02

Consultas dinâmicas com Criteria API

Transcrição

O velho problema da concatenação de Strings

Em alguns casos precisamos criar consultas com parâmetros mais avançados. Podemos optar por permitir ao usuário escolher alguns filtros para buscar um produto. Para fazer isso, precisamos dinamicamente criar uma query verificando os filtros solicitados e concatenando as condições.

```
String jpql = "select p from Produto p ";
```

Devemos testar, por exemplo, se a categoria do produto é passada ou não. Para isso, faremos um `if`:

```
if (categoriaId != null)
    jpql += "join fetch p.categorias c where c.id = :pCategoria and ";
```

Se a categoria for passada, queremos o produto desta categoria. E o mesmo podemos fazer para os demais filtros da nossa *query*. Certo?!

```
if (lojaId != null)
    jpql += "p.loja.id = :pLoja and ";
if (!nome.isEmpty())
    jpql += "p.nome like :pNome and ";
```

Ou seja, caso a loja e/ou o nome sejam passados queremos adicionar esses filtros na pesquisa.

Então, vamos simular como ficaria a String da nossa query caso haja uma categoria. Já que precisamos concatenar com a parte que consulta na tabela categoria.Teremos:

```
select p from Produto p join fetch p.categorias c where c.nome = 'Tecnologia' and
```

Entretanto, repare que há um pequeno problema: a palavra-chave `and` voando na query. Não podemos retirar ela, uma vez que precisamos verificar também os demais filtros. Porém, receberemos um erro ao executar essa query.

Por esse motivo, muitos programadores colocam, ao final da query, uma condição coringa associada ao `and`. Algo como:

```
jpql += "1=1";
```

No final, nosso código deverá ficar parecido com este:

```
if (categoriaId != null)
    jpql += "join fetch p.categorias c where c.id = :pCategoria and ";
else
    jpql += "where ";

if (lojaId != null)
    jpql += "p.loja.id = :pLoja and ";
if (!nome.isEmpty())
    jpql += "p.nome like :pNome and ";

jpql += "1=1";
```

Vamos imaginar, agora, que queremos buscar todos os produtos sem nenhum filtro. Ou seja, não passaremos por nenhuma das condições. Como ficaria nossa query nesse caso?

Teriamos a query inicial:

```
select p from Produto p join fetch p.categorias c where c.nome = 'Tecnologia' and
```

E como nessa situação não teremos nenhum `if`, pularemos direto para o último `append`. Nossa query ficará algo como:

```
select p from Produto p 1=1
```

O que resultará em outro problema, pois a condição `1=1` está voando na query. Para corrigir isso, podemos colocar um `else` junto ao primeiro `if`. Ele deverá fazer um `append` com um `where`:

```
if (categoriaId != null)
    jpql += "join fetch p.categorias c where c.id = :pCategoria and ";
else
    jpql += "where "
```

Dessa forma, caso não haja alguma categoria, nossa query ficará da seguinte maneira:

```
select p from Produto p where 1=1
```

E, ao final, precisamos novamente verificar os filtros passados para adicionar parâmetros da query:

```
TypedQuery<Produto> query = em.createQuery(jpql, Produto.class);

if (categoriaId != null)
    query.setParameter("pCategoria", categoriaId);
if (lojaId != null)
    query.setParameter("pLoja", lojaId);
if (!nome.isEmpty())
    query.setParameter("pNome", "%" + nome + "%");
```

Abaixo podemos observar o código completo para realizarmos essa busca utilizando concatenação de Strings:

```
String jpql = "select p from Produto p ";


if (categoriaId != null)
    jpql += "join fetch p.categorias c where c.id = :pCategoria and ";
else
    jpql += "where ";

if (lojaId != null)
    jpql += "p.loja.id = :pLoja and ";
if (!nome.isEmpty())
    jpql += "p.nome like :pNome and ";

jpql += "1=1";

TypedQuery<Produto> query = em.createQuery(jpql, Produto.class);

if (categoriaId != null)
    query.setParameter("pCategoria", categoriaId);
if (lojaId != null)
    query.setParameter("pLoja", lojaId);
if (!nome.isEmpty())
    query.setParameter("pNome", "%" + nome + "%");

List<Produto> resultList = query.getResultList();

return resultList;
```

Repare que cada parâmetro exige pelo menos dois `if's` e, assim, toda legibilidade do JPQL acaba perdida. Observe também que a String da *query* está em pedaços, tornando-a difícil de ser lida e mantida. O que queremos, na verdade, é apenas realizar uma consulta dinâmica (ou seja, que haja parâmetros opcionais).

Para isso, a JPA possui um conjunto de classes que nos ajudam nessa tarefa e de uma forma muito mais elegante.