o Entity tmb suporta transação

poderá usar os métodos BeginTransaction e UseTransaction,

disponíveis na propriedade Database da classe DbContext.

Suponha que sua aplicação deverá suportar o MySQL. Atualmente o código de LojaContext está acoplado diretamente ao SQL Server. Como podemos tornar LojaContext independente do banco de dados empregado na aplicação?

Usando o princípio da Injeção de Dependência! Criaremos um construtor que receberá um objeto do tipo DbContextOptions<LojaContext> com as opções necessárias para usar o MySQL. Ele chamará o construtor da classe base DbContext através da palavra reservada base, passando as opções como argumento. Para que o código continue compilando, manteremos o construtor sem argumentos.

Quando tanto o construtor como o método OnConfiguring são usados, o último sobrescreve as opções no contexto. Por isso precisamos colocar uma condição no método OnConfiguring perguntando se as opções já foram configuradas, o que é obtido através da verificação da propriedade booleana IsConfigured.

O código final fica assim:

using Microsoft.EntityFrameworkCore;

namespace Alura.Loja.Testes.ConsoleApp

{

public class LojaContext : DbContext

{

public DbSet<Produto> Produtos { get; set; }

public LojaContext()

{ }

public LojaContext(DbContextOptions<LojaContext> options): base(options)

{ }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder

.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=LojaDB;Trusted\_Connection=true;");

}

}

}

}

Mais Informações:

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/configuring-dbcontext>

pode adicionaor produtos assim

contexto.Produtos.Add(p1);

contexto.Produtos.Add(p2);

contexto.Produtos.Add(p3);

contexto.SaveChanges();

ou

contexto.Produtos.AddRange(p1, p2, p3);

contexto.SaveChanges();

´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´´

foreach (var e in contexto.ChangeTracker.Entries())

{

Console.WriteLine(e.State);

}

O comando a cima mostra como o entity **gerencia** seus objetos,

**Added**

O objeto é novo, foi adicionado ao contexto, e o método SaveChanges ainda não foi executado. Depois que as mudanças são feitas, o estado do objeto muda para **Unchanged**. Objetos no estado **Added** não têm seus valores rastreados em sua instância de EntityEntry.

**Deleted**

O objeto foi excluído do contexto. Depois que as mudanças foram salvas, seu estado muda para **Detached**.

**Detached**

O objeto existe, mas não está sendo monitorado. Uma entidade fica nesse estado imediatamente após ter sido criada e antes de ser adicionada ao contexto. Ela também fica nesse estado depois que foi removida do contexto através do método Detach ou se é carregada por um método com opção NoTracking. Não existem instâncias de EntityEntry associadas a objetos com esse estado.

**Modified**

Uma das propriedades escalares do objeto foi modificada e o método SaveChangesainda não foi executado. Quando o monitoramento automático de mudanças está desligado, o estado é alterado para **Modified** apenas quando o método DetectChanges é chamado. Quando as mudanças são salvas, o estado do objeto muda para **Unchanged**.

**Unchanged**

O objeto não foi modificado desde que foi anexado ao contexto ou desde a última vez que o método SaveChanges foi chamado.

**Inserindo o Migration**

Install-Package Microsoft.EntityFrameworkCore.Tools -Version 1.1.1

**Vendo Opções do Migration :**

Get-Help EntityFramework

* Add-Migration
* Remove-Migration
* Update-Database
* Script-Migration
* Drop-Database
* Scaffold-DbContext

Para mostrar o entity que mudas a estrutura do banco teremos que fazer 2 coisas,

1° rodar o comando que avisa-o que mudamos a estrutura

Add-Migration

2° criar o arquivo de script , e passar para o DBA executar a estrutura

Script-Migration

Ou

Update-Database, ele altera de fora imediata

Ná pratica:

Add-Migration nomeDaVersao (ao executar o comando é criado uma pasta no projeto com 2 arquivos, um é o nome que colocamos no parametro add com a data e hora da alteração)

Update-Database –Verbose (vai executar a migração, o Verbose vai mostrar o que o Entity faz por baixo dos panos)No banco foi dado um erro devido ele criar uma tabela com o nome já existia, porém ele criou um tabela com um nome dbo\_EFMMigrationHistority, isso é para controle de versão do entity

No caso do exemplo, antes de fazer os passos a cima, teremos que fazer o seguinte uma coisa antes... a classe Produto está do seguinte jeito

Id - Nome – Categoria – Preco

Ai criamos uma migration (Add-Migration inicial) pronto, vamos na pasta migration, que o comando gera automaticamente, e comentamos o up, depois disso , fazermos a alteracao da class Produto, :  
Id - Nome – Categoria – PrecoUnitario – Unidade  
  
rodamos o Update-Database – inicial (lembrando que temos que comentar o up gerado na pasta inicial) – na vdd ele vai dar um Insert no db com o historico de versão  
  
Após isso, fazeremos novamente o Add-Migration Unidade (antes de fazer isso, temos que descomentar o cód da pasta migration arquivo inicial...) depois podemos rodar o Update-Database (sem parametros)

Criando a class Compra,

public int Id { get; set; }

public double Preco { get; internal set; }

public int ProdutoId { get; set; }

public Produto Produto { get; internal set; }

public int Quantidade { get; internal set; }

colocamos o ProdutoId, devido o Entity colocar que ProdutoId pode ser null, e NESSE modelo, não se pode vender uma compra sem um produto relacionado a ela(compra), (se tivessemos colocado só Produto, automaticamente ele criaria a chave, mas como informei no comeco, ele iria permitir null)

Por convenção, o nome da propriedade que representará o relacionamento será o **nome da classe** seguido da palavra **ID**. Então para representar a classe **Produto** dentro da classe **Compra**, criamos a propriedade **ProdutoID** na classe **Compra**, do tipo **int**, pois não pode ser nulo.

LojaContext : adicionar o seguinte atributo, isso indica para o Entity que ira gerenciar a class tmb  
public DbSet<Compra> Compras { get; set; }

Depois é só gerar os scripits :  
Add-Migration nomeDaVersao  
Update-Database

**Shadow Properties** está escondida do Entity