02

Corrigindo a migração para refletir o relacionamento

Transcrição

Antes de criarmos a migração para representar a classe, entraremos na classe `Compra` e adicionaremos uma propriedade `Id`. Essa propriedade irá representar a chave primária da compra. Também removeremos o construtor da classe, que foi gerado automaticamente pelo Visual Studio.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    internal class Compra
    {
        public int Id { get; set; }
        public int Quantidade { get; internal set; }
        public Produto Produto { get; internal set; }
        public double Preco { get; internal set; }
    }
}
```

Abriremos o **Console do Gerenciador de Pacotes** do NuGet, e digitaremos o comando para adicionarmos a migração:

PM> Add-Migration Compra

Abriremos a classe de migração `Compra`. Podemos notar que os métodos `Up()` e `Down()` estão vazios. O que queremos é criar a tabela **Compra** no banco de dados, mas o que fizemos de errado?

Estamos esquecendo de um passo importante, que é dizer pro Entity gerenciar a persistência da classe `Compra`. Para fazermos isso, acessaremos a classe de contexto `LojaContext`, e criaremos uma propriedade `DbSet<Compra> Compras`.

```
using Microsoft.EntityFrameworkCore;
using System;

namespace Alura.Loja.Testes.ConsoleApp
{
    internal class LojaContext : DbContext
    {
        public DbSet<Produto> Produtos { get; set; }
        public DbSet<Compras> Compras { get; set; }

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

Removeremos a migração que acabamos de fazer executando o comando `Remove-Migration` no Console. A classe de migração com os métodos `Up()` e `Down()` vazias, será excluída.

Rodaremos novamente o comando `Add-Migration Compra`. A classe criada para essa migração possui código nos métodos `Up()` e `Down()`. Basicamente o método `Up()` irá criar uma tabela com as colunas `Id`, `Preco`, `ProdutoId` e `Quantidade`.

Não criamos nenhuma propriedade na classe `Compras` com o nome `ProdutoId`, então de onde veio essa coluna? Isso acontece porque no modelo relacional não é possível guardar uma referência para um objeto, como fazemos no modelo orientado a objetos. Justamente por isso, no modelo relacional é guardado uma *referência para a chave primária* do produto, e o Entity usa na coluna a convenção `[nome da classe]Id`.

É sempre importante analisarmos o código antes de atualizar as migrações. A linha de criação da coluna `ProdutoId` permite que um produto seja nulo:

```
ProdutoId = table.Column<int>(nullable: true);
```

Quando efetuamos uma compra, é obrigatório ter um produto. Essa migração não está atendendo as nossas necessidades. Por isso iremos removê-la com o comando `Remove-Migration`.

Para indicar que um `Produto` é obrigatório, iremos deixar explícito a chave estrangeira `ProdutoId`.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    internal class Compra
    {
        public int Id { get; set; }
        public int Quantidade { get; internal set; }
        public int ProdutoId { get; set; }
        public Produto Produto { get; internal set; }
        public double Preco { get; internal set; }
    }
}
```

Como um valor do tipo `int` não pode ser nulo, o Entity irá interpretar que a propriedade deve ser obrigatória. Executaremos o comando `Add-Migration Compra` para adicionar a migração.

Podemos ver na classe de migração `Compra`, que a coluna `ProdutoId` não permite valores nulos:

```
ProdutoId = table.Column<int>(nullable: false);
```

Finalmente podemos sincronizar essa migração com o banco de dados, usaremos o comando `Update-Database`. A tabela `Compras` foi criada, e a migração foi adicionada no histórico de migrações na tabela `__EFMigrationsHistory`.