



A <PROGRAMAÇÃO> PRECISA DE TI

DB Scaffolding in EF Core

Laboratório Web

Scaffolding?

- É uma técnica suportada por algumas frameworks MVC para gerar código de forma automática
- Scaffold significa andaime, é uma analogia para a estrutura de construção do projeto
- Aqui o objetivo é efetuar scaffolding a uma base de dados existente para criar um modelo EF
- O contexto e as entidades são gerados de forma automática

Como fazer scaffolding de uma DB?

- Podemos fazê-lo de 2 formas:
 - .NET Core CLI
 - Package Manager Console

.NET Core CLI

1. Criar o projeto através da consola
 - `dotnet new console -o ProjectName`
 - `cd ProjectName`
2. Adicionar os pacotes necessários
 - `dotnet add package MySql.EntityFrameworkCore`
 - `dotnet add package Microsoft.EntityFrameworkCore`
 - `dotnet add package Microsoft.EntityFrameworkCore.Tools`
3. Restaurar as dependências
 - `dotnet restore`
4. Criar o modelo da Entity Framework
 - `dotnet ef dbcontext scaffold "connection-string" MySql.EntityFrameworkCore -o Models -f`

.NET Core CLI

- **dotnet ef dbcontext scaffold** -> Comando para gerar o modelo
- **"connection-string"** -> String com os dados da ligação à base de dados
 - `"server=localhost;database=library;"user=root;password=password"`
- **MySQL.EntityFrameworkCore** -> Provider a utilizar, tipicamente é o nome do pacote NuGet e reflete o provider da base de dados
- **-o Models -f** -> Definimos a pasta de OUTPUT e forçamos a escrita de ficheiros existentes

Package Manager Console

1. Criar um novo projeto no Visual Studio
2. Abrir o Package Manager Console
3. Instalar todas as dependências necessárias (Pode ser através do Nuget Manager ou do comando Install-Package)
 - MySql.EntityFrameworkCore
 - Microsoft.EntityFrameworkCore
 - Microsoft.EntityFrameworkCore.Tools
4. Executar o seguinte comando
 - Scaffold-DbContext "connection-string" MySql.EntityFrameworkCore - OutputDir Models -f

Query data

- **Load all data**

```
var blogs = context.Blogs.ToList();
```

- **Load single entity**

```
var blog = context.Blogs.Single(b => b.BlogId == 1);
```

- **Filtering**

```
var blogs = context.Blogs  
    .Where(b => b.Url.Contains("dotnet"))  
    .ToList();
```


Load related data (Eager Loading)

- **Single relationship**

```
var blogs = context.Blogs  
.Include(blog => blog.Posts)
```

- **Multiple relationships**

```
var blogs = context.Blogs  
.Include(blog => blog.Posts)  
.Include(blog => blog.Owner)
```

- **Multiple Levels**

```
var blogs = context.Blogs  
.Include(blog => blog.Posts)  
.ThenInclude(post => post.Author)
```


Load related data (Filtered)

- Podemos utilizar outras operações quando estamos a incluir dados relacionados:
 - **Where, OrderBy, OrderByDescending, ThenBy, ThenByDescending, Skip, e Take**

```
using (var context = new BloggingContext())
{
    var filteredBlogs = context.Blogs
        .Include(
            blog => blog.Posts
                .Where(post => post.BlogId == 1)
                .OrderByDescending(post => post.Title)
                .Take(5))
        .ToList();
}
```

Save data (single operations)

- **Adding data**

```
context.Blogs.Add(blog);  
context.SaveChanges();
```

- **Update data**

```
var blog = context.Blogs.First();  
blog.Url = "http://example.com/blog";  
context.SaveChanges();
```

- **Delete data**

```
var blog = context.Blogs.First();  
context.Blogs.Remove(blog);  
context.SaveChanges();
```

Save data (multiple operations)

```
using (var context = new BloggingContext())
{
    // seeding database
    context.Blogs.Add(new Blog { Url = "http://example.com/blog" });
    context.Blogs.Add(new Blog { Url = "http://example.com/another_blog" });
    context.SaveChanges();
}

using (var context = new BloggingContext())
{
    // add
    context.Blogs.Add(new Blog { Url = "http://example.com/blog_one" });
    context.Blogs.Add(new Blog { Url = "http://example.com/blog_two" });

    // update
    var firstBlog = context.Blogs.First();
    firstBlog.Url = "";

    // remove
    var lastBlog = context.Blogs.OrderBy(e => e.BlogId).Last();
    context.Blogs.Remove(lastBlog);

    context.SaveChanges();
}
```

Cascade delete

- **EF Core** representa relações entre as tabelas utilizando as chaves estrangeiras
- Quando removemos uma entidade “parent” todas as chaves estrangeiras das entidades “child” deixam de ter significado
- Nestas situações temos 2 opções:
 - Alterar o valor das chaves estrangeiras para null (se forem marcadas como nullable)
 - Apagar todas as entidades child (cascade delete)
- Se as chaves estrangeiras forem marcadas como non-nullable, num delete de um parent todas as entidades child serão apagadas

Cascade delete

```
public class Blog
{
    public int Id { get; set; }

    public string Name { get; set; }

    public IList<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public int Id { get; set; }

    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

Cascade delete

```
using var context = new BlogsContext();  
  
var blog = context.Blogs.OrderBy(e => e.Name).Include(e => e.Posts).First();  
  
context.Remove(blog);  
  
context.SaveChanges();
```


Severing a relationship

- Colocar a entidade parent como null:

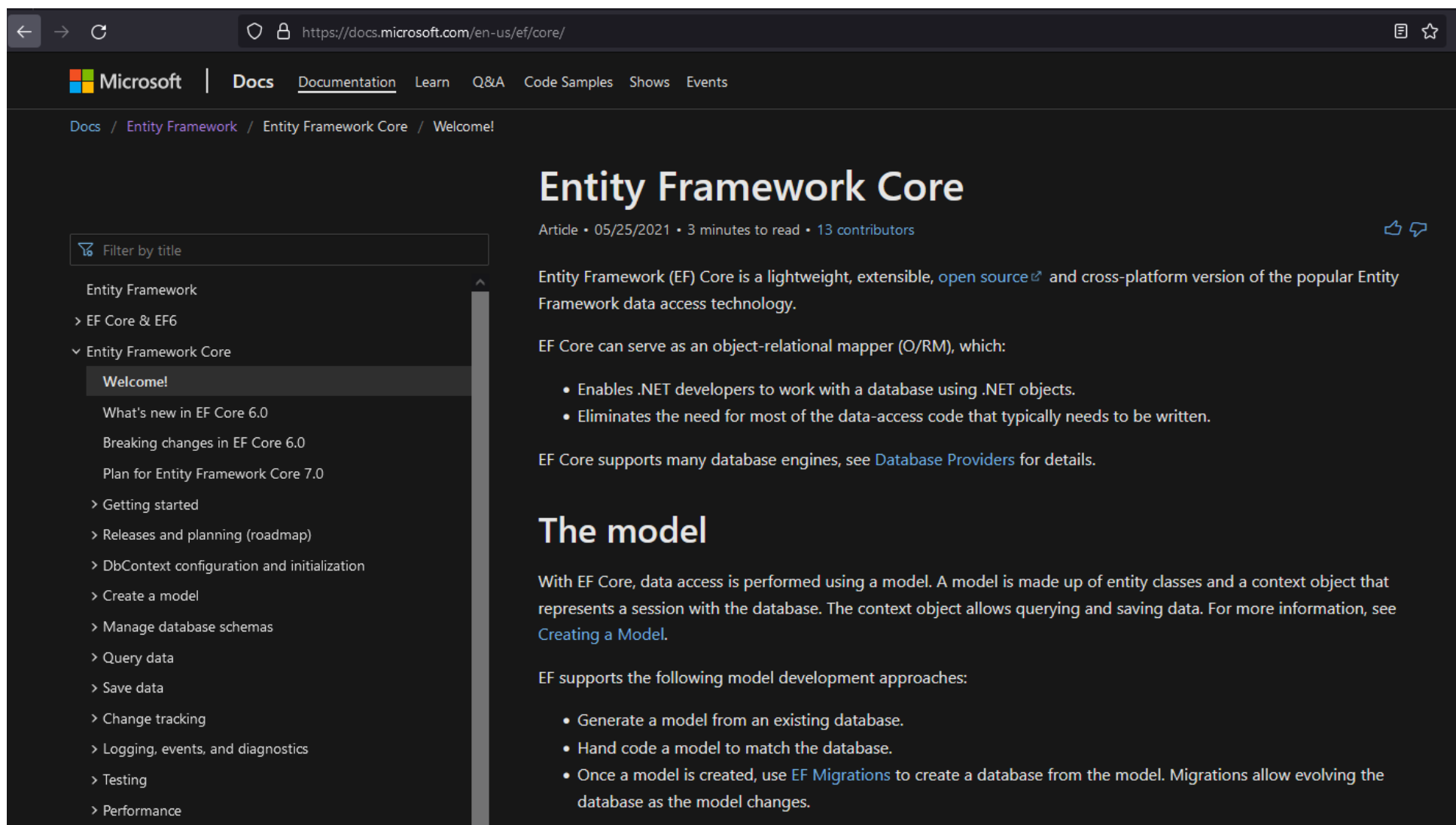
```
using var context = new BlogsContext();  
  
var blog = context.Blogs.OrderBy(e => e.Name).Include(e => e.Posts).First();  
  
foreach (var post in blog.Posts)  
{  
    post.Blog = null;  
}  
  
context.SaveChanges();
```


Severing a relationship

- Limpar a lista de entidades child no parent:

```
using var context = new BlogsContext();  
  
var blog = context.Blogs.OrderBy(e => e.Name).Include(e => e.Posts).First();  
  
blog.Posts.Clear();  
  
context.SaveChanges();
```

Referências



The screenshot shows the Microsoft Docs website for Entity Framework Core. The browser address bar displays <https://docs.microsoft.com/en-us/ef/core/>. The navigation bar includes the Microsoft logo and links to Docs, Documentation, Learn, Q&A, Code Samples, Shows, and Events. The breadcrumb trail reads: Docs / Entity Framework / Entity Framework Core / Welcome!.

Entity Framework Core

Article • 05/25/2021 • 3 minutes to read • 13 contributors

Entity Framework (EF) Core is a lightweight, extensible, [open source](#) and cross-platform version of the popular Entity Framework data access technology.

EF Core can serve as an object-relational mapper (O/RM), which:

- Enables .NET developers to work with a database using .NET objects.
- Eliminates the need for most of the data-access code that typically needs to be written.

EF Core supports many database engines, see [Database Providers](#) for details.

The model

With EF Core, data access is performed using a model. A model is made up of entity classes and a context object that represents a session with the database. The context object allows querying and saving data. For more information, see [Creating a Model](#).

EF supports the following model development approaches:

- Generate a model from an existing database.
- Hand code a model to match the database.
- Once a model is created, use [EF Migrations](#) to create a database from the model. Migrations allow evolving the database as the model changes.

Left sidebar navigation:

- Filter by title
- Entity Framework
 - > EF Core & EF6
 - ▼ Entity Framework Core
 - Welcome!**
 - What's new in EF Core 6.0
 - Breaking changes in EF Core 6.0
 - Plan for Entity Framework Core 7.0
 - > Getting started
 - > Releases and planning (roadmap)
 - > DbContext configuration and initialization
 - > Create a model
 - > Manage database schemas
 - > Query data
 - > Save data
 - > Change tracking
 - > Logging, events, and diagnostics
 - > Testing
 - > Performance

qualificar

TAL

X

</>

A <PROGRAMAÇÃO