Sumário’

[Criando um modelo para um banco de dados existente no Entity Framework Core 2](#_Toc77947301)

[Entity Framework Core 2](#_Toc77947302)

[Instalar Entity Framework Core 2](#_Toc77947303)

[Instalando Pacotes 3](#_Toc77947304)

[Microsoft.EntityFrameWorkCore.SqlServer 3](#_Toc77947305)

[Microsoft.EntityFrameWorkCore.Tools 4](#_Toc77947306)

[Instale EF Core Tools para dotnet CLI 5](#_Toc77947307)

[Criando o modelo 6](#_Toc77947308)

[Comando Scaffold-DbContext 7](#_Toc77947309)

# Criando um modelo para um banco de dados existente no Entity Framework Core

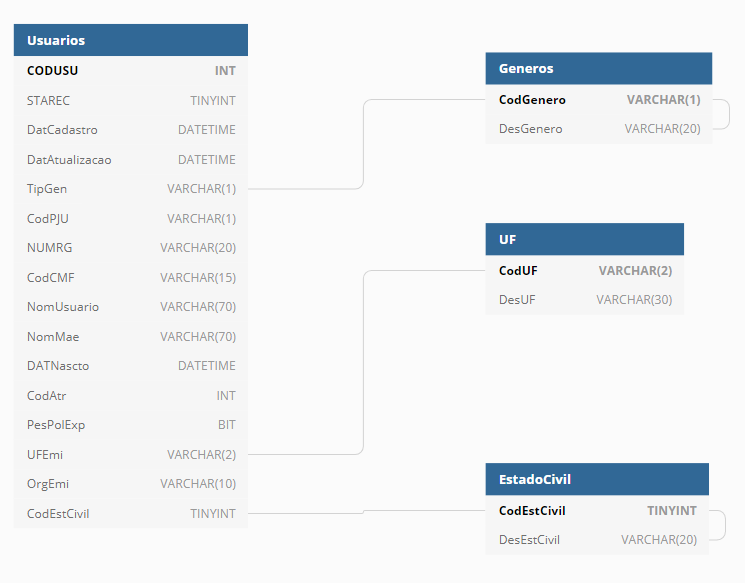
## Entity Framework Core

Entity Framework Core é a nova versão do Entity Framework após EF 6.x. É uma versão de código aberto, leve, extensível e de plataforma cruzada da tecnologia de acesso a dados Entity Framework.

Entity Framework é uma estrutura de Mapeamento Objeto / Relacional (O / RM). É um aprimoramento do ADO.NET que oferece aos desenvolvedores um mecanismo automatizado para acessar e armazenar os dados no banco de dados.

EF Core deve ser usado com aplicativos .NET Core. No entanto, ele também pode ser usado com aplicativos baseados na estrutura .NET 4.5+ padrão.

Diagrama ER da base de dados proposta



# Instalar Entity Framework Core

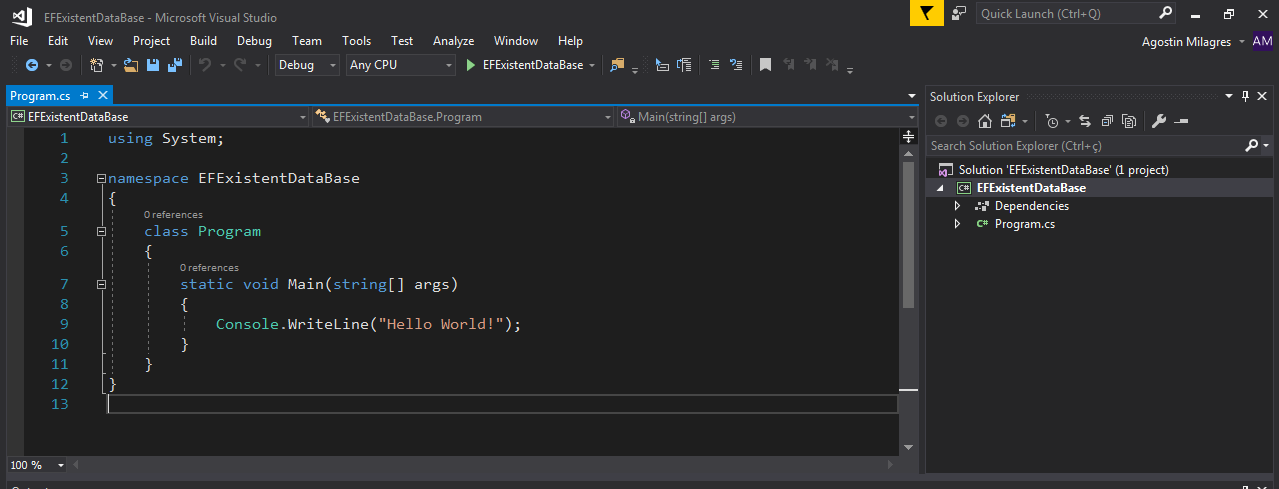
O Entity Framework Core pode ser usado com aplicativos baseados em .NET Core ou .NET 4.6. Aqui, você aprenderá a instalar e usar o Entity Framework Core 2.0 em aplicativos .NET Core usando o Visual Studio 2017.

O EF Core não faz parte do .NET Core e do .NET framework padrão. Ele está disponível como um pacote NuGet. Você precisa instalar os pacotes NuGet para as duas coisas a seguir para usar o EF Core em seu aplicativo:

EF Core DB Provider

EF Core Tools

Vamos instalar os pacotes NuGet acima no aplicativo de console .NET Core no Visual Studio 2017.



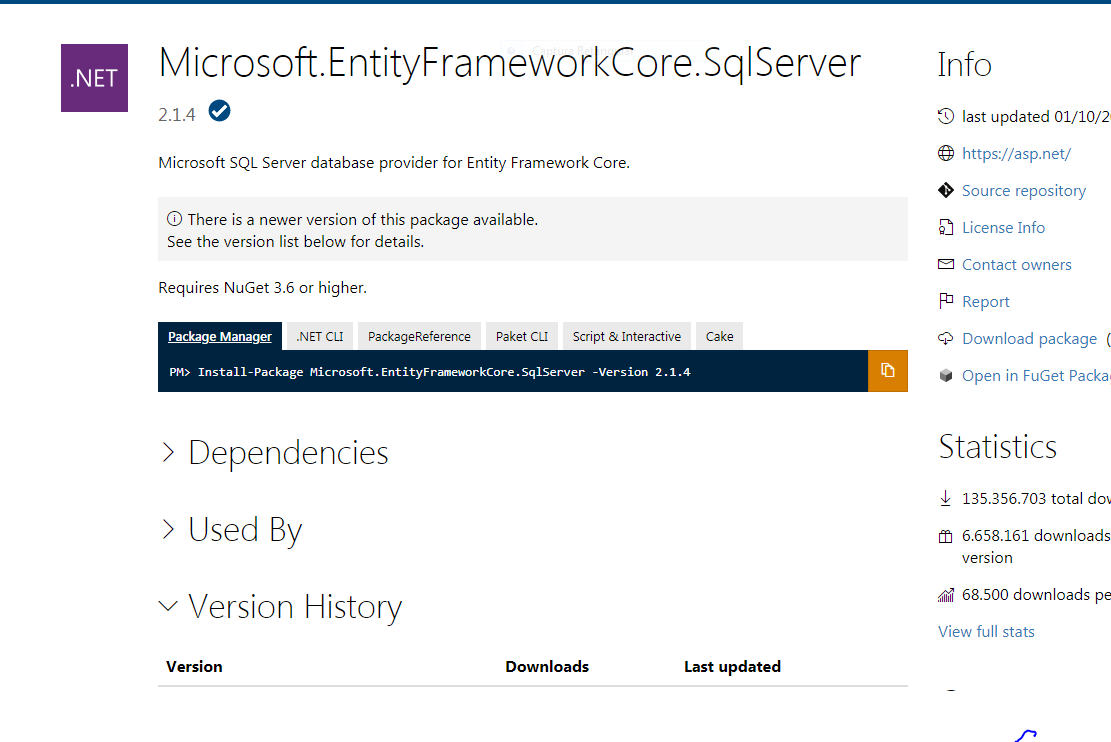
# Instalando Pacotes

## Microsoft.EntityFrameWorkCore.SqlServer

EF Core nos permite acessar bancos de dados através do modelo de provedor. Existem diferentes provedores EF Core DB disponíveis para os diferentes bancos de dados. Esses provedores estão disponíveis como pacotes NuGet.

Primeiro, precisamos instalar o pacote NuGet para o provedor do banco de dados que desejamos acessar. Aqui, queremos acessar o banco de dados MS SQL Server, portanto, precisamos instalar o pacote Microsoft.EntityFrameworkCore.SqlServer.

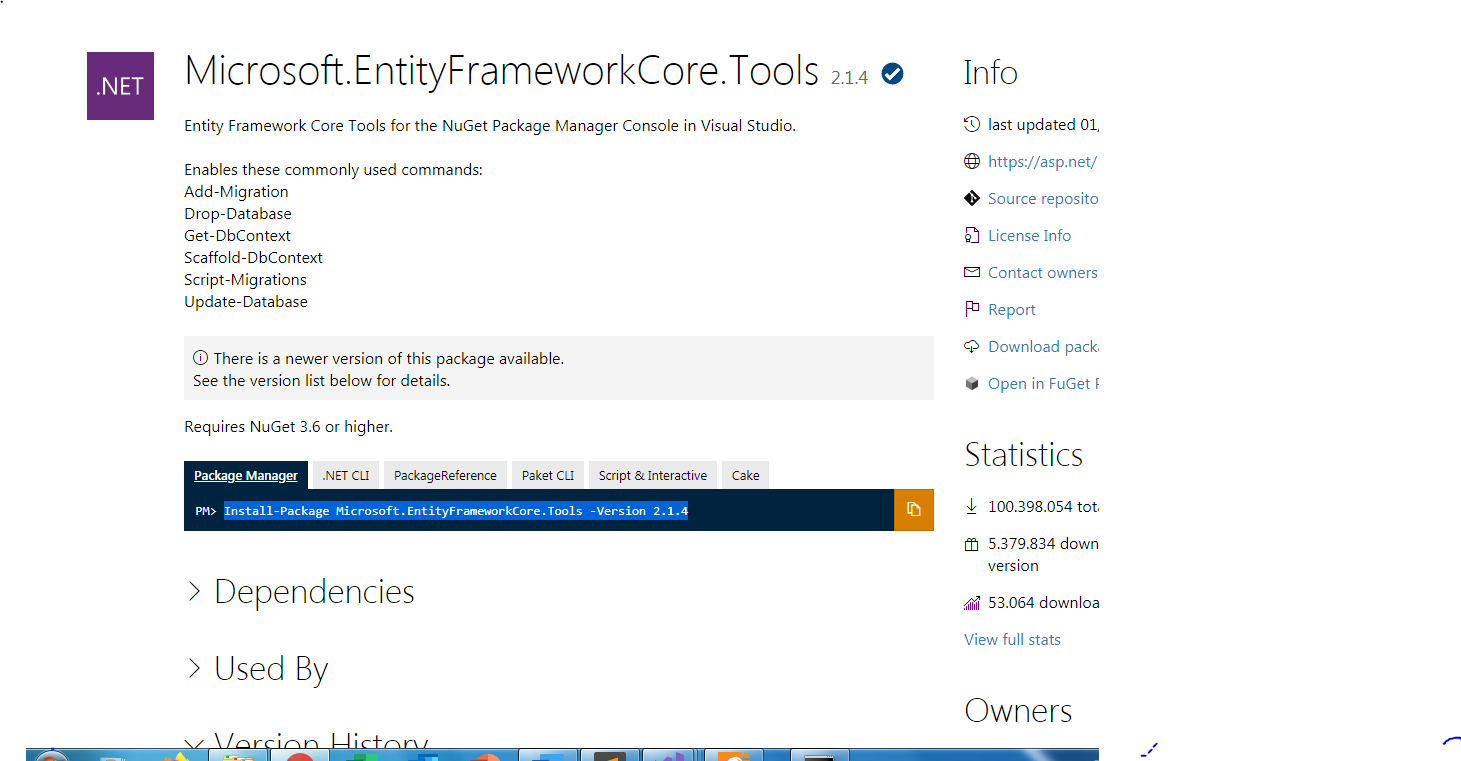
Para instalar o pacote NuGet do provedor de banco de dados, clique com o botão direito do mouse no projeto no Gerenciador de Soluções no Visual Studio e selecione Gerenciar Pacotes NuGet .. (ou selecione no menu: Ferramentas -> Gerenciador de Pacotes NuGet -> Gerenciar Pacotes NuGet para Solução).



## Microsoft.EntityFrameWorkCore.Tools

Junto com o pacote de provedor de banco de dados, você também precisa instalar ferramentas EF para executar comandos EF Core. Isso torna mais fácil realizar várias tarefas relacionadas ao EF Core em seu projeto em tempo de design, como migrações, andaimes, etc.

EF Tools estão disponíveis como pacotes NuGet. Você pode instalar o pacote NuGet para ferramentas EF dependendo de onde deseja executar os comandos: usando o Console do Gerenciador de Pacotes (versão PowerShell dos comandos EF Core) ou usando o dotnet CLI.



# Instale EF Core Tools para dotnet CLI

Se você deseja executar comandos EF Core da CLI (Command Line Interface do .NET Core, primeiro instale o pacote NuGet Microsoft.EntityFrameworkCore.Tools.DotNet usando NuGet UI.

Após instalar o pacote Microsoft.EntityFrameworkCore.Tools.DotNet, edite o arquivo .csproj clicando com o botão direito do mouse no projeto no Solution Explorer e selecione Edit <projectname> .csproj. Adicione o nó <DotNetCliToolReference> conforme mostrado abaixo. Esta é uma etapa extra que você precisa realizar para executar comandos EF Core 2.0 do dotnet CLI no VS2017.

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>netcoreapp2.1</TargetFramework>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.1.4" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.1.4">

**<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.0" />**

<PrivateAssets>all</PrivateAssets>

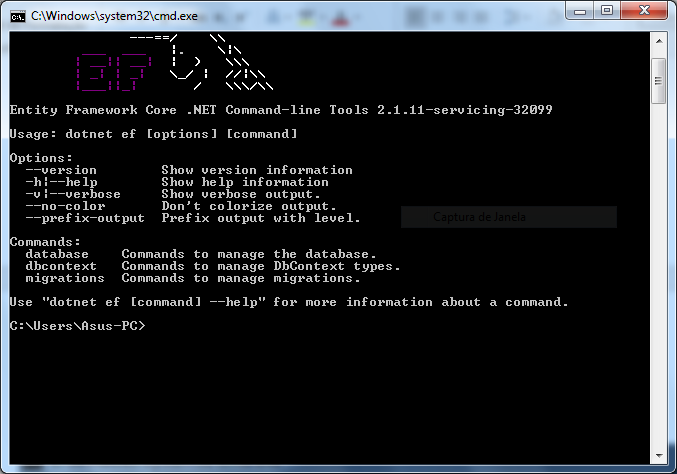
<IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>

</PackageReference>

</ItemGroup>

</Project>

Agora, abra o prompt de comando (ou terminal) da pasta raiz do seu projeto e execute os comandos EF Core da CLI começando com dotnet ef, conforme mostrado abaixo.

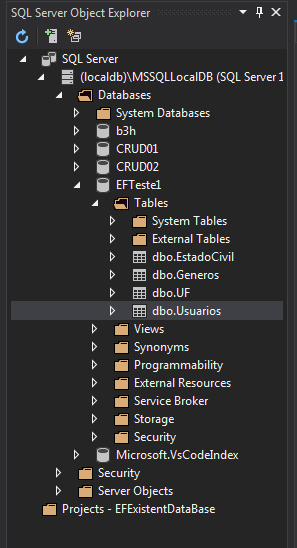


# Criando o modelo

Aqui, você aprenderá como criar o contexto e as classes de entidade para um banco de dados existente no Entity Framework Core. A criação de classes de entidade e contexto para um banco de dados existente é chamada de abordagem **Database-First**.

EF Core não suporta designer visual para modelo de banco de dados e assistente para criar as classes de entidade e contexto semelhantes ao EF 6. Portanto, precisamos fazer engenharia reversa usando o comando **Scaffold-DbContext.** Este comando de engenharia reversa cria classes de entidade e contexto (derivando **DbContext**) com base no esquema do banco de dados existente.

Vamos criar classes de entidade e contexto para o seguinte banco de dados EFTeste1 no MS SQL Server local mostrado abaixo.



# Comando Scaffold-DbContext

Gera código para DbContext tipos de entidade e para um banco de dados. Para Scaffold-DbContext que o gere um tipo de entidade, a tabela de banco de dados deve ter uma chave primária.

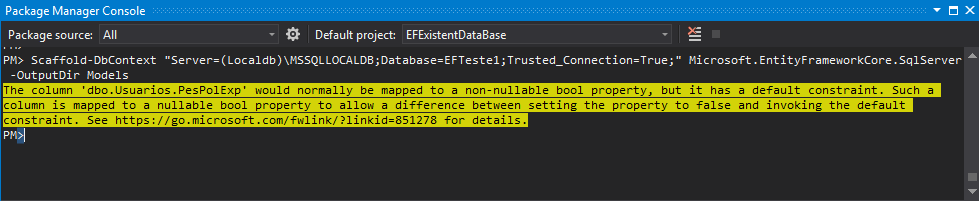
Parâmetros:

|  |  |
| --- | --- |
| **SCAFFOLD-DBCONTEXT** | |
| **Parâmetro** | **Descrição** |
| -Conexão <String> | A cadeia de conexão para o banco de dados. Para projetos ASP.NET Core 2. x, o valor pode ser *nome = <name of connection string>*. Nesse caso, o nome é proveniente das fontes de configuração que estão configuradas para o projeto. Esse é um parâmetro posicional e é necessário. |
| -Provedor <String> | O provedor a ser usado. Normalmente, esse é o nome do pacote NuGet, por exemplo: Microsoft.EntityFrameworkCore.SqlServer . Esse é um parâmetro posicional e é necessário. |
| -OutputDir <String> | O diretório no qual colocar arquivos. Os caminhos são relativos ao diretório do projeto. |
| -ContextDir <String> | O diretório no qual colocar o DbContext arquivo. Os caminhos são relativos ao diretório do projeto. |
| -Namespace <String> | O namespace a ser usado para todas as classes geradas. O padrão é gerado a partir do namespace raiz e do diretório de saída. Adicionado no EF Core 5,0. |
| -ContextNamespace <String> | O namespace a ser usado para a DbContext classe gerada. Observação: substituições -Namespace . Adicionado no EF Core 5,0. |
| -Contexto <String> | O nome da DbContext classe a ser gerada. |
| -Esquemas <String[]> | Os esquemas de tabelas para os quais gerar tipos de entidade. Se esse parâmetro for omitido, todos os esquemas serão incluídos. |
| -Tabelas <String[]> | As tabelas para as quais gerar tipos de entidade. Se esse parâmetro for omitido, todas as tabelas serão incluídas. |
| -Annotations | Use atributos para configurar o modelo (sempre que possível). Se esse parâmetro for omitido, somente a API fluente será usada. |
| -UseDatabaseNames | Use nomes de tabela e coluna exatamente como aparecem no banco de dados. Se esse parâmetro for omitido, os nomes de banco de dados serão alterados para se adequar melhor às convenções de estilo de nome C#. |
| -Force | Substitui os arquivos existentes. |
| -NoOnConfiguring | Não gerar DbContext.OnConfiguring . Adicionado no EF Core 5,0. |
| -Despluralize | Não use o pluralizer. Adicionado no EF Core 5,0. |

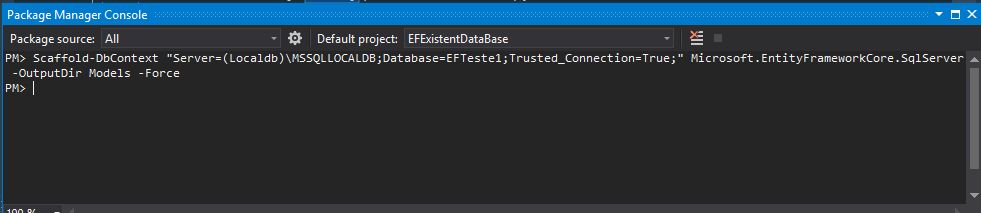
Os [parâmetros comuns](https://docs.microsoft.com/pt-br/ef/core/cli/powershell#common-parameters) são listados acima.

Exemplo:

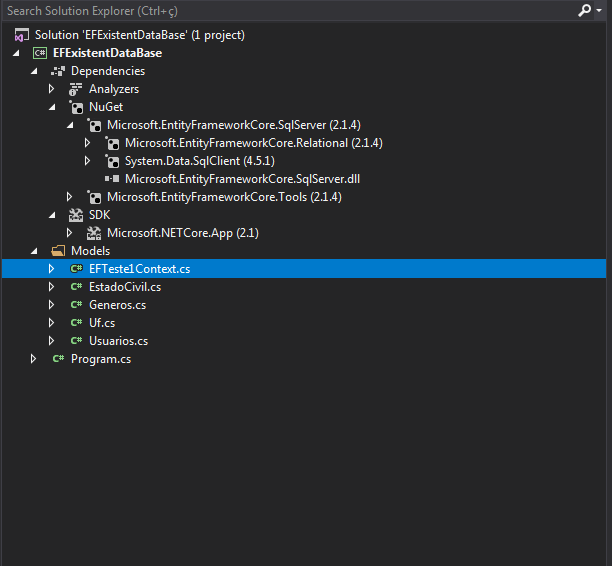
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=EFTeste1;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models



pelo resultado acima, percebemos que houve um alerta sobre a definição de uma determinada coluna do modelo, ao efetuar o ajuste necessário, basta executarmos novamente o mesmo comando acrescido de um parâmetro adicional **-Force.**



O comando Scaffold-DbContext acima cria classes de entidade para cada tabela no banco de dados EFTeste1 e classe de contexto (derivando DbContext) com configurações de Fluent API para todas as entidades na pasta Models.



Script SQL

CREATE TABLE [dbo].[UF] (

[CodUF] VARCHAR (2) NOT NULL,

[DesUF] VARCHAR (30) NOT NULL,

CONSTRAINT [PK\_UF] PRIMARY KEY CLUSTERED ([CodUF] ASC)

);

GO

CREATE TABLE [dbo].[Generos] (

[CodGenero] VARCHAR (1) NOT NULL,

[DesGenero] VARCHAR (20) NOT NULL,

CONSTRAINT [PK\_Generos] PRIMARY KEY CLUSTERED ([CodGenero] ASC)

);

GO

CREATE TABLE [dbo].[EstadoCivil] (

[CodEstCivil] TINYINT NOT NULL,

[DesEstCivil] VARCHAR (20) NOT NULL,

CONSTRAINT [PK\_EstadoCivil] PRIMARY KEY CLUSTERED ([CodEstCivil] ASC)

);

GO

CREATE TABLE [dbo].[Usuarios] (

[CODUSU] INT NOT NULL,

[STAREC] TINYINT DEFAULT (1) NOT NULL,

[DatCadastro] DATETIME DEFAULT (getdate()) NOT NULL,

[DatAtualizacao] DATETIME DEFAULT (getdate()) NOT NULL,

[TipGen] VARCHAR (1) NOT NULL,

[CodPJU] VARCHAR (1) DEFAULT ('I') NOT NULL,

[NUMRG] VARCHAR (20) NOT NULL,

[CodCMF] VARCHAR (15) NOT NULL,

[NomUsuario] VARCHAR (70) NOT NULL,

[NomMae] VARCHAR (70) NOT NULL,

[DATNascto] DATETIME NOT NULL,

[CodAtr] INT DEFAULT (1) NOT NULL,

[PesPolExp] BIT DEFAULT (0) NOT NULL,

[UFEmi] VARCHAR (2) NOT NULL,

[OrgEmi] VARCHAR (10) NOT NULL,

[CodEstCivil] TINYINT DEFAULT (0) NOT NULL,

CONSTRAINT [PK\_Usuarios] PRIMARY KEY CLUSTERED (CODUSU ASC),

CONSTRAINT [FK\_Usuarios\_EstadoCivil] FOREIGN KEY ([CodEstCivil]) REFERENCES [dbo].[EstadoCivil] ([CodEstCivil]),

CONSTRAINT [FK\_Usuarios\_Generos] FOREIGN KEY ([TipGen]) REFERENCES [dbo].[Generos] ([CodGenero]),

CONSTRAINT [FK\_Usuarios\_UF] FOREIGN KEY ([UFEmi]) REFERENCES [dbo].[UF] ([CodUF])

);

GO