عنوان: EF Code First #3
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۷:۳۷:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

بررسی تعاریف نگاشت‌ها به کمک متادیتا در EF Code first

در قسمت قبل مروری سطحی داشتیم بر امکانات مهیای جهت تعاریف نگاشت‌ها در EF Code first. در این قسمت، حالت استفاده از متادیتا یا همان data annotations را با جزئیات بیشتری بررسی خواهیم کرد.
برای این منظور پروژه کنسول جدیدی را آغاز نمائید. همچنین به کمک NuGet، ارجاعات لازم را به اسمبلی EF، اضافه کنید. در ادامه مدل‌های زیر را به پروژه اضافه نمائید؛ یک شخص که تعدادی پروژه منتسب می‌تواند داشته باشد:

```
using System;
using System.Collections.Generic;

namespace EF_Sample02.Models
{
    public class User
    {
        public int Id { set; get; }
        public DateTime AddDate { set; get; }
        public string Name { set; get; }
        public string LastName { set; get; }
        public string Email { set; get; }
        public string Description { set; get; }
        public byte[] Photo { set; get; }
        public IList<Project> Projects { set; get; }
    }
}
```

```
using System;

namespace EF_Sample02.Models
{
    public class Project
    {
        public int Id { set; get; }
        public DateTime AddDate { set; get; }
        public string Title { set; get; }
        public string Description { set; get; }
        public virtual User User { set; get; }
    }
}
```

به خاصیت public virtual User User در کلاس Project اصطلاحا Navigation property هم گفته می‌شود.
دو کلاس زیر را نیز جهت تعریف کلاس Context که بیانگر کلاس‌های شرکت کننده در تشکیل بانک اطلاعاتی هستند و همچنین کلاس آغاز کننده بانک اطلاعاتی سفارشی را به همراه تعدادی رکورد پیش فرض مشخص می‌کنند، به پروژه اضافه نمائید.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using EF_Sample02.Models;

namespace EF_Sample02
{
    public class Sample2Context : DbContext
    {
```

```
        public DbSet<User> Users { set; get; }
        public DbSet<Project> Projects { set; get; }
    }

    public class Sample2DbInitializer : DropCreateDatabaseAlways<Sample2Context>
    {
        protected override void Seed(Sample2Context context)
        {
            context.Users.Add(new User
            {
                AddDate = DateTime.Now,
                Name = "Vahid",
                LastName = "N.",
                Email = "name@site.com",
                Description = "-",
                Projects = new List<Project>
                {
                    new Project
                    {
                        Title = "Project 1",
                        AddDate = DateTime.Now.AddDays(-10),
                        Description = "..."
                    }
                }
            });

            base.Seed(context);
        }
    }
}
```

به علاوه در فایل کانفیگ برنامه، تنظیمات رشته اتصالی را نیز اضافه نمائید:

```
<connectionStrings>
    <add
       name="Sample2Context"
       connectionString="Data Source=(local);Initial Catalog=testdb2012;Integrated Security = true"
       providerName="System.Data.SqlClient"
      />
  </connectionStrings>
```

همانطور که ملاحظه می‌کنید، در اینجا name به نام کلاس مشتق شده از DbContext اشاره می‌کند (یکی از قراردادهای توکار EF Code first است).

یک نکته:
مرسوم است کلاس‌های مدل را در یک class library جداگانه اضافه کنند به نام DomainClasses و کلاس‌های مرتبط با DbContext را در پروژه class library دیگری به نام DataLayer. هیچکدام از این پروژه‌ها نیازی به فایل کانفیگ و تنظیمات رشته اتصالی ندارند؛ زیرا اطلاعات لازم را از فایل کانفیگ پروژه اصلی که این دو پروژه class library را به خود الحاق کرده، دریافت می‌کنند. دو پروژه class library اضافه شده تنها باید ارجاعاتی را به اسمبلی‌های EF و data annotations داشته باشند.

در ادامه به کمک متد Database.SetInitializer که در قسمت دوم به بررسی آن پرداختیم و با استفاده از کلاس سفارشی Sample2DbInitializer فوق، نسبت به ایجاد یک بانک اطلاعاتی خالی تشکیل شده بر اساس تعاریف کلاس‌های دومین پروژه، اقدام خواهیم کرد:

```
using System;
using System.Data.Entity;

namespace EF_Sample02
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new Sample2DbInitializer());
            using (var db = new Sample2Context())
            {
                var project1 = db.Projects.Find(1);
                Console.WriteLine(project1.Title);
            }
        }
    }
}
```

تا زمانیکه وهله‌ای از Sample2Context ساخته نشود و همچنین یک کوئری نیز به بانک اطلاعاتی ارسال نگردد، Sample2DbInitializer در عمل فراخوانی نخواهد شد.

ساختار بانک اطلاعاتی پیش فرض تشکیل شده نیز مطابق اسکریپت زیر است:

```
CREATE TABLE [dbo].[Users](
 [Id] [int] IDENTITY(1,1) NOT NULL,
 [AddDate] [datetime] NOT NULL,
 [Name] [nvarchar](max) NULL,
 [LastName] [nvarchar](max) NULL,
 [Email] [nvarchar](max) NULL,
 [Description] [nvarchar](max) NULL,
 [Photo] [varbinary](max) NULL,
 CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
 [Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
 IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
CREATE TABLE [dbo].[Projects](
 [Id] [int] IDENTITY(1,1) NOT NULL,
 [AddDate] [datetime] NOT NULL,
 [Title] [nvarchar](max) NULL,
 [Description] [nvarchar](max) NULL,
 [User_Id] [int] NULL,
 CONSTRAINT [PK_Projects] PRIMARY KEY CLUSTERED
(
 [Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Projects]  WITH CHECK ADD  CONSTRAINT [FK_Projects_Users_User_Id] FOREIGN
KEY([User_Id])
REFERENCES [dbo].[Users] ([Id])
GO

ALTER TABLE [dbo].[Projects] CHECK CONSTRAINT [FK_Projects_Users_User_Id]
GO
```

توضیحاتی در مورد ساختار فوق، جهت یادآوری مباحث دو قسمت قبل:

- خواصی با نام Id تبدیل به primary key و identity field شده‌اند.
- نام جداول، همان نام خواص تعریف شده در کلاس Context است.
- تمام رشته‌ها به nvarchar از نوع max نگاشت شده‌اند و null پذیر می‌باشند.
- خاصیت تصویر که با آرایه‌ای از بایت‌ها تعریف شده به varbinary از نوع max نگاشت شده است.
- بر اساس ارتباط بین کلاس‌ها فیلد User_Id در جدول Projects اضافه شده است که توسط قیدی به نام FK_Projects_Users_User_Id، جهت تعریف کلید خارجی عمل می‌کند. این نام گذاری پیش فرض هم بر اساس نام خواص در دو

کلاس انجام می‌شود.
- schema پیش فرض بکارگرفته شده، dbo است.
- null پذیری پیش فرض فیلدها بر اساس اصول زبان مورد استفاده تعیین شده است. برای مثال در سی شارپ، نوع int نال پذیر نیست یا نوع DateTime نیز به همین ترتیب یک value type است. بنابراین در اینجا این دو نوع به صورت not null تعریف شده‌اند (صرفنظر از اینکه در SQL Server هر دو نوع یاد شده، null پذیر هم می‌توانند باشند). بدیهی است امکان تعریف nullable types نیز وجود دارد.

مروری بر انواع متادیتای قابل استفاده در EF Code first

1) Key
همانطور که ملاحظه کردید اگر نام خاصیتی Id یا ClassName+Id باشد، به صورت خودکار به عنوان primary key جدول، مورد استفاده قرار خواهد گرفت. این یک قرارداد توکار است.
اگر یک چنین خاصیتی با نام‌های ذکر شده در کلاس وجود نداشته باشد، می‌توان با مزین سازی خاصیتی مفروض با ویژگی Key که در فضای نام System.ComponentModel.DataAnnotations قرار دارد، آن‌را به عنوان Primary key معرفی نمود. برای مثال:

```
public class Project
{
     [Key]
     public int ThisIsMyPrimaryKey { set; get; }
```

و ضمنا باید دقت داشت که حین کار با ORMs فرقی نمی‌کند EF باشد یا سایر فریم ورک‌های دیگر، داشتن یک key جهت عملکرد صحیح فریم ورک، ضروری است. بر اساس یک Key است که Entity معنا پیدا می‌کند.

2) Required
ویژگی Required که در فضای نام System.ComponentModel.DataAnnotations تعریف شده است، سبب خواهد شد یک خاصیت به صورت not null در بانک اطلاعاتی تعریف شود. همچنین در مباحث اعتبارسنجی برنامه، پیش از ارسال اطلاعات به سرور نیز نقش خواهد داشت. در صورت نال بودن خاصیتی که با ویژگی Required مزین شده است، یک استثنای اعتبارسنجی پیش از ذخیره سازی اطلاعات در بانک اطلاعاتی صادر می‌گردد. این ویژگی علاوه بر EF Code first در ASP.NET MVC نیز به نحو یکسانی تاثیرگذار است.

3) MaxLength و MinLength
این دو ویژگی نیز در فضای نام System.ComponentModel.DataAnnotations قرار دارند (اما در اسمبلی EntityFramework.dll تعریف شده‌اند و جزو اسمبلی پایه System.ComponentModel.DataAnnotations.dll نیستند). در ذیل نمونه‌ای از تعریف این‌ها را مشاهده می‌کنید. همچنین باید درنظر داشت که روش دیگر تعریف متادیتا، ترکیب آن‌ها در یک سطر نیز می‌باشد. یعنی الزامی ندارد در هر سطر یک متادیتا را تعریف کرد:

```
[MaxLength(50, ErrorMessage = "حداکثر 50 حرف"), MinLength(4, ErrorMessage = "حداقل 4 حرف")]
public string Title { set; get; }
```

ویژگی MaxLength بر روی طول فیلد تعریف شده در بانک اطلاعاتی تاثیر دارد. برای مثال در اینجا فیلد Title از نوع nvarchar با طول 30 تعریف خواهد شد.
ویژگی MinLength در بانک اطلاعاتی معنایی ندارد.
هر دوی این ویژگی‌ها در پروسه اعتبار سنجی اطلاعات مدل دریافتی تاثیر دارند. برای مثال در اینجا اگر طول عنوان کمتر از 4 حرف باشد، یک استثنای اعتبارسنجی صادر خواهد شد.

ویژگی دیگری نیز به نام StringLength وجود دارد که جهت تعیین حداکثر طول رشته‌ها به کار می‌رود. این ویژگی سازگاری بیشتر با ASP.NET MVC دارد از این جهت که Client side validation آن‌را نیز فعال می‌کند.

4) Table و Column

این دو ویژگی نیز در فضای نام System.ComponentModel.DataAnnotations قرار دارند، اما در اسمبلی EntityFramework.dll تعریف شده‌اند. بنابراین اگر تعاریف مدل‌های شما در پروژه Class library جداگانه‌ای قراردارند، نیاز خواهد بود تا ارجاعی را به اسمبلی EntityFramework.dll نیز داشته باشند.

اگر از نام پیش فرض جداول تشکیل شده خرسند نیستید، ویژگی Table را بر روی یک کلاس قرار داده و نام دیگری را تعریف کنید. همچنین اگر Schema کاربری رشته اتصالی به بانک اطلاعاتی شما dbo نیست، باید آن‌را در اینجا صریحا ذکر کنید تا کوئری‌های تشکیل شده به درستی بر روی بانک اطلاعاتی اجرا گردند:

```
[Table("tblProject", Schema="guest")]
public class Project
```

توسط ویژگی Column سه خاصیت یک فیلد بانک اطلاعاتی را می‌توان تعیین کرد:

```
[Column("DateStarted", Order = 4, TypeName = "date")]
public DateTime AddDate { set; get; }
```

به صورت پیش فرض، خاصیت فوق با همین نام AddDate در بانک اطلاعاتی ظاهر می‌گردد. اگر برای مثال قرار است از یک بانک اطلاعاتی قدیمی استفاده شود یا قرار نیست از شیوه نامگذاری خواص در سی شارپ در یک بانک اطلاعاتی پیروی شود، توسط ویژگی Column می‌توان این تعاریف را سفارشی نمود.

توسط پارامتر Order آن که از صفر شروع می‌شود، ترتیب قرارگیری فیلدها در حین تشکیل یک جدول مشخص می‌گردد.

اگر نیاز است نوع فیلد تشکیل شده را نیز سفارشی سازی نمائید، می‌توان از پارامتر TypeName استفاده کرد. برای مثال در اینجا علاقمندیم از نوع date مهیا در SQL Server 2008 استفاده کنیم و نه از نوع datetime پیش فرض آن.

نکته‌ای در مورد Order:

Order پیش فرض تمام خواصی که قرار است به بانک اطلاعاتی نگاشت شوند، به int.MaxValue تنظیم شده‌اند. به این معنا که تنظیم فوق با Order=4 سبب خواهد شد تا این فیلد، پیش از تمام فیلدهای دیگر قرار گیرد. بنابراین نیاز است Order اولین خاصیت تعریف شده را به صفر تنظیم نمود. (البته اگر واقعا نیاز به تنظیم دستی Order داشتید)

نکاتی در مورد تنظیمات ارث بری در حالت استفاده از متادیتا:

حداقل سه حالت ارث بری را در EF code first می‌توان تعریف و مدیریت کرد:

الف) Table per Hierarchy - TPH

حالت پیش فرض است. نیازی به هیچگونه تنظیمی ندارد. معنای آن این است که «لطفا تمام اطلاعات کلاس‌هایی را که از هم ارث بری کرده‌اند در یک جدول بانک اطلاعاتی قرار بده». فرض کنید یک کلاس پایه شخص را دارید که کلاس‌های بازیکن و مربی از آن ارث بری می‌کنند. زمانیکه کلاس پایه شخص توسط DbSet در کلاس مشتق شده از DbContext در معرض استفاده EF قرار می‌گیرد، بدون نیاز به هیچ تنظیمی، تمام این سه کلاس، تبدیل به یک جدول شخص در بانک اطلاعاتی خواهند شد. یعنی یک table به ازای سلسله مراتبی (Hierarchy) که تعریف شده.

ب) Table per Type - TPT

به این معنا است که به ازای هر نوع، باید یک جدول تشکیل شود. به عبارتی در مثال قبل، یک جدول برای شخص، یک جدول برای مربی و یک جدول برای بازیکن تشکیل خواهد شد. دو جدول مربی و بازیکن با یک کلید خارجی به جدول شخص مرتبط می‌شوند.

تنها تنظیمی که در اینجا نیاز است، قرار دادن ویژگی Table بر روی نام کلاس‌های بازیکن و مربی است. به این ترتیب حالت پیش فرض الف (TPH) اعمال نخواهد شد.

ج) Table per Concrete Type - TPC

در این حالت فقط دو جدول برای بازیکن و مربی تشکیل می‌شوند و جدولی برای شخص تشکیل نخواهد شد. خواص کلاس شخص، در هر دو جدول مربی و بازیکن به صورت جداگانه‌ای تکرار خواهد شد. تنظیم این مورد نیاز به استفاده از Fluent API دارد.

توضیحات بیشتر این موارد به همراه مثال، موکول خواهد شد به مباحث استفاده از Fluent API که برای تعریف تنظیمات پیشرفته نگاشت‌ها طراحی شده است. استفاده از متادیتا تنها قسمت کوچکی از توانایی‌های Fluent API را شامل می‌شود.

5) ConcurrencyCheck و Timestamp

هر دوی این ویژگی‌ها در فضای نام System.ComponentModel.DataAnnotations و اسمبلی به همین نام تعریف شده‌اند.

در EF Code first دو راه برای مدیریت مسایل همزمانی وجود دارد:

```
[ConcurrencyCheck]
public string Name { set; get; }

[Timestamp]
public byte[] RowVersion { set; get; }
```

زمانیکه از ویژگی ConcurrencyCheck استفاده می‌شود، تغییر خاصی در سمت بانک اطلاعاتی صورت نخواهد گرفت، اما در برنامه، کوئری‌های update و delete ایی که توسط EF صادر می‌شوند، اینبار اندکی متفاوت خواهند بود. برای مثال برنامه جاری را به نحو زیر تغییر دهید:

```
using System;
using System.Data.Entity;

namespace EF_Sample02
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new Sample2DbInitializer());
            using (var db = new Sample2Context())
            {
                //update
                var user = db.Users.Find(1);
                user.Name = "User name 1";
                db.SaveChanges();
            }
        }
    }
}
```

متد Find بر اساس primary key عمل می‌کند. به این ترتیب، اول رکورد یافت شده و سپس نام آن تغییر کرده و در ادامه، اطلاعات ذخیره خواهند شد.

اکنون اگر توسط SQL Server Profiler کوئری update حاصل را بررسی کنیم، به نحو زیر خواهد بود:

```
exec sp_executesql N'update [dbo].[Users]
set [Name] = @0
where (([Id] = @1) and ([Name] = @2))
',N'@0 nvarchar(max) ,@1 int,@2 nvarchar(max) ',@0=N'User name 1',@1=1,@2=N'Vahid'
```

همانطور که ملاحظه می‌کنید، برای به روز رسانی فقط از primary key جهت یافتن رکورد استفاده نکرده، بلکه فیلد Name را نیز دخالت داده است. از این جهت که مطمئن شود در این بین، رکوردی که در حال به روز رسانی آن هستیم، توسط کاربر دیگری در شبکه تغییر نکرده باشد و اگر در این بین تغییری رخ داده باشد، یک استثناء صادر خواهد شد.
همین رفتار در مورد delete نیز وجود دارد:

```
//delete
var user = db.Users.Find(1);
db.Users.Remove(user);
db.SaveChanges();
```

که خروجی آن به صورت زیر است:

```
exec sp_executesql N'delete [dbo].[Users]
where (([Id] = @0) and ([Name] = @1))',N'@0 int,@1 nvarchar(max) ',@0=1,@1=N'Vahid'
```

در اینجا نیز به علت مزین بودن خاصیت Name به ویژگی ConcurrencyCheck، فقط همان رکوردی که یافت شده باید حذف شود و نه نمونه تغییر یافته آن توسط کاربری دیگر در شبکه.
البته در این مثال شاید این پروسه تنها چند میلی ثانیه به نظر برسد. اما در برنامه‌ای با رابط کاربری، شخصی ممکن است اطلاعات یک رکورد را در یک صفحه دریافت کرده و 5 دقیقه بعد بر روی دکمه save کلیک کند. در این بین ممکن است شخص دیگری در شبکه همین رکورد را تغییر داده باشد. بنابراین اطلاعاتی را که شخص مشاهده می‌کند، فاقد اعتبار شده‌اند.

ConcurrencyCheck را بر روی هر فیلدی می‌توان بکاربرد، اما ویژگی Timestamp کاربرد مشخص و محدودی دارد. باید به خاصیتی از نوع byte array اعمال شود (که نمونه‌ای از آن‌را در بالا در خاصیت public byte []RowVersion مشاهده نمودید). علاوه بر آن، این ویژگی بر روی بانک اطلاعاتی نیز تاثیر دارد (نوع فیلد را در SQL Server تبدیل به timestamp می‌کند و نه از نوع varbinary مانند فیلد تصویر). SQL Server با این نوع فیلد به خوبی آشنا است و قابلیت مقدار دهی خودکار آن‌را دارد. بنابراین نیازی نیست در حین تشکیل اشیاء در برنامه، قید شود.
پس از آن، این فیلد مقدار دهی شده به صورت خودکار توسط بانک اطلاعاتی، در تمام updateها و deleteهای EF Code first حضور خواهد داشت:

```
exec sp_executesql N'delete [dbo].[Users]
where ((([Id] = @0) and ([Name] = @1)) and ([RowVersion] = @2))',N'@0 int,@1 nvarchar(max) ,
@2 binary(8)',@0=1,@1=N'Vahid',@2=0x00000000000007D1
```

از این جهت که اطمینان حاصل شود، واقعا مشغول به روز رسانی یا حذف رکوردی هستیم که در ابتدای عملیات از بانک اطلاعاتی دریافت کرده‌ایم. اگر در این بین RowVesrion تغییر کرده باشد، یعنی کاربر دیگری در شبکه این رکورد را تغییر داده و ما در حال حاضر مشغول به کار با رکوردی غیرمعتبر هستیم.
بنابراین استفاده از Timestamp را می‌توان به عنوان یکی از best practices طراحی برنامه‌های چند کاربره ASP.NET درنظر داشت.

6) NotMapped و DatabaseGenerated

این دو ویژگی نیز در فضای نام System.ComponentModel.DataAnnotations قرار دارند، اما در اسمبلی EntityFramework.dll تعریف شده‌اند.
به کمک ویژگی DatabaseGenerated، مشخص خواهیم کرد که این فیلد قرار است توسط بانک اطلاعاتی تولید شود. برای مثال خواصی از نوع public int Id به صورت خودکار به فیلدهایی از نوع identity که توسط بانک اطلاعاتی تولید می‌شوند، نگاشت خواهند شد و نیازی نیست تا به صورت صریح از ویژگی DatabaseGenerated جهت مزین سازی آن‌ها کمک گرفت. البته اگر

علاقمند نیستید که primary key شما از نوع identity باشد، می‌توانید از گزینه DatabaseGeneratedOption.None استفاده نمائید:

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
public int Id { set; get; }
```

DatabaseGeneratedOption در اینجا یک enum است که به نحو زیر تعریف شده است:

```
public enum DatabaseGeneratedOption
{
        None = 0,
        Identity = 1,
        Computed = 2
}
```

تا اینجا حالت‌های None و Identity آن، بحث شدند.
در SQL Server امکان تعریف فیلدهای محاسباتی و Computed با T-SQL نویسی نیز وجود دارد. این نوع فیلدها در هربار insert یا update یک رکورد، به صورت خودکار توسط بانک اطلاعاتی مقدار دهی می‌شوند. بنابراین اگر قرار است خاصیتی به این نوع فیلدها در SQL Server نگاشت شود، می‌توان از گزینه DatabaseGeneratedOption.Computed استفاده کرد.
یا اگر برای فیلدی در بانک اطلاعاتی default value تعریف کرده‌اید، مثلا برای فیلد date متد getdate توکار SQL Server را به عنوان پیش فرض درنظر گرفته‌اید و قرار هم نیست توسط برنامه مقدار دهی شود، باز هم می‌توان آن‌را از نوع DatabaseGeneratedOption.Computed تعریف کرد.
البته باید درنظر داشت که اگر خاصیت DateTime تعریف شده در اینجا به همین نحو بکاربرده شود، اگر مقداری برای آن در حین تعریف یک وهله جدید از کلاس User درکدهای برنامه درنظر گرفته نشود، یک مقدار پیش فرض حداقل به آن انتساب داده خواهد شد (چون value type است). بنابراین نیاز است این خاصیت را از نوع nullable تعریف کرد (public DateTime? AddDate).

همچنین اگر یک خاصیت محاسباتی در کلاسی به صورت ReadOnly تعریف شده است (توسط کدهای مثلا سی شارپ یا وی بی):

```
[NotMapped]
public string FullName
{
    get { return Name + " " + LastName; }
}
```

بدیهی است نیازی نیست تا آن‌را به یک فیلد بانک اطلاعاتی نگاشت کرد. این نوع خواص را با ویژگی NotMapped می‌توان مزین کرد.
همچنین باید دقت داشت در این حالت، از این نوع خواص دیگر نمی‌توان در کوئری‌های EF استفاده کرد. چون نهایتا این کوئری‌ها قرار هستند به عبارات SQL ترجمه شوند و چنین فیلدی در جدول بانک اطلاعاتی وجود ندارد. البته بدیهی است امکان تهیه کوئری LINQ to Objects (کوئری از اطلاعات درون حافظه) همیشه مهیا است و اهمیتی ندارد که این خاصیت درون بانک اطلاعاتی معادلی دارد یا خیر.

7) ComplexType

ComplexType یا Component mapping مربوط به حالتی است که شما یک سری خواص را در یک کلاس تعریف می‌کنید، اما قصد ندارید این‌ها واقعا تبدیل به یک جدول مجزا (به همراه کلید خارجی) در بانک اطلاعاتی شوند. می‌خواهید این خواص دقیقا در همان جدول اصلی کنار مابقی خواص قرار گیرند؛ اما در طرف کدهای ما به شکل یک کلاس مجزا تعریف و مدیریت شوند.
یک مثال:
کلاس زیر را به همراه ویژگی ComplexType به برنامه مطلب جاری اضافه نمائید:

```
using System.ComponentModel.DataAnnotations;

namespace EF_Sample02.Models
{
    [ComplexType]
    public class InterestComponent
    {
        [MaxLength(450, ErrorMessage = "حداکثر 450 حرف")]
        public string Interest1 { get; set; }

        [MaxLength(450, ErrorMessage = "حداکثر 450 حرف")]
        public string Interest2 { get; set; }
    }
}
```

سپس خاصیت زیر را نیز به کلاس User اضافه کنید:

```
public InterestComponent Interests { set; get; }
```

همانطور که ملاحظه می‌کنید کلاس InterestComponent فاقد Id است؛ بنابراین هدف از آن تعریف یک Entity نیست و قرار هم نیست در کلاس مشتق شده از DbContext تعریف شود. از آن صرفا جهت نظم بخشیدن به یک سری خاصیت مرتبط و هم‌خانواده استفاده شده است (مثلا آدرس یک، آدرس 2، تا آدرس 10 یک شخص، یا تلفن یک تلفن 2 یا موبایل 10 یک شخص).
اکنون اگر پروژه را اجرا نمائیم، ساختار جدول کاربر به نحو زیر تغییر خواهد کرد:

```
CREATE TABLE [dbo].[Users](
 ---...
 [Interests_Interest1] [nvarchar](450) NULL,
 [Interests_Interest2] [nvarchar](450) NULL,
 ---...
```

در اینجا خواص کلاس InterestComponent، داخل همان کلاس User تعریف شده‌اند و نه در یک جدول مجزا. تنها در سمت کدهای ما است که مدیریت آن‌ها منطقی‌تر شده‌اند.

یک نکته:
یکی از الگوهایی که حین تشکیل مدل‌های برنامه عموما مورد استفاده قرار می‌گیرد، null object pattern نام دارد. برای مثال:

```
namespace EF_Sample02.Models
{
    public class User
    {
        public InterestComponent Interests { set; get; }
        public User()
        {
            Interests = new InterestComponent();
        }
    }
}
```

در اینجا در سازنده کلاس User، به خاصیت Interests وهله‌ای از کلاس InterestComponent نسبت داده شده است. به این

ترتیب دیگر در کدهای برنامه مدام نیازی نخواهد بود تا بررسی شود که آیا Interests نال است یا خیر. همچنین استفاده از این الگو حین کار با یک ComplexType ضروری است؛ زیرا EF امکان ثبت رکورد جاری را در صورت نال بودن خاصیت Interests (صرفنظر از اینکه خواص آن مقدار دهی شده‌اند یا خیر) نخواهد داد.

8) ForeignKey

این ویژگی نیز در فضای نام System.ComponentModel.DataAnnotations قرار دارد، اما در اسمبلی EntityFramework.dll تعریف شده‌است.

اگر از قراردادهای پیش فرض نامگذاری کلیدهای خارجی در EF Code first خرسند نیستید، می‌توانید توسط ویژگی ForeignKey، نامگذاری مورد نظر خود را اعمال نمائید. باید دقت داشت که ویژگی ForeignKey را باید به یک Reference property اعمال کرد. همچنین در این حالت، کلید خارجی را با یک value type نیز می‌توان نمایش داد:

```
[ForeignKey("FK_User_Id")]
public virtual User User { set; get; }
public int FK_User_Id { set; get; }
```

در اینجا فیلد اضافی دوم FK_User_Id به جدول Project اضافه نخواهد شد (چون توسط ویژگی ForeignKey تعریف شده است و فقط یکبار تعریف می‌شود). اما در این حالت نیز وجود Reference property ضروری است.

9) InverseProperty

این ویژگی نیز در فضای نام System.ComponentModel.DataAnnotations قرار دارد، اما در اسمبلی EntityFramework.dll تعریف شده‌است.

از ویژگی InverseProperty برای تعریف روابط دو طرفه استفاده می‌شود.

برای مثال دو کلاس زیر را درنظر بگیرید:

```
public class Book
{
    public int ID {get; set;}
    public string Title {get; set;}

    [InverseProperty("Books")]
    public Author Author {get; set;}
}

public class Author
{
    public int ID {get; set;}
    public string Name {get; set;}

    [InverseProperty("Author")]
    public virtual ICollection<Book> Books {get; set;}
}
```

این دو کلاس همانند کلاس‌های User و Project فوق هستند. ذکر ویژگی InverseProperty برای مشخص سازی ارتباطات بین این دو غیرضروری است و قراردادهای توکار EF Code first یک چنین مواردی را به خوبی مدیریت می‌کنند.

اما اکنون مثال زیر را درنظر بگیرید:

```
public class Book
{
    public int ID {get; set;}
    public string Title {get; set;}

    public Author FirstAuthor {get; set;}
    public Author SecondAuthor {get; set;}
}

public class Author
{
```

```
    public int ID {get; set;}
    public string Name {get; set;}

    public virtual ICollection<Book> BooksAsFirstAuthor {get; set;}
    public virtual ICollection<Book> BooksAsSecondAuthor {get; set;}
}
```

این مثال ویژه‌ای است از کتابخانه‌ای که کتاب‌های آن، تنها توسط دو نویسنده نوشته شده‌اند. اگر برنامه را بر اساس این دو کلاس اجرا کنیم، EF Code first قادر نخواهد بود تشخیص دهد، روابط کدام به کدام هستند و در جدول Books چهار کلید خارجی را ایجاد می‌کند. برای مدیریت این مساله و تعین ابتدا و انتهای روابط می‌توان از ویژگی InverseProperty کمک گرفت:

```
public class Book
{
    public int ID {get; set;}
    public string Title {get; set;}

    [InverseProperty("BooksAsFirstAuthor")]
    public Author FirstAuthor {get; set;}
    [InverseProperty("BooksAsSecondAuthor")]
    public Author SecondAuthor {get; set;}
}

public class Author
{
    public int ID {get; set;}
    public string Name {get; set;}

    [InverseProperty("FirstAuthor")]
    public virtual ICollection<Book> BooksAsFirstAuthor {get; set;}
    [InverseProperty("SecondAuthor")]
    public virtual ICollection<Book> BooksAsSecondAuthor {get; set;}
}
```

اینبار اگر برنامه را اجرا کنیم، بین این دو جدول تنها دو رابطه تشکیل خواهد شد و نه چهار رابطه؛ چون EF اکنون می‌داند که ابتدا و انتهای روابط کجا است. همچنین ذکر ویژگی InverseProperty در یک سر رابطه کفایت می‌کند و نیازی به ذکر آن در طرف دوم نیست.

نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۹:۰۸:۳۲

سلام . آقای نصیری بابت زحمات متشکر خیلی خیلی.
یه سوال. برای سیلورلایت و استفاده از قابلیت های بایندینگ اون ، مدلها باید INotifyPropertyChanged رو پیاده سازی کنن ولی در Code First پیاده سازی نشده ، آیا در ادامه شرح میدید یا باید خودمون دستی اونو پیاده سازی کنیم؟
یاحق.

نویسنده: Ali
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۹:۳۲:۱۷

خیلی عالی. امیدوارم به زودی شاهد کتاب ام.وی.سی و ای.اف شما باشیم. (کتاب چاپ شده! البته)

نویسنده: Sirwan Afifi
تاریخ: ۱۳۹۱/۰۲/۱۶ ۲۲:۲۷:۳۷

خیلی ممنون

واقعا عالی بود هرچند از اول بصورت کامل نخوندم ولی این سری آموزش هاتون واقعا کیفیتش عالیه،برای من هم که مبتدی هستم خیلی خوب و قدم به قدم توضیح دادید.خیلی ممنون

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۱۶ ۲۲:۴۶:۱۱

سلام
به یاد دارم قبلا NH را به عنوان قویترین ORM موجود آموزش می دادید.
با توجه به ویژگی های EF 5 قبول دارید در حال حاضر EF قویترین ORM موجود در دنیای Developing است؟
آیا نقطه ضعف یا کمبودی شما در آن مشاهده می کنید؟

نویسنده: AhmadalliShafiee
تاریخ: ۱۳۹۱/۰۲/۱۶ ۲۳:۵۰:۳۶

با سلام
۲ تا سوال داشتم: اول این‌که دوست‌دارم از EF Code First توی نرم‌افزارهای ویندوزی استفاده کنم ولی راه‌حلی براش پیدا نکردم(NuGet فقط توی Visual Web Developer کار میکنه)
دوم این‌که امکانش وجود داره که مجموعه آموزش‌های MVCتون را به صورت یک فایل PDF در سایت قرار بدید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۷ ۰۰:۰۷:۲۹

- بله. تعاریف کلاس رو که دارید. این‌ها رو هم باید دستی اضافه کنید.
- در قسمت اول اشاره کردم به db.Blogs.Local . این خاصیت Local از نوع ObservableCollection است که در برنامه‌های WPF و Silverlight می‌تونه جذاب باشه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۷ ۰۰:۱۳:۰۶

- NuGet فقط یک ابزار دریافت و افزودن خودکار اسمبلی‌ها به پروژه است. کاری به برنامه وب یا ویندوز ندارد. حتی نیازی به

ویژوال استودیو هم ندارد. از طریق خط فرمان هم قابل اجرا است: (^)
- فایل CHM سایت در دسترس هست. بالای سایت قسمت گزیده‌ها. خلاصه وبلاگ.

نویسنده: m_dabirsiaghi
تاریخ: ۱۳۹۱/۰۴/۲۳ ۱۰:۵۶

آقای نصیری سپاسگزار از بابت مطالب

نویسنده: فرید صالحی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۹:۴۳

ممکنه این سوال مستقیما به اینجا مربوط نشه، اما به هر حال اینجا هم خودشو نشون میده. چرا امکان دسترسی به نام propertyها به صورت strongly type وجود نداره؟ آیا تو پیاده سازی مشکلی داره؟
مثلا در مثال inverse property، باید اسم فیلد معادل به صورت رشته ای ذکر بشه. حالا اگه این اسم تغییر کرد چطور باید ردیابی بشه.
این مساله به فرض تو بایندینگ ها، مثلا برای dropdownlist ، هم یه مقدار آدم رو نگران میکنه و یکی از ویژگی‌های مثبت استفاده از Linq رو که وجود intelisence و بررسی در زمان کامپایل هست نقض میکنه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۷ ۹:۵۶

در قسمت جاری زمانی که با attributes کار می‌کنید، محدود هستید به امکانات زبان مورد استفاده. در تعریف و مقدار دهی ویژگی‌ها امکان استفاده از lambda expressions وجود ندارد و مقادیر تعریف شده در آن باید در زمان کامپایل ثابت باشند.
+
قسمت‌های بعدی رو که مطالعه کنید به روش دوم تعریف‌های نگاشت‌ها به نام **Fluent API** خواهید رسید. در آنجا همه چیز strongly typed است.

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۶/۲۸ ۹:۳۵

اگر بخواهیم فیلدی به اسم Id کلید جدول باشد ولی Identity نباشد چکار باید کرد؟
من میخوام یک سری دیتا رو از یک تیبل دیتابیس قدیمی، منتقل کنم به دیتابیس جدیدم ولی اگر Identity باشه نمیشه دیتا رو Paste کرد توی تیبل دیتابیس جدید.
دیتابیس من SQL CE 4.0 هستش. ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۸ ۹:۴۵

- در متن فوق قسمت ششم توضیح داده شده: «اگر علاقمند نیستید که primary key شما از نوع identity باشد، می‌توانید از گزینه DatabaseGeneratedOption.None استفاده نمائید»
- ضمنا این روش کار نیست برای انتقال اطلاعات. اگر از sql server 2008 استفاده می‌کنید، امکان تهیه خروجی به صورت اسکریپت را دارد. یکی از نکاتی که در این اسکریپت لحاظ می‌شود، دو دستور IDENTITY_INSERT زیر است که با SQL CE هم کار می‌کند:

```
SET IDENTITY_INSERT [table1] ON;
GO
INSERT INTO [table1] ([Id],...) VALUES (1,...);
GO
SET IDENTITY_INSERT [table1] OFF;
GO
```

برای اجرای اسکریپت نهایی می‌تونید از sql ce toolbox استفاده کنید.

نویسنده: kia
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۶:۴۷

در مورد **مسئله همزمانی** بهترین راهکار چیست از نظر شما؟ (منظور در همین EF هست)
استفاده از ConcurrencyChe ck یا Timestamp و به چه صورتی؟ (فرقشون رو از لحاظ فنی می‌دونم, اینکه کدوم رو در کجا و چه مسائلی باید استفاده کرد رو می‌خوام بدونم)

مثلا استفاده از فیلدی جداگانه (مثلا LastModifiedTime) در جداول مهم که امکان تداخل همزمانی در شبکه را دارند, و مزین کردن این فیلد با [ConcurrencyCheck]؟
یا ستونهای جدول رو همگی مزین کنیم به [ConcurrencyCheck]؟
یا یک فیلد از جنس timestamp تعریف کنیم؟
یا جور دیگه ای حل کنیم این قضیه رو در جاهای مختلف؟

ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۷:۱۰

بهترین راه حل استفاده از ویژگی Timestamp بر روی خاصیتی مانند RowVersion است که در متن با مثال و خروجی SQL متناظر توضیح داده شد. مقداری که در این فیلد به صورت خودکار مدیریت شونده، ذخیره می‌شود تاریخ یا زمان نیست. یک عدد ترتیبی است که با هر با آپدیت رکورد، افزایش می‌یابد. بنابراین به صورت خودکار بر روی تمام فیلدها اعمال می‌شود و زحمت تعریف و مدیریت آن از ConcurrencyCheck کمتر و نهایتا سریعتر است.
یک مثال کامل در مورد نحوه استفاده از آن.

نویسنده: علی
تاریخ: ۱۳۹۲/۰۱/۰۷ ۹:۴۷

با سلام
شما اشاره کردید
" مرسوم است کلاس‌های مدل را در یک class library جداگانه اضافه کنند به نام DomainClasses و کلاس‌های مرتبط با DbContext را در پروژه class library دیگری به نام DataLayer "
اگر امکان دارد یک توضیح مختصری راجب پیاده سازی معماری 3 لایه برای همین مثال (BlogوPost) بدید
مثلا برای افزودن یک پست باید یک متد به کلاس Post اضافه کنم یا مکان آن در جایی دیگر است ؟ منطق سیستم را کجا قرار بدم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۷ ۹:۵۰

در قسمت 12 این سری توضیح داده شده به تفصیل.

نویسنده: بهروز
تاریخ: ۱۳۹۲/۰۱/۱۷ ۱۱:۷

با سلام
اگه بخوام که همین کلاس User فیلد Id آن کلید باشد ولی Identity نباشد چه کار باید انجام دهیم لطفا به هر دو صورت Meta Data و Fluent API توضیح دهید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۷ ۱۱:۴۷

متن رو یکبار کامل مطالعه کنید: «...اگر علاقمند نیستید که primary key شما از نوع identity باشد، می‌توانید از گزینه DatabaseGeneratedOption.None استفاده نمائید ... »

نویسنده: میثم خوشقدم
تاریخ: ۱۳۹۲/۰۲/۰۹ ۱۷:۴۲

سلام

خسته نباشید

ضمن تشکر از مطالب پربارتون

سوالی که برای من پیش اومده این است که در پروژه خوب است که یک کلاس Dbocontextداشته باشیم و تمام جداول در آن تعریف بشوند و یا برای یک یا گروهی از جداول DbContext مجزا داشته باشیم؟

اگر در مواردی خوب است که چند DbContext داشته باشیم چگونه به همه DbContextها یک کانکشن بدون تحریف متد Base اون‌ها ست کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۹ ۱۷:۴۹

یک کلاس DbContext باید داشته باشید:
تمام مباحث ردیابی تغییرات EF در یک context کار می‌کنند (در یک قسمت مجزا به این موضوع پرداخته شده).
همچنین به روز رسانی خودکار ساختار بانک اطلاعاتی هم بر اساس اطلاعات یک context صورت می‌گیرد؛ بر این اساس، یک هش را در بانک اطلاعاتی در جدولی خاص ذخیره خواهد کرد و هر بار این هش را با هش اطلاعات context موجود مقایسه می‌کند.
ضمن اینکه در قسمت 11 این سری به مفهومی به نام unit of work پرداخته شده. در EF کلاس DbContext پیاده سازی کننده الگوی واحد کار است.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۲/۱۰ ۹:۵۵

در مواردی که تعداد جداول زیاد باشند، یکی گرفتن DBContext کارایی رو پایین نمیاره؟ به خصوص اگه entityها با روابط ارث بری و Self referencing توی مدلمون وجود داشته باشن. برای این موارد چه راهی وجود داره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۰:۱۱

- تا EF 5.0 اینطوری طراحی شده و طراحی صحیحی هم هست؛ چون از دیدگاه الگوی واحد کار شما در آن واحد نیاز خواهید داشت در یک تراکنش با چندین موجودیت کار کنید. نه اینکه تعدادی موجودیت در یک تراکنش و دیگری در تراکنشی دیگر قرار داشته باشند.
- این مساله تاثیری روی کارآیی ندارد. چون تمام روابط در آغاز برنامه خوانده شده و کش می‌شوند. تنها تاثیری که تعداد مدل‌های

زیاد دارند، کند کردن آغاز برنامه است (همان زمان کش کردن اولیه). راه حل برای آن وجود دارد ؛ همچنین این مساله در EF6 که به زودی منتشر خواهد شد به صورت جداگانه‌ای بررسی و بهبود کلی داده شده است.

نویسنده: میثم خوشقدم
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۳:۴۲

در مورد SimpleMembership چطور؟
پروژه پیش فرض Visual Studio آبجکت DbContext رو به صورت زیر ست می‌کند.

```
Database.SetInitializer<UsersContext>(null);
```

ست کردن آن با DbMigration در آینده مشکلی ایجاد نمی‌کند و یا در شیوه فراخوانی SimpleMigration

خود مایکروسافت در مثال خود چرا از Migration استفاده نکرده است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۳:۵۲

این تنظیمات مرتبط است به غیرفعال سازی مباحث Migration جهت اعمال دستی اسکریپت تولیدی آن‌ها؛ برای توضیحات مرتبط با آن مراجعه کنید به انتهای قسمت پنجم در مورد «استفاده از DB Migrations در عمل». این تنظیم، ارتباطی به تشکیل روابط بین کلاس‌های مدل‌های برنامه در ابتدای کار آن ندارد.
حتی در حالت دستی هم پاورشل، اطلاعات را از DbContext دریافت و با ساختار بانک اطلاعاتی مقایسه می‌کند. سپس بر این اساس می‌تواند فایل SQL قابل اجرای بر روی بانک اطلاعاتی را تولید کند.

نویسنده: سید مهدی فاطمی
تاریخ: ۱۳۹۲/۰۵/۰۱ ۲۲:۴۳

چطور من می‌تونم در code first یک جدول بدون کلید داشته باشم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۱ ۲۳:۲۹

کلا در EF (تمام نگارش‌ها و حالت‌های مختلف آن) نمی‌توانید جدول بدون PK داشته باشید چون EF از آن برای سیستم ردیابی و همچنین تولید کوئری‌های به روز رسانی اطلاعات استفاده می‌کند. یک سری راه حل عجیب و غریب هم ممکن است پیدا کنید ولی بهترین کار همان تعریف یک کلید ساده است.

نویسنده: مصطفی حسینی
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۹:۳۱

سلام.
من طبق برنامه و حرف شما در اینجا کد رو به صورت زیر نوشتم :

```
public class Post : BaseEntity
    {
        public new int Id { get; set; }
        public virtual ICollection<Comment.Comment> Comments { get; set; }
```

```
        [NotMapped]
        public int CommentsCount
        {
            get
            {
                if (Comments == null || !Comments.Any())
                    return 0;
                return Comments.Count;
            }
        }

        public Post()
        {
            Comments = new List<Comment.Comment>();
        }
    }
```

و زمان استفاده از آن :

```
var post = _tEntities.Include(p => p.User).Include(p => p.Comments)
            .Select(p => new PostListViewModels
            {
                Id = p.Id,
                Username = p.Username,
                CommentCount = p.CommentsCount
            });
```

خطای زیر صادر میشود :

```
The specified type member 'CommentsCount' is not supported in LINQ to Entities. Only initializers,
entity members, and entity navigation properties are supported
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۲۰:۱۵

بله. علت اینجا است که کوئری‌های LINQ to Entities بر روی دیتابیس اجرا می‌شوند و خاصیت NotMapped شما سمت کلاینت محاسبه خواهد شد. ترکیب این‌دو با هم در select و projection نگارش فعلی EF میسر نیست. اطلاعات خاصیت سمت کلاینت NotMapped فقط پس از فراخوانی ToList و یا AsEnumerable بر روی کوئری انجام شده قابل دسترسی است و نه قبل از آن.

نویسنده: مصطفی حسینی
تاریخ: ۱۳۹۲/۰۵/۲۱ ۲۱:۰

به نظر شما بهتر نیست به جای استفاده از این گونه فیلدها که باید بعد از ToList و یا AsEnumerable استفاده شوند، به شکل زیر به فرض مثال عمل کرد؟ :

```
var post = _tEntities.Include(p => p.User).Include(p => p.Comments).Select(p => new PostListViewModels
{
       Id = p.Id,
       Username = p.Username,
        CommentCount = p.Comments.Count(c => c.IsApproved != true)
});
```

از جهت کوئری SQL ایجاد شده میگم. کل فیلدها رو ابتدا میگیره و بعد Select روی اون انجام میشه.
کدوم راه به نظر شما بهینه‌تر هستش؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۲۱:۳۳

بستگی داره. اگر تمام فیلدها مورد نیاز باشند، روش NotMapped یک sub query کمتر دارد. اگر فقط سه فیلد مدنظر شما باید واکشی شوند، بله؛ محاسبه آن در سمت دیتابیس بهتر است.

نویسنده: rezaei
تاریخ: ۱۳۹۲/۰۸/۲۵ ۹:۴۲

با سلام؛ در database first ما میتونیم به صورت دستی در جداولمون رکورد وارد کنیم مثلا نام کاربری و کلمه عبور مدیر یا برخی جداول که دارای اطلاعات اولیه دارند. در code First ما چطور باید اینکار رو انجام بدیم به نحوی که فقط یکبار مقدار دهی اولیه صورت بگیره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۵ ۹:۴۶

به متد **Seed** protected override void در مطلب جاری و همچنین قسمت‌های بعدی این بحث، دقت کنید.

نویسنده: امیر
تاریخ: ۱۳۹۲/۰۹/۱۷ ۱۰:۴۵

سلام
من DataLayer رو درون یک پروژه class library ایجاد کردم سوالی که دارم اینه که آیا تو تنظیمات کانکشن استرینگ برنامه تو پروژه mvc باید کار خاصی کنم یا فقط با add کردن refrence تو پروژه mvc و نوشتن نام کلاس برای name کافیست؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۷ ۱۱:۱۴

در قسمت اول بحث شده؛ باید نام رشته اتصالی ذکر شده در وب کانفیگ، FullNamespace.DbContextClassName باشد.

نویسنده: حمید حسین وند
تاریخ: ۱۳۹۳/۰۱/۲۵ ۲۳:۳

سلام
آیا روش دیگه برای درج کلید خارجی هست بدون اینکه یک select انجام بدیم و اونو از دیتابیس بخونیم به صورت زیر؟

```
var user = db.Users.FirstOrDefault(x=>x.UserName == "hamid");

db.Post.Add(new Post
{
      Title = txtTitle.Text,
      Content = txtContent.Text,
      User = user
}
db.SaveChanges();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۵ ۲۳:۵۴

- کار با کلیدهای اصلی و خارجی در EF Code first
- چند نکته کاربردی درباره Entity Framework

+ در ذیل هر مطلب، «مطالب مرتبط» و همچنین «ارجاع دهنده‌های داخلی» نیز جهت مطالعه و یافتن پاسخ‌ها بسیار مفید هستند.