02

Relacionamento entre Entidades

section Cadastro de produtos para a loja

Para facilitar a busca de produtos, queremos dividí-los em categorias. Vamos então criar a entidades `Categoria` :

```
public class Categoria
{
    public virtual int Id { get; set; }
    public virtual string Nome { get; set; }
}
```

E seu mapeamento:

```
<class name="Categoria">
  <id name="Id">
    <generator class="identity"/>
  </id>
</class>
```

No c#, para representarmos que um produto possui uma categoria, precisamos apenas criar uma nova propriedade do tipo `Categoria` na classe `Produto` :

```
public class Produto
{
    // outras propriedades
    public virtual Categoria Categoria;
}
```

Para representarmos esse relacionamento no banco de dados, precisamos guardar o id da categoria na tabela de produtos. Uma coluna que guarda o id para outra tabela é chamada de chave estrangeira.

Como a chave estrangeira está na tabela produtos, podemos ter vários produtos associados a mesma categoria, o que caracteriza um relacionamento muitos para um (many to one).

O mapeamento do relacionamento many to one deve ser feito dentro do xml de configuração da entidade, nesse caso, o `Produto.hbm.xml` :

```
<class name="Produto">
  <id name="Id">
    <generator class="identity"/>
  </id>
  <property name="Nome"/>
  <property name="Preco"/>
</class>
```

Para criarmos o relacionamento many to one, utilizamos a tag `many-to-one`. Essa tag recebe o nome da propriedade que representa o relacionamento, nesse caso a Categoria.

```
<many-to-one name="Categoria" />
```

No mapeamento, podemos definir o nome da coluna que será a chave estrangeira desse relacionamento pelo atributo `column` da tag `many-to-one`. Quando não configuramos o nome da coluna, o NHibernate utiliza o nome da propriedade como nome da coluna. Definiremos que esse relacionamento utilizará a coluna `CategoriaId` como chave estrangeira:

```
<many-to-one name="Categoria" column="CategoriaId"/>
```

O mapeamento final do produto fica da seguinte forma:

```
<class name="Produto">
  <id name="Id">
    <generator class="identity"/>
  </id>
  <property name="Nome"/>
  <property name="Preco"/>
  <many-to-one name="Categoria" column="CategoriaId"/>
</class>
```

Lembre-se que, para cada novo mapeamento de entidade que fazemos, precisamos executar novamente o método GeraSchema da classe NHibernateHelper para que as tabelas do banco de dados sejam criadas/atualizadas.

Adicionando um produto com categoria

Quando queremos gravar um novo produto no banco de dados, precisamos, inicialmente, criar e inicializar uma nova instância do objeto Produto

```
Produto produto = new Produto();
produto.Nome = "Camiseta";
produto.Preco = 10.0;
```

E depois, para gravá-lo no banco, utilizamos o método `Save` do `ISession` dentro de uma transação:

```
ITransaction transacao = session.BeginTransaction();
session.Save(Produto);
transacao.Commit();
```

Para associarmos o produto com uma categoria, no c#, precisamos apenas colocar uma instância de `Categoria` no atributo `Categoria` do `Produto` antes de gravá-lo no banco de dados.

```
Produto produto = new Produto();
// inicializa o produto
Categoria categoria = new Categoria();
```

```
// inicializa a categoria
produto.Categoria = Categoria;

ITransaction transacao = session.BeginTransaction();
session.Save(produto);
transacao.Commit();
```

Porém quando executamos esse código, o NHibernate joga uma exceção pois tentamos associar uma nova categoria, que está no estado transient, com o produto que está sendo gravado. O nome dessa exceção é `TransientObjectException`.

A `TransientObjectException` ocorre sempre que tentamos gravar um relacionamento com um objeto que está no estado transient. Para evitar a exceção, precisamos fazer com que a categoria que será associada com o produto esteja no estado persistent, ou seja, antes de gravar o produto precisamos gravar a categoria:

```
Produto produto = new Produto();
// inicializa o produto
Categoria categoria = new Categoria();
// inicializa a categoria
produto.Categoria = Categoria;

ITransaction transacao = session.BeginTransaction();
session.Save(categoria);
session.Save(produto);
transacao.Commit();
```

Categoria com lista de produtos

Agora que definimos o mapeamento do relacionamento many to one do produto com a categoria, estamos interessados em pegar todos os produtos associados a uma determinada categoria.

Quando olhamos o relacionamento do ponto de vista de um produto, temos muitos produtos associados a uma instância de categoria, porém do ponto de vista da categoria, temos uma categoria associada a vários produtos, ou seja, temos um relacionamento one to many.

No mundo relacional, toda vez que temos um relacionamento many to one, existe, automaticamente, um one to many, ou seja, no mundo relacional, os relacionamentos são sempre bidirecionais. Na orientação a objetos, não temos relacionamentos bidirecionais automáticos, toda vez que queremos o relacionamento one to many, devemos fazer o mapeamento explícito desse relacionamento.

Para mapear a lista de produtos na categoria, colocaremos, inicialmente, a lista como uma propriedade da categoria:

```
public class Categoria
{
    public virtual int Id { get; set; }
    public virtual string Nome { get; set; }
    public virtual IList<Produto> Produtos { get; set; }
}
```

Agora que colocamos a lista, precisamos falar para o NHibernate que essa lista representa um relacionamento one to many, fazemos essa configuração dentro do arquivo `Categoria.hbm.xml`.

Para mapearmos uma propriedade do tipo `IList` do C#, utilizamos a tag `bag` passando o nome da propriedade dentro do atributo `name` da tag:

```
<bag name="Produtos">

</bag>
```

Dentro da tag `bag`, definiremos que essa propriedade representa um relacionamento do tipo one to many, através da tag `one-to-many`. Dentro dela, devemos informar que queremos fazer um relacionamento com o `Produto` através do atributo `class`:

```
<bag name="Produtos">
    <one-to-many class="Produto"/>
</bag>
```

Depois de configurar que a lista representa um relacionamento one to many, precisamos informar qual é o nome da chave estrangeira que está guardada na tabela dos produtos. Essa configuração é feita através do atributo column da tag `key` que deve ficar dentro da tag `bag`:

```
<bag name="Produtos">
  <key column="CategoriaId"/>
  <one-to-many class="Produto"/>
</bag>
```

A configuração final do Categoria.hbm.xml fica da seguinte forma:

```
<class name="Categoria">
  <id name="Id">
    <generator class="identity"/>
  </id>
  <property name="Nome"/>
  <bag name="Produtos">
    <key column="CategoriaId"/>
    <one-to-many class="Produto"/>
  </bag>
</class>
```

Repare que o relacionamento one to many que acabamos de mapear é determinado pela chave estrangeira do produto. O lado que determina o relacionamento é chamado de lado forte, ou dono (owner) do relacionamento, o outro lado é chamado de lado fraco do relacionamento.

Agora que mapeamos a lista de produtos na categoria, podemos recuperar todos os produtos de uma determinada categoria simplesmente acessando a propriedade `Produtos` da `Categoria`:

```
Categoria categoria = session.Get<Categoria>(1);
IList<Produto> produtos = categoria.Produtos;
foreach(var produto in produtos)
{
```

```
    Console.WriteLine(produto.Nome);
}
```

Cuidando do relacionamento bidirecional

Imagine que temos a seguinte categoria no banco de dados:

```
+----+--------------+
| Id | Nome         |
+----+--------------+
|  1 | Informática  |
+----+--------------+
```

Associados a essa categoria, temos os seguintes produtos:

```
+----+---------+--------+-------------+
| Id | Nome    | Preco  | CategoriaId |
+----+---------+--------+-------------+
|  1 | Teclado |  20.00 |           1 |
|  2 | Monitor | 300.00 |           1 |
+----+---------+--------+-------------+
```

Podemos recuperar a lista de produtos utilizando o código abaixo:

```
ITransaction transacao = session.BeginTransaction();

Categoria categoria = session.Load<Categoria>(1);
IList<Produto> produtos = categoria.Produtos;
transacao.Commit();
```

Quando executamos esse código, podemos ver que o NHibernate executa uma query que recupera apenas a categoria do banco de dados, os produtos relacionados a essa categoria, por padrão, não são carregados.

O NHibernate carrega os relacionamentos apenas quando necessário (modo lazy). Quando pedimos qualquer informação sobre o relacionamento, ele é forçado a realizar a busca no banco de dados. Por exemplo, o código abaixo força o hibernate a buscar os produtos:

```
ITransaction transacao = session.BeginTransaction();

Categoria categoria = session.Load<Categoria>(1);
Console.WriteLine(categoria.Produtos.Count);

transacao.Commit();
```

Esse código imprime 2 no terminal, pois no banco de exemplo temos 2 produtos associados à categoria de id 1. Agora vamos adicionar mais um produto com a categoria 1 e imprimir novamente o número de produtos dessa categoria:

```
ITransaction transacao = session.BeginTransaction();

Categoria categoria = session.Load<Categoria>(1);
Console.WriteLine(categoria.Produtos.Count);

Produto produto = new Produto()
{
    Categoria = categoria,
    Nome = "nome",
    Preco = 200.0
};
session.Save(produto);
Console.WriteLine(categoria.Produtos.Count);

transacao.Commit();
```

Nesse código, o primeiro `WriteLine` imprime 2 como esperado, porém depois de gravarmos o produto, o segundo `WriteLine` continua imprimindo 2 no terminal, ou seja, o NHibernate não buscou novamente a lista de produtos, é trabalho da aplicação manter a consistência das duas pontas do relacionamento.

Para que o número de produtos da categoria fique correto, a aplicação precisa adicionar o novo produto à lista da categoria:

```
Produto produto = new Produto();
// inicializa o produto
produto.Categoria = categoria;
categoria.Produtos.Add(produto);
session.Save(produto);
```

Carregando elementos relacionados

O NHibernate carrega as informações do relacionamento apenas quando elas são necessárias (carregamento lazy), porém podemos modificar esse comportamento fazendo com que ele busque os relacionamentos assim que a entidade for carregada (carregamento eager).

Para utilizarmos o carregamento eager de um relacionamento, many to one, por exemplo, devemos configurar o atributo `fetch` da tag `many-to-one`. Esse atributo aceita dois valores: `join`, que faz com que o relacionamento seja carregado de forma eager, ou `select`, que faz com que o relacionamento seja lazy. O valor padrão do atributo é `select`.

O relacionamento many to one do produto com carregamento eager fica da seguinte forma:

```
<many-to-one name="Categoria" column="CategoriaId" fetch="join"/>
```

Da mesma forma que configuramos o relacionamento many to one, também podemos configurar o one to many para realizar o carregamento eager, porém no caso do one to many, colocamos o atributo fetch na tag bag:

```
<bag name="Produtos" fetch="join">
    <key column="Categoria"/>
    <one-to-many class="Produto"/>
</bag>
```