02

Buscas dinâmicas com criteria

Queremos colocar na aplicação um novo relatório de produtos. Esse relatório filtrará os produtos por nome, nome da categoria e preço mínimo, porém essas informações são opcionais.

Quando a o nome é fornecido para a busca, devemos colocar a condição que compara o nome do produto na busca, senão, devemos ignorar essa condição. Faremos o mesmo para a categoria e o preço do produto.

Como essa busca acessa o banco de dados para procurar informações sobre produtos, colocaremos sua implementação dentro da classe `ProdutosDAO` .

```
public class ProdutosDAO
{
    private ISession session;
    // implementação dos outros métodos do DAO.
    public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
                double precoMinimo, string nomeCategoria)
    {

    }
}
```

Dentro do método `BuscaPorNomePrecoMinimoECategoria` , queremos criar uma HQL que busca uma lista de produtos:

```
from Produto p
```

Se o nome que recebemos como parâmetro for diferente de nulo, precisamos colocar uma condição na HQL:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    string hql = "from Produto p ";

    if(nome != null)
    {
        hql += " where p.Nome = :nome ";
    }
}
```

Depois precisamos verificar se o preço mínimo que recebemos é válido. Queremos considerar o preço na query apenas se ele for maior do que zero:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    string hql = "from Produto p ";

    if(nome != null)
    {
```

```
        {
            hql += " where p.Nome = :nome ";
        }
        if(precoMinimo > 0.0)
        {
            hql += " and p.Preco > :precoMinimo ";
        }
    }
```

Mas e se o nome for nulo e o preço maior do que zero? Nessa situação, teremos a seguinte hql:

```
from Produto p and p.Preco > :precoMinimo
```

Acabamos de criar uma HQL inválida! Montar a HQL dinamicamente não é uma tarefa fácil, além disso, depois de construirmos a query, precisamos colocar seus parâmetros, porém se colocarmos um parâmetro que não é utilizado, o NHibernate lança uma exceção. Então precisamos colocar novamente os ifs para chamar o `SetParameter`:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    string hql = "from Produto p ";

    // monta a hql

    IQuery query = session.CreateQuery(hql);

    if(nome != null)
    {
        query.SetParameter("nome", nome);
    }
    if(precoMinimo > 0.0)
    {
        query.SetParameter("precoMinimo", precoMinimo);
    }
    // outros ifs para colocar os parâmetros
}
```

Podemos ver que a HQL não é a melhor opção quando queremos trabalhar com buscas que são definidas dinamicamente.

Buscas dinâmicas com ICriteria

Para resolver o problema de buscas dinâmicas, o NHibernate possui as criterias. A criteria do NHibernate é um jeito alternativo para montarmos as SQLs que serão enviadas para o banco de dados.

Criterias são criadas através do método `CreateCriteria` do `ISession`. Esse método recebe qual é a entidade na qual queremos fazer a busca e devolve um objeto do tipo `NHibernate.ICriteria`:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    ICriteria criteria = session.CreateCriteria<Produto>();
}
```

Para listar os resultados de um `ICriteria`, utiliza-se o método `List` passando qual será o tipo devolvido pela busca. Como acabamos de criar a criteria, se chamarmos o método List, ela devolverá a lista com todos os produtos (o mesmo resultado da query `from Produto`):

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    // from Produto p
    ICriteria criteria = session.CreateCriteria<Produto>();

    return criteria.List<Produto>();
}
```

Agora queremos restringir os resultados dessa criteria para que ela liste apenas produtos com nome igual ao que recebemos como argumento. Para criar uma restrição, utilizamos a classe `NHibernate.Criterion.Restrictions`. Dentro dessa classe, temos o método `Eq`, que cria uma nova restrição de igualdade.

Queremos que a restrição de igualdade compare a propriedade `Nome` do produto com a variável nome que recebemos, então devemos chamar `Restrictions.Eq` passando a string com o nome da propriedade do produto que queremos utilizar na comparação como primeiro argumento e a variável nome como segundo argumento:

```
Restrictions.Eq("Nome", nome)
```

Para restringir o resultado de um `ICriteria`, precisamos adicionar restrições:

```
criteria.Add(Restrictions.Eq("Nome", nome));
```

Com a restrição acima, a query que será executada no banco de dados fica equivalente a seguinte HQL:

```
from Produto p where p.Nome = :nome
```

O código com a restrição na criteria fica da seguinte forma:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    // from Produto p where p.Nome = :nome
    ICriteria criteria = session.CreateCriteria<Produto>();
    criteria.Add(Restrictions.Eq("Nome", nome));

    return criteria.List<Produto>();
}
```

Repare que quando utilizamos as restrições da criteria, estamos colocando uma condição na query e ao mesmo tempo definindo seus parâmetros.

Utilizando o `Add`, podemos adicionar várias restrições na mesma criteria. Vamos adicionar a restrição do preco maior do que o valor mínimo:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    // from Produto p where p.Nome = :nome and p.Preco >= :precoMinimo
    ICriteria criteria = session.CreateCriteria<Produto>();
    criteria.Add(Restrictions.Eq("Nome", nome));
    criteria.Add(Restrictions.Ge("Preco", precoMinimo));

    return criteria.List<Produto>();
}
```

Para comparar o nome da categoria do produto, precisamos navegar no relacionamento da classe `Produto`. Para navegarmos em um relacionamento com a criteria, utilizamos o método `CreateCriteria` do `ICriteria` passando uma string com o nome da proriedade que representa o relacionamento:

```
ICriteria criteriaCategoria = criteria.CreateCriteria("Categoria");
```

Todas as restrições que aplicarmos a `criteriaCategoria` serão aplicadas na categoria. Queremos que o nome da categoria seja igual ao parâmetro `nomeCategoria` que recebemos no método, então utilizaremos novamente o `Restrictions.Eq`:

```
ICriteria criteriaCategoria = criteria.CreateCriteria("Categoria");
criteriaCategoria.Add(Restrictions.Eq("Nome", nomeCategoria));
```

Temos agora o seguinte método:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    // from Produto p where p.Nome = :nome and p.Preco >= :precoMinimo
    // and p.Categoria.Nome = :nomeCategoria
    ICriteria criteria = session.CreateCriteria<Produto>();
    criteria.Add(Restrictions.Eq("Nome", nome));
    criteria.Add(Restrictions.Ge("Preco", precoMinimo));
    ICriteria criteriaCategoria = criteria.CreateCriteria("Categoria");
    criteriaCategoria.Add(Restrictions.Eq("Nome", nomeCategoria));

    return criteria.List<Produto>();
}
```

Agora, colocando a verificação dos argumentos, conseguimos resolver o problema da query dinâmica:

```
public IList<Produto> BuscaPorNomePrecoMinimoECategoria(string nome,
        double precoMinimo, string nomeCategoria)
{
    ICriteria criteria = session.CreateCriteria<Produto>();
    if(nome != null)
    {
        criteria.Add(Restrictions.Eq("Nome", nome));
```

```
            criteria.Add(Restrictions.Eq("Nome", nome));
        }
        if(precoMinimo > 0.0)
        {
            criteria.Add(Restrictions.Ge("Preco", precoMinimo));
        }
        if(nomeCategoria != null)
        {
            ICriteria criteriaCategoria = criteria.CreateCriteria("Categoria");
            criteriaCategoria.Add(Restrictions.Eq("Nome", nomeCategoria));
        }


        return criteria.List<Produto>();
    }
```