06

Corrigindo o modelo para recuperar objetos relacionados

Transcrição

Exploraremos o relacionamento **um para um** entre `Cliente` e `Endereco`. Como já fizemos anteriormente, vamos colocar todo o conteúdo do método `Main()` e um método chamado `ExibeProdutosDaPromocao()`. Dentro do `Main()` criaremos apenas um objeto de contexto do Entity e deixaremos os *Logger*.

```
static void Main(string[] args)
{
    using (var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());
    }
}
```

Buscaremos a propriedade `Logradouro` do endereço de um `Cliente`. Porém antes precisamos buscar o cliente, faremos isso usando o `FirstOrDefault()`. Como a propriedade `EnderecoDeEntrega` já está disponível, podemos acessar o `Logradouro`.

```
static void Main(string[] args)
{
    using (var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());

        var cliente = contexto.Clientes.FirstOrDefault();

        Console.WriteLine($"Endereço de entrega: {cliente.EnderecoDeEntrega.Logradouro}");

    }
}
```

Ao executarmos a aplicação ocorrerá um exceção. Isso aconteceu porque acessamos a propriedade `Logradouro` nula, o Entity não carregou essa propriedade. Incluiremos a entidade em nossa busca:

```
static void Main(string[] args)
{
    using (var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());

        var cliente = contexto
            .Clientes
            .Include(c => c.EnderecoDeEntrega)
            .FirstOrDefault();
```

```
            .FirstOrDefault();

        Console.WriteLine($"Endereço de entrega: {cliente.EnderecoDeEntrega.Logradouro}");
    }
}
```

Executaremos a aplicação e veremos que funcionou perfeitamente. O `SELECT` que o Entity utilizou foi com `LEFT JOIN`, pelo fato do endereço não ser obrigatório para o cliente.

Continuaremos explorando as consultas com relacionamento, pegaremos as compras de um produto. Primeiro pegaremos um produto qualquer - no nosso caso o produto com uma compra era o com Id 9004 - passando o seu `Id` pelo `Where()`.

```
static void Main(string[] args)
{
    using (var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());

        var cliente = contexto
            .Clientes
            .Include(c => c.EnderecoDeEntrega)
            .FirstOrDefault();

        Console.WriteLine($"Endereço de entrega: {cliente.EnderecoDeEntrega.Logradouro}");

        var produto = contexto
            .Produtos
            .Where(p => p.Id == 9004)
            .FirstOrDefault();
    }
}
```

Com esse produto faremos um `foreach` na listagem `produto.Compras`. Porém não temos a lista de `Compras` pelo fato de que o Entity não nos obrigar a criá-la. Adicionaremos a lista na classe `Produto`:

```
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<PromocaoProduto> Promocoes { get; set; }
        public IList<Compra> Compras { get; set; }

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

Antes de inciarmos a iteração pela lista, precisamos incluir as compras no `Include()`.

```
static void Main(string[] args)
{
    using (var contexto = new LojaContext())
    {
        var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());

        var cliente = contexto
            .Clientes
            .Include(c => c.EnderecoDeEntrega)
            .FirstOrDefault();

        Console.WriteLine($"Endereço de entrega: {cliente.EnderecoDeEntrega.Logradouro}");

        var produto = contexto
            .Produtos
            .Include(p => p.Compras)
            .Where(p => p.Id == 9004)
            .FirstOrDefault();

        Console.WriteLine($"Mostrando as compras do produto {produto.Nome}");
        foreach(var item in produto.Compras)
        {
            Console.WriteLine(item);
        }
    }
}
```

Executaremos a aplicação e como resultado, veremos que o Entity buscou o produto com o `Id` **9004**, em seguida ele fez um `INNER JOIN` de `Compras` com `Produtos` para pegar as compras, e por último foi mostrado a compra. A única coisa que precisamos fazer é uma propriedade, onde navegaremos para o lado onde temos muitos.

Vimos como trabalhar com buscas de relacionamentos **muitos para muitos**, **um para um** e **um para muitos**. Mas e se quiséssemos buscar as compras acima de um valor determinado? É o nosso próximo passo!