05

Inserindo objetos relacionados

Transcrição

Anteriormente, criamos uma nova classe chamada `Compra` e sincronizamos o modelo de classe com o banco usando o recurso de *Migrations*. Temos tudo que precisamos para registrar a compra do banco de dados.

Na classe `Program`, iremos instanciar um objeto do contexto de `LojaContext`. Com o objeto, iremos adicionar a compra ao contexto do Entity e salvar as alterações. Para visualizarmos as ações, colocaremos o código de *Logger*. A classe ficará da seguinte maneira:

```
class Program
{
    static void Main(string[] args)
    {
        //compra de 6 pães franceses
        var paoFrances = new Produto();
        paoFrances.Nome = "Pão Francês";
        paoFrances.PrecoUnitario = 0.40;
        paoFrances.Unidade = "Unidade";
        paoFrances.Categoria = "Padaria";

        var compra = new Compra();
        compra.Quantidade = 6;
        compra.Produto = paoFrances;
        compra.Preco = paoFrances.PrecoUnitario * compra.Quantidade;

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

            contexto.Compras.Add(compra);

            contexto.SaveChanges();
        }
    }
}
```

Sabemos que o estado de `compra` vai ser ***Added***, mas qual será o estado do `paoFrances`? Nós não pegamos o `paoFrances` do banco de dados, ele é um produto novo. Tivemos todo o trabalho de refazer a **migração** `Compra` justamente para não ter um produto nulo, como será que o Entity irá reagir nessa situação?

Quando adicionarmos o `compra` no ***Change Tracker***, o Entity perceberá que existe uma referência ao produto `paoFrances` e também o incluirá para ser supervisionado.

Comentaremos a linha do `SaveChanges()`, e colocaremos na classe `Program` o método `ExibeEntries()` que usamos nas aulas anteriores.

```
class Program
```

```
{
    static void Main (string [] args)
    {
        //compra de 6 pães franceses
        var paoFrances = new Produto();
        paoFrances.Nome = "Pão Francês";
        paoFrances.PrecoUnitario = 0.40;
        paoFrances.Unidade = "Unidade";

        paoFrances.Categoria = "Padaria";

        var compra = new Compra();
        compra.Quantidade = 6;
        compra.Produto = paoFrances;
        compra.Preco = paoFrances.PrecoUnitario * compra.Quantidade;

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

            contexto.Compras.Add(compra);

            ExibeEntries(contexto.ChangeTracker.Entries());

            //contexto.SaveChanges();
        }

    }

    private static void ExibeEntries(IEnumerable<EntityEntry> entries)
    {
        foreach(var e in entries)
        {
            Console.WriteLine(e.Entity.ToString() + " - " + e.State);
        }
    }
}
```

Executaremos a aplicação, como resultado veremos que temos duas entidades, um do tipo `Compra` e outro do tipo `Produto`. As duas entidades estão com o estado *Added*, por isso o Entity irá gerar o comando SQL `INSERT` para cada um deles. O contexto foi esperto o suficiente para adicionar a referência do produto contida na compra.

Removeremos o comentário da linha `contexto.SaveChanges()` e executaremos a aplicação. Veremos que o Entity inseriu primeiro o produto no banco de dados, depois inseriu a compra, justamente por ser necessário ter um produto preenchido para incluirmos a compra. Se olharmos o banco, veremos todas as informações persistidas.