05

Relacionamentos e Migrações

Transcrição

De um lado, temos as classes do modelo em C# e, do outro, o banco de dados vazio. Portanto, precisamos fazer a geração das tabelas a partir do modelo, e para isso utilizamos o Entity Framework Core. Acessaremos os seus comandos, clicando na opção "Tools", na barra de menu superior. Na sequência, selecionaremos "NuGet Package Manager > Package Manager Console".

No console, podemos digitar os comandos que permitirão gerar tabelas. Criaremos um pacote de atualizações do esquema do banco de dados e, em seguida, o aplicaremos. Ele é chamado de migração, ou *migration*. Ele contém informações de quais tabelas geraram quais chaves, ou colunas, a partir do código. Para isso, utilizaremos o comando `Add-Migration` e, em seguida, teremos que inserir o nome da migração. Daremos o nome `Inicial`.

```
PM> Add-Migration Inicial
```

Pressionaremos a tecla "Enter" para rodarmos a aplicação, e ele gerará o pacote de migração. Ao finalizar, abriremos o projeto, e na pasta "Migrations" observamos que há uma classe contendo a data, seguida do nome `Inicial.cs`, nela temos todo o esquema que foi gerado a partir da entidade `Produto`, são instruções para a criação da tabela de produtos.

Retornaremos ao console. Uma vez que geramos a migração, precisamos fazer a aplicação dela. Para isso, utilizamos o comando `Update-Database`, seguido de `-verbose`, que indica ao Entity que ele deve gerar uma série de logs, para podermos visualizar o que está acontecendo. Pressionaremos a tecla "Enter" para executarmos.

Os logs foram gerados, assim como a respectiva tabela no banco de dados, como podemos observar em "SQL Server Object Explorer > CasaDoCodigo > Tables > dbo.Produto". Clicaremos com o botão direito do mouse sobre ela, e selecionaremos a opção "View Designer".

Podemos visualizar que foi gerada uma tabela contendo uma chave primária `id`, que é uma das propriedades da nossa classe e, em seguida, mais três linhas, com código, nome e preço do produto. Assim, concluímos a criação da nossa primeira migração.

Retornaremos ao `ApplicationContext` para fazermos o mapeamento do restante das entidades. Assim como fizemos anteriormente, utilizaremos o `modelBuilder`:

```
namespace CasaDoCodigo
{
    public class ApplicationContext : DbContext
    {
        public ApplicationContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Produto>().HasKey(t => t.Id);
```

```
                        modelBuilder.Entity<Pedido>().HasKey(t => t.Id);

                        modelBuilder.Entity<ItemPedido>().HasKey(t => t.Id);

                        modelBuilder.Entity<Cadastro>().HasKey(t => t.Id);
        }
    }
}
```

Agora que relacionamos, fizemos o mapeamento de todas as entidades, precisamos descrever como serão os relacionamentos. Utilizaremos os comandos para criar estas conexões, como o `HasMany()`, acompanhado da expressão lambda `t.Itens`, indicando que queremos trabalhar com muitos itens. Por fim, vamos inserir o novo método `WithOne()`, que indica o relacionamento de volta, ou seja, cada item de pedido se relaciona a um pedido individual:

```
namespace CasaDoCodigo
{
    public class ApplicationContext : DbContext
    {
        public ApplicationContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Produto>().HasKey(t => t.Id);

                        modelBuilder.Entity<Pedido>().HasKey(t => t.Id);
                        modelBuilder.Entity<Pedido>().HasKey(t => t.Id).HasMany(t => t.Itens).WithOr

                        modelBuilder.Entity<ItemPedido>().HasKey(t => t.Id);

                        modelBuilder.Entity<Cadastro>().HasKey(t => t.Id);
        }
    }
}
```

A seguir, faremos o relacionamento entre o pedido e o cadastro. A diferença é que esta relação se dá de um para um, ou seja, cada pedido está relacionado a um cadastro e vice-versa. Acrescentaremos ainda um método indicando que o atributo é obrigatório, que é o `IsRequired()`.

```
namespace CasaDoCodigo
{
    public class ApplicationContext : DbContext
    {
        public ApplicationContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
```

```
            modelBuilder.Entity<Produto>().HasKey(t => t.Id);

                        modelBuilder.Entity<Pedido>().HasKey(t => t.Id);
                        modelBuilder.Entity<Pedido>().HasMany(t => t.Itens).WithOne(t => t.Pedido);
                        modelBuilder.Entity<Pedido>().HasOne(t => t.Cadastro).WithOne(t => t.Pedido)

                        modelBuilder.Entity<ItemPedido>().HasKey(t => t.Id);

                        modelBuilder.Entity<Cadastro>().HasKey(t => t.Id);
        }
    }
}
```

Dando continuidade, faremos isso com o `ItemPedido`. Cada um estará relacionado a um `Pedido` e a um `Produto`. Da mesma forma, o `Cadastro` está relacionado a um `Pedido`.

```
namespace CasaDoCodigo
{
    public class ApplicationContext : DbContext
    {
        public ApplicationContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Produto>().HasKey(t => t.Id);

                        modelBuilder.Entity<Pedido>().HasKey(t => t.Id);
                        modelBuilder.Entity<Pedido>().HasMany(t => t.Itens).WithOne(t => t.Pedido);
                        modelBuilder.Entity<Pedido>().HasOne(t => t.Cadastro).WithOne(t => t.Pedido)

                        modelBuilder.Entity<ItemPedido>().HasKey(t => t.Id);
                        modelBuilder.Entity<ItemPedido>().HasOne(t => t.Pedido);
                        modelBuilder.Entity<ItemPedido>().HasOne(t => t.Produto);

                        modelBuilder.Entity<Cadastro>().HasKey(t => t.Id);
                        modelBuilder.Entity<Cadastro>().HasOne(t => t.Pedido);
        }
    }
}
```

Com isso, concluímos o mapeamento e podemos criar uma nova migração. Retornaremos ao console e digitaremos o comando `Add-Migration Modelo`. Pressionaremos a tecla "Enter" para executarmos. O novo pacote será criado. Aplicaremos a nova migração com o comando `Update-Database -verbose`, pressionando a tecla "Enter" para executá-la.

Agora, ao abrir o SQL Object Explorer vemos as tabelas Cadastro, ItemPedido, Pedido e Produto.