nesse