05

Manipulando promoções e produtos

Transcrição

Criamos a tabela `Promocao` junto com a tabela de `JOIN` chamada de `PromocaoProduto`. Agora cadastraremos uma promoção no banco de dados, para vermos o comportamento do Entity com o relacionamento muito para muitos.

Na classe `Program`, tiraremos os comentários das linhas onde são incluídos os produtos. Mas temos um problema, por conta da configuração que fizemos para ter o relacionamento **muitos para muitos**, o método `Produtos.Add()` não recebe mais um `Produto`, e sim um `PromocaoProduto`.

Para não poluirmos o código de produção com uma classe interna, criaremos um método na classe `Promocao` chamado `IncluiProduto()`.

```
class Program
{
    static void Main(string[] args)
    {
        var promocaoDePascoa = new Promocao();
        promocaoDePascoa.Descricao = "Páscoa Feliz";
        promocaoDePascoa.DataInicio = DateTime.Now;
        promocaoDePascoa.DataTermino = DateTime.Now.AddMonths(3);

        promocaoDePascoa.IncluiProduto(p1);
        promocaoDePascoa.IncluiProduto(p2);
        promocaoDePascoa.IncluiProduto(p3);

        using(var contexto = new LojaContext())
        {
            var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
            var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
            loggerFactory.AddProvider(SqlLoggerProvider.Create());
        }
    }
    // ...
}
```

Não podemos esquecer de instanciar os produtos `p1`, `p2` e `p3`.

```
static void Main(string[] args)
{
    var p1 = new Produto();
    var p2 = new Produto();
    var p3 = new Produto();

    var promocaoDePascoa = new Promocao();
    promocaoDePascoa.Descricao = "Páscoa Feliz";
    promocaoDePascoa.DataInicio = DateTime.Now;
    promocaoDePascoa.DataTermino = DateTime.Now.AddMonths(3);

    promocaoDePascoa.IncluiProduto(p1);
    promocaoDePascoa.IncluiProduto(p2);
```

```
        promocaoDePascoa.IncluiProduto(p2);
        promocaoDePascoa.IncluiProduto(p3);

        using(var contexto = new LojaContext())
        {
            var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
            var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
            loggerFactory.AddProvider(SqlLoggerProvider.Create());
        }
    }
```

Em seguida, implementaremos o método `IncluiProduto()` na classe `Promocao`. Para não recebermos um erro, precisaremos instanciar a lista `Produtos` no construtor da classe. Ela ficará da seguinte maneira:

```
public class Promocao
{
    public int Id { get; set; }
    public string Descricao { get; internal set; }
    public DateTime DataInicio { get; internal set; }
    public DateTime DataTermino { get; internal set; }
    public IList<PromocaoProduto> Produtos { get; internal set; }

    public Promocao()
    {
        this.Produtos = new List<PromocaoProduto>();
    }

    public void IncluiProduto(Produto produto)
    {
        this.Produtos.Add(new PromocaoProduto() { Produto = produto });
    }
}
```

Incluiremos os produtos no banco de dados, por isso os popularemos com alguns valores. Em seguida, adicionaremos o `promocaoDePascoa` ao contexto do Entity e exibir com o método `ExibeEntries()`.

```
static void Main(string[] args)
{
    var p1 = new Produto() { Nome = "Suco de Laranja", Categoria = "Bebidas", PrecoUnitario = 8.79,
    var p2 = new Produto() { Nome = "Café", Categoria = "Bebidas", PrecoUnitario = 12.45, Unidade =
    var p3 = new Produto() { Nome = "Macarrão", Categoria = "Alimentos", PrecoUnitario = 4.23, Unida

    var promocaoDePascoa = new Promocao();
    promocaoDePascoa.Descricao = "Páscoa Feliz";
    promocaoDePascoa.DataInicio = DateTime.Now;
    promocaoDePascoa.DataTermino = DateTime.Now.AddMonths(3);

    promocaoDePascoa.IncluiProduto(p1);
    promocaoDePascoa.IncluiProduto(p2);
    promocaoDePascoa.IncluiProduto(p3);

    using(var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
```

```
            loggerFactory.AddProvider(SqlLoggerProvider.Create());

            contexto.Promocoes.Add(promocaoDePascoa);
            ExibeEntries(contexto.ChangeTracker.Entries());

        }

        private static void ExibeEntries(IEnumerable<EntityEntry> entries)
        {
            // ...
        }
    }
```

Executaremos a aplicação. Como resultado, veremos que o Entity passou a monitorar as entidade `Promocao`, `PromocaoProduto`, além dos três produtos adicionados. Todos os elementos adicionados ficaram com o estado **Added**. Podemos colocar o `SaveChanges()` no código para que as promoções seja persistidas no banco de dados.

```
    static void Main(string[] args)
    {
        var p1 = new Produto() { Nome = "Suco de Laranja", Categoria = "Bebidas", PrecoUnitario = 8.79,
        var p2 = new Produto() { Nome = "Café", Categoria = "Bebidas", PrecoUnitario = 12.45, Unidade =
        var p3 = new Produto() { Nome = "Macarrão", Categoria = "Alimentos", PrecoUnitario = 4.23, Unida

        var promocaoDePascoa = new Promocao();
        promocaoDePascoa.Descricao = "Páscoa Feliz";
        promocaoDePascoa.DataInicio = DateTime.Now;
        promocaoDePascoa.DataTermino = DateTime.Now.AddMonths(3);

        promocaoDePascoa.IncluiProduto(p1);
        promocaoDePascoa.IncluiProduto(p2);
        promocaoDePascoa.IncluiProduto(p3);

        using(var contexto = new LojaContext())
        {
            var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
            var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
            loggerFactory.AddProvider(SqlLoggerProvider.Create());

            contexto.Promocoes.Add(promocaoDePascoa);
            contexto.SaveChanges();

        }

        private static void ExibeEntries(IEnumerable<EntityEntry> entries)
        {
            // ...
        }

    }
```

Novamente executando a aplicação, tudo funciona como o esperado. Olhando no banco de dados, veremos a promoção cadastrada na tabela `Promocao`, e na tabela `Produto` os três novos produtos também estão cadastrados.

Mas o que acontece se deletarmos a promoção? Faremos um teste pegando a promoção do banco e chamando o método `contexto.Promocoes.Remove()`.

```
static void Main(string[] args)
{
    var p1 = new Produto() { Nome = "Suco de Laranja", Categoria = "Bebidas", PrecoUnitario = 8.79,
    var p2 = new Produto() { Nome = "Café", Categoria = "Bebidas", PrecoUnitario = 12.45, Unidade =
    var p3 = new Produto() { Nome = "Macarrão", Categoria = "Alimentos", PrecoUnitario = 4.23, Unida

    var promocaoDePascoa = new Promocao();
    promocaoDePascoa.Descricao = "Páscoa Feliz";
    promocaoDePascoa.DataInicio = DateTime.Now;
    promocaoDePascoa.DataTermino = DateTime.Now.AddMonths(3);

    promocaoDePascoa.IncluiProduto(p1);
    promocaoDePascoa.IncluiProduto(p2);
    promocaoDePascoa.IncluiProduto(p3);

    using(var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());

        //contexto.Promocoes.Add(promocaoDePascoa);
        var promocao = contexto.Promocoes.Find(3);
        contexto.Promocoes.Remove(promocao);
        contexto.SaveChanges();

    }
    // ...
}
```

A promoção será deletada, junto com o elemento de `PromocaoProduto` pelo efeito de cascata, já que eles são completamente interligados. Mas os produtos cadastrados ainda permanecem no banco.