01

Relacionamento muitos para muitos e a classe de join

Transcrição

Na aula anterior introduzimos a classe `Compra`, para representarmos a compra de um produto com base no preço e quantidade. Utilizamos essa funcionalidade para introduzir o conceito de relacionamento entre entidades. Nessa aula continuaremos falando sobre relacionamentos, porém em vez de ter relacionamento apenas com a instância de uma única classe, faremos o relacionamento para uma *coleção de instâncias*.

Vamos imaginar que a nossa loja crie regularmente, promoções para baixar o estoque dos produtos. Representaremos esse conceito criando uma classe chamada de `Promocao`.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    public class Promocao
    {

    }
}
```

Agora na classe `Program`, dentro do método `Main()` criaremos um cenário para essas promoções. Deixaremos dentro do método apenas a instancia do contexto do Entity e o código do *Logger*.

Instanciaremos uma promoção e armazenaremos em um objeto chamado `promocaoDePascoa`. A partir desse objeto, chamaremos as propriedades `Descricao`, `DataInicio`, `DataTermino`. Além dessas propriedades, adicionaremos três produtos na propriedade `Produtos`.

```
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.Produtos.Add(new Produto());
        promocaoDePascoa.Produtos.Add(new Produto());
        promocaoDePascoa.Produtos.Add(new Produto());

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

    private static void ExibeEntries(IEnumerable<EntityEntry> entries)
    {
        foreach(var e in entries)
        {
```

```
            Console.WriteLine(e.Entity.ToString() + " - " + e.State);
        }
    }
}
```

As propriedades ainda não existem na classe `Promocao`, por isso iremos criá-la.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    public class Promocao
    {
        public string Descricao { get; internal set; }
        public DateTime DataInicio { get; internal set; }
        public DateTime DataTermino { get; internal set; }
        public IList<Produto> Produtos { get; internal set; }
    }
}
```

Para podermos usar a propriedade `Produtos`, é necessário colocarmos a classe `Produto` como `public`. Como o Entity irá refletir isso no banco de dados? Sabemos que a classe `Promocao` ainda não tem a sua tabela representativa, faremos essa configuração.

Na classe `LojaContext`, criaremos uma nova propriedade `DbSet<Promocao> Promocoes`.

```
using Microsoft.EntityFrameworkCore;
using System;

namespace Alura.Loja.Teste.ConsoleApp
{
    public class LojaContext : DbContext
    {
        public DbSet<Produto> Produtos { get; set; }
        public DbSet<Compra> Compras { get; set; }
        public DbSet<Promocao> Promocoes { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=LojaDB;Trusted_Conn
        }
    }
}
```

Se tentarmos usar o comando `Add-Migration` receberemos um erro informando que não possuímos um chave primária, por isso criaremos na classe promoção a propriedade `Id`. Em seguida, podemos finalmente adicionar a migração que chamaremos de `Promocao`. O comando completo será `Add-Migration Promocao`.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    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<Produto> Produtos { get; internal set; }
    }
}
```

Analisando a **classe de migração** `Promocao` , veremos que ele está criando um coluna `PromocaoId` na tabela `Produto` . O Entity interpretou que cada produto está associado a **apenas um promoção**. O que não faz sentido para a nossa aplicação. O que queremos é que a promoção acesse uma coleção de produtos, e o produto acesse uma coleção de promoções. Removeremos essa migração com `Remove-Migration` .

Para resolvermos isso, adicionaremos na classe `Produto` uma lista de promoções.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    public class Produto
    {
        public int Id { get; internal set; }
        public int Nome { get; internal set; }
        public int Categoria { get; internal set; }
        public int PrecoUnitario { get; internal set; }
        public string Unidade { get; set; }
        public IList<Promocao> Promocoes { get; set; }

        public override string ToString()
        {
            return $"Produto: {this.Id}, {this.Nome}, {this.Categoria}, {this.PrecoUnitario}";
        }
    }
}
```

Dessa forma temos o relacionamento de muitas promoções para muitos produtos. Porém se tentarmos adicionar a migração teremos um erro. A versão **6** que é a anterior do Entity Framework Core, conseguia identificar as tabelas `JOIN` automaticamente. Mas na versão atual, que é um ***subset*** da versão anterior, foi deixado de fora (por enquanto) essa função.

No próximo vídeo veremos como resolver esse relacionamento.