Entity Framework Core: Mapeando bancos de dados com Database First

[Facebook](https://www.facebook.com/sharer/sharer.php?u=http://www.devmedia.com.br/entity-framework-core-mapeando-bancos-de-dados-com-database-first/36814" \o "Compartilhe no Facebook" \t "_blank) [Twitter](https://twitter.com/home?status=Entity%20Framework%20Core:%20Mapeando%20bancos%20de%20dados%20com%20Database%20First%0D%0Ahttp://www.devmedia.com.br/entity-framework-core-mapeando-bancos-de-dados-com-database-first/36814" \o "Compartilhe no Twitter" \t "_blank)

 (5)  (0)

Neste artigo veremos como efetuar a engenharia reversa de uma base de dados já existente e mapeá-la com o Entity Framework Core, utilizando Fluent API e/ou Data Annotations através de ferramentas de linha de comando.

**Motivação**

Adequar uma aplicação orientada a objetos a uma base de dados já existente pode se mostrar um processo trabalhoso, capaz de tomar mais da metade do tempo de desenvolvimento. Esse tipo de procedimento normalmente acontece em projetos que estão em execução ou em projetos legados, e a principal tarefa nesses casos é realizar o mapeamento e adequação das tabelas físicas em entidades lógicas (classes).

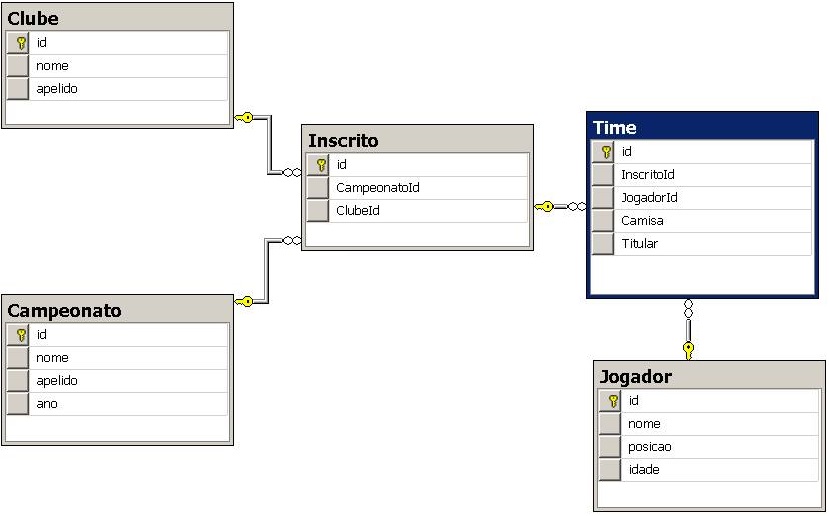
Por meio do modelo Database First (em que se parte de um banco de dados já existente para, a partir dele, gerar as classes do sistema) o Entity Framework já era capaz de auxiliar nesse processo desde suas versões anteriores, porém, o EF Core foi otimizado e agora oferece suporte a múltiplas plataformas (Linux, Mac OS e Windows).

Nesse artigo utilizaremos o Entity Framework Core na versão 1.0.0, enquanto as suas ferramentas de linha de comando estarão na versão 1.0.0-preview2.

http://www.devmedia.com.br/layout/icon/information-button.png Saiba mais sobre o [Entity Framework Core e o modelo Code First](http://www.devmedia.com.br/entity-framework-core-criando-bases-de-dados-com-migrations/36776" \t "_blank).

**A base de dados**

Na **Figura 1** podemos ver o diagrama com as tabelas da nossa base de dados de exemplo, que nesse caso se chama *PlantelDb*.

  
**Figura 1.** Base de dados de exemplo

Aqui utilizamos o SQL Server, no entanto, o EF Core é capaz de efetuar os mesmos procedimentos em outros bancos de dados, como MySQL e PostgreSQL.

**Configurando o projeto e suas dependências**

Para iniciar nosso projeto devemos criar um diretório (chamado Ef7BaseExistente, nesse exemplo) e dentro dele criar uma nova Class Library, chamada Projeto.Repositorio.

Como o Entity Framework Core não permite que uma Class Library receba o conteúdo gerado pelo procedimento de engenharia reversa - de acordo com a documentação oficial, apenas projetos que contêm um método Main podem ser utilizados para esse fim -, precisamos implementar esse método.

Para isso, vamos adicionar uma classe chamada Principal.cs na raiz do projeto e nela implementar um método Main vazio. Veja o código apresentado na **Listagem 1**.

|  |
| --- |
| 01 namespace Projeto.Repositorio  02 {  03    class Principal  04    {  05        static void Main(string[] args)  06        { }  07    }  08 } |

**Listagem 1.** Classe Principal.cs

Em seguida, é preciso instalar os pacotes necessários para que seja possível utilizar a instrução de linha de comandos ef, que é responsável por realizar os procedimentos relativos ao Entity Framework Core.

Para isso, em projetos do tipo Class Library, devemos adicionar o pacote Microsoft.EntityFrameworkCore.Design na seção dependencies do arquivo *project.json*, enquanto que em projetos de interação com o usuário, como MVC ou Console, devemos utilizar o pacote Microsoft.EntityFrameworkCore.Tools.

Na **Listagem 2** podemos ver como deve ficar o arquivo *project.json*.

|  |
| --- |
| 01 {  02  "version": "1.0.0",  03  "authors": ["Gabriel Simas"],  04  "description": "Camada de Infraestutura da Aplicação",  05  "copyright": "DevMedia 2016 - Todos os direitos reservados.",  06  07  "buildOptions": {  08    "emitEntryPoint": true,  09    "optimize": true  10  },  11  12  "dependencies": {  13    "Microsoft.NETCore.App": {  14      "version": "1.0.0",  15      "type": "platform"  16    },  17    "Microsoft.EntityFrameworkCore.SqlServer":"1.0.0",  18    "Microsoft.EntityFrameworkCore.Design" :"1.0.0-preview2-final",  19    "Microsoft.EntityFrameworkCore.SqlServer.Design" : "1.0.0"  20  },  21  22 "tools": {  23  "Microsoft.EntityFrameworkCore.Tools" : "1.0.0-preview2-final"  24 },  25  26  "frameworks": {  27  "netcoreapp1.0": {  28      "imports": [  29        "dotnet5.6",  30        "dnxcore50",  31        "portable-net45+win8"  32      ]  33    }  34 },  35  36  "tooling": {  37    "defaultNamespace": "Projeto.Repositorio"  38  }  39 } |

**Listagem 2.** Configuração das dependências necessárias

Linhas 12 a 20: definição dos pacotes dos quais a aplicação depende para realizar o mapeamento das tabelas do SQL Server em entidades do projeto;

Linhas 22 a 24: configuração das ferramentas que serão necessárias nesse projeto. Nesse caso, trata-se da ferramenta de linha de comando ef.

Agora, podemos executar o comando de restauração e instalação dos pacotes. Para isso, digite a seguinte instrução no prompt de comandos, estando dentro do diretório do projeto:

|  |
| --- |
| dotnet restore |

**Database First via linha de comandos**

Para efetuar a engenharia reversa da base de dados o EF Core requer a execução de um comando bastante simples, cuja estrutura nos permite utilizar apenas as opções de que precisamos. Sua sintaxe é a seguinte:

|  |
| --- |
| dotnet ef dbcontext scaffold [argumentos] [opções] |

Em [argumentos] devemos informar a string de conexão do banco de dados, seguida do provedor a ser utilizado, como o Microsoft.EntityFrameworkCore.SqlServer.

Já em [opções], podemos informar alguns parâmetros adicionais que definirão comportamentos mais específicos, como a forma de mapeamento, se via Fluent API ou Data Annotations, como veremos a seguir.

**Mapeamento com Fluent API**

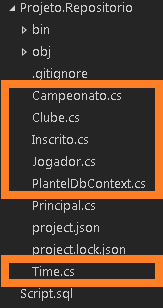
Inicialmente, vamos executar o EF com as opções padrão, informando apenas a string de conexão e o provedor de dados, como mostra o comando abaixo:

|  |
| --- |
| dotnet ef dbcontext scaffold "Data Source=SERVIDOR\INSTANCIA;Initial Catalog=PlantelDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer |

*Observação: É necessário que estejamos dentro do diretório do projeto, pois a ferramenta dotnet, executada via linha de comando, irá procurar o arquivo project.json.*

Com essa instrução, direcionamos a criação dos objetos à raiz do projeto e o mapeamento será feito via FluentAPI, que é o comportamento padrão.

Caso tudo ocorra sem problemas, receberemos uma mensagem de “Done” ao final do processo e, observando nosso projeto, veremos que algumas classes foram criadas, como mostra a **Figura 2**.

  
**Figura 2.** Base de dados de exemplo

Como podemos notar, foram geradas as entidades (Campeonato, Inscrito, Jogador e Time) e a classe PlantelDbContext, essa última herdando de DbContext.

Na **Listagem 3** temos um trecho da classe PlantelDbContex, no qual podemos ver um exemplo de mapeamento (da entidade Jogador) realizado via Fluent API.

|  |
| --- |
| modelBuilder.Entity<Jogador>(entity =>  {      entity.Property(e => e.Id).HasColumnName("id");      entity.Property(e => e.Idade).HasColumnName("idade");      entity.Property(e => e.Nome)      .IsRequired()      .HasColumnName("nome")      .HasColumnType("varchar(100)");      entity.Property(e => e.Posicao)      .HasColumnName("posicao")      .HasColumnType("varchar(30)");  }); |

**Listagem 3.** Exemplo de mapeamento resultante com Fluent API

**Mapeamento com Data Annotations**

Para alterar o méteodo de mapeamento de FluentAPI para o Data Annotations, basta adicionar o parâmetro –a ou --data-annotations como opção no comando que vimos anteriormente. Veja um exemplo na instrução abaixo:

|  |
| --- |
| dotnet ef dbcontext scaffold "Data Source=SERVIDOR\INSTANCIA;Initial Catalog=PlantelDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer –a [ou --data-annotations] --force |

Nesse caso, utilizamos ainda o parâmetro --force, para evitar que o ef lance uma exceção de “objeto já criado”, devido a já termos as classses no projeto, e termine o processo.

Na **Listagem 4** temos um exemplo de entidade gerada, dessa vez com as anotações sobre os atributos.

|  |
| --- |
| public partial class Jogador  {      public Jogador()      {          Time = new HashSet<Time>();      }        [Column("id")]      public int Id { get; set; }      [Required]      [Column("nome", TypeName = "varchar(100)")]      public string Nome { get; set; }      [Column("posicao", TypeName = "varchar(30)")]      public string Posicao { get; set; }      [Column("idade")]      public int? Idade { get; set; }        [InverseProperty("Jogador")]      public virtual ICollection<Time> Time { get; set; }  } |

**Listagem 4.** Exemplo de mapeamento resultante com Data Annotations

Como foi possível verificar, por meio de uma sintaxe simples e flexível o Entity Framework Core facilita a adaptação de projetos a bases de dados já existentes, realizando o mapeamento das tabelas para classes do sistema.

http://www.devmedia.com.br/layout/icon/information-button.png Saiba mais sobre [Data Annotations](http://www.devmedia.com.br/asp-net-mvc-data-annotations-e-custom-validation/30751).