04

Consultas com Criteria API

Transcrição

[00:00] Olá pessoal! Agora que já vimos que esse código fica com os *ifs* duplicados utilizando o JPQL, vamos ver uma solução que a JPA buscou para tentar resolver isso. Na verdade, foi uma solução que já existia no *hibernate*, só que só na versão 2 da JPA que ela entrou para a especificação, utilizando um recurso chamado de *Criteria*.

[00:21] Nada mais é do que uma outra maneira de você realizar consultas, mas não é dessa maneira que fizemos, com o JPQL, é sem usar o JPQL. Vamos ver como ficaria essa versão com o *criteria* ao invés do JPQL. Eu vou duplicar esse método `buscarPorParamentros`, vou trazer para o final. Só vou dar uns "Enters" para subir para o topo.

[00:45] Só que agora esse método vai se chamar `public List<Produto> buscarPorParametrosComCriteria`. É o mesmo esquema, ele recebe o nome, o preço e a data de cadastro, que são parâmetros opcionais. Agora vamos ver como é que fica essa consulta - só vou fechar as chaves aqui do método - com o API de *criteria*.

[01:04] A ideia é: precisaremos fazer o seguinte `em.`. O *entity manager*, ele tem um método chamado `.getCriteriaBuilder()`. Nós precisamos do *builder*, que é a classe que sabe criar um objeto *criteria* para nós. Vou jogar essa linha para uma variável local, vou chamar essa variável de `builder`, `CriteriaBuilder builder = em.getCriteriaBuilder();`.

[01:22] Esse é o primeiro passo: *entity manager*, me dá o *builder* de *criteria*. Ele te devolve um objeto do tipo `CriteriaBuilder`. Na sequência, pedimos para o `builder`, `builder.createQuery()`, crie uma *query* para mim, *builder*. Só que agora não é aquela *typed query*, é uma *criteria query*. Da mesma forma, temos que passar qual é a classe que ele vai trabalhar, se basear, para saber qual é o retorno e se basear na consulta.

[01:54] Esse método, eu vou jogar em uma variável local. Ele devolve um objeto do tipo *criteria query*, que vou chamar aqui de `query`. Aqui vem a diferença, no outro método, nós tínhamos o *typed query*, aqui nós temos o *criteria query*. Só que esse `CriteriaQuery` não é criado pelo *entity manager*, é criado pelo `CriteriaBuilder`. Por isso que tivemos que pegar esse `.getCriteriaBuilder()` no primeiro momento.

[02:17] Agora é em cima dessa variável `query` que nós vamos trabalhar. Como fazemos para dizer de onde ele vai disparar o *select*, o *from*, o *where*, como é que funciona? Esse objeto `query`, ele tem alguns métodos que podemos chamar. `query.from()`, precisamos dizer para ele de onde ele vai disparar o *from*.

[02:35] No caso, é da entidade "Produto", `(Produto.class);`. Eu estou dizendo: *query*, você vai fazer a consulta, o *from* é do `Produto.class`. Se o nosso *select* for igual ao *from*, ou seja, estamos fazendo um `SELECT p FROM Produto p`, então o meu *select* e o meu *from* estão trabalhando com a mesma entidade e você não precisa chamar o *select*.

[02:58] Senão, você precisaria fazer isso `query.select` - está vendo que tem um *select*? Só que, no nosso caso, o *select* seria igual o *from*, então é opcional. Já *setei* o *from*, ele assume que o *select* é igual ao *from*. Esse `.from`, ele devolve um objeto, também vou guardar em uma variável, que é uma classe esquisita, chamada de `Root<Produto>`.

[03:18] Vou chamar essa classe de *from*, `Root<Produto> from = query.from(Produto.class);`. Já tenho o *builder*, já criei a *query*, já disse para a *query* a partir de onde ela vai disparar o *select* e o *from*, que é do `Produto.class`. E agora? Como é que eu faço para filtrar os parâmetros, como é que eu faço o *where*?

[03:32] O que acontece? Agora é que vão entrar aqueles *ifs* e *elses*. Eu vou copiar eles do código anterior. Só que vem a diferença, ao invés de termos duas vezes os *ifs* e *elses* - os *ifs*, na verdade, não temos *elses* - ele vem apenas uma única vez. E vai ser parecido. Vou verificar se os parâmetros foram preenchidos. Só que como era a ideia? Na versão JPQL, tivemos que fazer a gambiarra do `WHERE 1=1` e concatenar os *ands*.

[03:59] Então o tipo de filtro que eu estou fazendo é um *and*. Eu tenho que fazer vários *ands*, vai depender dos parâmetros que foram preenchidos. Então, antes de eu começar os meus *ifs* e *elses*, eu preciso pedir para o *builder*, para ele criar para mim um objeto, que é o objeto que vai fazer os *ands*, que vai grudando esses *ands* conforme entrar nos *ifs*.

[04:19] O *builder*, ele tem um outro método aqui, chamado `.and`, `builder.and()`. Esse método, ele devolve um objeto do tipo `Predicate`. Vou jogar em uma variável local. Olha só, ele retorna uma variável do tipo `Predicate`.

[04:33] Vou chamar aqui de `filtros`, `Predicate filtros = builder.and();`. É em cima dessa variável que vamos trabalhar dentro dos *ifs*, é com essa variável que vamos fazer os filtros. Por enquanto, eu criei um `.and()` vazio, ele não faz nada. Se ele entrar nos *ifs*, eu vou complementando esses *ands*, vou fazendo os nossos filtros.

[04:53] Se entrou nesse primeiro *if*, preencheu o nome, então eu falo: `builder.and()`. Eu vou chamar o `.and` de novo. Só que na primeira vez que eu

passei o *and*, eu não passei parâmetro, ele tava criando o *and* do zero. Agora, no *if*, eu vou falar: *and*, eu quero te substituir. Eu faço: `and = builder.and()`.

[05:29] Eu passo como parâmetro - aliás, não precisa atribuir aqui. É só `builder.and()`. Passo o `(and, )` atual. Ele vai pegar o *and* atual, que o primeiro não tinha nada, mas nos próximos teria, se tivesse entrado nos *ifs* anteriores. *Builder*, faz um novo *and*, considerando esse primeiro *and*, que já estava em `Predicate filtros`. E qual é o tipo de filtro que você quer fazer, que ele vai fazer o *and*?

[05:55] No nosso caso, é esse filtro `p.nome = :nome`. Como eu traduzo isso para o *criteria query*? Eu tenho que usar de novo `builder.` e eu vou dizer qual é o tipo de filtro. No nosso caso, o filtro que eu estou fazendo é um filtro de igualdade, é um igual. Só que tem o `=`, tem o `!=`, tem o `>`, `>=`, `<`, `<=`. Vai depender do tipo de filtro.

[06:23] O *builder*, ele tem vários métodos, vai depender do tipo de filtro. Tem o `.ge`, que seria o maior ou igual, *greater and equal*. Tem o `.gt`, maior, `.equal`, que seria o igual. Vai depender do tipo de filtro que você quer fazer. Nesse `builder.and(and, builder.equal());`, você passa qual é o campo que é para filtrar e qual é o parâmetro para substituir.

[06:43] O parâmetro é o parâmetro do método, o `nome`. O filtro, o campo, como é que eu passo para ele o campo, que seria esse `p.nome` do nosso JPQL? Nós usamos o nosso `(from.get("nome"), nome)` e passamos uma *string* com o nome do atributo. Percebe, é meio esquisito isso, já está começando a complicar um pouco, mas é dessa maneira que nós trabalhamos com o *criteria*.

[07:12] Aqui, em `builder.and(and, builder.equal(from.get("nome", nome));` não é `(and`, é `(filtros`, eu chamei a variável de filtros. É assim que funciona: `Predicate filtros = builder.and();`, crio um `and()` vazio. Entrou no `if`,

`builder`, crie um novo `and`, usando o `and` atual e faça com essa fórmula `builder.equal`.

[07:31] É um filtro de igualdade, no atributo `"nome"` com o parâmetro `nome`. Nos outros *ifs* é a mesma coisa. Vou copiar essa linha, vou colar e vou colar. Colei em algum lugar errado. Colei. Porém, nesse segundo é o parâmetro `("preco")`, no terceiro é `("dataCadastro")`.

[07:49] No segundo, não é `nome`, é `preco`. Muito cuidado nos "Ctrl + C", "Ctrl + V", que sempre acabamos esquecendo de renomear. No terceiro é `dataCadastro`. Preenchi todos os parâmetros. E agora, como eu falo para ele rodar a *query*? Agora eu preciso dizer para ele que é para usar esses `filtros`.

[08:09] Percebe, eu criei os filtros, mas eu não passei para a minha *query*. Eu tenho que dizer que esses `filtros` vão fazer parte do *where*. Então vou pegar a minha `query.where(filtros)`, e eu passo esses `filtros` como parâmetro. Agora ele já sabe que é para pegar esses `filtros` e jogar no `where`.

[08:24] Percebe, essa aqui é a API de *criterias*. Ao invés de usarmos uma *string*, igual o JPQL, e montar toda a consulta via *string*, aqui nós montamos via classes e métodos, então tem métodos. Tem o método `and`, tem o método `equal`, tem o método `from`, tem o método `where`, tem o método `select`.

[08:42] É tipo um orientado a objetos, seria um JPQL orientado a objetos, trazendo para classes e métodos. Mas o problema você já viu, fica um código meio estranho, meio complexo de entender, mas vamos lá. Fiz o `query.where(filtros);`. Agora, como eu disparo a *query*?

[09:00] Vai ser parecido com o que já tínhamos aqui no `return query.getResultList();`. A *query*, nós não tínhamos criado a partir de `em.createQuery`, do *entity manager*? Aqui nós também precisamos fazer isso, `return em.createQuery(query);`, só que olha só, ao invés de passar uma *string* com JPQL, podemos passar a nossa *query*, que é o objeto `CriteriaQuery`.

[09:27] Então eu o `createQuery` recebe uma *string* ou um *criteria query*. `return em.createQuery(query).getResultList();`, crie uma *query* baseado nesse `query`, nesse *criteria query*. No *criteria query* já tem todos os parâmetros, o *from*, o *where*, ele já sabe como vai montar, gerar o JPQL, gerar o SQL.

[09:48] Eu mando ele retornar o `.getResultList` para ele disparar de fato a consulta no banco e trazer os registros. Então essa é a versão utilizando *criteria*, que é essa API, que foi baseada na API de *criterias* do *hibernate*. Mas a do *hibernate* era muito mais simples, essa da JPA ficou um pouco complicada.

[10:04] Perceba, eu só tenho uma única vez os *ifs*, diferente do JPQL, que eu tenho duas vezes os *ifs*. Porém, fica com esse código meio confuso. Será que funciona? Vamos testar. Eu já criei aqui uma classe, chamada "TesteCriteria", bem parecida com aquelas outras classes *main*. Só que no meu `popularBancoDeDados` eu só crio três categorias e três produtos, um produto em cada categoria.

[10:28] Não tem pedido, não tem cliente. Criei o *entity manager*, criei o `ProdutoDao`, agora eu vou chamar `ProdutoDao.buscarPorParamentrosComCriteria()`. Olha só, vou passar só um parâmetro aqui, que é o `nome`, passar só um nome aqui, `("PS5", null, null)`, o produto PS5. Vou deixar nulo no preço e na data de cadastro, então ele não é para fazer um *and* aqui, é para fazer só um *where* com um único parâmetro.

[10:58] Nem preciso jogar em uma variável, só isso aqui já funciona, ele já vai mostrar a *query* aqui no console. Então vamos rodar. Olha lá.

[11:05] "select", etc., "where 1=1". Perceba, eu tinha falado que era uma gambiarra, mas o próprio *hibernate* faz isso. Ele fez um "where 1=1", mas ele não filtrou pelo nome. Vamos dar uma olhada no código, acho que eu esqueci algum detalhe. Deixa eu verificar aqui, `buscarPorParamentosComCriteria`,

passei um nome, `ProdutoDao`, `buscarParamentrosComCriteria`, `if (nome != null && !nome.trim() .isEmpty())`.

[11:34] `builder.and`, passa os próprios filtros. `builder.equal(from.get("nome"), nome)`, `builder.and`, `builder.and`, `query.where(filtros);`, passei os filtros aqui. `.createQuery`, `.getResultList`. A princípio esqueci algum detalhe que não estou percebendo. Deixa eu dar uma olhada mais com calma.

[11:59] Perceba, esse código é meio tenso, meio complicado. `.getCriteriaBuilder`, `.createQuery`, `from`. `from.get`, `builder.equal`. É *equal* mesmo? É *equal*. É *equal*, que é o parâmetro que eu quero filtrar, o parâmetro que veio do método. Nome diferente de nulo, nome não veio vazio, `builder.and`. Esse *and*. Será que eu tenho que substituir o `filtros`?

[12:29] Fazer isso, `filtros = builder.and`. Não lembro agora, eu acredito que sim. Vou colocar isso. Vamos rodar e eu espero que seja isso. Vamos ver. Era isso mesmo, tem que reatribuir a variável `filtros`.

[12:43] Seguindo, "where 1=1", ele mesmo colocou "1=1", então se até o *hibernate* faz isso, não é mais gambiarra, é um padrão. "and produto0_.nome=?". Filtrou o nome. Agora eu vou passar um nome e vou passar a data: `("PS5", null, LocalDate.now());`, pegar uma data atual. Vamos rodar, ver se ele vai colocar o *and* e os dois parâmetros. Olha lá.

[13:10] ".nome=?", ".dataCadastro=?". Vou tirar o nome, que é o primeiro parâmetro, só para ver se vai dar pau. Vou botar nulo, nulo no preço e vou filtrar só pela data: `(null, null, LocalDate.now())`. Vamos rodar.

[13:23] Mesma coisa, "where 1=1" e filtrou só pela ".dataCadastro". Perceba, continuou com o mesmo resultado, continuamos tendo uma *query* que é dinâmica e tem parâmetros opcionais, posso passar nulo. Só que se eu passar nulo, eu não quero buscar registros que estejam nulos no banco de dados, eu quero ignorar o parâmetro.

[13:40] Essa é a versão JPQL, que tínhamos visto até então, tudo com JPQL. E essa é a versão com *criteria*, você pode montar todas as suas *queries* com *criteria* ao invés de JPQL. Tem vantagens e desvantagens. Aqui nós vimos, é mais vantajoso quando temos parâmetros opcionais, que não duplicamos aqueles *ifs*.

[14:00] Porém, a grande desvantagem, esse código é muito complicado, é difícil de ler, de entender. Me esqueci desse detalhe dos filtros, e eu fiquei lendo bastante tempo esse código antes de gravar esse vídeo, vou confessar para vocês, para memorizar ao máximo, até fui bem, eu pensei que ia travar mais.

[14:20] Porque eu não estou acostumado com esse tipo de código. Eu, particularmente, detesto a API de *criterias*, eu não uso para nada. Eu prefiro ter o *if* duplicado mesmo, do que ter que escrever com *criteria*. Eu acho muito complexo o *criteria*, para dar manutenção depois, outro programador ou outra programadora que for ler esse código, que não foi ele ou ela que escreveu, é difícil.

[14:43] Você vai ter que botar uns comentários e toda vez que você for mexer nesse código, você vai ter que relembrar como funciona, pesquisar no Google, *Stack Overflow* para lembrar como funciona a API de *criteria*. Então eu, particularmente, não gosto muito, embora ache interessante ter esse negócio do parâmetro opcional, de não duplicar o *if*.

[15:00] Mas eu prefiro um *if* duplicado e um código mais simples do que um *if* a menos e um código mais complexo. Mas é gosto, você vai pensar bem e escolher qual das consultas você gostou mais. Assim fechamos a aula de hoje, vejo vocês na próxima aula, onde vamos discutir mais recursos da JPA. Um abraço e até lá.