02

Escondendo informações específicas do banco

Transcrição

Abriremos o Console do NuGet, e usaremos o comando `Add-Migration Cliente` para adicionar essa migração. Ao executarmos receberemos um erro informando que a classe `Endereco` precisa de um `Id`.

Como o Entity soube que queríamos persistir a classe `Endereco` se não criamos uma propriedade no `LojaContext`? Quando definimos uma entidade no `LojaContext`, ele navega por suas propriedades. Dessa forma, caso o valor seja um referência de outra classe - como a `Endereco` -, ela automaticamente passa a ser controlada pelo Entity.

A decisão de mapear uma classe no `LojaContext` fica de acordo com a lógica de negócio. Caso seja **necessário** manipular os atributos da classe de forma independente, criar um propriedade em `LojaContext` fará sentido. Em nosso caso não criaremos, nós apenas iremos manipular o endereço por meio da classe `Cliente`.

Para resolvermos nosso problema, bastaríamos criar uma propriedade `Id` ou `EnderecoId`, para que o Entity mapeie automaticamente como chave primária. Mas existem cenários de relacionamentos **um para um**, onde a tabela dependente - `Endereco` - assume o `Id` da tabela principal, a tabela `Cliente`.

Usaremos este exemplo para aprendermos que não precisamos necessariamente ter um chave primária em nossa classe. Como colocar um tributo `ClienteId` não faz sentido para o modelo, podemos ensinar ao Entity por meio do método `OnModelCreating()` da classe `LojaContext`.

Dentro do método `OnModelCreating()`, chamaremos o `modelBuilder` dizendo que a entidade `Endereco` possui uma propriedade `ClienteId` e por último criaremos a chave, o método ficará da seguinte maneira:

```
internal class LojaContext : DbContext
{
    // ...

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder
            .Entity<PromocaoProduto>()
            .HasKey(pp => new { pp.PromocaoId, pp.ProdutoId });

        modelBuilder
            .Entity<Endereco>()
            .Property("ClienteId");
    }
    // ...
}
```

O nome desse conceito para o Entity é ***Shadow Property***, ou seja, uma propriedade que está escondida, ficando apenas no banco de dados. Podemos rodar o comando `Add-Migration Cliente`.

Ao executarmos receberemos outro erro, mas desta vez é informado que para usarmos ***Shadow Property*** é necessário passar o tipo da propriedade. Basta adicionarmos na definição da propriedade `"ClienteId"`.

```
internal class LojaContext : DbContext
{
    // ...

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder
            .Entity<PromocaoProduto>()
            .HasKey(pp => new { pp.PromocaoId, pp.ProdutoId });

        modelBuilder
            .Entity<Endereco>()
            .Property<int>("ClienteId");

        modelBuilder
            .Entity<Endereco>()
            .HasKey("ClienteId");
    }
    // ...
}
```

Executaremos o comando `Add-Migration Cliente` novamente. A migração será criada, mas o Entity interpretou errado e considerou que a classe principal era a `Endereco` e a dependente era a `Cliente`, por esse motivo a coluna `ClienteId` foi criada em `Cliente`. Outro detalhe é que o nome da tabela ficou como `Endereco` no singular, o que não é o ideal já que a tabela armazenará diversos endereços.

Novamente no Console, removeremos a migração com `Remove-Migration`. Veremos como alterar do nome da tabela e resolver a dependência entre `Cliente` e `Endereco`.